UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS
Faculdade de Sistemas de Informação
Bacharelado em Sistemas de Informação

Trabalho de Conclusão de Curso

SIAC: SISTEMA DE APOIO AO CALOURO COM INTEGRAÇÃO AO CHATBOT MICA (MÓDULO INTERATIVO DE CONTEÚDOS ACADÊMICOS)

GUSTAVO PAIXÃO MACHADO,
GABRIEL MORANDI DE MELLO

Marabá - PA
2024

GUSTAVO PAIXÃO MACHADO,
GABRIEL MORANDI DE MELLO

SIAC: SISTEMA DE APOIO AO CALOURO COM INTEGRAÇÃO AO CHATBOT MICA (MÓDULO INTERATIVO DE CONTEÚDOS ACADÊMICOS)

Trabalho de Conclusão de Curso apresentado a Universidade Federal do Sul e Sudeste do Pará, como parte dos requisitos necessários para obtenção do Título de Bacharel de Sistemas de Informação.

Orientador: Prof. Dr. Warley Muricy Valente Junior

Marabá - PA
2024

RESUMO

O primeiro contato do calouro com a universidade apresenta desafios que podem impactar seu desempenho acadêmico e até mesmo levar à evasão. Diante disso, este trabalho propõe desenvolver uma plataforma web de apoio ao calouro, integrada a um chatbot, para auxiliar estudantes ingressantes em suas dúvidas sobre o curso e a universidade. A plataforma centralizará informações essenciais, orientará o aluno sobre conteúdos e disponibilizará recursos de aprendizagem ao longo do curso. O chatbot é uma ferramenta importante para responder dúvidas e será alimentado com informações dos coordenadores dos cursos. Espera-se que essa abordagem contribua para a adaptação dos calouros, reduzindo dificuldades iniciais e fortalecendo o vínculo entre os estudantes e a instituição, promovendo maior sucesso acadêmico e menor evasão.

Palavras-chave: Calouros, Universidade, Plataforma web, Chatbot, Suporte acadêmico,

ABSTRACT

First-year students face challenges in their initial contact with the university, which can impact their academic performance and even lead to dropout. To address this, this work proposes the development of a web-based platform to support freshmen, integrated with a chatbot, to assist incoming students with their questions about the course and the university. The platform will centralize essential information, provide guidance on course content, and offer learning resources throughout the academic journey. The chatbot will be a important tool for answering questions and will be fed with information from course coordinators. It is expected that this approach will contribute to freshmen's adaptation, reducing initial difficulties and strengthening the bond between students and the institution, promoting greater academic success and lower dropout rates.

Keywords: Freshmen, University, Web platform, Chatbot, Academic support

LISTA DE FIGURAS

Figura 1 – CLN e a GLN pertencem a PLN 6
Figura 2 – Resultados da pesquisa no Scopus por ano para “chatbot” ou “agente de conversação” ou “interface de conversação” como palavras-chave de 2000 a 2019. 7
Figura 3 – Arquitetura Geral Chatbot 8
Figura 4 – Rasa Architecture Overview 13
Figura 5 – Talk to Your Bot 14
Figura 6 – Scrum Sprints 20
Figura 7 – Desempenho do Astro comparado a outros frameworks de front-end . . 22
Figura 8 – Gráfico Quantitativo Pergunta Fechada 1 27
Figura 9 – Gráfico Quantitativo Pergunta Fechada 2 27
Figura 10 – Gráfico Quantitativo Pergunta Fechada 3 28
Figura 11 – Quadro de Palavras-chave sobre a Pergunta Q5 28
Figura 12 – Caso de Uso de Alto Nível SIAC 34
Figura 13 – Diagrama UML de Classe Mongodb 36
Figura 14 – Padrão de Arquitetura MVC Aplicado 38
Figura 15 – Diagrama UML de Componentes *Back-end* 39
Figura 16 – Páginas SIAC 41
Figura 17 – Arquivo Figma 42
Figura 18 – Matriz de Confusão - Intents 62
Figura 19 – Matriz de Confusão - Entidades 63
Figura 20 – Arquitetura de Contêineres e Integração com Back-end e Front-end . . 67
Figura 21 – Página Inicial SIAC 70
Figura 22 – Página de Chatbot MICA 71
Figura 23 – Página Inicial SIAC 72
Figura 24 – Página Roadmap Bem-vindo(a) 73
Figura 25 – Página Roadmap Lista de Cursos 73
Figura 26 – Página do Roadmap [Curso] 74
Figura 27 – Página Sobre Chatbot 75
Figura 28 – Página Chatbot Chat 76
Figura 29 – Página de Bem-vindo(a) Coordenador(a) 76
Figura 30 – Página de Login Coordenador(a)] 77
Figura 31 – Página de Alterar Senha Coordenador(a) 78
Figura 32 – Página de Dashboard Coordenador(a) 79
Figura 33 – Página Administrador Bem-vindo 79
Figura 34 – Página de Login Administrador 80

Figura 35 – Página de Dashboard Administrador 81
Figura 36 – Arquitetura Final SIAC 82
Figura 37 – Componente Endpoints e Camada de Distribuição 83
Figura 38 – Camada de Treinamento e Camada do Chatbot 83
Figura 39 – Camada de Ação e Componente de Banco de Dados 84

LISTA DE TABELAS

Tabela 1 – Tabela Comparativa Chatbots 14
Tabela 2 – Tabela Comparativa dos Trabalhos Relacionados. 16
Tabela 3 – Estrutura do Formulário de Pesquisa com Coordenadores 26
Tabela 4 – Categorias e Perguntas Escolhidas do CRCA 30
Tabela 5 – Requisitos Funcionais 31
Tabela 6 – Requisitos Não Funcionais 32
Tabela 7 – Intents e Exemplos de Frases 49
Tabela 8 – Ações Configuradas no Chatbot 50
Tabela 9 – Stories Configuradas no Chatbot 55

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CRCA	*Centro de Registro e Controle Acadêmico*
CSS	*Cascading Style Sheets*
IA	*Inteligência Artificial*
JSON	*JavaScript Object Notation*
ML	*Machine Learning*
MICA	*Módulo Interativo de Conteúdos Acadêmicos*
NLU	*Natural-language Understanding*
ORM	*Object Relational Mapper*
PLN	*Processamento de Linguagem Natural*
PPC	*Projeto Pedagógico do Curso*
REST	*Representational State Transfer*
SDK	*Software Development Kit*
SIAC	*Sistema de Apoio ao Calouro*
SIGAA	*Sistema Integrado de Gestão de Atividades Acadêmicas*
UML	*Unified Modeling Language*
YAML	*YAML Ain't Markup Language*

LISTA DE ALGORITMOS

5.1 Comando Python para criar e utilizar o ambiente virtual 44
5.2 Comando Pip Instalação Rasa 44
5.3 Comando Inicialização Rasa 44
5.4 Saída rasa init 44
5.5 Alteração config.yml 46
5.6 Alteração rules.yml 47
5.7 domain.yml Exemplo de Utter Listada 51
5.8 domain.yml Listagem das Entidades e Slots 52
5.9 action.py Função da action_localizacao_curso 52
5.10 stories.yml 55
5.11 Alteração config.yml 56
5.12 Treinando um modelo Rasa 57
5.13 Saída 1 rasa train 57
5.14 Saída 2 rasa train 58
5.15 Saída 3 rasa train 58
5.16 Saída 4 rasa train 59
5.17 Saída 5 rasa train 59
5.18 Rodando o agente no terminal 60
5.19 Rodando Servidor de Ações 60
5.20 Saída Servidor de Ações 60
5.21 Saída Rasa Shell 60
5.22 Saída 2 Rasa Shell 60
5.23 Comando rasa test 61
5.25 Dockerfile do Servidor Rasa 64

SUMÁRIO

LISTA DE ALGORITMOS . . . .

1 – INTRODUÇÃO **1**
1.1 Motivação 1
1.2 Objetivo Geral 2
1.3 Objetivos Específicos 2
1.4 Estrutura do Trabalho 2

2 – REVISÃO DE LITERATURA **3**
2.1 Adaptação à Universidade 3
2.1.1 A inadaptação e a evasão do ensino superior 4
2.1.2 Acolhimento ao calouro 4
2.1.3 Sistemas de Apoio Acadêmico 5
2.2 Processamento de Linguagem Natural (PLN) 5
2.3 Chatbot 7
2.3.1 Arquitetura Geral e Funcionamento 7
2.3.2 Chatbot na Educação 9
2.3.3 *Benchmark* Chatbots 9
2.3.3.1 IBM Watson Assistant 9
2.3.3.2 Microsoft Bot Framework 10
2.3.3.3 Amazon Lex 10
2.3.3.4 Wit.ai 10
2.3.3.5 Botpress 10
2.3.3.6 Dialogflow 10
2.3.3.7 Rasa 11
2.3.3.7.1 Conceitos Básicos do Rasa 11
2.3.3.7.2 Rasa Open Source vs Rasa Pro 13
2.3.3.8 Tabela Comparativa Chatbots 14

3 – TRABALHOS RELACIONADOS **16**

4 – METODOLOGIA **18**
4.1 Metodologia de Projeto 18
4.1.1 Metodologias Ágeis no Desenvolvimento de Softwares 18
4.1.2 Metodologia Scrum 19
4.1.3 Metodologia Scrum Aplicada ao Projeto 19
4.2 Tecnologias e Ambientes de Desenvolvimento 20

4.2.1 Pesquisa e Modelagem . . . 21
4.2.2 Design e Prototipagem . . . 21
4.2.3 *Front-end* . . . 21
4.2.4 *Back-end* . . . 22
4.2.5 Chatbot . . . 24
4.2.6 Ambientes de Desenvolvimento . . . 24

5 – PROPOSTA . . . 25
5.1 Execução e Acompanhamento das Sprints . . . 25
5.1.1 Levantamento de Requisitos . . . 25
5.1.1.1 Requisitos Funcionais e Não Funcionais; . . . 25
5.1.1.2 Pesquisa Coordenadores . . . 26
5.1.1.3 Pesquisa Discentes . . . 29
5.1.1.4 Tabela de Requisitos . . . 30
5.1.2 Modelagem . . . 32
5.1.2.1 Caso de uso de Alto Nível . . . 33
5.1.2.2 Banco de Dados . . . 34
5.1.2.2.1 Modelagem UML do Banco de Dados . . . 35
5.1.2.3 Padrão de Arquitetura MVC . . . 37
5.1.2.3.1 Diagrama de Componentes *Back-end* . . . 38
5.1.3 Prototipação . . . 39
5.1.3.1 Fluxos de Usuários . . . 39
5.1.3.2 Sitemap SIAC . . . 40
5.1.3.3 Prototipação das Telas . . . 41
5.1.3.4 Protótipos Resultantes . . . 42
5.1.4 Desenvolvimento Back-end . . . 42
5.1.4.1 Estrutura do Repositório . . . 42
5.1.4.2 Rotas da API . . . 43
5.1.5 Configuração Chatbot . . . 43
5.1.5.1 Ambiente Virtual . . . 43
5.1.5.2 Iniciando Projeto Rasa . . . 44
5.1.5.3 Inicialização do Projeto Rasa . . . 44
5.1.5.4 Alterando Configurações Padrões . . . 46
5.1.5.4.1 Arquivo de Configurações . . . 46
5.1.5.4.2 Arquivo de Regras . . . 47
5.1.5.5 Definição de Intents e Criação das Actions e Stories . . . 47
5.1.5.5.1 Intents: . . . 47
5.1.5.5.2 Criação das Actions . . . 49
5.1.5.5.3 Desenvolvimento de Stories: . . . 54
5.1.5.6 Configuração dos Endpoints . . . 56

5.1.5.7 Treinamento do Modelo . . . 57
5.1.5.7.1 Validação Inicial do Modelo: . . . 57
5.1.5.7.2 Treinamento dos Componentes da Pipeline: . . . 58
5.1.5.7.3 Métricas de Treinamento: . . . 58
5.1.5.7.4 Treinamento das Políticas: . . . 59
5.1.5.7.5 Conclusão do Treinamento: . . . 59
5.1.5.8 Rodando e Testando o Rasa . . . 59
5.1.5.8.1 Rodando o Agente . . . 59
5.1.5.8.2 Testando o Modelo . . . 61
5.1.6 Containerização com Docker . . . 63
5.1.6.0.1 Containers . . . 63
5.1.6.0.2 Estrutura de Containers para o Chatbot . . . 64
5.1.6.0.3 Configuração do `docker-compose.yml` . . . 66
5.1.6.0.4 Vantagens da Conteinerização no Chatbot . . . 67
5.1.7 Integração do Back-end com o Chatbot . . . 68
5.1.7.0.1 Rotas Abertas: . . . 68
5.1.7.0.2 Rotas de Administração: . . . 68
5.1.7.0.3 Rotas dos Coordenadores: . . . 68
5.1.8 Desenvolvimento Front-end e Integração com o Back-end . . . 69
5.1.8.1 Estrutura do Repositório . . . 69
5.1.8.2 Desenvolvimento com Astro . . . 70
5.1.8.2.1 Componentização . . . 70
5.1.8.2.2 Integração com o Back-end . . . 71
5.1.8.2.3 Apresentação de Páginas Desenvolvidas . . . 71
5.2 Arquitetura Resultante . . . 81

6 – CONSIDERAÇÕES FINAIS . . . 85
6.1 Contribuições . . . 85
6.2 Limitações encontradas . . . 85
6.3 Trabalhos Futuros . . . 86
6.3.1 Aprimoramento do SIAC . . . 86
6.3.2 Aprimoramento da MICA . . . 86
6.3.3 Continuidade das Sprints . . . 86
6.4 Conclusão do Trabalho . . . 87

Referências . . . 88

1 INTRODUÇÃO

O primeiro contato do calouro com a universidade tem grande importância ao decorrer do curso, segundo (PRICKETT et al., 2020) os primeiros anos da graduação, sobretudo em um contexto de Computação, são os mais complexos, ao serem onde se tem as matérias fundamentais, e pode-se observar-se a diferenciação de disciplinas comparada ao ensino básico. A partir disso, é construída toda a sua relação com o curso escolhido e junto desta, vem os desafios para adaptação à vida acadêmica — distintos dos existentes no ensino básico —, como a utilização de sistemas de apoio acadêmico, a obrigatoriedade no cumprimento de atividades complementares, estágios e práticas de extensões.

Nas palavras de (TEIXEIRA et al., 2008) o acompanhamento e diligência pelo aluno é reduzida no ambiente universitário comparada ao ensino básico, e apesar de muitas vezes, a universidade apresentar todo o funcionamento relacionado aos cursos, deveres e obrigações dos universitários, ocorre posteriormente a dificuldade em se consultar ou compreender as informações antes repassadas, ou não, tal como calendário acadêmico, bolsas e auxílios, grade curricular, projetos de extensão e perguntas frequentes sobre o curso ou a universidade em si. Essas dificuldades iniciais, vivenciadas por estudantes recém-integrados no ambiente universitário podem ser um agente indireto ou direto influente na reprovação e posterior evasão em cursos de nível superior, pois segundo (LOBO, 2012) uma das principais causas gerais da evasão é a inadaptação do calouro ao ensino superior com a falta de maturidade em lidar com esses novos desafios.

1.1 Motivação

Assim, torna-se o foco deste trabalho a promoção de uma alternativa sob este impasse, propondo-se a criação de uma plataforma *web* de apoio a calouros universitários do setor público de ensino, juntamente de uma tecnologia de *chatbot*, que visa orientar o calouro não somente nos períodos ingressantes, mas também ao decorrer de todo o curso. O sistema centralizará informações pré-cadastradas dos cursos e da universidade, que estão disponíveis nos *websites*, e as apresentará de maneira mais acessível e intuitiva para os estudantes. Essas informações servirão como a base de dados do *chatbot*, garantindo que o sistema forneça respostas precisas e relevantes para as dúvidas dos calouros, enquanto mantém uma interface amigável e personalizada. Por outro lado, informações mais específicas de cada curso poderão ser totalmente personalizadas pelos coordenadores, permitindo a inclusão de dados exclusivos para cada curso. Além disso, o sistema oferecerá a consulta à grade curricular dos cursos cadastrados, onde os calouros poderão visualizar os tópicos associados pelos coordenadores a cada disciplina, tópicos que vinculam *links* de artigos, vídeos ou *websites* que visam auxiliar o discente naquela disciplina.

1.2 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma plataforma *web* que esteja integrada a um *chatbot* capaz de auxiliar os calouros ingressantes em relação a dúvidas de seu curso e da universidade em si.

1.3 Objetivos Específicos

No escopo do objetivo geral, pretende-se também alcançar os seguintes objetivos específicos:

- Levantamento de dados por meio de pesquisa com coordenadores sobre a necessidade de um sistema de apoio aos calouros, incluindo a viabilidade de um chatbot e sugestões de funcionalidades adicionais para suporte aos estudantes de períodos iniciais;
- Realizar uma revisão das ferramentas mais adequadas para o desenvolvimento da plataforma *web*;
- Fazer um *benchmark* de chatbots para escolher o que melhor se adeque à plataforma web;
- Criação e validação de protótipos da plataforma para testar e aprimorar sua interface e funcionalidades;
- Desenvolvimento da plataforma web utilizando as ferramentas escolhidas na revisão;
- Criação, Configuração e Integração do Chatbot com a plataforma web;
- Identificar possíveis melhorias e ajustes na plataforma, visando a sua evolução como trabalho futuro.

1.4 Estrutura do Trabalho

Este trabalho está estruturado em seis capítulos organizados da seguinte forma:

Capítulo I Apresenta a introdução, incluindo a motivação, objetivos gerais e específicos, e a estrutura do trabalho.

Capítulo II Apresenta uma revisão literária acerca da adaptação à universidade, processamento de linguagem natural (PLN) e chatbots.

Capítulo III Mostra alguns trabalhos relacionados com a proposta da monografia, destacando suas principais contribuições e temáticas abordadas.

Capítulo IV Descreve a metodologia utilizada, relacionada a metodologia de projeto adotada, aos ambientes de desenvolvimento e as tecnologias utilizadas.

Capítulo V Discorre a proposta do trabalho, o sistema web SIAC integrado ao chatbot MICA.

Capítulo VI Apresenta as considerações finais da monografia.

2 REVISÃO DE LITERATURA

Este capítulo irar apresentar de forma conceitual e detalhada os fundamentos teóricos que sustentam todo o processo de idealização deste projeto.

A seção 2.1 aborda os principais desafios enfrentados pelos calouros nos períodos iniciais. Em particular, o tema da inadaptação e suas consequências, como a possível evasão, será tratado na subseção 2.1.1. Enquanto na subseção seguinte temos 2.1.2, que exibira possíveis medidas em que a universidade pode tomar para melhorar o acolhimento e direcionamentos de novos ingressantes. Por fim, na subseção 2.1.3 apresentamos estratégias que em conjunto com as tecnologias possam ser utilizadas para promover a permanência dos estudantes.

A seção seguinte 2.2 desenvolverá o processamento de linguagem natural, um tema importante na inteligência artificial em que tecnologias do tipo Chatbot faz uso.

Finalmente, como ultima seção teremos 2.3, onde apresentara os conceitos que permeiam a tecnologia chatbot. Bem como irar esclarecer na 2.3.1 o funcionamento geral da arquitetura de um chatbot e seu funcionamento. Da mesma maneira que irar exemplificar estudos de casos em que chatbots foram aplicados ou desenvolvidos no ambiente educacional na subseção 2.3.2, e por fim irar realizar um benchmark na subseção 2.3.3 a respeitos das principais tecnologias chatbots e a composição de uma tabela comparativa entre elas.

2.1 Adaptação à Universidade

A inserção de calouros ao ambiente universitário é uma temática de extrema importância, que abre para os discentes um leque de possibilidades, isso, pois, segundo (ALMEIDA; SOARES; FERREIRA, 2001), a Universidade emerge como um contexto de facilitação para o desenvolvimento pessoal dos jovens, promovendo a integração e o ajustamento acadêmico, pessoal, social e afetivo do aluno, constituindo um suporte do desenvolvimento presente e de projeção futura, facilitando tanto a transição do ensino secundário para o ensino superior, quanto do ensino superior para o mundo social e profissional.

No entanto, entende-se que esta adaptação à universidade, além de trazer grandes oportunidades, pode muitas vezes vir com diversos desafios, sobretudo se mal realizada por parte da instituição de ensino, isso, pois, de acordo com (CAVALLINI; CARAMASCHI, 2012) o ingresso no ensino superior representa uma transição que poderá repercutir no desenvolvimento social e pessoal assim como nos relacionamentos interpessoais dos jovens universitários, tanto no início do curso, quanto em seu decorrer, pois nos períodos iniciais, os ingressantes lidam com um ambiente totalmente diferente do ensino básico, onde surgem novas responsabilidades acadêmicas, que se assemelham a uma metodologia de

aprendizagem autodirigida, onde aluno é, em muitos momentos, o seu próprio professor. Dessa forma, segundo (TEIXEIRA et al., 2008), o acompanhamento dos docentes do ensino superior sobre o aluno diminui potencialmente, se comparado ao ensino básico e nesse contexto, o aluno precisa desenvolver habilidades de independência e autonomia sobre os estudos e o professor passa a ter o papel de somente fornecer orientação, suporte, recurso e acompanhar o progresso de seus alunos.

2.1.1 A inadaptação e a evasão do ensino superior

Então, a partir de uma compreensão acerca das dificuldades desse processo de adaptação ao ambiente universitário, elucida-se aqui a relação entre este fator e a problemática citada por (FILHO et al., 2007), sobre a taxa de evasão dos discentes no primeiro ano de cursos de ensino superior, sendo duas a três vezes maior do que a dos anos seguintes. Assim, mesmo que a evasão acadêmica seja uma problemática multifatorial, que afeta as mais diversas instituições de ensino superior, segundo (LOBO, 2012) uma das principais causas gerais para esta, é a desambientação do calouro ao funcionamento deste novo ambiente, tal qual o pouco discernimento acadêmico destes ingressantes, como também trabalhado por (TEIXEIRA; MENTGES; KAMPFF, 2019), que descreve como um dos fatores que influem negativamente na vivência e permanência de alunos no ensino superior, envolve o próprio sistema educacional, englobando tanto a qualidade e as dificuldades com a didática empregada, como com o suporte oferecido pela instituição aos discentes. Logo, essa inadaptação de calouros, visualizada na pouca compreensão do ambiente e de informações acadêmicas, são um dos agentes internos da universidade que contribuem direta e indiretamente aos índices de evasão ao longo dos períodos iniciais de curso, uma vez que há, em muitos casos, uma escassez ou falta total de orientação acadêmica, que pode resultar no desnorteamento do estudante universitário em relação à vida acadêmica.

2.1.2 Acolhimento ao calouro

Dessa maneira, para uma maior adequação a este novo ambiente, que possibilite que esses alunos permaneçam e sejam bem sucedidos no decorrer de sua graduação, faz-se importante um melhor acolhimento e direcionamento aos novos ingressantes, assim, sendo necessário que as instituições de ensino superior utilizem-se das mais diversas alternativas e encaminhem recursos apropriados para este fim. Sobre isso, temos como exemplo o trabalhado realizado por (HOLANDA et al., 2022), onde em um contexto da alta taxa de evasão no curso de Ciência da Computação, foram realizadas diversas mudanças neste, tais quais a reformulação do currículo do curso, onde foram feitas alterações nas disciplinas dos períodos ingressantes e uma reformulação das disciplinas iniciais de programação, entre outras modificações, nesse contexto, implementou-se então a disciplina de Introdução ao Ambiente Universitário da Computação (IAUC), que possuía como objetivo trazer uma melhor recepção a esses calouros e que teve bons resultados.

2.1.3 Sistemas de Apoio Acadêmico

Assim, entrando aqui em ênfase essa problemática, fica evidente a necessidade de novas estratégias acerca do suporte oferecido pelas instituições, a fim de atuar positivamente na formação destes. Sobre isso, em sua revisão sistemática, (TEIXEIRA; MENTGES; KAMPFF, 2019) descreve serem escassas as iniciativas institucionais adequadas para promover a permanência dos estudantes, mas entre as poucas estratégias adotadas, uma das que se destaca é a utilização de sistemas de informação para o apoio à permanência desses alunos. Assim, entende-se que as tecnologias servem como importantes ferramentas nesse meio acadêmico, que se usadas de maneira eficiente, podem contribuir efetivamente, realizando uma função de apoio a discentes e potencializando o desenvolvimento destes no contexto universitário, surgindo como exemplos a isso, os sistemas de apoio acadêmico, sendo uma forma possível para intervir sob as dimensões envolvidas nessa problemática.

2.2 Processamento de Linguagem Natural (PLN)

O PLN é um termo encaixado na inteligência artificial atualmente, e seu uso consegue analisar, processar e extrair as informações de língua natural, especificamente no que diz respeito à interação entre humanos e máquinas.

O PLN é baseado na aplicação de métodos e técnicas para extrair informações importantes a partir de escritos, como a análise de palavras-chave, a estrutura da língua, os sujeitos mencionados e o contexto semântico. A compreensão da língua se mostra como uma tarefa complicada, pois segundo (GOLDBERG, 2022), devido à ambiguidade no idioma humano, palavras e expressões podem ter diferentes significações em diferentes circunstâncias.

Segundo (MARTIN, 2009), o PLN abrange várias etapas e segue procedimentos para funcionar de acordo, temos como métodos fundamentais que compõem todo o seu funcionamento final.

A CLN (Compreensão de Linguagem Natural), compreende toda a função de possibilitar que a máquina compreenda a linguagem humana que está sendo repassada, por meio de processos como análise e extração de significados de palavras ou frases. Assim como possui outras etapas, como:

- **Tokenização:** Trata-se em dividir a entrada de texto em tamanhos menores, como palavras ou frases;
- **Análise Sintática:** Onde ocorre uma identificação das conjunções gramaticais entre as palavras para compreender a estrutura da frase em geral;
- **Análise Semântica:** Concessão dos significados das palavras e construção das relações entre elas;
- **Desambiguação:** Trabalhar com as ambiguidades de sentidos que uma palavra possui, assim indicando o sentido mais coerente;

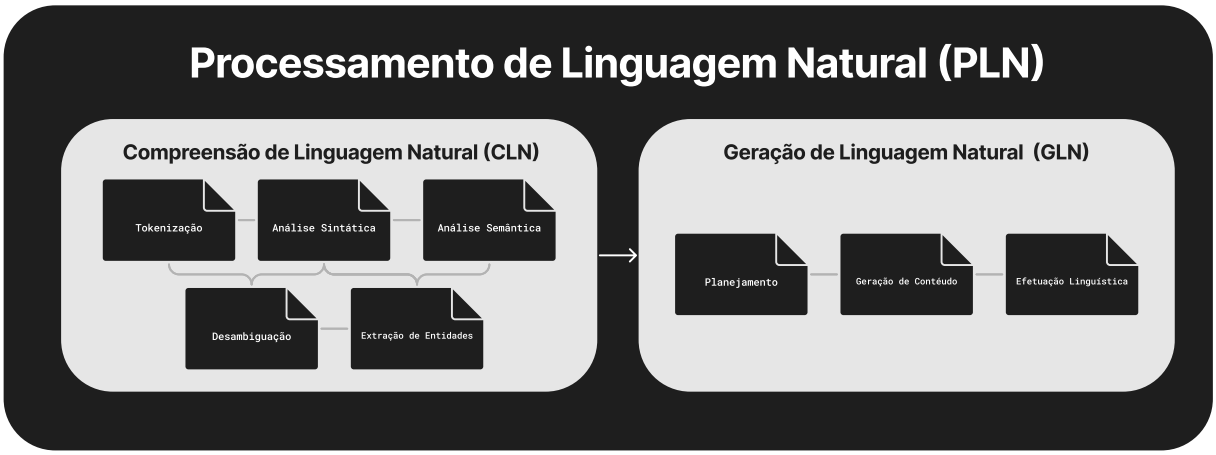
- **Extração de Entidades:** Assimilação de entidades identificadas, como nomes próprios sejam de pessoas, lugares ou outro tipos, como datas.

Por outro lado, a GLN (Geração de Linguagem Natural), aborda o processo de gerar texto ou a fala com base na linguagem humana, isso utilizando dados e informações referentes a linguagem natural. Esse trabalho envolve algumas etapas, como:

- **Planejamento:** Estabelece a estrutura e organização do texto;
- **Geração de Conteúdo:** Geração do conteúdo textual, com base nos dados e informações repassadas anteriormente, e seguindo regras ou diretrizes estabelecidas;
- **Efetuação Linguística:** Trata-se de uma adequação final, para que o texto gerado respeite um modelo gramatical, de forma natural e exata.

Observando a Figura 1, conseguimos identificar ambos os fundamentos que compõe o processamento de linguagem natural, e podemos identificar todas as etapas citadas anteriormente.

Figura 1 – CLN e a GLN pertencem a PLN



Fonte: Autoria Própria

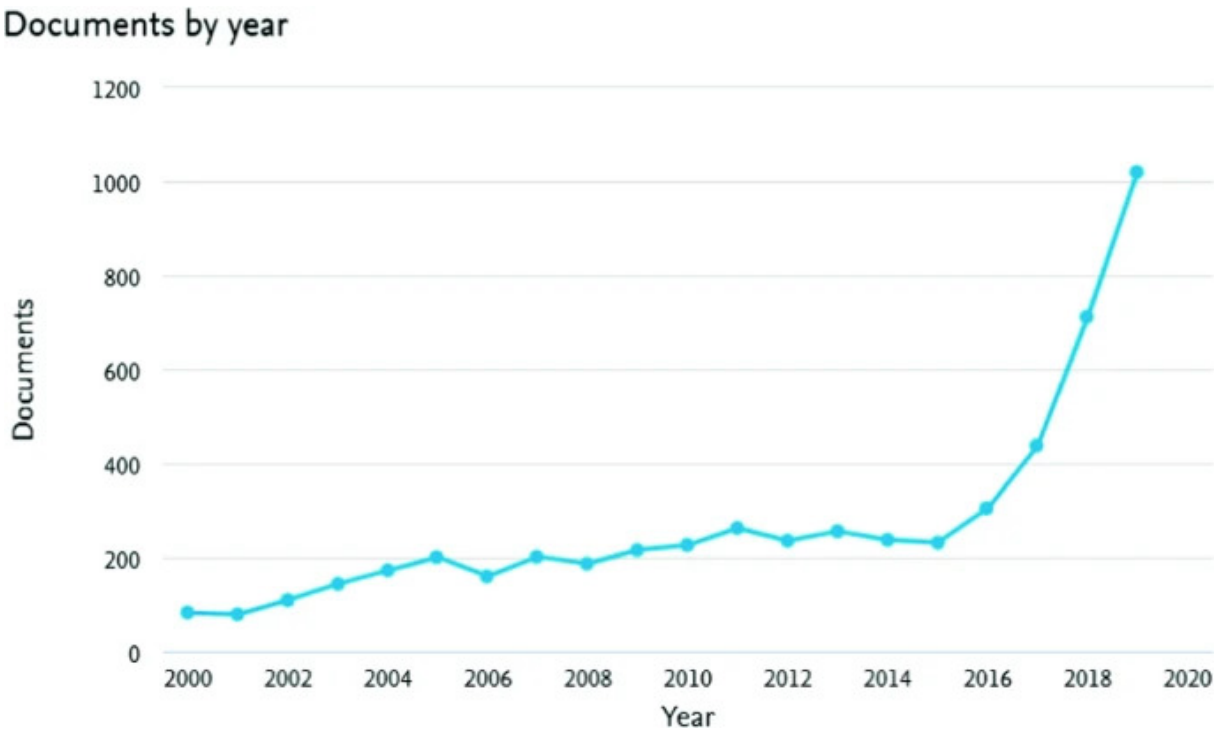
O seu uso abre possibilidades na inteligência artificial, como o treinamento de modelos de aprendizado de máquina, temos exemplos como redes neurais, para ensinar algoritmos em grandes conjuntos de dados de texto. Esses modelos conseguem aprender padrões e relacionamentos em dados, o que permite que os sistemas generalizem o conhecimento e sejam mais eficientes em novas tarefas de processamento de linguagem.

Em síntese esse campo de atuação está em crescimento, segundo (GOLDBERG, 2022), essa evolução vem sendo impulsionada pelos avanços nas técnicas de aprendizado de máquina e pela maior disponibilidade de grandes conjuntos de dados de texto. À medida que a tecnologia PLN continua a se desenvolver, espera-se que apareça cada vez mais em vários aplicativos e serviços, proporcionando uma interação humano-computador mais natural e eficiente.

2.3 Chatbot

Os chatbots são sistemas de inteligência artificial que possibilitam a interação com seres humanos por meio de uma conversa natural, utilizando a linguagem falada ou escrita. De acordo com (HAPKE; HOWARD; LANE, 2019), ao contrário das linguagens formais, a linguagem natural não pode ser traduzida diretamente em operações matemáticas precisas, mas utiliza-se dela para a extração, processamento e resposta de informações e instruções presentes em conversas do dia a dia. Como demonstrado na Figura 2, de acordo com (SCOPUS, 2020) houve um aumento significativo no que se refere às buscas por Chatbots, conforme (COLACE et al., 2018), esse crescimento ocorreu especialmente após o ano de 2016, onde muitos Chatbots foram desenvolvidos para soluções industriais, enquanto há uma grande variedade de Chatbots menos famosos relevantes voltados para pesquisas e aplicações.

Figura 2 – Resultados da pesquisa no Scopus por ano para "chatbot" ou "agente de conversação" ou "interface de conversação" como palavras-chave de 2000 a 2019.



Fonte: (SCOPUS, 2020)

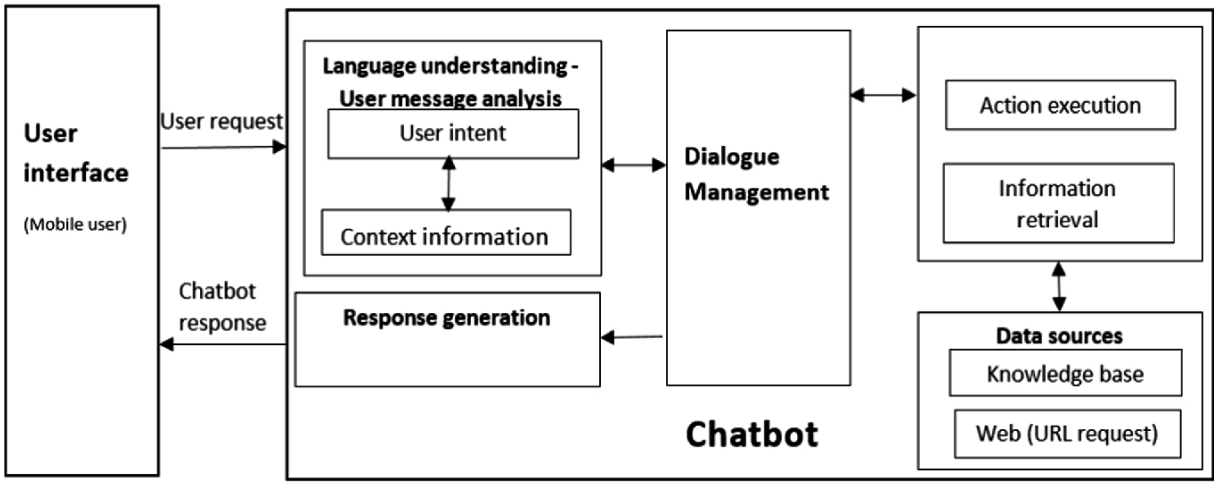
2.3.1 Arquitetura Geral e Funcionamento

A arquitetura geral de um chatbot envolve diferentes etapas. Inicialmente, com processamento de linguagem natural, os chatbots analisam a entrada do usuário e identificam intenções e entidades relevantes. O chatbot então usa essas informações para buscar respostas apropriadas em sua base de conhecimento ou sistema de gerenciamento

de informações. Por fim, o chatbot elabora as respostas apresentadas ao usuário de forma coerente e natural.

Ao observar a Figura 3, conforme (ADAMOPOULOU; MOUSSIADES, 2020), podemos identificar as diferentes etapas que ocorrem na arquitetura geral de um chatbot, explorando cada uma dela temos as seguintes etapas:

Figura 3 – Arquitetura Geral Chatbot



Fonte: (ADAMOPOULOU; MOUSSIADES, 2020)

- **Receber Entrada:** Ocorre uma requisição do primeiro bloco nomeado "User Interface", nesse caso um dispositivo mobile (Podendo ser qualquer dispositivo em outros contextos), diretamente ao bloco do Chatbot;
- **Tratar Requisições:** A requisição chega ao bloco do Chatbot, onde existem módulos que irão tratar essa requisição. O primeiro módulo, "Language understanding — User message analysis", analisa e compreende a intenção do usuário e as informações associadas ao que foi dito. Caso o chatbot não compreenda completamente a mensagem, ele solicitará mais informações ao usuário.
- **Repassar Requisições:** Essas requisições após interpretadas são gerenciadas e repassadas por um módulo gerenciador de diálogos para um próximo módulo que irar realizar ações;
- **Executar Ações:** Após obter a melhor interpretação possível da mensagem e decidir como agir, o Chatbot recupera a informação necessária e executa a ação solicitada.
- **Buscar Dados:** A execução pode envolver a execução de tarefas específicas ou a busca de dados em uma fonte de dados designada, ou em módulo, chamada de "Data Sources". Essas fontes de dados podem ser um banco de dados denominado "Knowledge", contendo a base de conhecimento do chatbot, ou recursos externos acessados por meio de chamadas de API.
- **Gerar Resposta:** Após recuperar os dados das "Data Sources", essa informação é enviada para o "Dialogue Management", e em seguida é direcionada para o "Response

Generation". Este módulo é responsável por utilizar um gerador de linguagem natural (GLN) para produzir uma resposta ao usuário em linguagem natural, semelhante à resposta que um humano daria, com base nas intenções e no contexto das informações fornecidas anteriormente.

A arquitetura citada pode variar conforme o Chatbot que está sendo implementado, mas geralmente segue esta arquitetura como base e um desses modelos para gerar respostas apropriadas: baseado em regras, baseado em recuperação e modelo generativo.

2.3.2 Chatbot na Educação

O uso de Chatbots no ambiente educacional ganha uma proporção, pois segundo (LUCCHESI et al., 2018), os chatbots já vem sendo utilizado desde 1966 e com os avanços da tecnologia e os estudos nas áreas de NLP e ML, fazem com que a utilização de chatbots se dissemina, tornando cada vez mais presente no contexto diário das pessoas.

Acerca de estudos de casos de Chatbots na educação temos alguns, como dos autores (LEONHARDT et al., 2003), em que se baseando no Chatbot ALICE, desenvolveram o chatbot "Profª Elektra". A Elektra tem um objetivo educacional, volta dono auxílio dos estudantes de curso de educação à distância no entendimento e fixação de conteúdos voltados para a área de redes de computadores, e também ter um pequeno conhecimento voltado para o ensino de física. Assim como os autores (FERREIRA, 2008), que propõem desenvolver um chatbot para auxiliar no ensino de língua estrangeira. Vendo a dificuldade de alunos em aprender espanhol, os autores pensaram em criar algo que pudessem auxiliar os alunos a praticar sempre e quando desejarem, então foi desenvolvido um chatbot denominado "Esteban".

Deste modo, pode-se observar que o chatbots conquistaram seu espaço no ambiente educacional, e a cada dia surgem novas funções que aprimoram esta experiência, tanto para o discente quanto os docentes.

2.3.3 *Benchmark* Chatbots

Existem diversas soluções, ferramentas e frameworks para criação e utilização de Chatbots, consequentemente alavancando uma grande concorrência e deixando a vista todas as vantagens e desvantagens no uso de cada tecnologia, como preço, licença, funcionalidades, integração e plataforma. Em seguida discutiremos sobre cada solução e posteriormente compara-lós por meio de uma tabela, destacando suas diferenças e semelhanças.

2.3.3.1 IBM Watson Assistant

O (IBM, 2023) pertence a uma plataforma de chatbot da IBM. Ele oferece recursos avançados de processamento de linguagem natural (NLP) e permite criar chatbots personalizados e com diversas opções de integração para canais, como sites, aplicativos

móveis e assistentes virtuais. Uma ferramenta que tem grande potencial ao suportar multiconexões em um mesmo chatbot, fazendo merecimento ao seu custo, pois se trata de um software de código fechado e com custo de uso quando se trata de mais de 10.000 requisições no chatbot.

2.3.3.2 Microsoft Bot Framework

O (Microsoft, 2023) é uma ferramenta de código aberto que possibilita a criação, teste, implementação e gestão de bots inteligentes. O Bot Framework tem compatibilidade com Azure e pode-se utilizado tanto grátis quanto pago, sua SDK abrange de forma versátil e flexível o desenvolvimento de bots e estabelecimento de conexões com serviços de IA. Por meio dessa estrutura, os desenvolvedores podem criar bots que utilizam a fala, compreendem a linguagem natural e respondem a questões.

2.3.3.3 Amazon Lex

A (Amazon, 2023) é uma plataforma baseada em nuvem que possibilita a criação de chatbots, sendo de autoria da Amazon. Por meio do uso de tecnologias de inteligência artificial, como o processamento de linguagem natural, é possível realizar reconhecimento de voz e interpretação de linguagem natural, altamente utilizada para criar chatbots interativos, sendo completamente paga e de código fechado.

2.3.3.4 Wit.ai

O (Wit.ai, 2023) é uma plataforma de processamento de linguagem natural, no qual possibilita a criação de chatbots interativos. No momento pertence à Meta, possui planos gratuitos e pagos, assim como alta integração com diversas plataformas.

2.3.3.5 Botpress

(Botpress, 2023) é uma plataforma de código aberto, gratuito ou pago que proporciona a criação de chatbots baseados em regras ou com inteligência artificial avançada, utilizando aprendizado de máquina, possui recursos como análise e fluxo de conversa e diversa possibilidade de integração.

2.3.3.6 Dialogflow

O **Dialogflow** (Google, 2023), como a própria documentação o define, é uma IA de conversação realista, que possui agentes virtuais de última geração. Possuindo duas versões: Dialogflow CX (avançado) e Dialogflow ES (padrão).

Com o uso de sua plataforma é possível realizar o processamento de linguagem natural de forma fácil, ao possuir um design intuitivo e fácil integração com aplicativos para dispositivos móveis e aplicativos Web.

2.3.3.7 Rasa

O Rasa (Rasa, 2023a) é uma plataforma de código aberto amplamente utilizada para a construção de assistentes virtuais. Ele é dividido em duas versões: Rasa Open Source, focado em soluções personalizáveis, e Rasa Pro, que oferece funcionalidades adicionais de análise, segurança e escalabilidade, uma opção mais comercial e com possibilidade de ser *low-code* com o uso do Rasa X/Enterprise.

2.3.3.7.1 Conceitos Básicos do Rasa

A arquitetura do Rasa se baseia em dois componentes principais: a compreensão de linguagem natural (NLU) e o gerenciamento de diálogo. O NLU é responsável por classificar as intenções e extrair entidades das mensagens dos usuários, enquanto o gerenciamento de diálogo decide a próxima ação com base no contexto da conversa e nas políticas de diálogo.

Antes de detalharmos o funcionamento da arquitetura do Rasa, especialmente o Open Source, devemos entender alguns conceitos básicos, pois semelhante a outras tecnologias chatbot citadas anteriormente, o Rasa utiliza conceitos básicos da NLP 2.2, a seguir explicado algumas delas:

- **Intents:** As intenções, que no Rasa são os objetivos do usuário na conversa. Elas são mapeadas a partir de frases de exemplo usadas no treinamento do modelo NLU. Um exemplo seria um usuário que gostaria de saber a previsão do tempo, com isso a intenção configurada no Rasa seria: `inform_temperature`, em conjunto com frases que serviriam para treinar o chatbot na identificação daquela intenção.

```
{
  "intent": "inform_temperature",
  "examples": [
    "Qual é a previsão do tempo?",
    "Como está o clima agora?",
    "Vai chover amanhã?"
  ]
}
```

- **Entities:** No Rasa, as entidades fornecem detalhes adicionais sobre a intenção do usuário, permitindo que o chatbot capture informações importantes e personalize suas respostas. No mesmo contexto do último exemplo, uma entidade no fluxo de conversação poderia ser uma cidade, assim sendo utilizado para buscar informações de temperatura especificamente para aquele contexto identificado.
- **Slots:** São uma espécie de memória de curto prazo, armazenando informações ao longo da interação para garantir consistência nas respostas. O valor de entidade

capturado anteriormente poderia ser armazenado dentro daquele fluxo de conversa para apresentar outras intenções dentro daquele mesmo contexto.

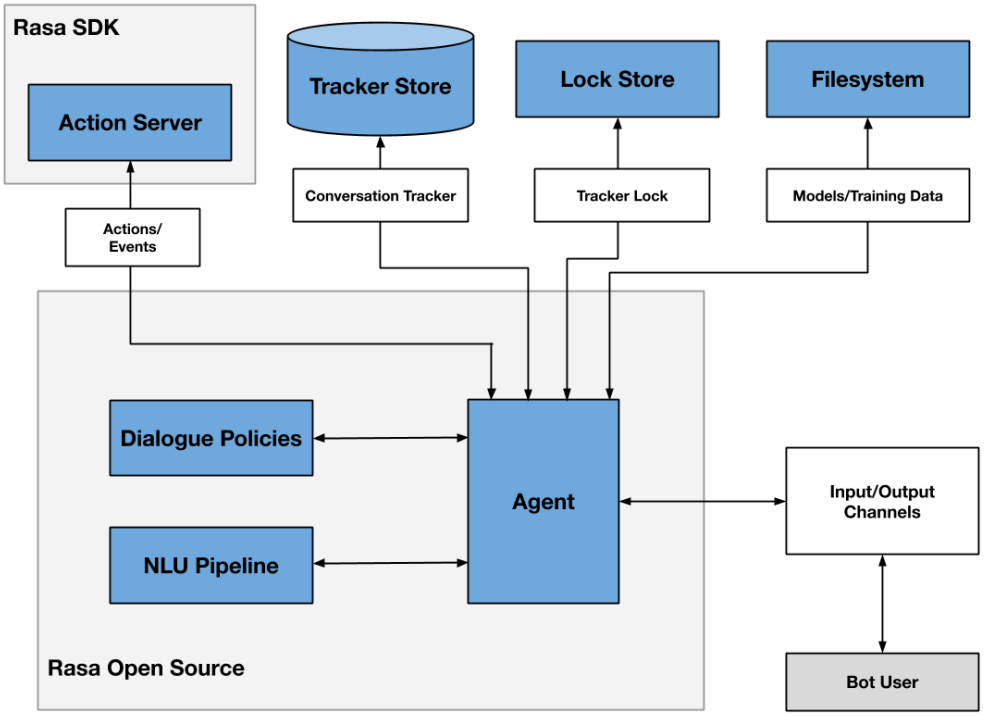
- **Actions:** As ações podem ser respostas simples ou chamadas de API externas via **Action Server**[1], permitindo maior flexibilidade na realização de tarefas personalizadas.
- **Policies:** Regras e algoritmos que determinam o próximo passo da conversa. O Rasa utiliza diferentes políticas para lidar com múltiplos cenários de diálogo, como preenchimento de formulários e respostas baseadas em regras.
- **NLU Component:** O elemento responsável por processar as mensagens recebidas. Utilizando as configurações fixadas, ele consegue extrair entidades, classificar intenções e realizar processamentos em um fluxo de conversação.

Observando o diagrama da Figura 4, podemos entender toda a arquitetura geral do Rasa Open Source, composta pelos seguintes componentes:

- O **Rasa Open Source**, onde o modelo treinado para o agente de conversação é carregado. Com base nas **Dialogue Policies** e na **NLU Pipeline** configuradas, o sistema decide as próximas ações durante a conversa.
- O **Tracker Store** é o componente responsável por armazenar as conversas, enquanto o **Lock Store** garante que uma conversa em processamento fique "trancada", impedindo alterações simultâneas.
- O **Filesystem** é utilizado para armazenar os dados de treinamento e os modelos já treinados.
- E o **Rasa SDK**, sendo um servidor separado responsável por executar ações personalizadas mais complexas e retornar respostas que serão enviadas pelo Rasa para o componente final, o qual são os canais de entrada e saída, onde ocorre a comunicação direta entre o agente e o usuário.

[1]Trata-se do servidor de ações personalizadas do Rasa, onde executa essas ações separadamente do Rasa. O Rasa mantém o Rasa SDK em Python para implementar ações personalizadas, embora também seja possível escrever ações personalizadas em outras linguagens. (Rasa, 2023b)

Figura 4 – Rasa Architecture Overview

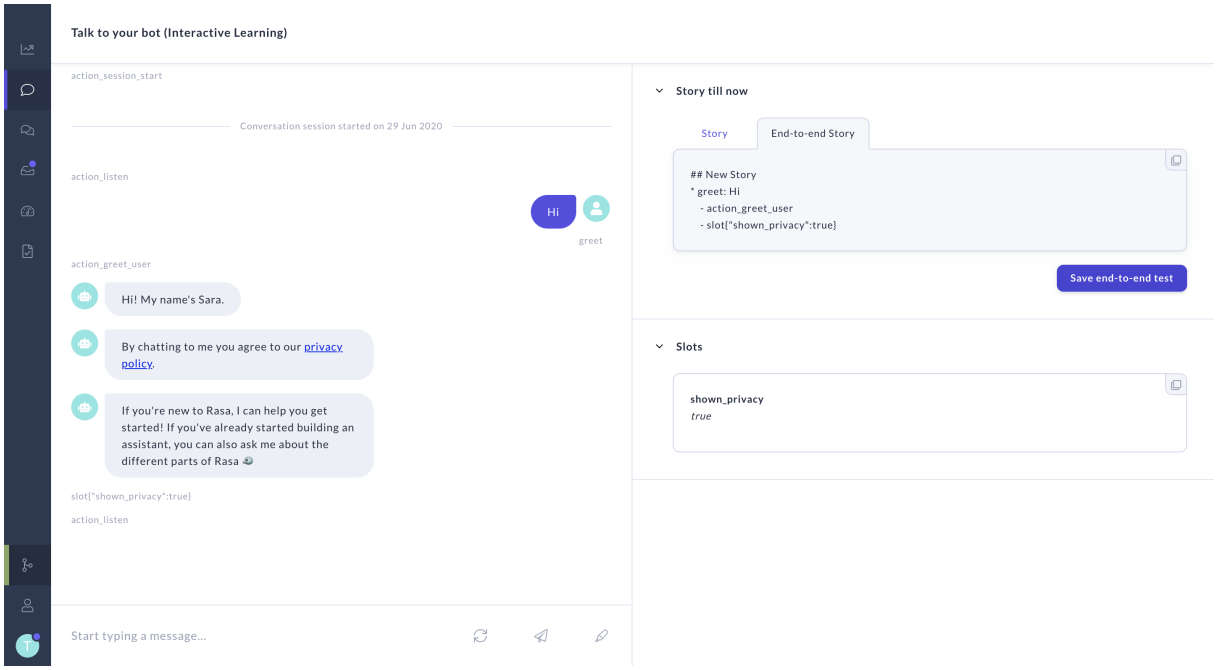


Fonte: (Rasa, 2023b)

2.3.3.7.2 Rasa Open Source vs Rasa Pro

A principal diferença entre as versões do Rasa está na abrangência e nas funcionalidades oferecidas. O Rasa Open Source é ideal para projetos que precisam de personalização e controle total sobre o fluxo de conversação. Já o Rasa Pro, com funcionalidades empresariais como segurança avançada, monitoramento e facilidade em hospedagem do agente treinado, sendo voltado para soluções maiores e mais complexas, e caso necessite pode ser todo configurado por meio de uma interface *low-code*, o Rasa X/Enterprise (Figura 5). Ambos permitem a integração com diversos sistemas externos e suportam uma ampla gama de canais de comunicação.

Figura 5 – Talk to Your Bot



Fonte: (Rasa, 2023b)

2.3.3.8 Tabela Comparativa Chatbots

A tabela a seguir foi idealizada com as tecnologias presentes na subseção *Benchmark* 2.3.3, onde definimos como critérios de comparação os seguintes aspectos: Custo, Código Aberto e Integrações.

Tabela 1 – Tabela Comparativa Chatbots

Chatbot	**Custo**	**Código Aberto**	**Integrações**
(IBM, 2023)	Pago	Não	IBM Cloud
(Microsoft, 2023)	Gratuito ou Pago	Sim	Microsoft Azure
(Amazon, 2023)	Pago	Não	Amazon Web Services (AWS)
(Rasa, 2023a)	Gratuito ou Pago	Sim	Integrações por meio de API
(Wit.ai, 2023)	Gratuito ou Pago	Não	Integração com plataformas de mensagens populares
(Botpress, 2023)	Gratuito ou Pago	Não	Integrações mediante API
(Google, 2023)	Gratuito ou Pago	Não	Firebase, Google Cloud, plataformas de mensagens populares

Analisando a tabela comparativa (Tabela 1), notamos semelhanças entre as tecnologias. Porém, ao comparar a tecnologia **Rasa Open Source** e a **Microsoft Azure Bot**, especialmente no contexto de custo e código aberto, no qual são semelhantes, as vantagens do Rasa em customização se destacam mais. Pois embora a versão gratuita da Microsoft ofereça integração com canais como Skype, ela limita a dependência da infraestrutura da Microsoft na versão gratuita. Por outro lado, o Rasa oferece total personalização e integrações sem estar vinculado a um ecossistema específico. Isso torna o Rasa ideal para projetos que necessitam de uma solução personalizada, garantindo melhor controle sobre regras de negócios e requisitos do projeto.

3 TRABALHOS RELACIONADOS

Este capítulo compara trabalhos relacionados ao uso de chatbots em universidades, analisando diferente tecnologias empregadas e os resultados obtidos. Esse comparativo permite uma visão clara sobre as abordagens tecnológicas adotadas e como elas foram aplicadas para superar obstáculos, além de oferecer uma perspectiva sobre o impacto e a eficiência das soluções desenvolvidas.

Tabela 2 – Tabela Comparativa dos Trabalhos Relacionados.

Solução	**Objetivo Geral**	**Tecnologia**	**Integração**
(MENDES; SIRQUEIRA, 2022)	Desenvolvimento e operação de um chatbot que auxilia alunos, principalmente novos ingressantes, em aspectos importantes da vida universitária.	Dialogflow	Mobile
(ARAUJO, 2020)	Estudo sobre o funcionamento dos chatbots, algumas ferramentas para a sua construção e uma efetiva implementação com a tecnologia Rasa e seu uso em um aplicativo de mensagens, o Telegram.	RASA	Telegram
(SILVA et al., 2021b)	Desenvolver um sistema do tipo Chatbot que apresente a solução completa, utilizando uma biblioteca de código aberto, até a interface de bate-papo e o sistema administrativo para gerenciamento do Chatbot.	Python Chatterbot	Web/Mobile
(MACIEL, 2019)	Análise de ferramentas para criação de chatbots e o desenvolvimento de um bot chamado Maciel.	Dialogflow	Whatsapp
(SILVA et al., 2021a)	Desenvolvimento de um chatbot baseado em regras para auxiliar a comunidade do IFB.	TelegramBotAPI	Telegram
Trabalho proposto	Desenvolvimento de uma plataforma web integrada a um chatbot, capaz de auxiliar os calouros ingressantes em relação a dúvidas de seu curso e da universidade.	Rasa	Web/Mobile

Os trabalhos listados (Tabela 2) demonstram um foco comum em desenvolver chatbots voltados para o auxílio de estudantes em suas interações acadêmicas. A utilização

de tecnologias como **Dialogflow**, **Rasa**, **Python** e **Telegram** reflete uma tendência em explorar diferentes plataformas para integração.

O trabalho de (MENDES; SIRQUEIRA, 2022) foca na adaptação de novos ingressantes à vida universitária, utilizando um chatbot desenvolvido com a tecnologia **Dialogflow**. O chatbot foi implementado em um aplicativo e projetado para responder a perguntas frequentes, auxiliando os alunos a se adaptarem mais rapidamente, reduzindo a demanda por informações nas secretarias acadêmicas.

Neste estudo de (ARAUJO, 2020), foi realizado um estudo sobre as tecnologias chatbot e uma implementação de uma dessas tecnologias, no caso o **Rasa** utilizando como canal o **Telegram** para o chatbot, onde visa sanar as dúvidas dos alunos em tempo real, no contexto universitário.

Enquanto no (SILVA et al., 2021b) o foco deste trabalho desenvolveu uma interface de gerenciamento de perguntas e respostas no contexto de Sistemas de Informação. O sistema permite que alunos, professores e administradores interajam de maneira eficiente, possibilitando um fluxo de informações claro e organizado. A abordagem utilizando a biblioteca **Python Chatterbot**, garantiu que as dúvidas sejam resolvidas de forma prática e que os usuários tenham um ponto de acesso centralizado para informações acadêmicas.

Por outro lado, (MACIEL, 2019) realiza um levantamento de ferramentas para chatbots e implementa um chat utilizado **Dialogflow** e o **WhatsApp** como canal de comunicação. O chatbot foi desenvolvido para atender dúvidas de discentes, docentes e servidores da universidade, promovendo uma comunicação eficiente e dinâmica.

Finalmente, (SILVA et al., 2021a) implementa um chatbot para buscar conteúdo no portal da universidade, utilizando o **Telegram** como meio de interação. O foco está em facilitar o acesso dos alunos a informações relevantes sobre cursos, atividades acadêmicas e informações da universidade, melhorando assim a comunicação entre a instituição e os estudantes.

Essas similaridades indicam que, embora as tecnologias e plataformas possam variar, o objetivo de auxiliar os estudantes na vida acadêmica é um ponto central. A proposta deste trabalho se insere nesse contexto ao desenvolver não apenas um chatbot para atender dúvidas relacionadas à universidade, mas também uma plataforma web integrada que visa apresentar informações do curso. Essa plataforma servirá como base de dados para as consultas do chatbot, com informações atualizadas gerenciadas pelos coordenadores.

4 METODOLOGIA

Este capítulo apresenta a metodologia adotada para o desenvolvimento do projeto, organizada em duas seções principais.

Na seção 4.1, é contextualizado o uso de metodologias ágeis, com ênfase no *Scrum*, detalhando sua aplicação no projeto. Serão discutidas as definições e estruturas das *sprints*, alinhadas às funcionalidades e requisitos necessários para o desenvolvimento.

Enquanto na seção 4.2, detalha os processos, ferramentas, bibliotecas, frameworks e os ambientes de desenvolvimento utilizados na etapa de desenvolvimento do SIAC e chatbot MICA.

4.1 Metodologia de Projeto

Esta seção irar tratar sobre a metodologia de projeto que será implementada durante todo o desenvolvimento do projeto. Inicialmente, abordaremos os conceitos das metodologias ágeis para o desenvolvimento de software, em seguida a abordagem *Scrum*, e por fim a metodologia aplicada ao projeto.

4.1.1 Metodologias Ágeis no Desenvolvimento de Softwares

O uso de metodologia ágeis se tornou presente em diversos campos de atuação, especialmente no desenvolvimento de software, pois segundo (BESSA; ARTHAUD, 2018), os requisitos de um sistema são remodelados inúmeras vezes em um projeto, assim as metodologias tradicionais não se adequavam a essas mudanças. Foi onde se fez presente o uso das metodologias ágeis, com o seu uso é possível uma adaptação e entrega contínua incrementalmente, juntamente de uma validação por parte dos clientes envolvidos, assim possibilitando uma flexibilidade às mudanças em prioridades e requisitos.

De modo a facilitar toda a criação e uso das metodologias ágeis para o desenvolvimento softwares, criou-se então um manifesto que é uma definição fundamental dos valores e princípios das metodologias ágeis. Alguns valores como, a priorização das pessoas, a colaboração, a entrega contínua de software funcional e a respostas ajustável as mudanças nos requisitos. De acordo com (AL., 2001), *"Indivíduos e interações mais que processos e ferramentas"* e *"Software em funcionamento mais que documentação abrangente"*, entre outros princípios.

Em suma a metodologia ágil possui atributos encontrados na engenharia de software, como a capacidade de adaptação e orientação ao cliente, conforme citado anteriormente. Este ponto encontra respaldo nas palavras de (SOMMERVILLE, 2011), que reforça que a engenharia de software tem como princípios fundamentais a adaptabilidade e

centralidade no cliente. Princípios no qual estão presente nas metodologias ágeis, reforçando assim cooperação entre os dois conhecimentos.

4.1.2 Metodologia Scrum

Seguindo fundamentos da metodologia ágil, foi criado a metodologia *Scrum*, onde inicialmente foi utilizada no desenvolvimento de softwares. O Scrum é uma das metodologias ágeis mais utilizadas, aplicada no gerenciamento de projetos que necessitam de colaboração, comunicação e a entrega iterativa. Com o seu uso é possível saber quais etapas do projeto estão em andamento ou foram concluídas.

O seu uso se baseia em *Sprints*, os quais são ciclos rápidos de iterações de desenvolvimento que possuem tempos estabelecidos para início e finalização e estão abertas para revisões e adaptações contínuas. Conforme o (SUTHERLAND; SCHWABER, 2017), os autores da metodologia *Scrum*, ela segue alguns princípios como:

- **Transparência:** Informações relevantes acerca do projeto devem estar disponíveis para os interessados;
- **Inspeção:** Realizar inspeções periódicas nos processos, de modo a identificar problemas e possíveis melhorias;
- **Adaptação:** Seguindo fundamentos das metodologias ágeis, o projeto tem que realizar adaptações quando necessário;
- **Colaboração:** Deve existir uma comunicação recorrente entre os membros do projeto;
- **Entrega Contínua:** Tem-se como objetivo a entrega incrementalmente a cada *sprint*, possibilitando *feedbacks* por parte dos interessados;

4.1.3 Metodologia Scrum Aplicada ao Projeto

Como citado no começo do capítulo, adotaremos a metodologia ágil *Scrum*, onde inicialmente foi elaborado o *Product Backlog*, no qual se trata de uma listagem das funcionalidades e requisitos necessários para todo o desenvolvimento do projeto, resultando na seguinte listagem:

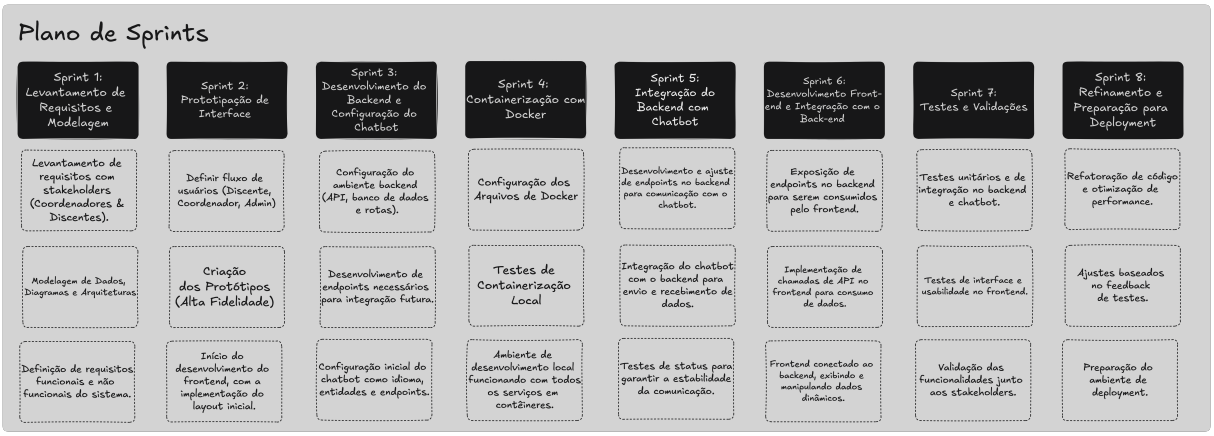
- **Levantamento de Requisitos e Modelagem:** Nesta sprint, são definidos os requisitos funcionais e não funcionais do sistema, mapeando as principais funcionalidades e restrições. Simultaneamente, ocorre a modelagem de dados e a criação de diagramas, garantindo uma estrutura sólida para a arquitetura e o fluxo de informações no sistema.
- **Prototipação:** Definição do fluxo e criação de protótipos de todo o sistema., com base nos requisitos coletados.
- **Desenvolvimento do *Back-end*:** Criação da *API REST* responsável pelo processamento dos dados e integração com serviços de chatbot e banco de dados.
- **Configuração do Chatbot:** Organização inicial do projeto do chatbot, como idioma, *endpoints* e fluxo de conversação.

- **Conteinerização com Docker:** Criação e configuração de contêineres Docker para o serviço de chatbot e MongoDB, garantindo a portabilidade.
- **Integração *Back-end* com Chatbot:** Integração do *back-end* com o chatbot.
- **Desenvolvimento do *Front-end* e Integração com o Back-end:** Implementação das interfaces com base nos protótipos e consumo dos dados da *API REST*.
- **Testes e Validações:** Testes de Integração, Unitário e de Usabilidade.
- **Refinamento e Deployment:** Ajustes baseados nos *feedbacks* e prepração para *deployment* (hospedagem).

Com base no Product Backlog, dividimos as funcionalidades e requisitos em *sprints*, planejadas para ocorrer em ciclos semanais, com início e término a cada sexta-feira. O objetivo de cada sprint é entregar um incremento funcional do produto, que será validado ao final. Caso aquele incremento entregue não esteja incompleto ou precise de ajustes, a sprint poderá ser renovada até que um incremento válido seja alcançado. Somente após essa validação é que se avança para a próxima sprint, garantindo assim a entrega contínua e incremental dos requisitos funcionais do sistema.

Na Figura 6, é possível visualizar todas as etapas de desenvolvimento do projeto, juntamente com uma listagem das atividades planejadas, as quais foram definidas com base no product backlog. Ao todo, foram identificadas 24 atividades, distribuídas ao longo de 8 sprints. Vale destacar que algumas etapas do backlog podem ocorrer simultaneamente, como o levantamento de requisitos e a modelagem, otimizando o processo de desenvolvimento

Figura 6 – Scrum Sprints



Fonte: Autoria Própria

4.2 Tecnologias e Ambientes de Desenvolvimento

Nesta seção, discutimos as técnicas, tecnologias e configurações do ambiente de desenvolvimento que apoiaram o desenvolvimento e implementação deste projeto. Serão explorados os componentes que possibilitaram a pesquisa e modelagem, design e prototipagem e planejamento do *front-end*, *back-end* e tecnologia chatbot. Visando destacar

como diferentes tecnologias se relacionam e se complementam ao longo do processo de desenvolvimento.

4.2.1 Pesquisa e Modelagem

Para a realização da pesquisa com os coordenadores, foi utilizado o **Google Forms** (Google, 2024) como ferramenta para o levantamento de informações. Na parte de modelagem do banco de dados, foi empregado o **Moon Modeler** (DATENSEN, 2024) para o desenvolvimento do diagrama UML de classe. Além disso, o **Excalidraw** (RHYNE; CONTRIBUTORS, 2024) foi utilizado para a representação de arquiteturas e outros esquemas explicativos sobre o projeto.

4.2.2 Design e Prototipagem

O design e a prototipagem foram realizados utilizando o **Figma** (Figma, 2024), o que possibilitou idealizar todos os possíveis componentes e telas que constituiriam o sistema. Além disso, o Figma permitiu demonstrar todas as interações por meio dos protótipos de alto nível.

4.2.3 *Front-end*

A camada de *Front-end* é toda a parte visual com a qual o usuário final irá interagir. Sua construção seguirá a idealização dos protótipos e será composta pelas seguintes tecnologias:

- **Astro:** A escolha do **Astro** (CONTRIBUTORS, 2024) se deu por ser um framework otimizado para sites, proporcionando um desempenho elevado, essencial para o carregamento rápido das páginas e para uma experiência de usuário fluida. O Astro permite a construção de aplicações modernas utilizando renderização estática de páginas **HTML** (BERNERS-LEE, 2023) e conjunto com a utilização da linguagem **JavaScript** (EICH, 2024) para desenvolver as lógicas das páginas, como estados dos componentes e também para realizar requisições à camada de *back-end* que será explicada posteriormente.

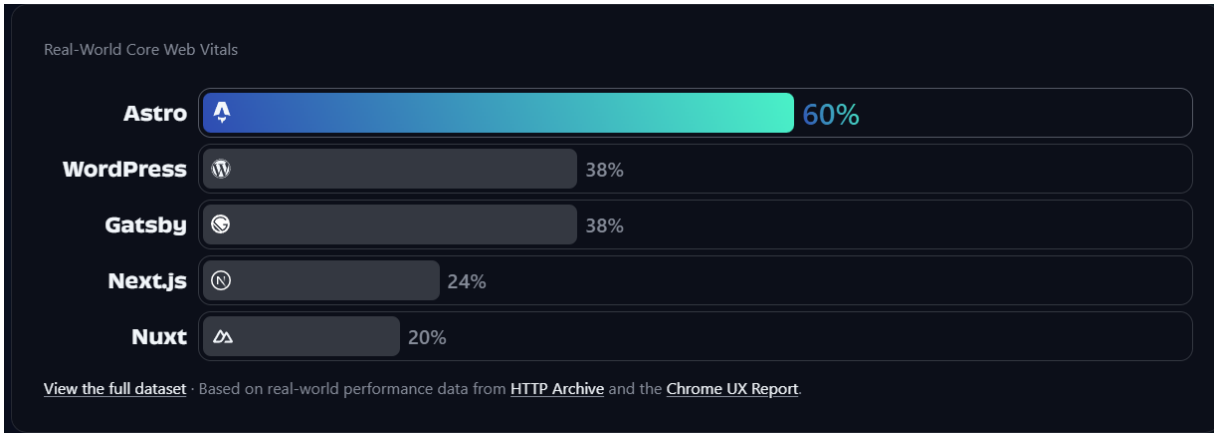


Figura 7 – Desempenho do Astro comparado a outros frameworks de front-end

Fonte: (CONTRIBUTORS, 2024)

- **Tailwind CSS:** Para a estilização das páginas, foi utilizado o **Tailwind CSS** (INC., 2024). Diferente do **CSS** (LIE, 2023) (Cascading Style Sheets), esse framework de utilitários CSS facilita a criação de interfaces consistentes e responsivas, permitindo que os estilos sejam aplicados diretamente aos componentes HTML. Além disso, ele se integra perfeitamente ao Astro, simplificando o processo de estilização sem a necessidade de criar folhas de estilo CSS extensas e separadas.

4.2.4 *Back-end*

O *back-end* será responsável por conter toda a lógica da aplicação e por intermediar as requisições do *front-end* para o banco de dados e as interações com o chatbot, além de implementar as regras de negócio relacionadas à manipulação das informações.

Para o desenvolvimento do *back-end*, foi utilizado **JavaScript**, mais especificamente no *framework* **Node.js** (CONTRIBUTORS, 2024). O Node.js permite a execução de códigos JavaScript fora do ambiente do navegador, possibilitando a criação de aplicações escaláveis e de alto desempenho.

No ambiente Node.js, foi utilizado o framework **Express** (Express.js, 2023), que oferece uma estrutura mais rápida e flexível em comparação com a utilização unicamente do Node.js. O Express facilita a criação de rotas e o gerenciamento de requisições HTTP, tornando o desenvolvimento mais ágil e organizado.

Como Sistema de Gerenciamento de Banco de Dados (SGBD), optou-se pelo **MongoDB** (MongoDB, 2024), uma tecnologia de banco de dados não relacional que armazena dados em documentos, especialmente no formato JSON. O MongoDB é altamente flexível, suportando grandes volumes de dados e permitindo escalabilidade horizontal.

Para a manipulação do banco de dados através do *back-end*, foi utilizado um ORM *(Object-Relational Mapping)*, neste caso o **Mongoose** (AUTOMATTIC, 2024), é possível gerenciar os esquemas para os documentos e coleções, garantindo que os dados

estejam sempre estruturados conforme as necessidades da aplicação, além de fornecer funcionalidades adicionais como validação e *middleware* para gerenciar a lógica de negócios.

Adicionalmente, foram integradas as seguintes tecnologias e bibliotecas para fortalecer a funcionalidade e segurança do *back-end*:

- **JWT (jsonwebtoken):** Utilizado para autenticação baseada em tokens, garantindo que apenas usuários autorizados possam acessar determinadas rotas da API. O **JWT** (AUTH0, 2024) permite a troca segura de informações entre o cliente e o servidor.
- **Axios:** O Axios (GitHub, Acesso em 1 de agosto de 2023) é uma biblioteca JavaScript utilizada para realizar requisições HTTP de forma fácil e intuitiva.
- **bcrypt:** Biblioteca utilizada para realizar o hashing e a verificação de senhas, proporcionando uma camada extra de segurança no armazenamento de credenciais de usuários. Através do **bcrypt** (GALLAGHER, 2024), é possível armazenar senhas de forma segura, evitando o acesso não autorizado.
- **Nodemailer:** Utilizado para o envio de e-mails, como notificações e recuperação de senhas. O **Nodemailer** (RYS, 2024) facilita a integração com serviços de e-mail, permitindo uma comunicação eficaz entre a aplicação e os usuários.
- **Swagger:** Ferramenta que permite a documentação automática da API, proporcionando uma interface interativa para que desenvolvedores e usuários testem as rotas disponíveis. O **Swagger** (SOFTWARE, 2024) melhora a usabilidade e facilita a integração com outras aplicações.
- **Vitest e Supertest:** Utilizadas para realizar testes de unidade e de integração, assegurando que os componentes do *back-end* funcionem como esperado. O **Vitest** (VITE, 2024) é um framework de testes leve e rápido, enquanto o **Supertest** (VISIONMEDIA, 2024) é utilizado para testar as rotas HTTP da API.

Além disso, foram utilizadas as seguintes dependências e ferramentas para melhorar o fluxo de trabalho no ambiente de desenvolvimento:

- **Nodemon (SHARP, 2024):** Uma ferramenta que permite a recarga automática do servidor durante o desenvolvimento, aumentando a eficiência ao evitar a necessidade de reiniciar o servidor manualmente após cada alteração de código.
- **Morgan (WILSON, 2024):** Um middleware que fornece *logging* de requisições HTTP, permitindo o monitoramento das interações com a API e facilitando a identificação de problemas.
- **Swagger-autogen (G., 2024):** Ferramenta que gera automaticamente a documentação do Swagger com base nas rotas definidas na aplicação, assegurando que a documentação esteja sempre atualizada e alinhada com a implementação.
- **BiomeJS (BIOME, 2024):** Utilizado para análise de qualidade de código, ajudando a manter as melhores práticas de codificação e garantindo que o código da aplicação esteja limpo e sustentável.
- **Postman (Postman, 2024):** Ele permite que os desenvolvedores enviem requisições

HTTP com diferentes métodos (GET, POST, PUT, DELETE) e analisem as respostas da API, facilitando o processo de depuração e validação das APIs desenvolvidas.

- **MongoDBCompass (MONGODB, 2024a):** O MongoDB Compass é uma interface gráfica oficial para o MongoDB que permite visualizar, explorar e manipular dados em bancos de dados MongoDB intuitivamente.

4.2.5 Chatbot

Acerca da tecnologia de chatbot adotada, conforme o comparativo na tabela de chatbots (Tabela 1), o **Rasa** (Rasa, 2023a) foi escolhido por oferecer melhor flexibilidade e personalização conforme as regras de negócios e os possíveis requisitos do sistema. Em conjunto com o Rasa, foram utilizadas as seguintes bibliotecas Python (Python Software Foundation, 2024):

- **rasa-sdk** Biblioteca do Rasa para suportar ações personalizadas (TECHNOLOGIES, 2024).
- **tensorflow** Biblioteca para aprendizado de máquina (AUTHORS, 2024).
- **spacy:** Biblioteca para processamento de linguagem natural (AI, 2024).
- **pydantic** Biblioteca para validação de dados (COLVIN, 2024).
- **pymongo** Para conexão e manipulação do banco de dados MongoDB (MONGODB, 2024b).
- **python-dotenv** Para gerenciar variáveis de ambiente sensíveis (VAUGHAN, 2024).

4.2.6 Ambientes de Desenvolvimento

Para o desenvolvimento do projeto, foram utilizadas várias tecnologias que facilitaram o fluxo de trabalho e a organização do projeto. O ambiente de desenvolvimento foi configurado com o **Visual Studio Code** (MICROSOFT, 2024), sendo o editor de código adotado para o projeto, no qual oferece uma interface intuitiva e suporte a diversas linguagens de programação.

Para controle de versão, foi utilizado o **Git** (STRAUB, 2024), um sistema de controle de versões amplamente adotado, permitindo o rastreamento de alterações no código e facilitando a colaboração no projeto.

Os repositórios do projeto foram hospedados no **GitHub** (GITHUB, 2024), um serviço de hospedagem de código-fonte que oferece recursos de colaboração e gerenciamento de projetos.

Além disso, para a conteinerização de partes do sistema, foi utilizado o **Docker** (Docker Inc., 2024), uma plataforma que permite a criação e gerenciamento de contêineres, garantindo um ambiente consistente para o desenvolvimento.

5 PROPOSTA

Neste capítulo será apresentada uma visão geral da proposta do projeto, a qual está organizada em duas seções principais. A primeira, **5.1 Execução e Acompanhamento das Sprints**, abrange o planejamento e a execução das principais etapas do desenvolvimento conforme a metodologia de projeto apresentada previamente, enquanto a segunda, **5.2 Arquitetura Resultante**, descreve a arquitetura final construída ao longo de todo desenvolvimento.

5.1 Execução e Acompanhamento das Sprints

Este subcapítulo aborda o andamento das sprints, começando pelo levantamento de requisitos, seguido pela modelagem, prototipação, desenvolvimento *back-end*, configurações do chatbot, conteinerização com Docker, integração do *back-end* com o chatbot e, por fim, o desenvolvimento *front-end.* e integração com o *back-end.*

5.1.1 Levantamento de Requisitos

Esta subseção aborda todo o levantamento de requisitos para o projeto, incluindo a explicação dos dois tipos de requisitos: funcionais e não funcionais. Também serão apresentadas ambas as pesquisas realizadas com o público-alvo do sistema e, por fim, a composição das tabelas de requisitos.

5.1.1.1 Requisitos Funcionais e Não Funcionais;

O levantamento de requisitos, sejam funcionais ou não, é fundamental para o êxito no desenvolvimento de um software, ao serem responsáveis por descrever o que o sistema deve fazer e como ele deve fazer para atender possíveis demandas dos usuários, pois em conjunto com o escopo necessitam ser documentados e compreendidos, pois caso não sejam estabelecidos, tem potencial de trazer possíveis problemas futuros, que podem atrasar todo o andamento do projeto. Para realizar um levantamento adequado desses requisitos é preciso compreende-lós.

Os requisitos funcionais trata-se de funcionalidades específicas que o sistema deve oferecer. Descrevendo assim comportamentos esperados do sistema em relação às interações dos usuários.

Por outro lado, os requisitos não funcionais refere-se a características e particularidades do sistema que não estão exatamente interligadas com as suas funcionalidades específicas. Representando deste modo como o sistema deve executar suas ocupações e quais são seus aspectos de desempenho, segurança, usabilidade e confiabilidade.

Para o levantamento dos requisitos do sistema, foram realizadas pesquisas com os *Stakeholders*, isso é, o grupo de interesse no qual o sistema atenderá, refere-se aos discentes e os coordenadores.

5.1.1.2 Pesquisa Coordenadores

A pesquisa com os coordenadores foi realizada digitalmente por meio de um formulário do **Google Forms** (Google, 2024), encaminhado via e-mail para os coordenadores dos cursos da UNIFESSPA. O formulário foi composto pelas seguintes seções:

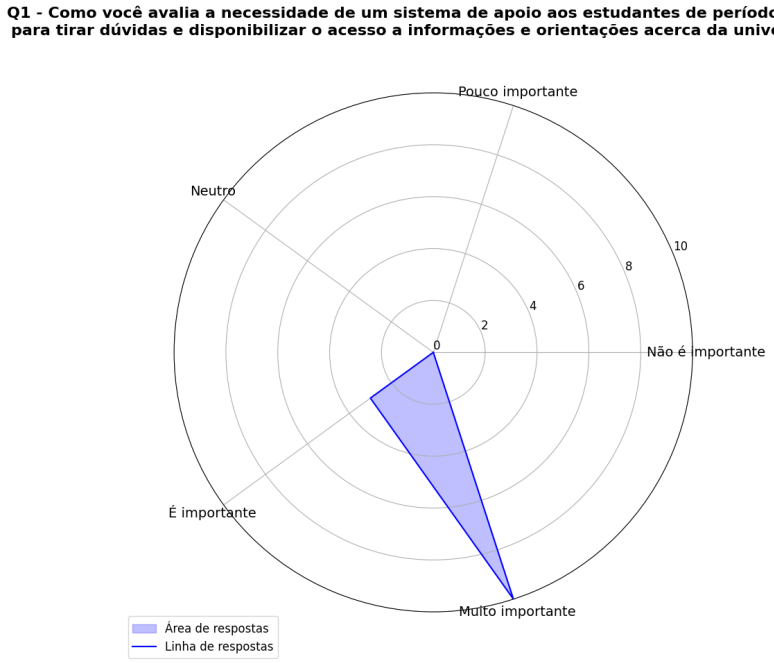
Tabela 3 – Estrutura do Formulário de Pesquisa com Coordenadores

Item	**Descrição**
Termo de Consentimento Livre e Esclarecido (TCLE)	Termo que garante que uma pessoa compreendeu todas as informações relevantes sobre o estudo antes de dar sua autorização para participar.
E-mail institucional	Campo para o preenchimento do e-mail oficial do coordenador.
Curso que representa	Campo para informar o curso no qual faz parte da coordenação.
Q1 — Como você avalia a necessidade de um sistema de apoio aos estudantes de períodos iniciais, para tirar dúvidas e disponibilizar o acesso a informações e orientações acerca da universidade?	Opções: “Não é importante”, “Pouco importante”, “Neutro”, “É importante” e “Muito importante”.
Q2 — Você vê benefícios na implementação de um Chatbot (um sistema de conversação virtual que responde a perguntas) como parte de um sistema de apoio aos calouros para oferecer suporte aos alunos que estão ingressando na instituição de ensino?	Opções: “Sim”, “Não” e “Talvez”.
Q3 — Você acha interessante que o sistema permita aos diretores/coordenadores alimentarem o Chatbot com informações específicas sobre o curso?	Opções: “Sim”, “Não” e “Talvez”.
Q4 — Quais outras funcionalidades poderiam ser úteis em um sistema de apoio a calouros/estudantes dos períodos iniciais?	Campo Dissertativo.
Q5 — Considerando a experiência e conhecimento das direções de curso, quais informações você considera importantes para contemplar em um sistema de apoio para calouros e serem repassadas pelo Chatbot?	Campo Dissertativo.
Espaço aberto para observações e sugestões	Campo Dissertativo.

Foram recebidas 13 respostas ao formulário, gerando resultados quantitativos e qualitativos. As respostas quantitativas das perguntas fechadas (**Q1**, **Q2**, **Q3**) foram

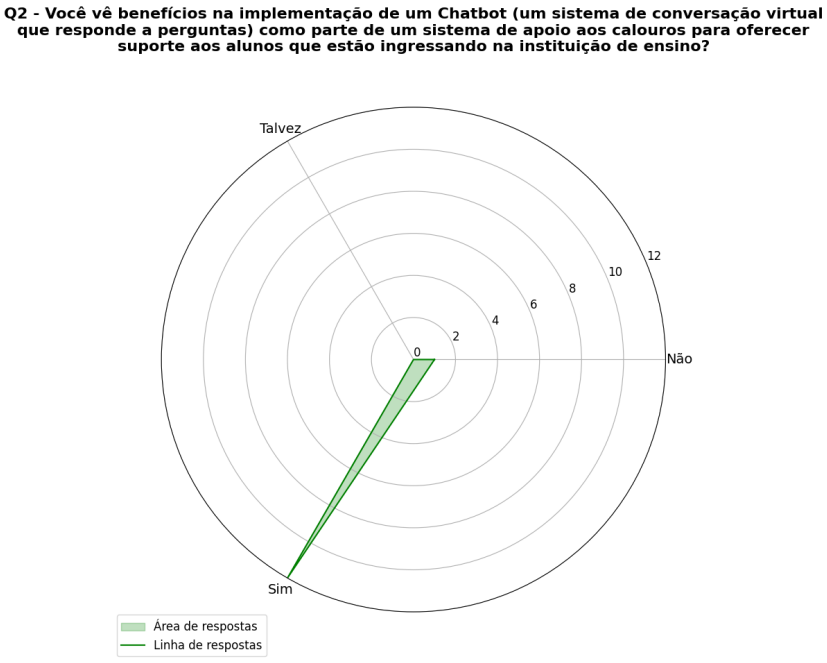
plotadas em gráficos com o auxílio das bibliotecas **Matplotlib** (HUNTER, 2007) e **Numpy** (HARRIS et al., 2020), apresentados a seguir:

Figura 8 – Gráfico Quantitativo Pergunta Fechada 1



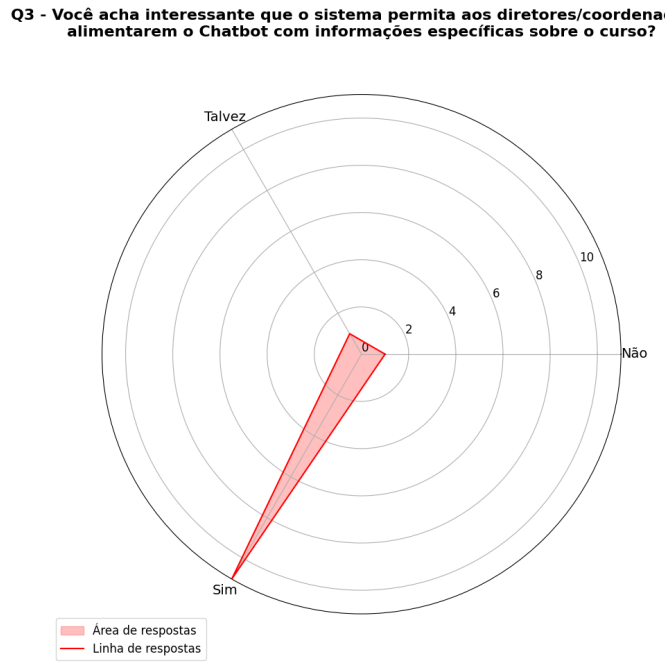
Fonte: Autoria Própria

Figura 9 – Gráfico Quantitativo Pergunta Fechada 2



Fonte: Autoria Própria

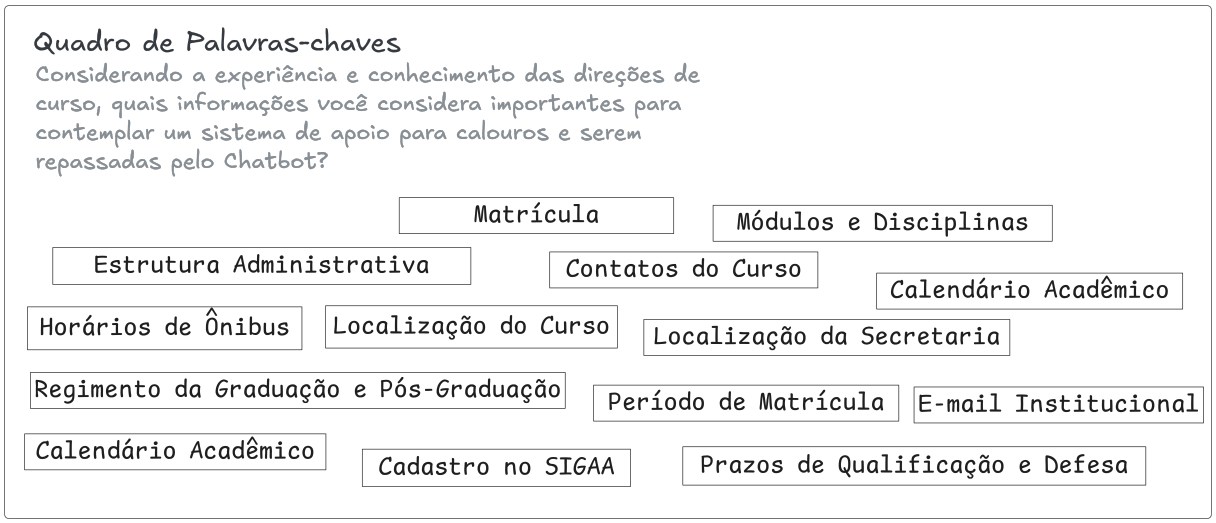
Figura 10 – Gráfico Quantitativo Pergunta Fechada 3



Fonte: Autoria Própria

Além dos dados quantitativos, as respostas qualitativas às perguntas **Q4** e **Q5** forneceram percepções valiosos. A pergunta **Q5**, especificamente, gerou um conjunto de palavras-chave a partir das respostas dissertativas dos coordenadores, como mostrado no quadro abaixo:

Figura 11 – Quadro de Palavras-chave sobre a Pergunta Q5



Fonte: Autoria Própria

A partir da análise da pergunta **Q5**, que questionava sobre as informações consideradas importantes para o sistema de apoio aos calouros, surgiram sugestões como: "Informações gerais, como horários de ônibus, horários de atendimento presencial na Uni-

versidade, atividades extraclasse, etc.". O quadro de palavras-chave extraído das respostas qualitativas dessa pergunta revela requisitos interligados e enfatiza a necessidade de incluir informações institucionais detalhadas no Chatbot, permitindo maior autonomia e clareza para os alunos.

Dessa forma, a análise dos gráficos e do quadro de palavras-chave evidencia a importância de um sistema integrado, que facilite o acesso a informações acadêmicas e administrativas por parte dos discentes, com uma ênfase especial na utilidade de um Chatbot para os períodos iniciais.

5.1.1.3 Pesquisa Discentes

Para a pesquisa em relação aos discentes, optou-se por utilizar perguntas frequentes do Regulamento do Ensino de Graduação — CRCA. Essa escolha foi realizada com as seguintes justificativas:

- **Relevância e Abrangência:** As categorias e as perguntas que as compõe, abordam tópicos recorrentes na experiência do aluno ingressante, temáticas como matrícula, trancamento de matrícula e dúvida sobre o SIGAA
- **Concordância as Normas Institucionais:** O regulamento de ensino de graduação disponível no CRCA, representa legalmente as políticas presentes na instituição, deste modo assegurando que essas informações quando utilizadas para levantamento de requisitos para os discentes e como base de informações para o chatbot, concordam com regulamento e assegurando fonte confiável de informação.
- **Eficiência no Desenvolvimento:** Ao utilizar informações tratadas, ganha-se tempo ao evitar o processamento dessas informações, tal como se assegura que os dados em parte foram validados gramaticalmente.

Portanto, com base no Regulamento do Ensino de Graduação — CRCA, dentre as 13 categorias presentes, foram escolhidas apenas 6, totalizando 15 perguntas. O critério selecionou as perguntas mais próximas do período de ingresso. Essas perguntas serão utilizadas para determinar os requisitos referentes aos discentes e servirão como fonte de informação para o chatbot.

Tabela 4 – Categorias e Perguntas Escolhidas do CRCA

Categoria	**Perguntas Escolhidas**
Habilitação de Candidatos Aprovados em Processos Seletivos	- Qual a diferença entre Habilitação e Matrícula? - Quais documentos devo apresentar na Habilitação?
Primeiro Acesso Discente ao SIGAA	- Como faço o cadastro no SIGAA? - Com o login usado no SIGAA posso ter acesso a quais sistemas da Universidade? - O que consigo fazer no Portal do Discente no SIGAA?
Matrículas	- Onde devo fazer minha primeira matrícula? - Tenho que fazer minha matrícula todo semestre? - Posso cursar o semestre sem estar matriculado? - Como faço a solicitação de matrícula on-line pelo SIGAA?
Frequência	- Qual o percentual mínimo de frequência para eu ser aprovado? - Em que situações posso ter minhas faltas abonadas?
Trancamentos de Matrículas	- O que quer dizer trancamento? - Como faço para solicitar trancamento? - Posso trancar o primeiro período do curso?
Conceitos e Frequências	- Que notas correspondem os Conceitos?

5.1.1.4 Tabela de Requisitos

Fundamentado nas pesquisas, foram criadas duas tabelas referentes aos dois tipos de requisitos do sistema.

A Tabela 5 mostra o conjunto de requisitos funcionais definidos, no qual são categorizados e ordenados por prioridade, eles definem funcionalidades essenciais dos usuários não autenticados, coordenadores, administradores e requisitos gerais em relação à utilização do sistema.

Tabela 5 – Requisitos Funcionais

ID	**Descrição**	**Prioridade**
Administrador		
RF1	O administrador deve gerenciar as intents globais e associar intents aos coordenadores para inserção de respostas personalizadas referentes ao seu curso.	Alta
RF2	O admnistrador deve realizar o cadastro inicial dos perfis dos coordenadores.	Alta
RF3	O administrador deve realizar o cadastro inicial dos cursos e vincular um perfil de coordenador existente ao curso criado.	Alta
RF4	O administrador deve gerenciar os perfis dos coordenadores (criação, edição, exclusão).	Média
RF5	O administrador deve gerenciar os cursos (criação, edição, exclusão).	Média
Coordenador		
RFC6	Permitir a autenticação do coordenador com os dados cadastrados e a mudança de senha inicial.	Alta
RFC7	Permitir ao coordenador gerenciar (criação, edição e exclusão) as respostas para intents associadas ao seu curso.	Alta
RFC8	Apresentar um dashboard para o coordenador visualizar informações referente aos dados do chatbot.	Alta
RFC9	Permitir ao coordenador gerenciar (criar, editar, excluir) módulos, disciplinas e tópicos do seu curso.	Média
RFC10	Apresentar um dashboard para o coordenador gerenciar as informações do curso, como descrição, objetivos e link para o PPC.	Baixa
Usuário Não Autenticado (Usuário Padrão)		
ID	**Descrição**	**Prioridade**
RF11	Permitir acesso à página de listagem de todos os cursos disponíveis.	Alta
RF12	Permitir acesso ao roadmap de um curso específico, visualizando seus módulos, disciplinas e tópicos associados.	Alta
RF13	Exibir uma página de chat para interação com o bot.	Alta
RF14	Permitir aos usuários não autenticados interagir com o chatbot sem a necessidade de login.	Alta
RF15	Exibir uma página explicativa sobre o chatbot e as tecnologias envolvidas.	Média
RF16	Permitir aos usuários realizar uma avaliação da conversação com o chatbot após a interação.	Média

A Tabela 6 apresenta o grupo de requisitos não funcionais estabelecidos. Em uma sequência de prioridade, eles definem as características e particularidades do sistema.

Tabela 6 – Requisitos Não Funcionais

ID	**Descrição**	**Prioridade**
RNF1	O sistema deve fornecer respostas em tempo real para as interações do chatbot, com uma latência máxima de 2 segundos.	Alta
RNF2	O tempo de carregamento das páginas não deve exceder 3 segundos para assegurar uma boa experiência do usuário.	Alta
RNF3	O sistema deve ser responsivo, adaptando-se automaticamente a dispositivos móveis, tablets e desktops.	Alta
RNF4	O sistema deve garantir a segurança dos dados por meio de autenticação robusta e criptografia de senhas e dados sensíveis.	Alta
RNF5	As interações do chatbot devem ser registradas para permitir análise futura de desempenho e melhoria contínua.	Alta
RNF6	O sistema deve ter uma interface amigável e intuitiva, adequada para usuários com pouca ou nenhuma experiência técnica.	Média
RNF7	O sistema deve informar aos usuários sobre os Termos de Uso, Políticas de Privacidade e Cookies, cumprindo as regulamentações de proteção de dados.	Média
RNF8	O sistema deve suportar o acesso simultâneo de múltiplos usuários sem degradação de desempenho.	Baixa
RNF9	O sistema deve implementar replicação automática de dados entre servidores para garantir alta disponibilidade e integridade.	Baixa
RNF10	O sistema deve realizar backups automáticos a cada 24 horas para garantir a persistência dos dados e rápida recuperação em caso de falha.	Baixa
RNF11	O sistema deve atender aos requisitos de acessibilidade, incluindo suporte a leitores de tela, navegação por teclado e cores acessíveis.	Baixa
RNF12	O sistema deve ser projetado para ser compatível com diferentes navegadores (Chrome, Firefox, Safari, Edge) e suas versões recentes.	Baixa

5.1.2 Modelagem

A modelagem do projeto envolve a definição dos atores em casos de uso de alto nível para uma compreensão simplificada, a estruturação dos dados que estarão presentes no banco de dados, tal como a definição de suas entidades que estarão sendo armazenadas e a adoção de padrões de arquitetura para o código *back-end*. Esta etapa é fundamental para garantir um desenvolvimento bem estruturado e eficiente, tanto em termos de manutenção quanto de escalabilidade.

5.1.2.1 Caso de uso de Alto Nível

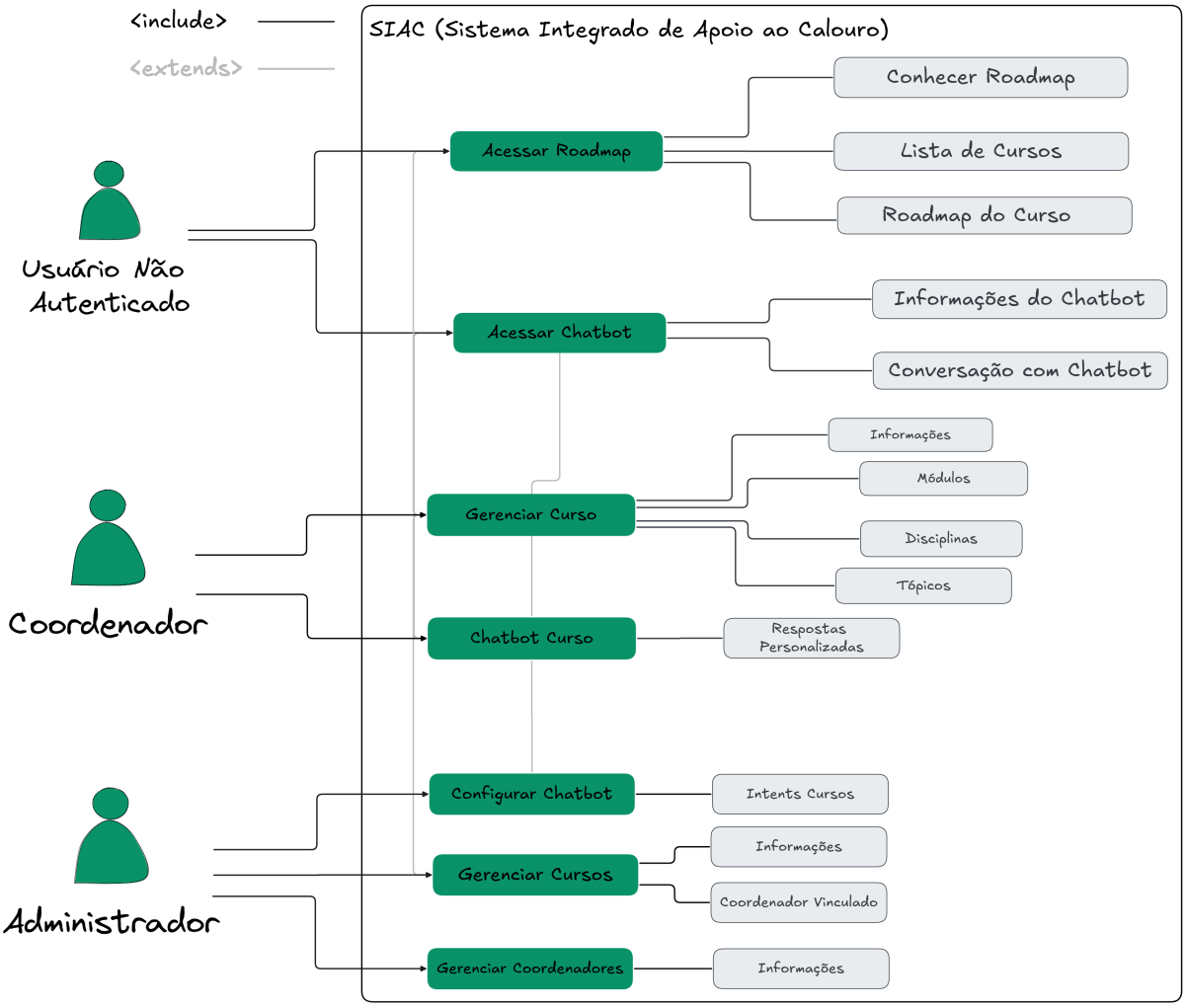
A utilização do diagrama de caso de uso de alto nível representa visualmente as principais interações entre os usuários e o sistema. Apresenta de forma simplificada e sem detalhes os casos de uso para os diversos usuários, onde na Figura 12 observam-se as representações dos atores (usuários) do sistema por meio de ícones e sua interligação aos principais casos de uso nas caixas verdes, que estão conectados a subcomponentes representando casos de uso específicos. Dentre as interligações dos casos de uso, temos dois tipos:

- **Include (<include>):** Indica que um caso de uso incorpora obrigatoriamente o comportamento de outro. No diagrama, embora não explicitamente marcado, infere-se que "Acessar Roadmap" inclui "Lista de Cursos" e "Roadmap do Curso". Isso significa que, ao acessar o roadmap, o usuário necessariamente terá acesso à lista de cursos e ao roadmap específico de cada curso.
- **Extend (<extends>):** Representa uma extensão opcional de um caso de uso. Embora não esteja explicitamente marcado no diagrama, pode-se considerar que o "Gerenciar Curso" pelo Coordenador estende o "Acessar Roadmap", pois as informações gerenciadas pelo coordenador eventualmente se refletem no roadmap acessível aos usuários não autenticados.

Já os usuários externos que interagem com o sistema temos os seguintes perfis:

1. **Usuário Não Autenticado:** Possui acesso limitado, podendo explorar o roadmap e interagir com o chatbot geral, sem necessidade de autenticação.
2. **Coordenador:** Responsável pela gestão do curso vinculado ao seu perfil e pela personalização das respostas do chatbot do seu curso.
3. **Administrador:** Detém o maior nível de acesso, gerenciando a configuração das intents do curso, informações dos cursos e perfis de coordenadores.

Figura 12 – Caso de Uso de Alto Nível SIAC



Fonte: Autoria Própria

Esta representação visual simplificada permite uma rápida compreensão da estrutura do SIAC, conforme os requisitos, destacando as principais funcionalidades disponíveis para cada tipo de usuário e as interações fundamentais com o sistema.

5.1.2.2 Banco de Dados

Conforme definido na Seção 4.2, o banco de dados **MongoDB** (MongoDB, 2024) foi selecionado devido às suas características de flexibilidade, escalabilidade horizontal, e facilidade de manuseio de grandes volumes de dados não estruturados ou semi-estruturados, como documentos JSON. Essas vantagens tornam o MongoDB uma escolha ideal para aplicações que lidam com dados dinâmicos, como chats e logs de interação com chatbots. Além disso, o suporte nativo as coleções distribuídas e a replicação automática de dados se alinha aos requisitos de disponibilidade e integridade da aplicação.

Como esse SGBD não trabalha com relacionamento tradicional, e sim com documentos, foi necessário representar quais documentos estariam presentes em coleções. Vale

lembrar que documentos e coleções no MongoDB seguem os seguintes conceitos:

- **Documento:** Uma unidade básica de armazenamento de dados, representando um registro em um banco de dados. No MongoDB, os documentos seguem a estrutura BSON (Binary JSON), permitindo a representação hierárquica e semi-estruturada dos dados. Um documento é construído com pares de chave-valor, onde a chave é uma String representando o campo, e o valor é o conteúdo associado à chave.
  Exemplo de documento:

```
{
    "nome": "Gustavo P.",
    "idade": 21,
    "endereco": {
        "rua": "Rua Principal",
        "numero": 123
    }
}
```

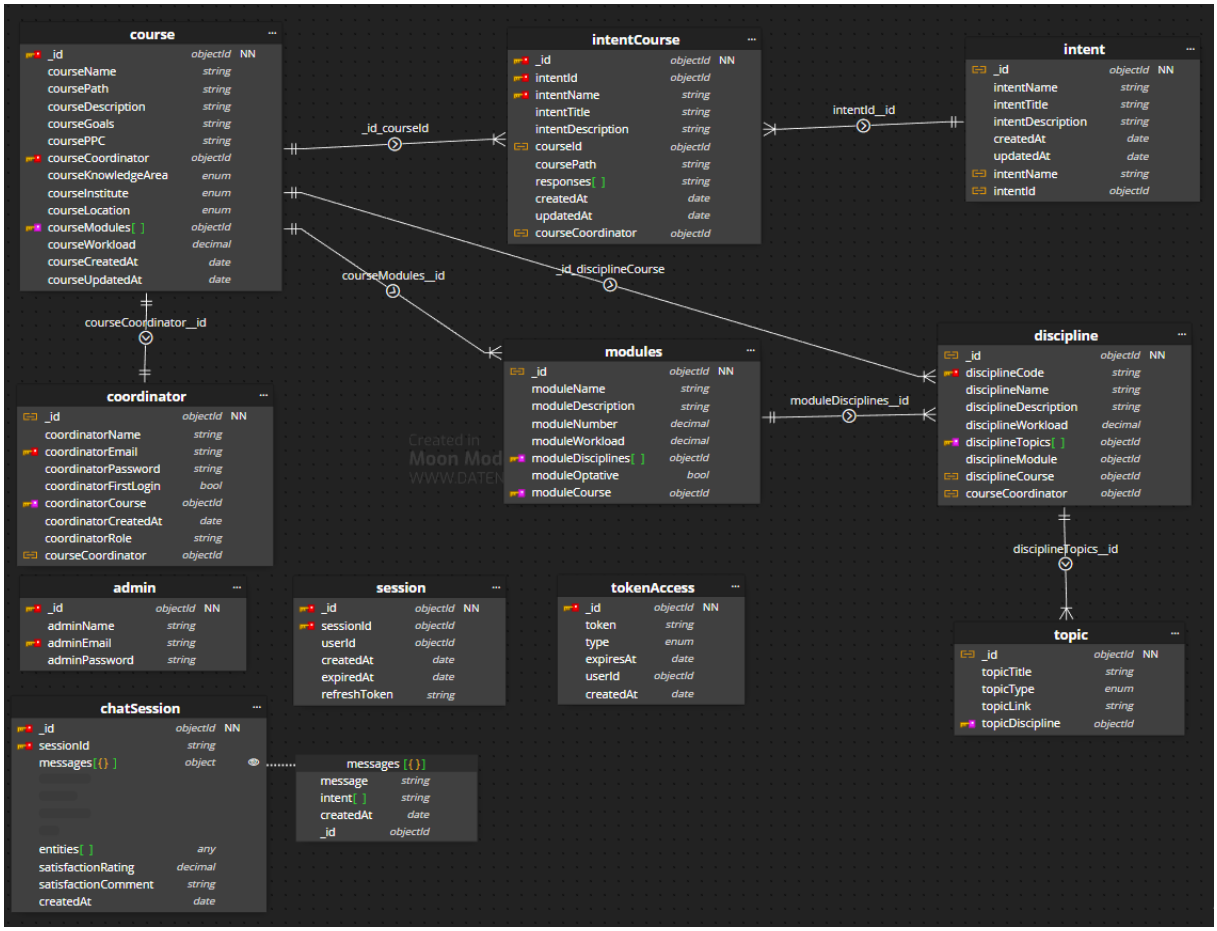
- **Coleção:** Um conjunto de documentos no MongoDB, semelhante a uma tabela em bancos de dados relacionais, mas sem esquema fixo, permitindo que os documentos em uma mesma coleção possuam campos distintos.

5.1.2.2.1 Modelagem UML do Banco de Dados

A modelagem do banco de dados foi elaborada com base no padrão de documentos do MongoDB. O foco principal foi criar uma estrutura flexível que permitisse a expansão e modificação do sistema ao longo do tempo, mantendo a integridade dos dados.

No diagrama desenvolvido utilizando a ferramenta de Modelagem **Moon Modeler** (DATENSEN, 2024), foi possível representar, por meio de um UML de classes (Figura 13), a estrutura principal do banco de dados MongoDB, com suas coleções e relações entre entidades. Cada coleção representa um conjunto de documentos, onde os campos são dinâmicos e podem ser expandidos conforme necessário, facilitando a adição de novas funcionalidades ao longo do projeto.

Figura 13 – Diagrama UML de Classe Mongodb



Fonte: Autoria Própria

Abaixo, descrevemos brevemente a função de cada coleção e o objetivo geral dos seus campos:

1. **Admin:** Guarda dados dos administradores do sistema, como credenciais e permissões.
2. **ChatSession:** Contém informações sobre as sessões de chat entre usuários e o chatbot, incluindo logs de interação.
3. **Coordinator:** Armazena dados dos coordenadores responsáveis por gerenciar os cursos.
4. **Course:** Armazena informações relacionadas aos cursos oferecidos. Inclui dados como nome do curso, descrição, objetivos, link para o PPC, e associações com módulos e coordenadores.
5. **Discipline:** Armazena as disciplinas de cada módulo dos cursos. As disciplinas estão associadas a um curso e um módulo específico e podem conter tópicos.
6. **Intent Course:** Contém intents específicas de cursos que o chatbot deve reconhecer e responder. Cada intent está diretamente relacionada a um curso e pode conter respostas personalizadas.
7. **Intent:** Armazena modelos genéricos de intents que servem como base para a criação

de intents específicas de cada curso.

8. **Module:** Contém os módulos que estruturam os cursos, agrupando disciplinas e outras informações.
9. **Session:** Armazena informações sobre as sessões dos usuários, tempo de expiração e armazenamento do "refreshToken", responsável por evitar a autenticação vários vezes.
10. **TokenAccess:** Gerencia os "tokenAccess", responsável por realizar a autenticação dos usuários que solicitarem recursos.
11. **Topic:** Representa tópicos específicos em uma disciplina, detalhando o conteúdo a ser abordado.

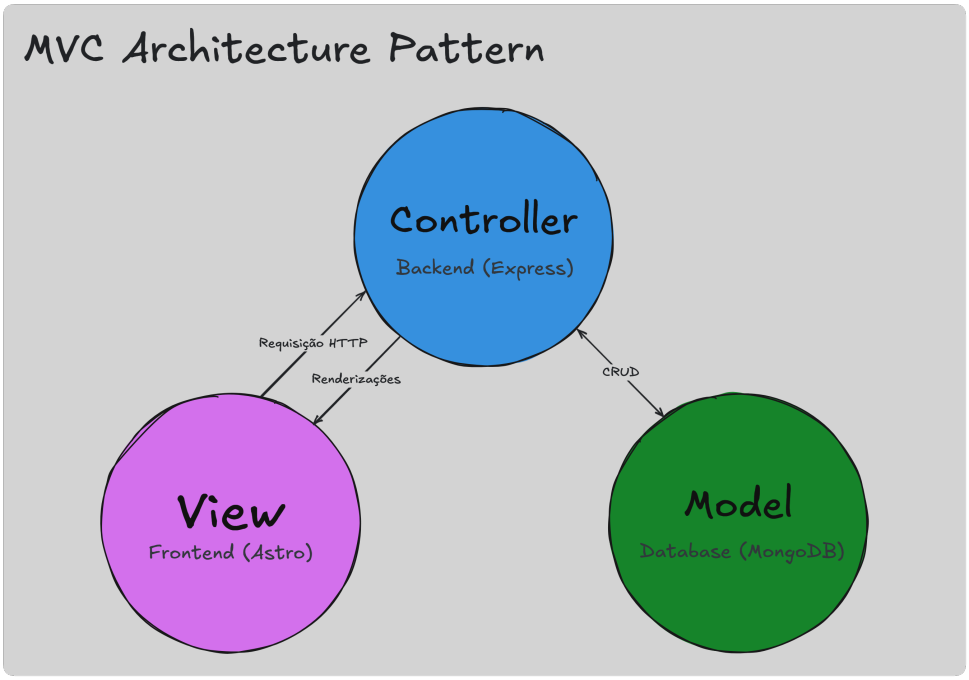
5.1.2.3 Padrão de Arquitetura MVC

A arquitetura MVC (Model-View-Controller) foi adotada para organizar o código do *back-end*. Este padrão separa as responsabilidades da aplicação em três camadas principais: Modelos, Controladores e Visões, permitindo uma melhor modularização, manutenção e expansão futura do sistema.

- **Model (Modelo):** Responsável pela representação dos dados e pela lógica de manipulação das informações no banco de dados.
- **View (Visão):** Camada de interface que apresenta as informações ao usuário, sendo um componente importante na interação com o chatbot e na exibição do conteúdo de todos os conteúdos.
- **Controller (Controlador):** Atua como intermediário entre o modelo e a visão, processando as requisições dos usuários e atualizando os modelos conforme necessário.

No projeto, o padrão MVC foi aplicado conforme descrito na Figura 14. O controlador é responsável por gerenciar as requisições do usuário, autenticado ou não, direcionando as ações apropriadas, como listar cursos, acessar roadmaps ou interagir com o chatbot. O modelo foi implementado utilizando o MongoDB, proporcionando uma estrutura flexível para armazenar informações sobre coordenadores, cursos e interações com o chatbot. Enquanto a camada de visão é de responsabilidade do *front-end*, encarregada de renderizar a interface e consumir os recursos fornecidos pelo *back-end*, como APIs para interação com o chatbot e exibição de dados sobre cursos.

Figura 14 – Padrão de Arquitetura MVC Aplicado

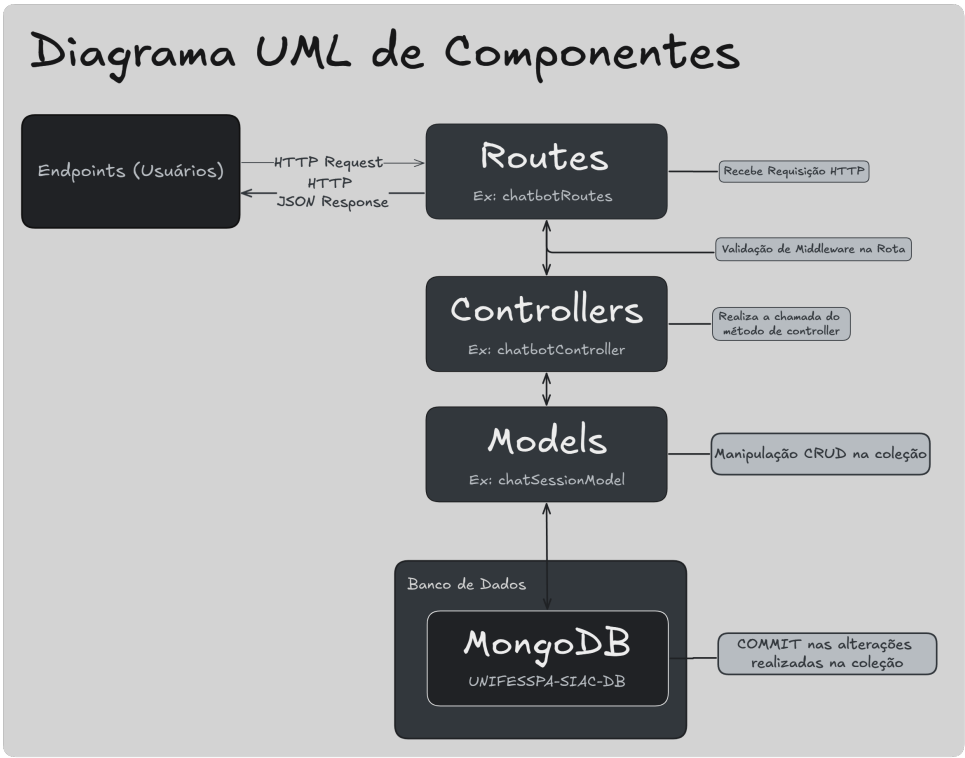


Fonte: Autoria Própria

5.1.2.3.1 Diagrama de Componentes *Back-end*

O diagrama de componentes, conforme ilustrado na Figura 15, apresenta o modelo de organização do código do *back-end* e descreve como toda a interação com o sistema funcionará. Observando o diagrama, temos o exemplo de um usuário que realiza uma requisição a uma rota — por exemplo, a rota do chatbot ("chatbotRoutes"). O componente de rota direciona a requisição ao controlador associado, neste caso, o "chatbotController", que executa o método correspondente à rota acionada. Se esse método requer interação com o banco de dados, o modelo associado aos dados, como o "chatSessionModel", é acionado para realizar as operações necessárias. Após a operação ser executada no modelo, ocorre o salvamento (ou "commit") no banco de dados. Em seguida, o fluxo retorna, passando pela mesma cadeia de componentes até que a resposta seja enviada de volta ao usuário, geralmente no formato de uma JSON Response.

Figura 15 – Diagrama UML de Componentes *Back-end*



Fonte: Autoria Própria

5.1.3 Prototipação

A etapa de prototipação é essencial para fornecer uma visão clara dos componentes e das interações finais que os usuários terão com o sistema. A construção dos protótipos foi baseada nas tabelas de requisitos funcionais (Tabela 5) e não funcionais (Tabela 6), além do caso de uso apresentado anteriormente (Figura 12). O processo seguiu uma abordagem de alta fidelidade, ou seja, as telas projetadas são interativas e muito próximas da versão final do *front-end*.

O desenvolvimento começou com a definição dos fluxos dos diferentes perfis de usuários (Subseção 5.1.3.1), seguido pela organização das páginas em agrupamentos (Subseção 5.1.3.2. O processo foi concluído com a criação dos protótipos de alta fidelidade (Subseção 5.1.3.3) e a apresentação dos resultados (Subseção 5.1.3.4).

5.1.3.1 Fluxos de Usuários

Foram definidos fluxos que detalham mais detalhadamente as possíveis ações que um usuário pode realizar e as páginas envolvidas nesse processo. Abaixo estão os fluxos:

- **Usuário não autenticado:**
  - **Fluxo 1:** Pode acessar a página inicial e visualizar o roadmap de cursos. O fluxo começa na página “Roadmap Bem-vindo(a)”, onde o usuário seleciona um curso

de interesse na “Roadmap Lista de Cursos” e, em seguida, visualiza o roadmap detalhado do curso na página “Roadmap Curso”.
  - **Fluxo 2:** Acessa a página sobre o chatbot, que contém informações referentes ao funcionamento do chatbot, e acessa posteriormente a página de conversação do chatbot.
- **Coordenador:** Após o login e a troca da senha inicial, o coordenador pode acessar a página de gerenciamento do curso. Nesse fluxo, ele pode gerenciar módulos, disciplinas, e interações com o chatbot, além de atualizar informações sobre os cursos.
- **Administrador:** Após realizar o login, o administrador pode acessar as páginas de gerenciamento de coordenadores e cursos. Ele também tem acesso ao gerenciamento do chatbot, permitindo a configuração de respostas personalizadas para os cursos.

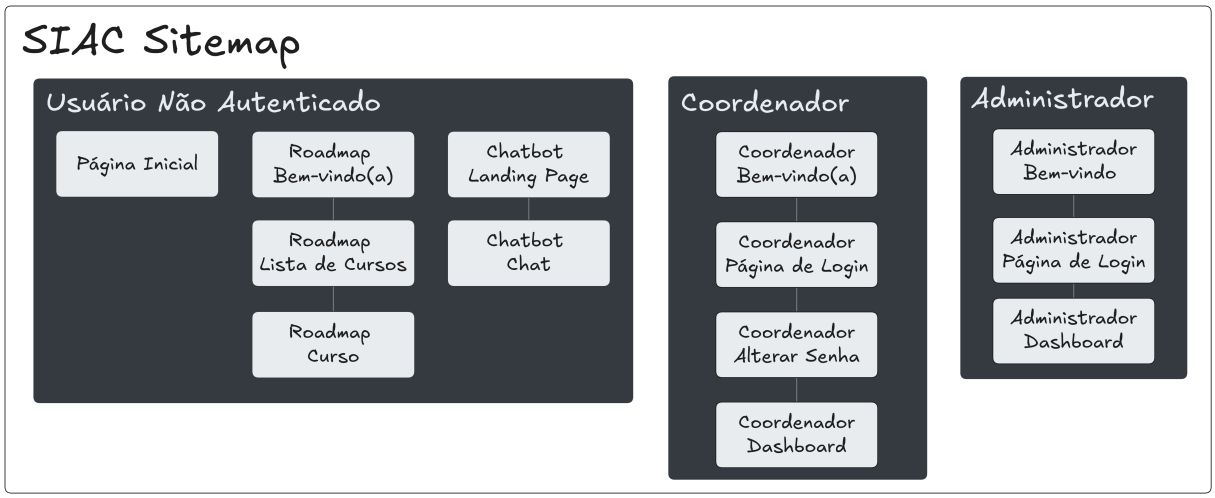
Esses fluxos mostram as ações principais que cada perfil de usuário pode realizar e as telas envolvidas em cada processo. Eles foram cuidadosamente planejados para garantir uma navegação intuitiva e eficiente no sistema.

5.1.3.2 Sitemap SIAC

Na Figura 16, podemos observar o conjunto de páginas propostas com base nos fluxos, organizadas nos seguintes grupos:

- **User:** Trata-se de páginas que não requerem autenticação. Inclui a página inicial e o fluxo de telas do Roadmap, começando pela “Roadmap Bem-vindo(a)”, seguida pela “Roadmap Lista de Cursos” e o “Roadmap Curso”, que detalha o roadmap do curso selecionado. Também estão incluídas as páginas relacionadas ao chatbot, como a página de informações e a página de conversação, onde ocorre a interação com o chatbot.
- **Coordinator:** Conjunto de páginas voltadas para o perfil do coordenador. A partir do segundo bloco, essas páginas requerem autenticação. Ao final desse grupo, temos a página de gerenciamento do curso e informações referentes ao chatbot.
- **Admin:** Páginas referentes ao perfil do administrador, começando pela página de boas-vindas e autenticação, e finalmente, pela página de gerenciamento dos perfis de coordenadores, cursos e informações do chatbot.

Figura 16 – Páginas SIAC



Fonte: Autoria Própria

5.1.3.3 Prototipação das Telas

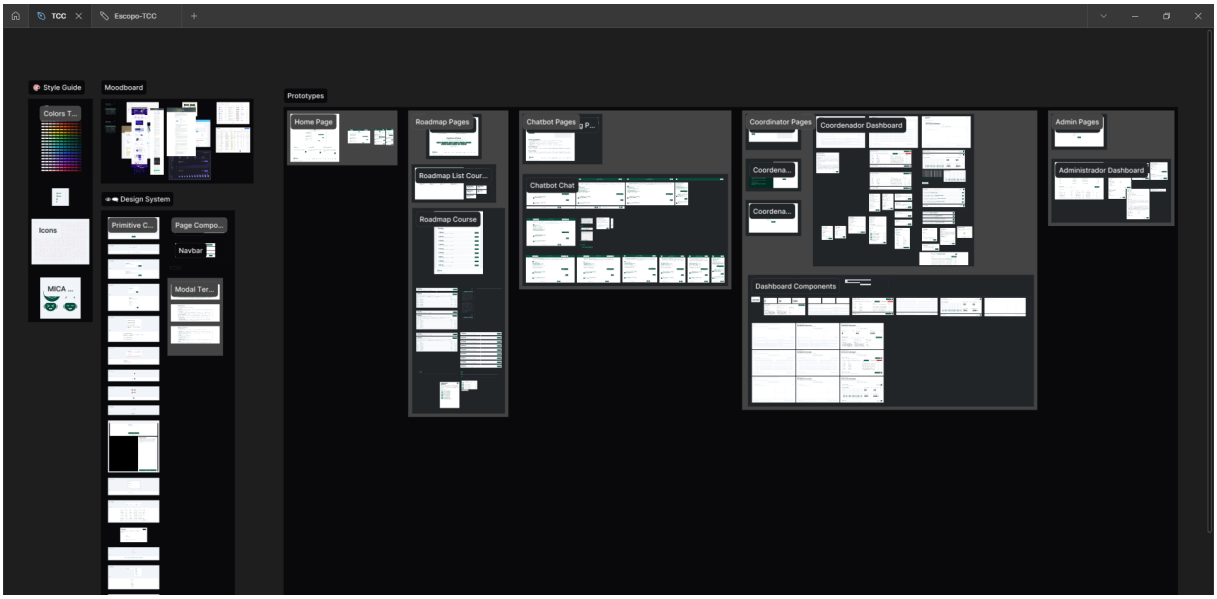
A etapa de prototipação das telas foi realizada utilizando o *software* **Figma** (Figma, 2024). Conforme o *sitemap* (Figura 16) e fluxo de usuários (Subseção 5.1.3.1) definidos precedentemente, o arquivo Figma foi organizado nas seguintes seções:

- **Style Guide:** Define as cores, fontes, ícones e imagens que serão utilizados no projeto.
- **Moodboard:** Quadro de referências que serão utilizadas para desenvolvimento dos protótipos.
- **Design System:** Organiza os componentes que serão utilizados nos protótipos. Existem componentes primitivos, como botões e seletores, que estarão presentes em várias páginas, assim como componentes mais específicos, como uma janela de visualização dos tópicos associados a disciplinas de um módulo em um roadmap, que pode ser utilizada apenas na página de roadmap do curso.
- **Prototypes:** Seção que agrupa os protótipos finalizados. Dentro dessa seção, encontramos os seguintes protótipos:
  - **Home Page:** Protótipos referentes à página inicial, com fluxo de navegação básico para usuários não autenticados.
  - **Roadmap Pages:** Protótipos das telas que envolvem o fluxo de acesso ao roadmap de um curso, detalhando as etapas e interações do usuário.
  - **Chatbot Pages:** Protótipos das telas voltadas para a interação com o chatbot, incluindo informações e a interface de conversação.
  - **Coordinator Pages:** Protótipos das páginas destinadas ao coordenador, como o painel de controle e as ferramentas de gerenciamento de cursos e módulos.
  - **Admin Pages:** Protótipos das páginas destinadas ao administrador, incluindo o gerenciamento de coordenadores, cursos e interações do chatbot.

5.1.3.4 Protótipos Resultantes

Com base na Figura 17, podemos observar o arquivo resultante no Figma — Protótipo SIAC Figma, que contém todas as seções mencionadas anteriormente. O arquivo está organizado em torno das páginas e fluxos de interação dos usuários, permitindo uma visualização clara de como cada perfil de usuário (usuário não autenticado, coordenador e administrador) irá navegar pelo sistema. Cada seção reflete as páginas correspondentes e suas funcionalidades, compondo um fluxo coeso e detalhado de todas as interações que os usuários podem realizar dentro da aplicação.

Figura 17 – Arquivo Figma



Fonte: Autoria Própria

5.1.4 Desenvolvimento Back-end

Conforme citado anteriormente, a etapa de desenvolvimento do *back-end* será realizada seguindo o padrão MVC *(Model-View-Controller)* (Subseção 5.1.2.3), em conjunto com o modelo de componentes ilustrado na Figura 15, que foi discutido na etapa de modelagem. Além disso, todas as tecnologias mencionadas (Subseção 4.2.4) serão utilizadas para garantir o funcionamento adequado do *back-end*.

5.1.4.1 Estrutura do Repositório

O repositório do *back-end* segue uma organização modular, com as seguintes pastas principais:

- **controllers/:** Contém os controladores com os métodos que gerencia as requisições e respostas da API, no qual serão acionados pelas rotas.

- **database/:** Configurações, conexão com o MongoDB e *scripts* de *seed* para o banco de dados. [1]
- **middlewares/:** Middlewares de autenticação, autorização e validação.
- **models/:** Definição dos schemas do MongoDB com base no diagrama de classes 13.
- **routes/:** Configuração das rotas da API.
- **services/:** Uso de serviços que estão fora da lógica dos controladores, como envio de e-mail.
- **src/:** Arquivos principais de inicialização do servidor.
- **tests/:** Scripts para testes unitários e de integração.
- **utils/:** Utilitários auxiliares como funções de validação e geração de tokens.

5.1.4.2 Rotas da API

A API foi estruturada em várias rotas, agrupadas por funcionalidade. Para fins de organização e documentação, o **Postman** (Postman, 2024) foi utilizado para testar e documentar as coleções de rotas, que estão divididas nas seguintes categorias:

- **Metadata Routes:** Informações dos cursos e do sistema SIAC (/api/metadata/...).
- **Auth Routes:** Autenticação e gerenciamento de tokens (/api/auth/...).
- **Admin Routes:** Gerenciamento dos perfis de administradores, coordenadores e informações dos cursos (/api/admin/...).
- **Coordinator Routes:** Gerenciamento de cursos e módulos pelos coordenadores (/api/coordinator/...).

Essas rotas são documentadas no **Swagger**, permitindo uma navegação fácil e centralizada de todas as funcionalidades da API. A ferramenta **Postman** também foi utilizada para testar as rotas e gerar coleções que facilitam o consumo dos recursos por parte do *front-end*, utilizando como base as documentações e as coleções.

5.1.5 Configuração Chatbot

Nesta etapa, são detalhadas as configurações e personalizações necessárias para o funcionamento adequado do chatbot. Para a configuração, utilizamos diversas tecnologias mencionadas na subseção 4.2.5, começando pela preparação do ambiente de desenvolvimento, em seguida por outros passos, como a definição de intents, stories e suas respectivas regras de conversação. Além disso, a configuração do armazenamento das conversas e interações é realizada com o MongoDB, permitindo a persistência dos dados.

5.1.5.1 Ambiente Virtual

Para a execução e configuração inicial do Rasa é fundamental que todas as dependências estejam corretamente instaladas, assim sendo necessário a criação utilização

[1] Scripts de seed são utilizados para preencher o banco de dados com dados utilizados para testes.

de um ambiente virtual, em que as bibliotecas cruciais estejam isoladas e com o interpretador Python com a versão suportada pelo Rasa.

Conforme a documentação do Rasa Open Source(Rasa, 2023b), o Rasa suporta atualmente as seguintes versões de Python: 3.8, 3.9 e 3.10.

Para criação do ambiente virtual rodaram-se os seguintes comandos:

```
1 python -m venv ./venv \\Criando ambiente virtual
2 .\venv\Scripts\activate \\Utilizando ambiente virtual
```

Listing 5.1 – Comando Python para criar e utilizar o ambiente virtual

Deste modo o ambiente virtual foi criado utilizando a versão 3.10.11 do Python (Python Software Foundation, 2024), isolando-se as dependências do projeto.

5.1.5.2 Iniciando Projeto Rasa

Com o ambiente virtual ativado, seguiu-se, com o auxílio do **Pip** (The pip developers, 2024), uma biblioteca para gerenciamento de pacote Python, a instalação das dependências do Rasa. Para isso, executou-se o seguinte comando:

```
1 pip3 install rasa
```

Listing 5.2 – Comando Pip Instalação Rasa

Esse comando instala as diversas bibliotecas necessárias para o ambiente Rasa, das quais podemos destacar:

- **rasa==3.6.20:** Framework principal para construir chatbots.
- **rasa-sdk==3.6.2:** Extensão para suportar ações personalizadas.
- **tensorflow==2.12.0:** Biblioteca para aprendizado de máquina.
- **spacy==3.5.0:** Biblioteca para processamento de linguagem natural.
- **pydantic==1.10.9:** Biblioteca para validação de dados.

Além dessas, foram instaladas bibliotecas adicionais para futura utilização:

- **pymongo==4.3.3:** Para conexão e manipulação do banco de dados MongoDB.
- **python-dotenv==1.0.1:** Para gerenciar variáveis de ambiente sensíveis.

5.1.5.3 Inicialização do Projeto Rasa

Para inicialização do projeto rasa foi executado o seguinte comando no ambiente virtual:

```
1 rasa init
```

Listing 5.3 – Comando Inicialização Rasa

Este comando cria a estrutura básica do projeto e treina um modelo inicial. A seguir está a saída ao rodar o comando:

```
Welcome to Rasa!
```

```
To get started quickly, an initial project will be created.
If you need some help, check out the documentation at https://rasa.com/docs/rasa
    .
Now let's start!

? Please enter a path where the project will be created [default: current
    directory] ./
Continue? Yes

Finished creating project structure.
? ? Do you want to train an initial model? Yes
Training an initial model...
INFO rasa.engine.training.hooks  - Finished training component '
    UnexpecTEDIntentPolicy'.
Your Rasa model is trained and saved at 'models/20240911-123655-cheerful-bronze.
    tar.gz'.

? Do you want to speak to the trained assistant on the command line?  Yes

Bot loaded. Type a message and press enter (use '/stop' to exit):
```

Listing 5.4 – Saída rasa init

Após a execução do comando criou-se toda a estrutura de pastas e arquivos do ambiente rasa, esse ambiente inclui pastas e arquivos como:

- **.rasa/:** Contém arquivos de cache.
- **actions/:** Inclui o arquivo Python com as ações personalizadas e arquivos de cache das ações.
- **data/:**
  - **nlu.yml:** Define as intents. Inicialmente, vem com intents básicas em inglês, como *greet* e *goodbye*.
  - **rules.yml:** Define regras e diretrizes que o agente deve seguir, como *"Say goodbye anytime the user says goodbye"*, no qual o agente chatbot deve responder sempre com a intent de "goodbye" quando o usuário disser adeus.
  - **stories.yml:** Contém as stories, sendo sequências de intents e ações.
- **models/:** Armazena os modelos treinados.
- **tests/:** Armazena as saídas de treinamento de uma ou mais stories.
- **config.yml:** Armazena informações sobre a configuração do modelo, como recipe, id do agente, linguagem e pipeline.
- **credentials.yml:** Armazena informações de integração via webhook.
- **domain.yml:** Define todas as intents cadastradas na **nlu.yml**, possíveis respostas e configurações de sessões de conversação.
- **endpoints.yml:** Armazena informações sobre canais de integração e armazenamento de conversas e modelos. [2]

[2] É importante ressaltar que o Rasa utiliza arquivos de marcação no formato YML para definir **intents**, **stories**, **regras** e **configurações**.

5.1.5.4 Alterando Configurações Padrões

Após a inicialização, algumas configurações foram ajustadas nos arquivos `config.yml` e `rules.yml` para adaptar o modelo de NLU às necessidades do projeto:

5.1.5.4.1 Arquivo de Configurações

Neste arquivo foram inseridas as seguintes informações:

```
language: pt
pipeline:
- name: WhitespaceTokenizer
- name: CountVectorsFeaturizer
- name: DIETClassifier
- name: FallbackClassifier
  threshold: 0.5
  ambiguity_threshold: 0.1
```

Listing 5.5 – Alteração config.yml

Cada atributo inserido tem grande importância no futuro treinamento do modelo, onde que no primeiro campo de **Linguagem:**, foi definido como pt (português), linguagem no qual o agente interpretará as mensagens e no campo seguinte de **Pipeline:**, utilizou-se uma configuração básica com os seguintes componentes:

- **WhitespaceTokenizer:** Configuração para aplicar a *tokenização* na entrada, assim com base nos espaços em branco é realizada a separação das palavras (conjuntos de caracteres), onde cada um se torna um *token*. Etapa fundamental no processamento de linguagem natural, como citado na subseção 2.2.
- **CountVectorsFeaturizer:** Realiza a transformação dos tokens em vetores numéricos, que posteriormente serão utilizados como entrada para os classificadores, onde poderão ser realizador cálculos vetoriais que indicam quantas vezes aquela palavra aparece no texto.
- **DIETClassifier**: O DIETClassifier (Dual Intent and Entity Transformer) é o componente principal que classifica intenções e extrai entidades. Sendo baseado em redes neurais e pode ser treinado para classificar as intenções cadastradas e reconhecer possíveis entidades nas entradas.
- **FallbackClassifier:** Acionado quando o chatbot tem baixa confiança na intenção detectada. Se a confiança estiver abaixo do **threshold** (0.5), ou se houver várias intenções com pontuação próxima, definidas pelo **ambiguity_threshold** (0.1), ele aciona o fallback.[3]

[3]Os parâmetros **threshold** e **ambiguity_threshold** são usados para determinar a confiança mínima e lidar com ambiguidades, respectivamente.

5.1.5.4.2 Arquivo de Regras

Por fim, na pasta `data`, o arquivo `rules.yml` foi configurado. Neste arquivo, as intents padrões fornecidas pelo Rasa foram traduzidas para o português e uma regra adicional foi inserida: `Pedir para reformular pergunta`. Esta regra utiliza a intent nativa `nlu_fallback` para lidar com casos em que a intenção do usuário não pode ser identificada com confiança, conforme configurado anteriormente (Parágrafo 5.1.5.4.1).

```
version: "3.1"

rules:
- rule: Diga oi sempre que o usuário disser oi
  steps:
    - intent: saudacao
    - action: action_hello_world

- rule: Diga adeus sempre que o usuário disser adeus
  steps:
    - intent: despedida
    - action: utter_despedida

- rule: Pedir para reformular pergunta
  steps:
    - intent: nlu_fallback
    - action: utter_reformular
```

Listing 5.6 – Alteração rules.yml

5.1.5.5 Definição de Intents e Criação das Actions e Stories

A definição das intents, assim como a criação das stories e actions, foi fundamentada pelo levantamento de requisitos descrito na Subseção 5.1.1. Além disso, perguntas e categorias extraídas da Tabela 4 foram utilizadas como base para a elaboração desses elementos, garantindo uma estrutura coesa e eficaz no atendimento às diferentes demandas dos usuários.

5.1.5.5.1 Intents:

As intents foram organizadas em cinco categorias principais, com base nas necessidades identificadas durante a análise das informações coletadas:

- **Padrão:** Estas são intents que são padrões e geralmente fornecidas inicialmente pelo Rasa. Elas servem como um ponto de partida ou auxílio para fluxo de comunicação. Exemplos incluem saudações e despedidas.

- **Curso:** Estas intents estão especificamente relacionadas as informações sobre os cursos oferecidos. Elas abrangem consultas sobre detalhes do curso, módulos e disciplinas. Por exemplo, um usuário pode perguntar sobre a estrutura do curso ou sobre os módulos específicos disponíveis.
- **Universidade CRCA:** Estas intents foram criadas com base nas perguntas extraídas da Tabela 4, que reúne informações sobre o CRCA. Elas incluem perguntas frequentes e informações específicas sobre a instituição, como o processo de habilitação de candidatos e orientações sobre matrículas.
- **Universidade:** Estas intents foram idealizadas para cobrir informações gerais relacionadas à universidade. Elas são de interesse para novos alunos e podem incluir informações sobre a universidade, como horários de funcionamento e serviços disponíveis.
- **Chatbot:** Estas intents são relacionadas ao funcionamento e às funcionalidades do chatbot. Elas lidam com perguntas sobre como utilizar o chatbot e suas capacidades tecnológicas.

Cada intent foi configurada no arquivo **nlu.yml** e listada no **domain.yml**[4]. Nesse processo, foram especificadas as frases que podem ativar cada intent, essas frases que ativam essas intents considerou tanto as perguntas extraídas da Tabela 4 quanto exemplos gerados com o auxílio de ferramentas de inteligência artificial (OpenAI, 2024). Isso possibilitou uma maior abrangência nas interações possíveis com o chatbot, garantindo respostas adequadas a uma variedade de consultas.

A Tabela 7 apresenta as intents criadas dentro da sua categorização correspondente. Na primeira coluna, estão listados os nomes das intents, enquanto na segunda coluna, podem ser observados exemplos de frases que compõem cada intent.

[4]O arquivo **domain.yml** centraliza todas as informações necessárias para o funcionamento do servidor Rasa, como intents, entidades (entities), slots, respostas e ações.

Tabela 7 – Intents e Exemplos de Frases

Intent	**Exemplo de Frases**
Padrão	
`saudacao`	"Oi", "Olá", "Bom dia"
`despedida`	"Tchau", "Até logo"
`agradecimento`	"Obrigado pela ajuda"
`informar`	"O curso é [Engenharia Civil](Curso)"
Curso	
`informacao_curso`	"Quais são as informações do curso de [Engenharia]?"
`modulos_curso`	"Quais são os módulos do curso de [Psicologia]?"
`disciplinas_curso`	"Quais disciplinas fazem parte do curso?"
`disciplinas_modulo`	"Quais disciplinas estão no módulo 1?"
`localizacao_curso`	"Onde fica o campus do curso de Medicina?"
`contatos_curso`	"Qual é o contato do curso de Direito?"
Universidade CRCA	
`habilitacao_candidatos`	"Quais documentos preciso para habilitação?"
`primeiro_acesso`	"Como faço para acessar o SIGAA pela primeira vez?"
`cadastro_sigaa`	"Como faço meu cadastro no SIGAA?"
`sistemas_sigaa`	"Quais sistemas posso acessar com o login do SIGAA?"
`orientacao_matricula`	"Como faço minha matrícula?"
`informacao_frequencia`	"Qual a frequência mínima para aprovação?"
`informacao_conceito`	"Quais notas correspondem aos conceitos?"
Universidade	
`informacao_ru`	"Quais são os horários do restaurante universitário?"
`calendario_academico`	"Quando começam as aulas?"
`horario_onibus`	"Qual é o horário dos ônibus?"
Chatbot	
`funcionamento_chatbot`	Como você foi criado?, "Qual o seu funcionamento?"

5.1.5.5.2 Criação das Actions

Antes de escrevermos as stories, é preciso configurar as actions. Essas actions são responsáveis por executar determinadas tarefas sempre que uma intent é identificada no fluxo de uma story. As actions podem variar de respostas simples a processos mais complexos, como fazer requisições a rotas ou buscar informações em um banco de dados. Com base nisso, as actions que irão compor as stories foram divididas em duas categorias principais, sendo a categoria Action subdividida em três subcategorias, conforme sua lógica:

- **Action:** Executada no servidor de ações do Rasa. As actions são configuradas no diretório de ações, no arquivo action.py, onde contém as funções referentes. As actions foram classificadas em três subcategorias:
  - **Media:** Ações cuja saída envolve botões, imagens ou outro tipo de conteúdo multimídia, além de textos.

  - **RequestRoute:** Ações que realizam requisições a rotas no *back-end* para buscar ou enviar dados.
  - **RequestBD:** Ações que buscam informações diretamente no banco de dados. Informações essas inseridas pelos coordenadores do curso segundo a IntentCourse cadastrada, como visto na modelagem 13.
- **Utter:** Ação executada no próprio servidor Rasa Core, em que responde com um texto simples previamente definido.

Com base nessa categorias e subcategorias, a tabela 8 a seguir foi elaborada, contendo as colunas: Nome da Ação, Descrição e Intent Associada. O Nome da Ação refere-se ao nome em que vai ser utilizado para referência-lá em uma story, a Descrição explica o que ela faz, e a Intent indica a qual intent a ação estará vinculada na story.

Tabela 8 – Ações Configuradas no Chatbot

Nome da Ação	Descrição	Intent Associada
Action		
Media		
`action_hello_world`	Envia mensagem de boas-vindas com opções de diálogo	saudacao
`action_primeiro_acesso`	Envia opções de diálogo para primeiro acesso	primeiro_acesso
RequestRoute		
`action_informacao_curso`	Busca informações sobre o curso	informacao_curso
`action_modulos_curso`	Busca os módulos do curso	modulos_curso
`action_disciplinas_curso`	Busca as disciplinas do curso	disciplinas_curso
`action_disciplinas_modulo`	Busca disciplinas de um módulo específico	disciplinas_modulo
RequestBD		
`action_localizacao_curso`	Busca localização do curso no banco de dados	localizacao_curso
`action_contatos_curso`	Busca informações de contato do curso no banco de dados	contatos_curso
Utter		
`utter_reformular`	Mensagem da FallbackClassifier [5]	nlu_fallback

[5]Como explicado anteriormente na subseção 5.1.5.4, esse classificador retornará essa mensagem caso não consiga definir a intenção com base nas condições configuradas.

Nome da Ação	Descrição	Intent Associada
utter_despedida	Envia mensagem de despedida	despedida
utter_agradecimento	Envia mensagem de agradecimento	agradecimento
utter_habilitacao_candidatos	Explica o processo de habilitação de candidatos	habilitacao_candidatos
utter_cadastro_sigaa	Explica o processo de cadastro no SIGAA	cadastro_sigaa
utter_sistemas_sigaa	Explica o uso do SIGAA em outros sistemas	sistemas_sigaa
utter_orientacao_matricula	Dá orientações sobre a matrícula	orientacao_matricula
utter_informacao_frequencia	Explica a frequência mínima exigida	informacao_frequencia
utter_informacao_conceito	Explica os conceitos de notas da universidade	informacao_conceito
utter_informacao_ru	Informa sobre o Restaurante Universitário	informacao_ru
utter_calendario_academico	Informa sobre o calendário acadêmico	calendario_academico
utter_horario_onibus	Informa sobre os horários de ônibus	horario_onibus
utter_funcionamento_chatbot	Explica o funcionamento do chatbot	funcionamento_chatbot

Em seguida foi realizada a configuração das ações, no qual ocorre em dois principais arquivos: **domain.yml** e **actions.py**. As ações do tipo utter, as quais são respostas pré-definidas enviadas pelo chatbot, são listadas somente no arquivo de domain.yml em um campo específico chamada responses:

```
responses:
  utter_reformular:
    - text: "Desculpe, não entendi. Você pode reformular a
       pergunta?"
```

Listing 5.7 – domain.yml Exemplo de Utter Listada

Além disso, as entidades e slots são definidos no mesmo arquivo. Entidades são usadas para identificar informações específicas nas mensagens dos usuários, enquanto os slots armazenam valores durante a conversa, informações essas que serão utilizadas em

ações específicas. A configuração das entidades e slots é feita da seguinte forma:

```
entities:
  - curso //Identificação da entidade curso em evidência
  - module_number //Identificação da entidade que identifica o nú
     mero do módulo em destaque

slots:
  curso: //Armazena a informação capturada referente ao curso
    type: text
    influence_conversation: false //Indica que o valor armazenado
       no slot não afeta diretamente o fluxo da conversa
    mappings:
      - type: from_entity
        entity: curso
  module_number: //Armazena a informação capturada referente ao nú
     mero de um módulo do curso
    type: text
    mappings:
      - type: from_entity
        entity: module_number
```

Listing 5.8 – domain.yml Listagem das Entidades e Slots

As ações mais complexas, que realizam lógica avançada ou requisições, são desenvolvidas em um arquivo Python no diretório actions. Essas ações são definidas em classes que herdam de Action, e suas funções principais incluem a execução da lógica necessária e a interação com o *dispatcher*[6] para enviar respostas ao usuário.

Por exemplo, a ação `ActionLocalizacaoCurso` é uma ação que busca uma informação personalizada sobre a localização de um curso em um banco de dados. A configuração desta ação no domain.yml não é diretamente visível, mas sua definição no actions.py é crucial para o funcionamento. Veja um exemplo de como essa ação é configurada:

```
class ActionInformacaoCurso(Action):
    def name(self) -> Text:
        return "action_informacao_curso"

    def run(
        self,
        dispatcher: CollectingDispatcher,
        tracker: Tracker,
        domain: Dict[Text, Any],
    ) -> List[Dict[Text, Any]]:
```

[6]O dispatcher está presente no servidor de ação do Rasa e é responsável por retornar ao servidor Rasa Core a resposta.

```
11
12          curso = get_course_from_tracker(tracker)
13          logger.info(f"Curso encontrado: {curso}")
14
15          if not curso:
16              dispatcher.utter_message(
17                  custom={
18                      "type": "informacao_curso",
19                      "title": "Nome do curso não fornecido",
20                      "content": "Por favor, me diga o nome do curso para
                            que eu possa buscar os módulos."
21                  }
22              )
23              return [FollowupAction("action_listen")]
24
25          course_path = format_course_name(curso)
26          logger.info(f"Course path formatado: {course_path}")
27
28          api_url = f"{BASE_URL_BACKEND}/api/metadata/courses/course/
                course-path/{course_path}"
29
30          try:
31              response = requests.get(api_url)
32              if response.status_code == 200:
33                  course_info = response.json()
34
35                  title_message = f"Informações sobre o curso {course_info
                        ['courseName']}:"
36
37
38                  content_message = (
39                      f"Descrição: {course_info['courseDescription']}\n"
40                      f"Objetivos: {course_info['courseGoals']}\n"
41                      f"Link PPC: [Curso PPC]({course_info['coursePPC']})\
                            n"
42                      f"Área de conhecimento: {course_info['
                            courseKnowledgeArea']}\n"
43                      f"Instituto: {course_info['courseInstitute']}\n"
44                      f"Local: {course_info['courseLocation']}"
45                  )
46
47                  dispatcher.utter_message(json_message={
48                      "type": "informacao_curso",
49                      "title": title_message,
50                      "content": content_message
51                  })
52
```

```
53              else:
54                  dispatcher.utter_message(
55                  custom={
56                      "type": "informacao_curso",
57                      "title": "Nome do curso não fornecido",
58                      "content": f"Desculpe, não encontrei informações
                           sobre o curso: [{curso}]. \nVerifique se o nome
                           está correto e tente novamente.\n Se o problema
                           persistir confira no [roadmap de cursos](/roadmap
                           /cursos) a lista com todos os cursos."
59                  }
60              )
61          except requests.exceptions.RequestException as e:
62              logger.error(f"Erro ao buscar informações do curso: {e}")
63              dispatcher.utter_message(
64                  custom={
65                      "type": "erro",
66                      "title": "Erro ao buscar o curso",
67                      "content": "Houve um problema ao buscar informações
                           do curso. Por favor, tente novamente mais tarde."
68                  }
69              )
70
71          return [SlotSet("curso", curso)]
```

Listing 5.9 – action.py Função da action_localizacao_curso

Expandindo o exemplo anterior, a ação `ActionLocalizacaoCurso` interage com o banco de dados MongoDB para recuperar informações sobre a localização de um curso, informações essas fornecidas pelos coordenadores dos respectivos cursos. A função *run* é o núcleo desta ação. Ela utiliza funções do Rasa para identificar e extrair a entidade curso do contexto da conversa. Uma vez detectada, a função executa a consulta ao banco de dados e o retornar ao usuário a resposta segundo as condições atendidas.

Todas as ações, incluindo `ActionLocalizacaoCurso`, foram definidas no arquivo `actions.py`, conforme mencionado anteriormente.

5.1.5.5.3 Desenvolvimento de Stories:

As stories no Rasa representam diálogos entre o usuário e o chatbot, ajudando a definir o fluxo de interação com base nas intenções reconhecidas. Elas são escritas no arquivo **stories.yml**, onde cada story é estruturada por passos, onde possui o reconhecimento de uma intenção do usuário (intent) seguido por uma ação (action) ou resposta pré-definida do chatbot (utter).

A partir das intents (Tabela 7) e actions (Tabela 8) definidas podemos construir uma story, um exemplo com o uso desses elementos seria a consulta de informações

do curso, utilizando as intents **informacao_curso** e **informar**, em conjunto da action **action_informacao_curso**, a story teria essa construção:

```
version: "3.1"

stories:
  - story: Informações sobre o curso
    steps:
      - intent: informacao_curso
      - action: action_informacao_curso
      - intent: informar
        entities:
          - curso
      - action: action_informacao_curso
```

Listing 5.10 – stories.yml

Essa story escreve um fluxo de diálogo específico que o chatbot deve seguir ao consultar informações sobre um curso. O fluxo é composto pelas seguintes etapas (steps):

- **Intent** `informacao_curso`: O usuário expressa a intenção de obter informações sobre um curso.
- **Action** `action_informacao_curso`: O chatbot executa a ação associada para buscar e retornar as informações sobre o curso.
- **Intent** `informar`: Caso o valor da entidade `curso` não seja informado ou capturado corretamente durante a conversa, o chatbot solicita mais informações para identificar essa entidade.
- **Action** `action_informacao_curso`: Logo após o usuarío informar o valor da entidade `curso`, a ação é realizada novamente, desse modo buscando as informações e retornando ao usuário.

Essas etapas garantem que o chatbot consiga lidar com casos em que a informação do curso não é capturada na primeira interação e continua a busca até obter a resposta adequada.

Todas as stories foram definidas para cobrir diferentes cenários de interação com o chatbot com uso das intent e actions definidas anteriormente. Abaixo está a tabela 9 completa das stories:

Tabela 9 – Stories Configuradas no Chatbot

Stories	
Story Name	**Steps**
Bem-vindo(a)	2
Primeiro acesso	6
Informações sobre o curso	4

Story Name	**Steps**
Módulos do curso	4
Disciplinas do curso	4
Disciplinas do módulo	4
Localização do curso	2
Contatos do curso	2
Despedida	2
Agradecimento	2
Habilitação de candidatos	2
Cadastro no SIGAA	2
Sistema do SIGAA	2
Orientações de matrícula	2
Informações sobre frequência	2
Conceitos e notas	2
Restaurante universitário	2
Calendário acadêmico	2
Horário de ônibus	2
Funcionamento do chatbot	2

5.1.5.6 Configuração dos Endpoints

Antes de realizarmos o treinamento do modelo com os dados preenchidos, devemos inserir alguns elementos no arquivo `endpoints.yml`, no qual será necessário para gerenciar a comunicação entre o servidor Rasa, o banco de dados MongoDB e o servidor de ações.

Abaixo, é apresentado o arquivo endpoints.yml, onde foram inseridas as informações:

```
action_endpoint:
  url: "http://localhost:5055/webhook/"
rasa:
  url: "http://localhost:5005/api"
rest:
  url: "http://localhost:5005/webhooks/rest/webhook"
tracker_store:
  type: mongod
  url: mongodb://localhost:27017
  db: ${DB_NAME}
  username:
  password:
```

Listing 5.11 – Alteração config.yml

A seguir, são explicados os principais elementos inseridos:

- `action_endpoint`: Define o endereço do servidor de ações, que está configurado para escutar as requisições no endereço `http://localhost:5055/webhook/`. Este endpoint é utilizado pelo Rasa para executar ações personalizadas através do servidor de ações.
- `rasa`: Configura o acesso à API principal do Rasa, utilizada para gerenciar as conversas e intents processadas. A URL configurada `http://localhost:5005/api` permite a comunicação direta com o servidor Rasa para manipulação e análise das conversas.
- `rest`: Define a URL do webhook para requisições REST. Esta configuração permite que o chatbot receba e envie mensagens por meio de APIs REST.
- `tracker_store`: Este campo configura o uso do MongoDB como Tracker Store, onde todo o histórico das conversas é armazenado para futura recuperação das interações anteriores e utilização desse contexto em conversas futuras.

Essas configurações garantem a correta integração entre o Rasa, o servidor de ações e o MongoDB, permitindo o funcionamento completo do chatbot.

5.1.5.7 Treinamento do Modelo

Após a conclusão das etapas de configuração das *intents*, *actions*, *stories* e *endpoints*, o próximo passo foi realizar o treinamento do modelo do chatbot com base nos dados preenchidos. O processo de treinamento é fundamental para que o Rasa consiga gerar um modelo capaz de processar e interpretar as intenções dos usuários, executar as ações corretas e manter o fluxo da conversa de acordo com as *stories* definidas.

Para realizar o treinamento, foi utilizado o comando abaixo no terminal dentro do diretório do projeto:

```
1 rasa train
```

Listing 5.12 – Treinando um modelo Rasa

Este comando inicia o processo de treinamento, onde o Rasa compila todas as *intents*, entidades e *stories* configuradas, criando um modelo pronto para ser utilizado na interação com os usuários.

Durante o treinamento do modelo no Rasa, o processo pode ser dividido em várias etapas que incluem validação, treinamento de componentes, e exibição de métricas.

5.1.5.7.1 Validação Inicial do Modelo:

O processo começa com a validação dos dados do arquivo de `domain.yml` e `stories.yml`

```
INFO     rasa.cli.train  - Started validating domain and training data...
INFO     rasa.validator  - Validating intents...
```

```
INFO     rasa.validator  - No story structure conflicts found.
```

Listing 5.13 – Saída 1 rasa train

Nesta fase, o Rasa verifica a integridade das intents, stories e utterances. Isso assegura que não há conflitos ou inconsistências entre os elementos definidos, como intenções duplicadas ou histórias mal estruturadas.

5.1.5.7.2 Treinamento dos Componentes da Pipeline:

Em seguida, o Rasa inicia o treinamento dos componentes definidos no arquivo `config.yml`, como o `CountVectorsFeaturizer` e o `DIETClassifier`. O log indica o início e término do treinamento de cada componente:

```
INFO     rasa.engine.training.hooks  - Starting to train component '
MemoizationPolicy'.
INFO     rasa.engine.training.hooks  - Starting to train component '
TEDPolicy'.
```

Listing 5.14 – Saída 2 rasa train

Como previamente explicado na subseção 5.1.5.4, o componente `CountVectorsFeaturizer` transforma o texto das intenções em uma representação vetorial. Enquanto que o `DIETClassifier` é responsável por classificar as intenções e extrair entidades. Esses componentes são essenciais para a compreensão do chatbot.

5.1.5.7.3 Métricas de Treinamento:

Durante o treinamento de cada componente, o progresso é mostrado com algumas métricas importantes:

```
Epochs: 100%|||||||||||||||||||||||||||||||||||||||||||| 300/300
[08:34<00:00,  1.72s/it, t_loss=0.953, i_acc=0.996, e_f1=0.986]
```

Listing 5.15 – Saída 3 rasa train

- `Epochs`(**Épocas**): A quantidade de épocas refere-se ao número de vezes que o algoritmo de treinamento passa por todo o conjunto de dados. No exemplo acima, o valor 300/300 significa que o modelo foi treinado por 300 iterações completas sobre os dados.
- `t_loss`(**Training Loss**): Reflete o erro do modelo durante o treinamento. Quanto menor o valor, melhor o desempenho do modelo em relação aos dados de treinamento.
- `i_acc`(**Intent Accuracy**): Mostra a precisão do modelo na classificação das intenções. No exemplo, o valor de 0.996 indica que o modelo está classificando corretamente 99.6% das intenções.

- `e_f1`(**Entity F1 Score**): Avalia a performance da extração de entidades. O valor de 0.986 indica um F1 score de 98.6%, que combina precisão e recall, refletindo um ótimo equilíbrio entre os dois.

5.1.5.7.4 Treinamento das Políticas:

Depois do classificador de intenções, o Rasa treina as políticas padrões, como a `MemoizationPolicy` e a `TEDPolicy`, que ajudam a prever as próximas ações do agente com base nas histórias fornecidas:

```
 INFO     rasa.engine.training.hooks  - Starting to train component '
MemoizationPolicy'.
 INFO     rasa.engine.training.hooks  - Starting to train component '
TEDPolicy'.
```

Listing 5.16 – Saída 4 rasa train

Essas políticas garantem que o chatbot seja capaz de seguir fluxos de conversa complexos, baseando-se nas interações anteriores e regras definidas.

5.1.5.7.5 Conclusão do Treinamento:

O treinamento termina com o modelo salvo, pronto para ser utilizado:

```
 Your Rasa model is trained and saved at 'models/20240917-050008-crunchy-
tower.tar.gz'.
```

Listing 5.17 – Saída 5 rasa train

O modelo treinado fica armazenado dentro da pasta `models`, como explicado na subseção 5.1.5.2, seguindo a estruturação de pastas do Rasa.

5.1.5.8 Rodando e Testando o Rasa

A seguir o modelo treinado anteriormente foi carregado no agente, por meio de um comando no terminal foi viável a utilização do agente sem a necessidade de uma interface de conversação avançada. Por fim o modelo treinado foi testado utilizando ferramentas do Rasa, para aferir a precisão do modelo com as intents e stories definidas.

5.1.5.8.1 Rodando o Agente

Afim de realizar testes no modelo que foi treinado, sem a necessidade de integrar com uma interface de conversação, utilizasse um comando que inicia-se o agente no modo de `shell` interativo, possibilitando a simulação de interações e verificando a respostas do agente.

```
rasa shell
```

Listing 5.18 – Rodando o agente no terminal

Além disso, para que as ações personalizadas do agente sejam realizadas, se faz necessário a inicialização do servidor de ações com o seguinte comando: `rasa run actions`, conforme dentro da pasta `actions`, onde se encontra o arquivo `actions.py` com as ações personalizadas.

```
rasa run actions
```

Listing 5.19 – Rodando Servidor de Ações

Com o servidor de ação inicializado temos a seguinte saída no terminal:

```
INFO     rasa_sdk.executor  - Registered function for 'action_contatos_curso'.
[...]
INFO     rasa_sdk.endpoint  - Starting plugins...
INFO     rasa_sdk.endpoint  - Action endpoint is up and running on http
   ://0.0.0.0:5055
```

Listing 5.20 – Saída Servidor de Ações

Onde o servidor subiu e todas as funções de ações personalizas presentes no arquivo foram registradas.

Enquanto que a saída do comando de inicialização do agente via terminal é:

```
INFO     root  - Starting Rasa server on http://0.0.0.0:5005
INFO     rasa.core.processor  - Loading model 'models\20240917-231824-cerulean-
   absorption.tar.gz...'
INFO     root  - Rasa server is up and running.
Bot loaded. Type a message and press enter (use '/stop' to exit):
Your input ->
```

Listing 5.21 – Saída Rasa Shell

Em que ocorre a inicialização do agente por meio do conector de `cmdline`, isto é via terminal. Na linha seguinte o servidor é disponibilizado e em sequência, bem como efeutase o carregamento do modelo treinado antecedentemente.[7]. Após o carregamento do modelo o servidor encontra-se rodando e o terminal aceita entradas para o agente carregado.

Com a finalidade de testar as stories descritas na Tabela 9, foram realizadas as seguintes entradas e saídas correspondentes.

```
Oi! Aqui é a MICA do SIAC, pronta para te ajudar. O que você gostaria de saber?
   (Use arrow keys)
    1: Informações sobre o seu curso. (/action_informacao_curso)
    2: Seu primeiro acesso. (/action_primeiro_acesso)
   Type out your own message...
Your input ->  'onde fica o ru'
Custom json:
{
```

[7]No Rasa, quando o argumento de qual modelo carregar não é explicitamente definido, automaticamente é seleciona e carregado o último modelo gerado e salvo na pasta `models`. Isso ocorre porque o Rasa mantém os modelos em ordem cronológica de criação, e o mais recente é priorizado para carregamento.

```
  "type": "informacao_ru",
  "title": "Aqui está o link para informações sobre o RU:",
  "content": "[Informações sobre o RU](https://ru.unifesspa.edu.br/)\n"
}
Your input ->  'como posso utilizar o login do sigaa'
Custom json:
{
  "type": "sistemas_sigaa",
  "title": "O login do SIGAA também é utilizado para acessar outros sistemas da
    universidade, como:",
  "content": "- Wi-Fi dos campi da universidade com o domínio @unifesspa.edu.br\
    n- E-mail institucional: Exemplo: email.unifesspa.edu.br\n- [SIGEventos](
    https://sigeventos.unifesspa.edu.br): Sistema Integrado de Gestão de Eventos
    \n\nPara mais informações, consulte os seguintes links:\n1. [Primeiros
    Passos](https://crca.unifesspa.edu.br/index.php?option=com_content&view=
    article&id=185:primeiros-passos&catid=2)\n2. [Primeiro Acesso Discente](
    https://helpdesk.unifesspa.edu.br/wiki-unifesspa/primeiro-acesso-discente/)\
    n"
}
Your input ->  'dljnundjw'
Me desculpe, acho que não entendi direito. Que tal tentar novamente?
```

Listing 5.22 – Saída 2 Rasa Shell

Observando as entradas e saídas respectivas, é possível constatar que o agente identificou corretamente as intenções de acordo com as stories configuradas. A partir da intenção identificada, o chatbot executou a ação correspondente. No entanto, é importante destacar que as ações personalizadas mais complexas, como `action_informacao_curso` e `action_localizacao_curso`, não foram executadas, pois requerem integração com o *back-end*, conforme a arquitetura que será apresentada posteriormente.

5.1.5.8.2 Testando o Modelo

Para uma verificação final automatizada, foram realizados testes utilizando as stories e intents configuradas. A validação de dados assegura que não haja erros ou inconsistências significativas no domínio, nos dados de NLU ou nas stories. Para validar os dados, o seguinte comando foi executado:

```
1 rasa test -d .\domain.yml -s .\data\stories.yml
```

Listing 5.23 – Comando rasa test

Esse comando testa o modelo mais recente em todas as stories definidas no arquivo stories.yml e nas intents cadastradas no arquivo domain.yml[8].

A execução do comando resulta em saídas como a seguinte:

```
INFO     rasa.nlu.test  - Evaluation for entity extractor: 'DIETClassifier'
INFO     rasa.nlu.test  - Classification report saved to 'results\
   DIETClassifier_report.json.'
```

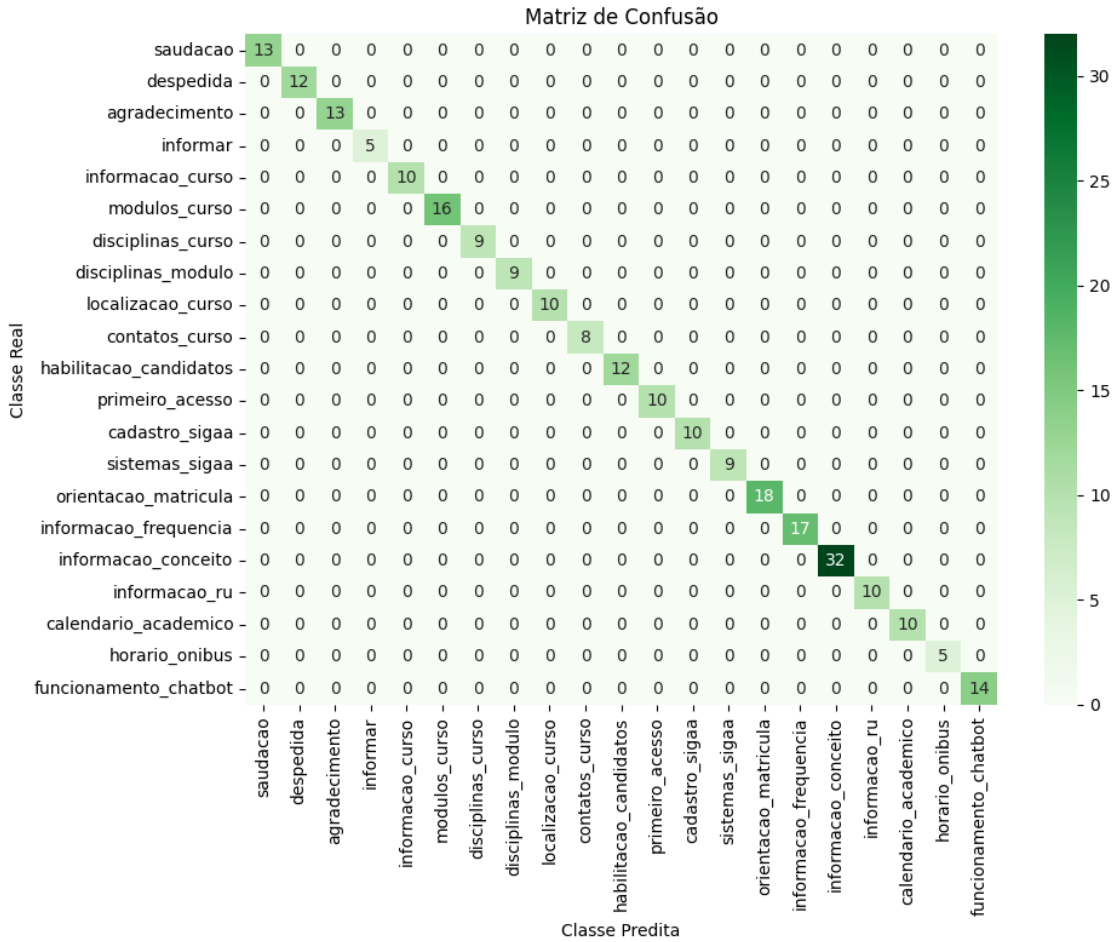
[8] É recomendado que, ao executar o comando, sejam passados como argumento os arquivos domain.yml e stories.yml.

```
INFO     rasa.nlu.test  - Evaluation for entity extractor: DIETClassifier
INFO     rasa.nlu.test  - Classification report saved to 'results\
   DIETClassifier_report.json.'
INFO     rasa.nlu.test  - Every entity was predicted correctly by the model.
INFO     rasa.utils.plotting  - Confusion matrix, without normalization:
'[[  68    0    0]
 [   0    2    0]
 [   0    0 1684]]'
```

Listing 5.24 – Saída rasa test

O relatório gerado pelos testes inclui arquivos relevantes, como a figura 18, que apresenta a matriz de confusão das intents.

Figura 18 – Matriz de Confusão - Intents

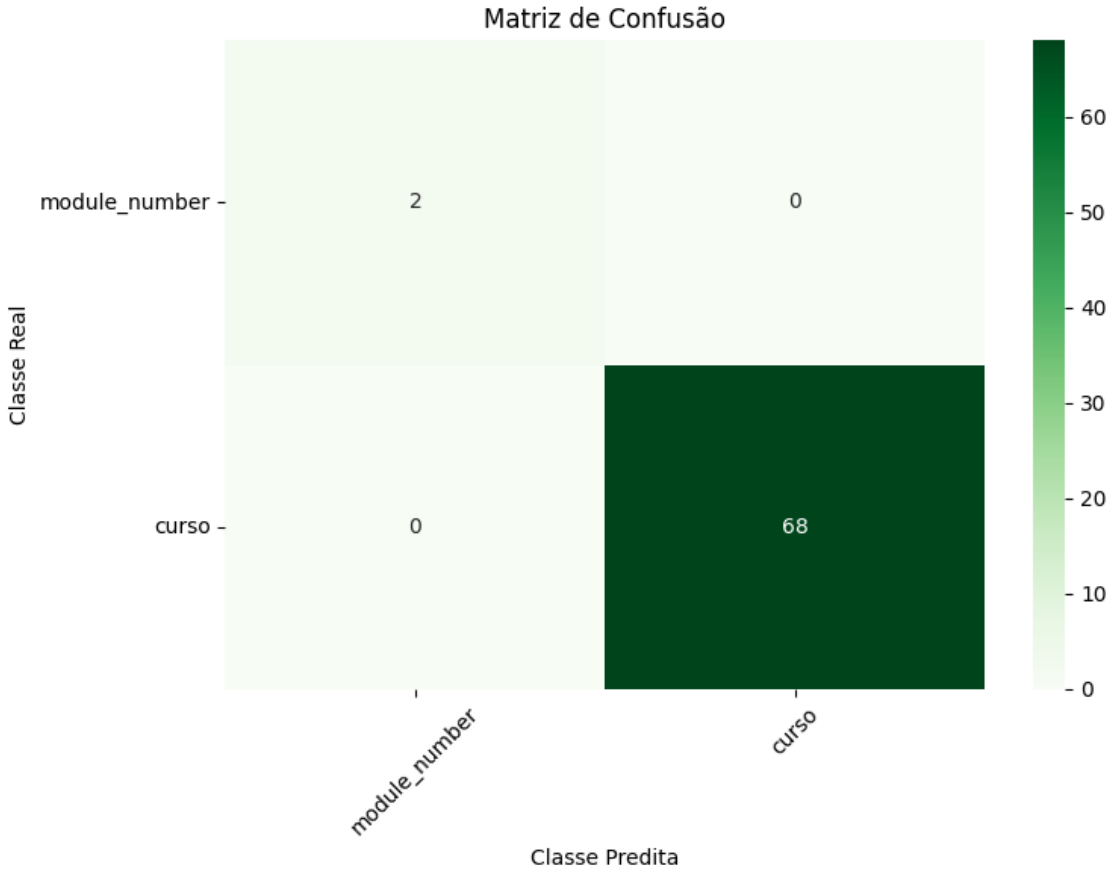


Fonte: Autoria Própria

Analisando a matriz, é possível visualizar que todas as intents foram classificadas corretamente pelo modelo. A matriz de confusão permite identificar quais intents o modelo teve maior dificuldade de classificar, indicando áreas para possíveis melhorias no treinamento.

Além disso, foram gerados outras matrizes do mesmo tipo, como a figura 19, onde demonstra a classificação de entidades para o componente `DIETClassifier`.

Figura 19 – Matriz de Confusão - Entidades



Fonte: Autoria Própria

Na matriz de confusão de entidades gerada (Figura 19), é possível observar que o classificador conseguiu capturar corretamente as entidades com base nos exemplos fornecidos, sem confusões significativas entre as duas classes de entidades.

Verifica-se, portanto, que o agente é capaz de processar e identificar corretamente todas as intenções cadastradas no fluxo das stories, bem como captar as entidades necessárias para a execução das ações personalizadas.

5.1.6 Containerização com Docker

Antes de realizarmos a integração do chatbot com o *back-end*, foi realizada a containerização tanto do chatbot quanto do banco de dados. Nos parágrafos a seguir, serão explicados o processo de containerização e suas vantagens.

5.1.6.0.1 Containers

Containers são uma tecnologia que permite empacotar uma aplicação juntamente com todas as suas dependências em um ambiente isolado e padronizado. Isso garante que a aplicação possa ser executada de maneira consistente em qualquer lugar, seja em um

servidor local ou em uma infraestrutura na nuvem. Diferentemente das máquinas virtuais, que incluem um sistema operacional completo, os containers compartilham o kernel do sistema operacional host, tornando-os mais leves e eficientes.

Para realizar a containerização do projeto, utilizamos o Docker (Docker Inc., 2024), uma tecnologia moderna amplamente adotada que simplifica a criação, distribuição e execução de containers. O Docker permite a criação de imagens de containers que podem ser executadas de forma consistente em diferentes ambientes, garantindo que a aplicação funcione de maneira previsível, independentemente da infraestrutura.

5.1.6.0.2 Estrutura de Containers para o Chatbot

Para o chatbot MICA, foram configurados três containers distintos para assegurar o funcionamento adequado da aplicação:

1. Container do Servidor Rasa Este container é responsável por executar o servidor principal do Rasa, que realiza o processamento das intenções dos usuários e gerencia o fluxo das conversas. O `Dockerfile` utilizado para a criação deste container é baseado na imagem `python:3.10.0-slim`, oferecendo um ambiente leve e otimizado para execução do chatbot.

Todos os arquivos do chatbot são copiados para o container, e as dependências necessárias são instaladas por meio do arquivo `requirements.txt`, que lista as bibliotecas e pacotes que o Rasa utiliza para operar corretamente.

Por fim, o servidor Rasa é configurado para expor sua API e habilitar o CORS[9], garantindo a comunicação com outras aplicações de forma segura e permitindo integrações flexíveis.

O arquivo Dockerfile correspondente está descrito abaixo:

```
1  FROM python:3.10.0-slim
2
3  COPY . /app
4
5  WORKDIR /app
6
7  RUN mkdir -p /.config /app/models && \
8      chown -R 1001:0 /app /.config && \
9      chmod -R 775 /app /.config
10
11 RUN pip install --no-cache-dir -r requirements.txt
12
```

[9]O comando `rasa run -model models/ -enable-api -cors "*--debug` é responsável por carregar o último modelo treinado, habilitar a API para integrações e permitir solicitações de qualquer domínio.

```
13 COPY clearCache.sh /app/
14 RUN chmod +x /app/clearCache.sh
15
16 USER 1001
17
18 CMD ["rasa", "run", "--enable-api", "--cors", "*"]
```

Listing 5.25 – Dockerfile do Servidor Rasa

2. Container do Servidor de Ação Este container é responsável por executar as ações personalizadas do chatbot. Para isso, a pasta `actions` é copiada, onde o arquivo `actions.py` contém as funções que definem essas ações personalizadas. Além disso, o arquivo `endpoints.yml` também é copiado, o qual é utilizado para indicar o endereço do container Rasa configurado anteriormente, permitindo que o servidor de ações envie corretamente as respostas para o chatbot principal.

O arquivo `requirements-actions.txt` localizado na pasta `actions` é igualmente copiado, contendo todas as bibliotecas necessárias para que o servidor de ações funcione corretamente.

A imagem base utilizada é a `rasa/rasa-sdk:latest`, que já inclui o SDK necessário para rodar as ações personalizadas do Rasa.

Por fim, o servidor de ações é inicializado após a instalação das bibliotecas contidas no arquivo `requirements-actions.txt`, e a execução do comando `start -actions actions` para rodar o servidor de ações dentro da pasta `actions`.

Abaixo está o Dockerfile utilizado para o servidor de ações:

```
19 FROM rasa/rasa-sdk:latest
20
21 WORKDIR /app
22 COPY actions /app/actions
23 COPY endpoints.yml /app/endpoints.yml
24 COPY actions/requirements-actions.txt /app/requirements-actions.
      txt
25
26 USER root
27 RUN python3 -m pip install --upgrade pip
28 RUN pip install -r /app/requirements-actions.txt
29
30 USER 1001
31 CMD ["start","--actions", "actions"]
```

Listing 5.26 – Dockerfile do Servidor de Ação

3. Container do MongoDB Este container utiliza a imagem `mongo:latest`, e como explicado previamente é responsável por armazenar os dados de conversação do chatbot e informações dos cursos.

5.1.6.0.3 Configuração do `docker-compose.yml`

A execução dos três containers é orquestrada pelo arquivo `docker-compose.yml`, que facilita a criação e gerenciamento de múltiplos containers. Abaixo está a configuração:

```
services:
  rasa:
    build:
      context: .
      dockerfile: ./Dockerfile
    volumes:
      - .:/app
    ports:
      - "5005:5005"
    environment:
      - MONGO_URI=mongodb://mongo:27017
      - DB_NAME=${DB_NAME}
    command: >
      sh -c "rasa train &&
            rasa run --model models/ --enable-api --cors '*' --
               debug"
  action-server:
    build:
      context: .
      dockerfile: ./actions/Dockerfile
    volumes:
      - ./actions:/app/actions
      - ./endpoints.yml:/app/endpoints.yml
    ports:
      - "5055:5055"
    environment:
      - MONGO_URI=mongodb://mongo:27017
      - DB_NAME=UNIFESSPA-SIAC-BACKEND
      - BASE_URL_BACKEND=localhost:3000
    command: >
      start --actions actions
  mongo:
    image: mongo:latest
    ports:
```

```
65          - "27017:27017"
```

Listing 5.27 – Configuração docker-compose.yml

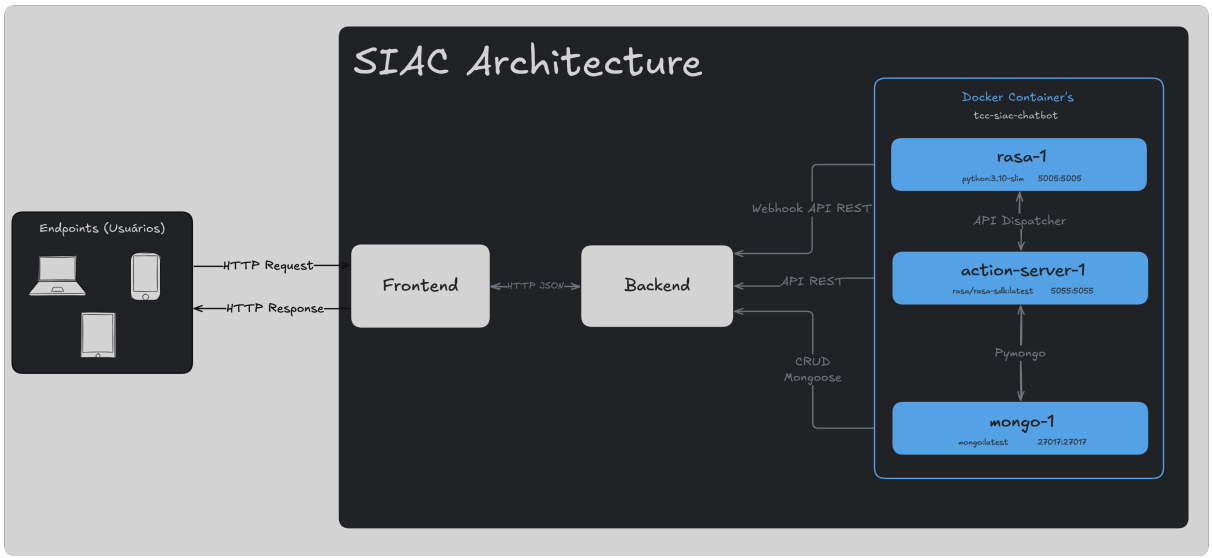
5.1.6.0.4 Vantagens da Conteinerização no Chatbot

O processo realizado anteriormente oferece diversas vantagens:

- **Portabilidade:** Os contêineres Docker podem ser executados em qualquer ambiente que suporte Docker, garantindo que o chatbot funcione de maneira idêntica em diferentes ambientes de desenvolvimento, teste e produção.
- **Isolamento:** Cada contêiner tem seu próprio conjunto de dependências e bibliotecas, evitando conflito de versão entre diferentes serviços.
- **Escalabilidade:** Com Docker, é fácil escalar o chatbot horizontalmente, criando múltiplas instâncias de contêineres para atender a mais usuários.
- **Facilidade de Deploy:** Docker permite que o chatbot seja facilmente empacotado e distribuído, simplificando o processo de implantação em qualquer ambiente.

Em suma, a arquitetura de contêineres, juntamente com a futura integração com *back-end* e *front-end*, será conforme ilustrada na Figura 20:

Figura 20 – Arquitetura de Contêineres e Integração com Back-end e Front-end



Fonte: Autoria Própria

Na figura, é possível visualizar o grupo de contêineres e os protocolos pelos quais cada componente irá se comunicar. Esta estrutura facilita a escalabilidade e a manutenção, garantindo a modularidade do sistema, aonde cada parte pode ser desenvolvida e atualizada de forma independente, mas integrada por meio de APIs e serviços definidos.

5.1.7 Integração do Back-end com o Chatbot

A integração do *back-end* com os contêineres do Chatbot foi realizada por meio da criação de novas rotas na *API REST*, desenvolvida anteriormente (Subseção 5.1.4.2). Foram adicionadas novas rotas que se dividem em três categorias:

5.1.7.0.1 Rotas Abertas:

- `api/chatbot/status`: Responsável por retornar o estado dos serviços do chatbot, indicando se estão em funcionamento ou não.
- `api/chatbot/statistics`: Retorna dados referentes ao chatbot, como o total de sessões, total de mensagens, uma lista com todas as intenções identificadas e o número de ativações, principais horários de acesso, além de uma média das avaliações de conversação.
- `api/chatbot/rate-session`: Permite avaliar uma sessão de conversação com uma nota de 1 a 5, juntamente com um comentário opcional.
- `api/chatbot/webhook/action/:action`: Envia ações personalizadas diretamente para o servidor de ações do chatbot.
- `api/chatbot/webhook/message`: Responsável por enviar mensagens diretamente para o servidor Rasa.

5.1.7.0.2 Rotas de Administração:

- `api/admin/chatbot/intents`: Permite ao administrador inserir as intenções (*intents*) que serão alimentadas pelos coordenadores.[10]
- `api/admin/chatbot/intents/id/:id`: Permite ao administrador visualizar, modificar ou excluir uma intenção inserida anteriormente.

5.1.7.0.3 Rotas dos Coordenadores:

- `api/coordinator/chatbot/intents-course/id/:id/responses`: Permite aos coordenadores inserir respostas personalizadas para as intenções vinculadas ao seu curso.
- `api/coordinator/chatbot/intents-course`: Retorna as intenções disponíveis do seu curso, para os coordenadores poderem gerenciar as respostas.
- `api/coordinator/chatbot/intents-course/id/:id/responses/:index`: Permite aos coordenadores modificar ou excluir respostas personalizadas cadastradas para seu curso.

[10]Ressalta-se que as intents que serão cadastradas no banco de dados para os cursos, estão previamente cadastradas no servidor rasa, em conjunto com as ações do servidor de ação.

A integração do *back-end* com os contêineres do chatbot ocorre por meio de requisições com a biblioteca **Axios** (GitHub, Acesso em 1 de agosto de 2023), é enviada para a sua *API REST*, especificamente o *webhook* do Rasa ou o servidor de ações, conforme visualizado na Figura 20. Este modelo de integração permite que o *front-end* manipule as informações no banco de dados e envie/receba dados para os servidores do chatbot por meio do *back-end*, que serve como intermediário para esses recursos.

5.1.8 Desenvolvimento Front-end e Integração com o Back-end

Nesta subseção, abordaremos o processo de desenvolvimento do *front-end* e sua integração com o *back-end*, crucial para garantir uma experiência de usuário fluida e eficaz. O desenvolvimento foi estruturado em várias etapas, cada uma focada em aspectos específicos do projeto.

5.1.8.1 Estrutura do Repositório

A organização do repositório foi projetada para facilitar a manutenção e a escalabilidade do projeto. Abaixo está uma descrição das principais pastas e arquivos:

- **data/:** Contém arquivos de dados estáticos ou modelos utilizados para alimentar as páginas durante o desenvolvimento.
- **public/:** Diretório que armazena arquivos públicos, como imagens e fontes, acessíveis diretamente na aplicação.
- **src/:** Diretório principal onde o código-fonte está localizado.
  - **components/:** Pasta que contém os componentes reutilizáveis da aplicação, como cabeçalhos, rodapés e botões.
  - **pages/:** Contém os arquivos de páginas individuais que utilizam os componentes para formar as diferentes views da aplicação.
  - **types/:** Armazena definições de tipos TypeScript, úteis para manter a tipagem estática e a integridade do código.
  - **utils/:** Contém funções auxiliares (utilities) reutilizadas em diversas partes do código, como formatação de dados ou manipulação de strings.
  - **types/:** Armazena definições de tipos TypeScript, úteis para manter a tipagem estática e a integridade do código.
  - **middleware.ts** Arquivo responsável por validar sessão em rotas protegidas.
- **astro.config.mjs:** Arquivo de configuração do Astro, onde estão definidas rotas, *plugins* e outras opções de build.
- **tailwind.config.mjs:** Arquivo de configuração do Tailwind, utilizado para customização.

5.1.8.2 Desenvolvimento com Astro

O desenvolvimento do *front-end* foi realizado utilizando uma combinação de tecnologias modernas, detalhadas na subseção 4.2.3, onde destacamos o uso de Astro para a construção das páginas e Tailwind CSS para estilização, proporcionando uma abordagem ágil e eficiente no desenvolvimento da interface, visando garantir uma interface de usuário responsiva, escalável e com alto desempenho. As telas foram construídas com base em protótipos previamente desenvolvidos na subseção 5.1.3.3, mantendo a consistência entre o design proposto e a implementação final. Enquanto a integração com o *back-end* é realizada por meio de chamadas API, garantindo a comunicação eficaz entre o cliente e o servidor.

5.1.8.2.1 Componentização

A *"componentização"* foi uma das principais vantagens oferecidas pelo uso do Astro. Componentes reutilizáveis, como cabeçalhos, rodapés, menus de navegação e *"cards"* de navegação, foram implementados de forma independente e integrados às páginas conforme necessário. Essa abordagem modular permitiu que o desenvolvimento fosse mais ágil e organizado, além de facilitar futuras atualizações no design ou nas funcionalidades, visto que cada componente pode ser modificado isoladamente.

Essa abordagem pode ser observada na Figura 21, que mostra a página inicial conforme o fluxo de páginas 16 proposto na etapa de prototipação.

Figura 21 – Página Inicial SIAC

Fonte: Autoria Própria

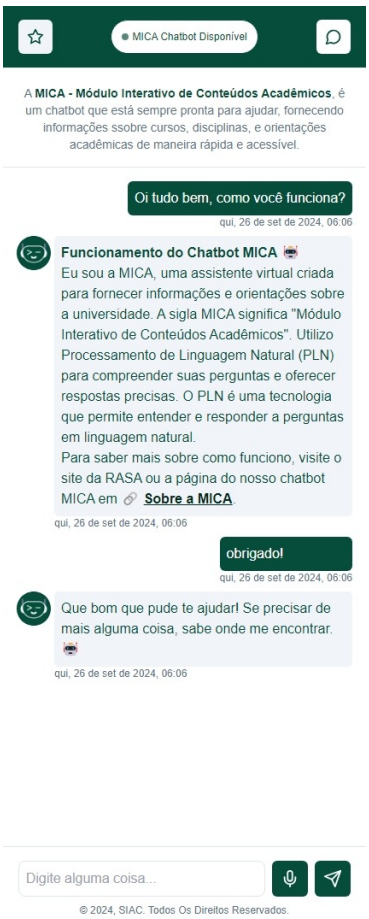
A construção da página foi inteiramente realizada utilizando componentes, como a barra de navegação, botões, *cards* de navegação e o rodapé. Esses componentes, que em alguns casos possuem variações, são reutilizados em todas as páginas.

5.1.8.2.2 Integração com o Back-end

A comunicação entre o *front-end* e o *back-end* foi feita utilizando a *API REST*, conforme descrito na subseção 5.1.4.2. As páginas e componentes do Astro interagem com as rotas da API para obter informações sobre os cursos, enviar mensagens para o chatbot, avaliar sessões de conversação e acessar outros dados do sistema.

Por exemplo, na página de interação com o chatbot, como visto na Figura 22, a mensagem do usuário é enviada por meio de uma requisição POST para a rota `api/chatbot/webhook/message`. A resposta do chatbot é recebida e exibida dinamicamente no componente de conversação. Da mesma forma, outras interações, como a avaliação de sessões (`api/chatbot/rate-session`) e a visualização de estatísticas (`api/chatbot/status`), são consumidas para que o *front-end* tenha acesso a esses dados.

Figura 22 – Página de Chatbot MICA



Fonte: Autoria Própria

5.1.8.2.3 Apresentação de Páginas Desenvolvidas

A estrutura das páginas foi planejada com base no sitemap do sistema, apresentado na Figura 16. O desenvolvimento das páginas seguiu o fluxo definido e o *design* proposto

na etapa de prototipação 5.1.3, garantindo que a navegação fosse intuitiva e funcional. Entre as páginas desenvolvidas estão:

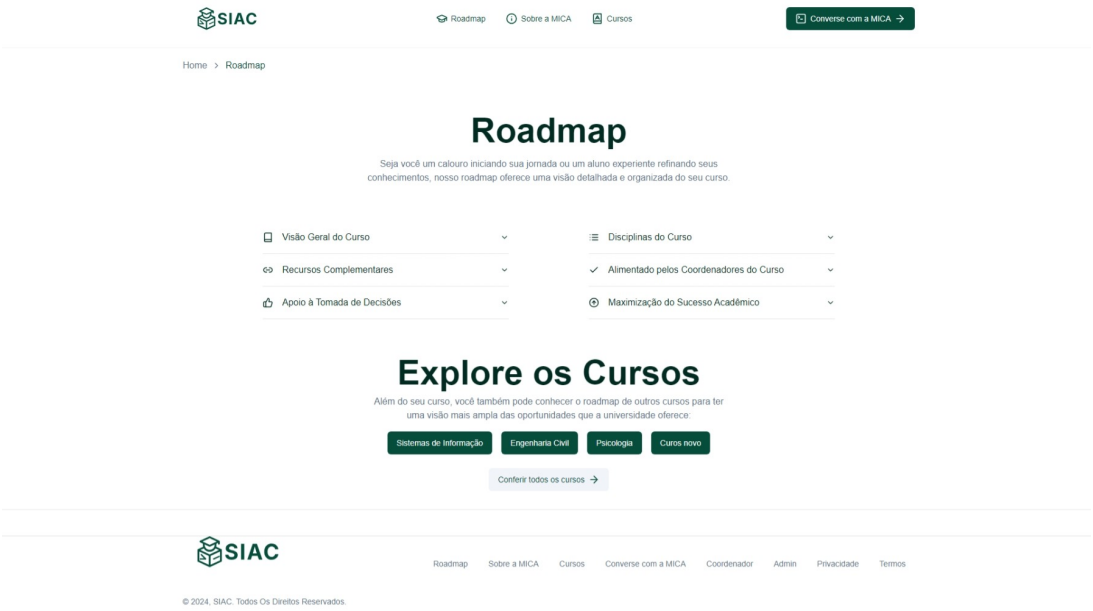
- **Página Inicial:** A página inicial do SIAC (Figura 23) contém diversos componentes importantes, como a barra de navegação, que inclui um menu de acesso às principais funcionalidades e um botão CTA (*Call to Action*) que direciona à MICA. Logo abaixo, há um título, acompanhado de uma descrição e uma imagem que representa um fluxo de conversação com a MICA. Em seguida, encontram-se *cards* que descrevem as funcionalidades do sistema. A página finaliza com um rodapé que contém *links* para outras páginas do sistema, além dos termos de uso e privacidade.

Figura 23 – Página Inicial SIAC

Fonte: Autoria Própria

- **Roadmap Bem-vindo(a):** Essa página "Roadmap Bem-vindo(a)" (Figura 24) apresenta a mesma barra de navegação, seguida de um *breadcrumb*, componente que indica a página atual, sendo um fator importante para a usabilidade, ao informar o estado do sistema. Logo após, há um título acompanhado de um texto descritivo da página, com um *accordion* que detalha as diversas vantagens da utilização do roadmap. Ao final, há uma chamada para explorar os cursos do roadmap, com uma descrição e *links* para alguns cursos, além de um *link* para visualizar a lista completa de cursos. O rodapé permanece o mesmo.

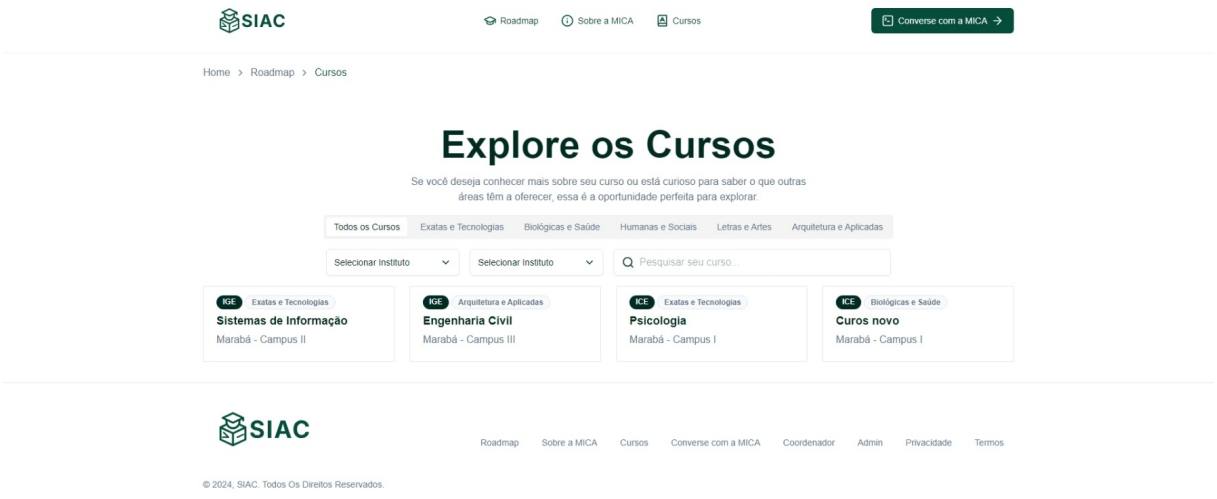
Figura 24 – Página Roadmap Bem-vindo(a)



Fonte: Autoria Própria

- **Roadmap Lista de Cursos:** A página (Figura 25) contém a barra de navegação, o *breadcrumb*, um título chamativo e uma descrição sobre a página. Logo em seguida, há um menu de filtragem, onde é possível filtrar a lista de cursos por área de conhecimento, instituto, localidade, e também uma barra de pesquisa. Após o filtro, são exibidos os cursos cadastrados em formato de *cards*, com *tags* que fornecem informações detalhadas sobre cada curso. Por fim, o rodapé permanece o mesmo, com *links* para outras páginas e para os termos de uso e privacidade.

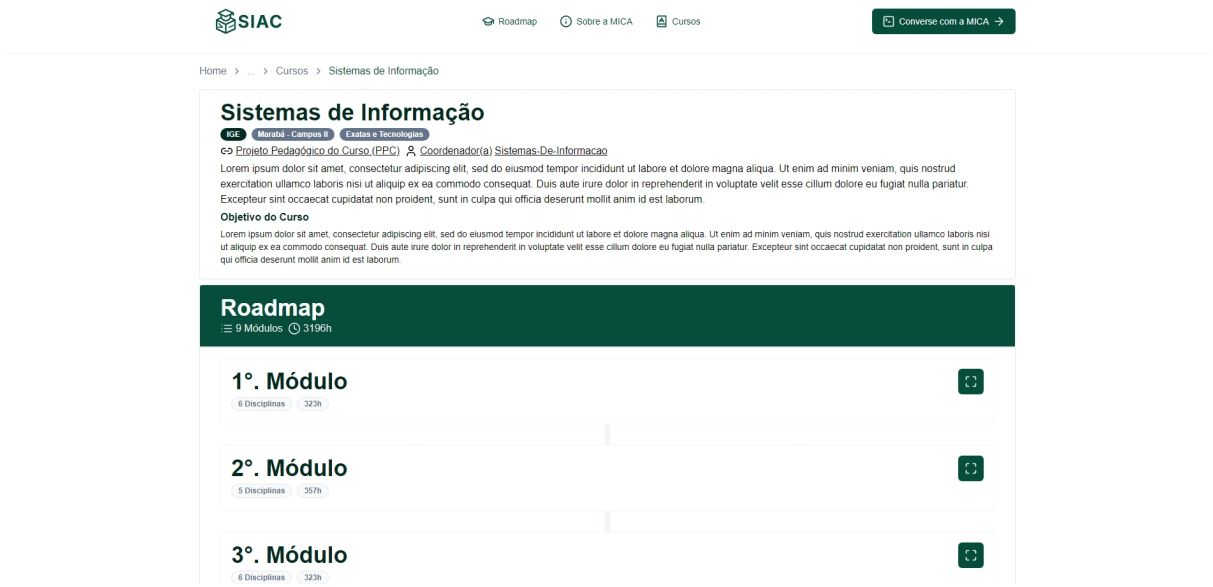
Figura 25 – Página Roadmap Lista de Cursos



Fonte: Autoria Própria

- **Roadmap [Curso]:** Esta página (Figura 26) é gerada dinamicamente com informações sobre o curso selecionado. Inicialmente, é exibido um cabeçalho com dados do curso, como nome, *tags* de instituto, localidade e área de conhecimento, acompanhado de um *link* para o PPC do curso e o nome do coordenador(a) vinculado, que ao ser clicado abre diretamente para o e-mail do coordenador. Logo após, há uma descrição e um texto sobre o objetivo do curso. A partir do componente Roadmap, são mostradas informações como a quantidade de módulos e o total da carga horária. Cada módulo apresenta seu nome, número de disciplinas e carga horária, com um botão que, ao ser clicado, expande o módulo e exibe a listagem de todas as disciplinas. Cada disciplina tem um botão que revela seus detalhes e mostra tópicos de apoio vinculados, como artigos, vídeos ou *websites*.

Figura 26 – Página do Roadmap [Curso]



Fonte: Autoria Própria

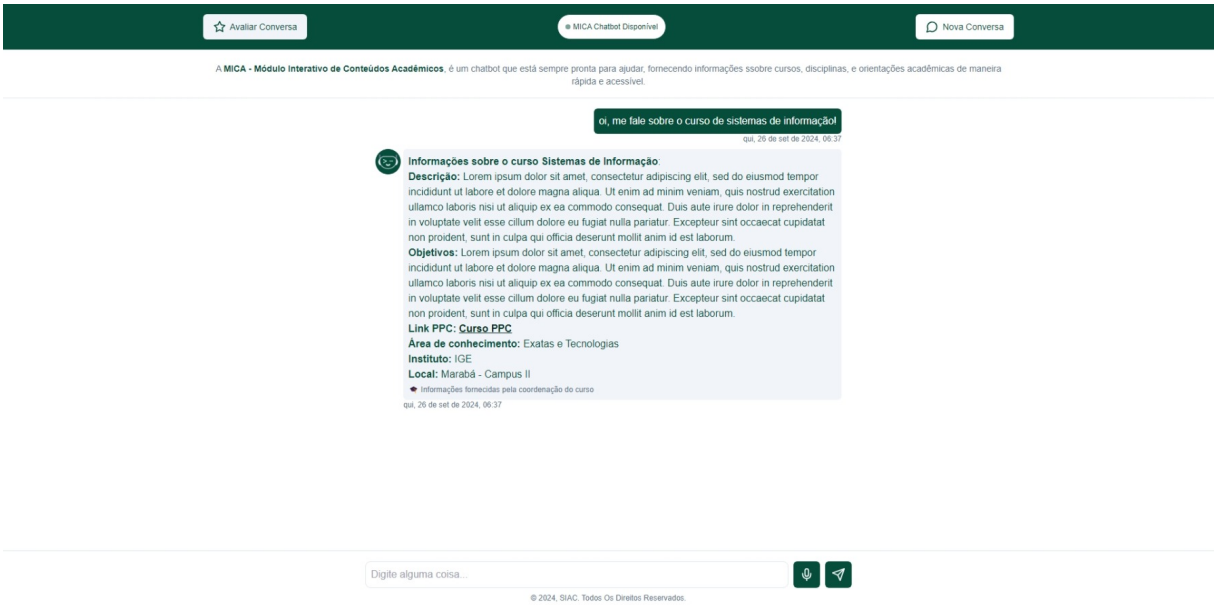
- **Chatbot Sobre:** Refere-se a uma página (Figura 27) explicativa, segue o formato de uma *landing page*, cujo objetivo é direcionar o usuário para a interação com a MICA. A página começa com um título que explica a sigla MICA, acompanhado de um texto descritivo e uma imagem. Essa figura ilustra um esquema de baixo nível do funcionamento do chatbot. Ao longo da página, são apresentadas, em formato de lista, as principais funcionalidades da MICA, seguidas de uma exemplificação do fluxo de conversação. Em seguida, são detalhadas todas as tecnologias que compõem tanto o SIAC quanto a MICA. Para finalizar, há uma chamada para que o usuário interaja com a MICA, incentivando-o a avaliar a conversa com o intuito de aprimorar o chatbot.

Figura 27 – Página Sobre Chatbot

Fonte: Autoria Própria

- **Chatbot Chat:** Esta página (Figura 28) exibe a interface de conversação com o chatbot. A barra de navegação contém os seguintes componentes: um botão de avaliação, que abre uma interface para avaliação da conversa (de 1 a 5 estrelas) juntamente com um campo opcional para comentários; um indicador central que mostra o *status* de disponibilidade da MICA; e, por fim, um botão "Nova conversa", que recarrega a página para iniciar uma nova interação. No centro da página, a conversa é renderizada. Na parte superior, há uma breve descrição sobre a MICA, destacando seu nome, que, ao ser clicado, direciona para a página explicativa mencionada anteriormente. As mensagens são exibidas no corpo central: as mensagens da MICA aparecem à esquerda, enquanto as do usuário ficam à direita. Na parte inferior da página, encontra-se o campo de entrada de texto, acompanhado por dois botões: um para entrada via microfone, que, ao ser clicado, permite ao navegador captar a fala do usuário e convertê-la em texto, e outro para enviar a mensagem. Por fim, há um texto referente aos direitos de uso do SIAC.

Figura 28 – Página Chatbot Chat



Fonte: Autoria Própria

- **Coordenador Bem-vindo(a):** A página de boas-vindas ao coordenador (Figura 29) contém um título de boas-vindas, acompanhado de um texto descritivo e um botão que direciona para a página de autenticação. Ao lado, há uma imagem ilustrativa.

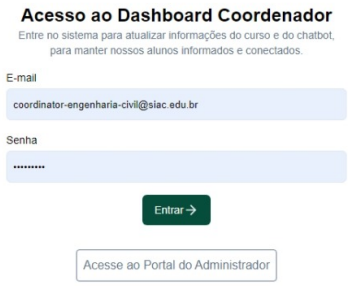
Figura 29 – Página de Bem-vindo(a) Coordenador(a)

Fonte: Autoria Própria

- **Coordenador Página de Login:** Página de autenticação (Figura 30) destinada ao coordenador. Ela contém um título que descreve o propósito da autenticação, seguido de uma breve explicação das funcionalidades disponíveis após o *login*. Abaixo, há

os campos para entrada de e-mail e senha, além de um botão para validar as credenciais. Logo abaixo, há um botão que permite alternar entre a autenticação de coordenador(a) e administrador.

Figura 30 – Página de Login Coordenador(a)]



Fonte: Autoria Própria

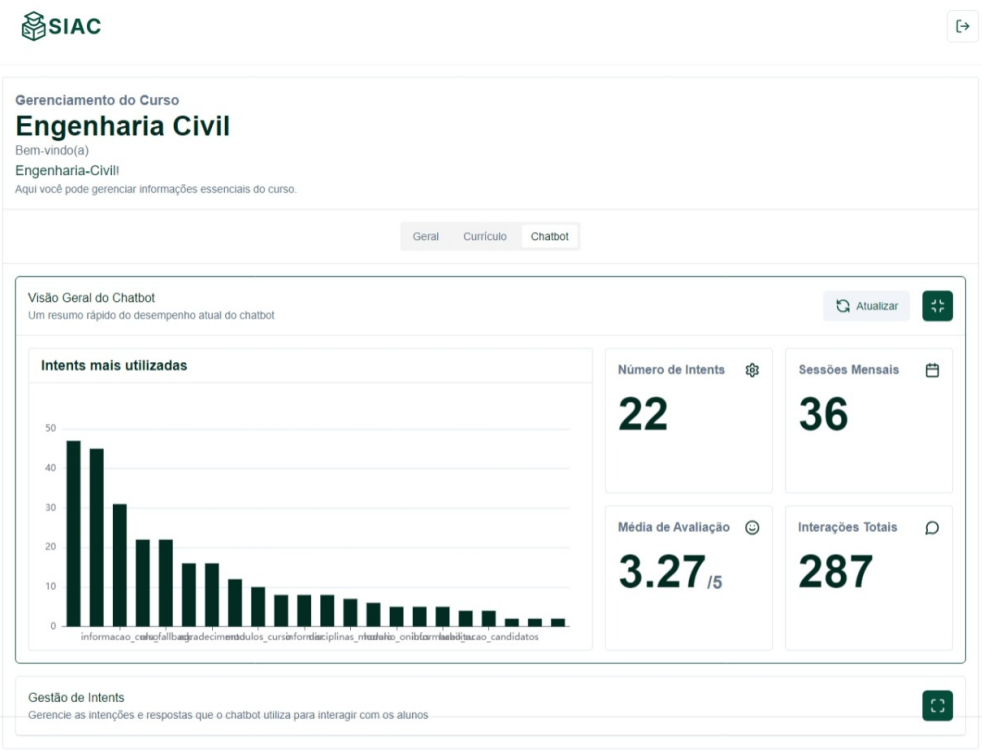
- **Coordenador Alterar Senha:** Página destinada à mudança de senha no primeiro acesso (Figura 31). Ela contém um título descritivo, uma descrição indicativa e os campos para a entrada da nova senha, além de um botão para validação.

Figura 31 – Página de Alterar Senha Coordenador(a)

Fonte: Autoria Própria

- **Coordenador Dashboard:** O *dashboard* principal do coordenador (Figura 32) apresenta as funcionalidades de gerenciamento, organizadas por páginas. O *dashboard* possui um cabeçalho fixo na parte superior, que indica o nome do curso e saúda o coordenador(a). Logo abaixo, há uma área de conteúdo, onde as informações são separadas em abas.
  A primeira aba, denominada "Geral", apresenta informações gerais sobre o curso, como nome, área de conhecimento, instituto, localidade, descrição, *link* do PPC e objetivos, permitindo que o coordenador(a) altere apenas as três últimas informações citadas.
  A aba do meio, nomeada "Currículo", refere-se ao gerenciamento dos módulos, disciplinas e tópicos do curso.
  A última aba é dedicada ao chatbot, contendo duas sub-abas expansíveis. A primeira, intitulada "Visão Geral do Chatbot", apresenta alguns *insights* sobre o chatbot, como um gráfico de barras com dados referentes à identificação das intenções no fluxo de conversação. Ao lado, há *cards* quantitativos: o primeiro exibe a quantidade de intenções cadastradas no chatbot, seguido pelo total de sessões mensais. Abaixo, o primeiro *card* é referente à média de avaliações das conversas e, por fim, um *card* mostra o total de mensagens enviadas pelos usuários. A segunda aba expansível refere-se ao gerenciamento das respostas personalizadas, onde o coordenador pode cadastrar respostas para as intenções previamente cadastradas pelo administrador para o seu curso, permitindo inserir, modificar e excluir respostas para essas intenções.

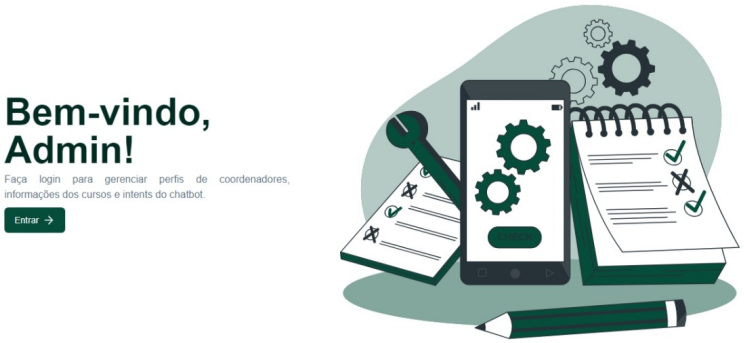
Figura 32 – Página de Dashboard Coordenador(a)



Fonte: Autoria Própria

- **Página Administrador Bem-vindo:** A página de boas-vindas ao administrador (Figura 33) contém um título de boas-vindas, acompanhado de um texto descritivo e um botão que direciona para a página de autenticação. Ao lado, há uma imagem ilustrativa.

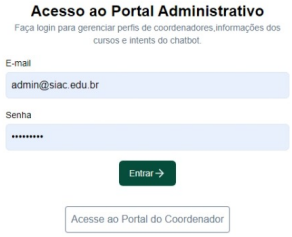
Figura 33 – Página Administrador Bem-vindo



Fonte: Autoria Própria

- **Administrador Página de Login:** Página de autenticação (Figura 34) destinada ao administrador. Ela contém um título que descreve o propósito da autenticação, seguido de uma breve explicação das funcionalidades disponíveis após o *login*. Abaixo, há os campos para entrada de e-mail e senha, além de um botão para validar as credenciais. Logo abaixo, há um botão que permite alternar entre a autenticação de administrador e coordenação.

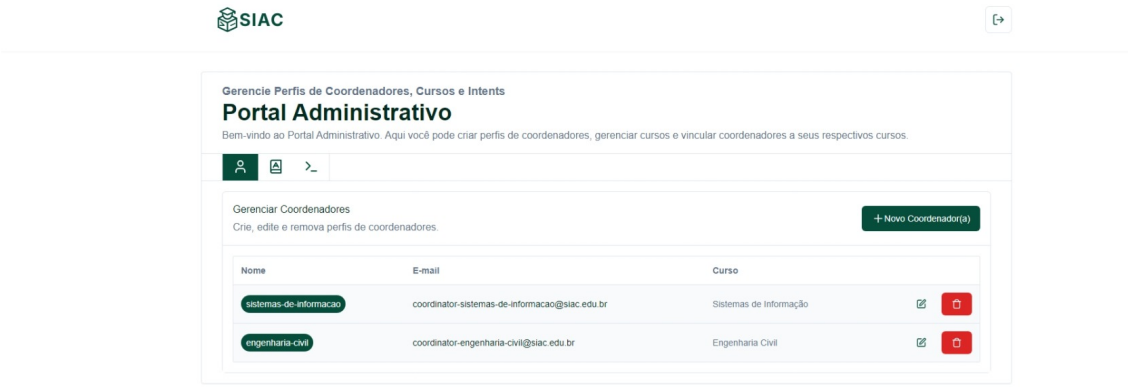
Figura 34 – Página de Login Administrador



Fonte: Autoria Própria

- **Dashboard Administrador:** O *dashboard* (Figura 35) principal do administrador contém um cabeçalho fixo que indica o portal administrativo, acompanhado de uma descrição. Abaixo, há abas que organizam as funcionalidades.
  A primeira aba refere-se ao gerenciamento dos perfis dos coordenadores, onde é possível visualizar uma listagem de coordenadores e as opções de inserir um novo coordenador, bem como excluir ou modificar os coordenadores listados. A segunda aba é dedicada aos cursos, permitindo ao administrador inserir novos cursos e vinculá-los aos perfis dos coordenadores criados anteriormente, além de visualizar os cursos cadastrados e alterá-los ou excluí-los.
  A última aba é voltada para as intenções do chatbot, onde estão listadas as intenções personalizadas previamente cadastradas. Ao criar uma nova intenção, ela é registrada para todo o curso, e o administrador também pode modificá-la ou excluí-la.

Figura 35 – Página de Dashboard Administrador



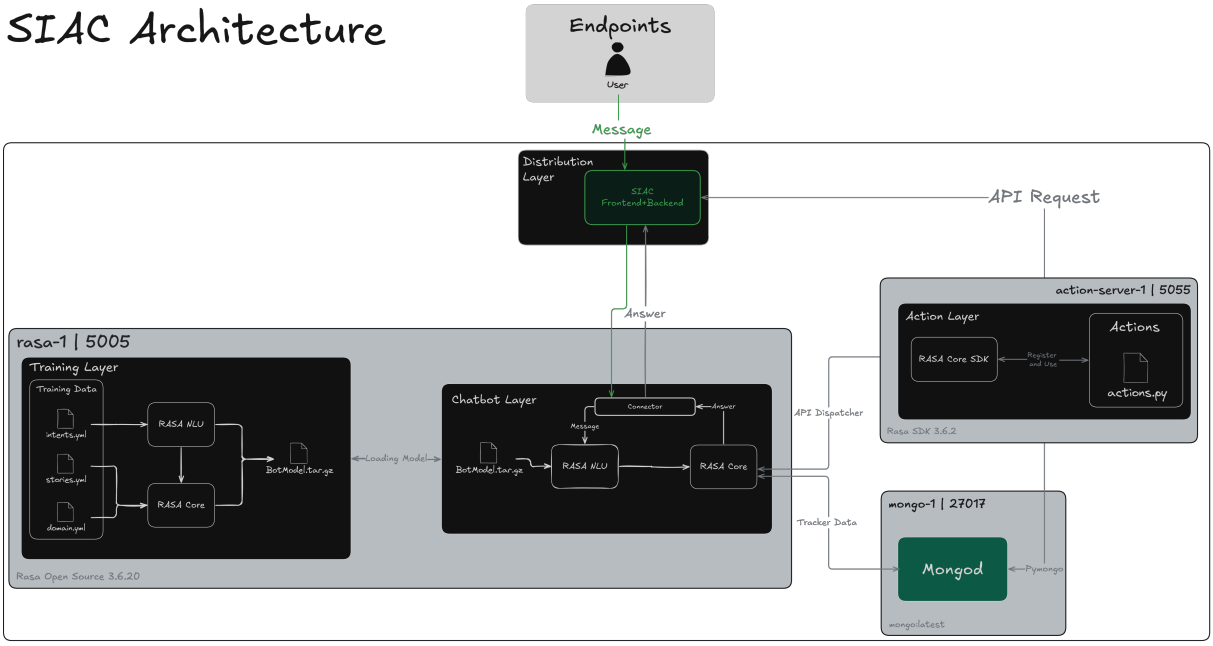
Fonte: Autoria Própria

Esse conjunto de páginas segue a estrutura e o fluxo proposto no *sitemap* (Figura 16), proporcionando uma navegação fluida e clara para os diferentes perfis de usuário, como coordenadores, administradores e usuários não autenticados.

5.2 Arquitetura Resultante

Neste subcapítulo, será apresentada a arquitetura final do Sistema de Apoio ao Calouro (SIAC) integrado ao chatbot MICA. A Figura 36 ilustra todos os contêineres e componentes que compõem o sistema, destacando a estrutura modular da arquitetura. A seguir, cada uma das camadas e componentes será detalhada, elucidando como eles interagem entre si e como contribuem para o funcionamento geral do SIAC.

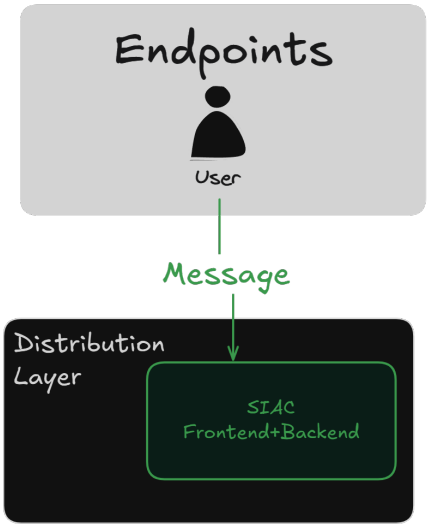
Figura 36 – Arquitetura Final SIAC



Fonte: Autoria Própria

- **Componente Endpoints:** Esse componente (Figura 37) representa o ponto de interação entre os usuários e o sistema. Os diferentes perfis de usuários, como alunos, coordenadores e administradores, utilizam esta camada para enviar requisições e receber respostas. Ele serve como interface externa, conectando-se à camada de distribuição.
- **Camada de Distribuição:** Responsável pela comunicação entre o *front-end* e o *back-end* (Figura 37), esta camada integra as solicitações do componente "Endpoints" com as funcionalidades internas do sistema. Ela distribui as requisições recebidas para os contêineres apropriados e retorna as respostas para o *front-end*.

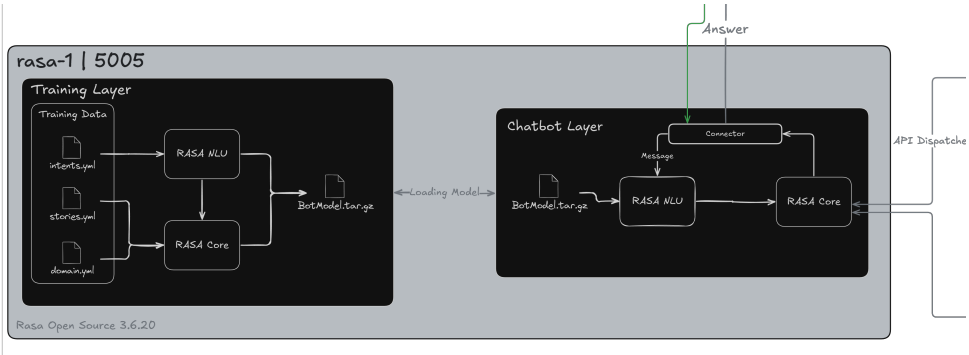
Figura 37 – Componente Endpoints e Camada de Distribuição



Fonte: Autoria Própria

- **Camada de Treinamento:** Parte integrante do contêiner Rasa (Figura 38), essa camada é responsável pelo treinamento dos modelos de linguagem natural (NLU). Ela processa os dados fornecidos, ajustando os parâmetros do modelo para melhorar a identificação de intenções e entidades nas mensagens enviadas pelos usuários. Os modelos treinados são então disponibilizados para as demais camadas do sistema.
- **Camada do Chatbot:** Dentro do contêiner `rasa-1` (Figura 38), essa camada utiliza o modelo NLU treinado para identificar a intenção das mensagens recebidas pelos usuários. O componente "RASA NLU" é responsável por processar as entradas e encaminhá-las ao "*Connector*", que conecta essa camada à camada de distribuição, possibilitando a troca de informações entre o chatbot e as demais funcionalidades do sistema.

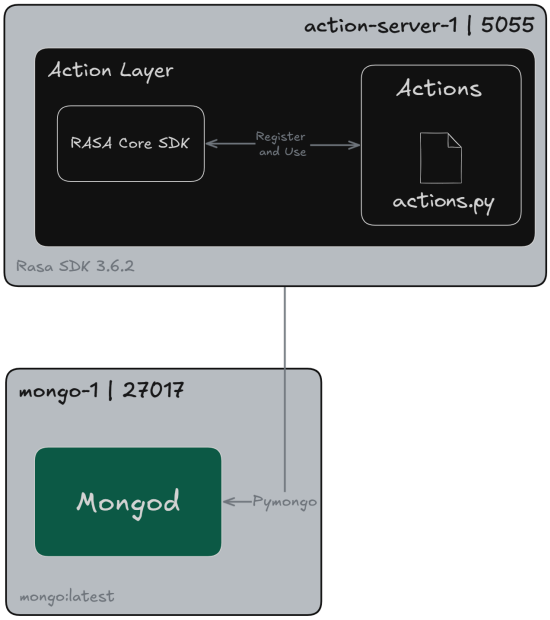
Figura 38 – Camada de Treinamento e Camada do Chatbot



Foonte: Autoria Própria

- **Camada de Ação:** Localizada no contêiner `action-server-1` (Figura 39), essa camada é ativada quando o chatbot reconhece uma intenção que requer a execução de uma ação personalizada. Utilizando o `Rasa Core SDK`, ela processa as instruções definidas no arquivo `actions.py`, podendo, quando necessário, realizar consultas ao banco de dados. Os resultados das ações são retornados por meio do `API Dispatcher`, que repassa as informações processadas para a camada de distribuição.
- **Componente Banco de Dados:** O banco de dados (Figura 39), hospedado no contêiner `mongo-1`, armazena informações essenciais para o funcionamento do sistema, incluindo dados do chatbot e do SIAC. Ele é acessado pela camada de ação sempre que há necessidade de buscar ou armazenar informações relevantes.

Figura 39 – Camada de Ação e Componente de Banco de Dados



Fonte: Autoria Própria

6 CONSIDERAÇÕES FINAIS

Este capítulo é destinado a concluir o trabalho, destacando as principais contribuições, limitações encontradas e as direções para trabalhos futuros.

6.1 Contribuições

Este projeto oferece uma contribuição significativa para a utilização de chatbots em ambientes acadêmicos, proporcionando uma abordagem inovadora para a personalização de respostas por parte dos coordenadores. Além disso, as pesquisas realizadas com coordenadores e discentes permitiram uma melhor compreensão das necessidades dos usuários, resultando em uma solução mais alinhada às expectativas do público-alvo. Assim como os artigos levantados na etapa de revisão de literatura sobre tecnologias na mesma área, espera-se que este material sirva de apoio para a construção de chatbots com objetivos semelhantes ou diferentes.

6.2 Limitações encontradas

Durante o desenvolvimento e implementação do SIAC integrado ao chatbot MICA, foram identificadas algumas limitações no fluxo de conversação do chatbot.

Uma das principais limitações encontradas foi a dificuldade do chatbot em reconhecer corretamente as intenções (intents) em fluxos de conversação mais complexos. Essa limitação se manifestou principalmente em situações onde as mensagens dos usuários eram formuladas de maneira diferente dos exemplos usados no treinamento. Quando as mensagens dos usuários eram muito diferentes dos exemplos fornecidos durante o treinamento, o chatbot apresentava dificuldades para identificar a intenção correta. Esse problema ocorreu principalmente em perguntas ambíguas ou muito longas, exigindo uma reformulação das mensagens para se aproximarem das amostras de dados utilizadas no treinamento. O modelo mostrou forte dependência dos dados utilizados no treinamento. Quando o *input* do usuário estava distante das expressões e palavras-chave treinadas, o chatbot apresentava dificuldade em reconhecer a intenção correta, resultando em respostas inadequadas ou genéricas.

Outra limitação observada foi o envio de respostas que, em alguns casos, não correspondiam à intenção esperada, impactando negativamente a experiência do usuário. Isso ocorreu principalmente devido a erros no reconhecimento da intenção ou baixa confiança do modelo na identificação correta. Em certas ocasiões, o chatbot retornou uma intenção completamente fora do esperado, mesmo quando a entrada do usuário era relativamente simples. Esse comportamento ocorreu mais frequentemente em entradas que

envolviam múltiplas intenções ou quando o modelo não tinha confiança suficiente para determinar a intenção correta, levando a respostas padrões ou inadequadas.

6.3 Trabalhos Futuros

6.3.1 Aprimoramento do SIAC

Como trabalho futuro para o SIAC, são sugeridos os seguintes tópicos:

- Implementar um sistema de priorização nas respostas, permitindo que certas respostas sejam destacadas conforme a relevância ou urgência da consulta.
- Adicionar uma seção de FAQ ao roadmap dos cursos, contendo respostas personalizadas inseridas pelos coordenadores. Essa seção pode incluir as dúvidas mais frequentes daquele curso, que posteriormente poderiam ser utilizadas pelo chatbot.

6.3.2 Aprimoramento da MICA

Os aprimoramentos sugeridos para a MICA incluem:

- Melhorar os fluxos de conversação, tornando-os mais naturais e eficientes, com base em interações reais dos usuários.
- Adicionar mais dados às intents, ampliando o escopo de temas que o chatbot pode tratar e melhorando sua capacidade de resposta.
- Aprimorar as configurações do chatbot para melhorar a classificação de intents e entidades, garantindo uma maior precisão no entendimento das consultas.

6.3.3 Continuidade das Sprints

A organização das sprints foi fundamental para o sucesso do desenvolvimento do projeto, permitindo um planejamento eficiente e uma execução organizada em todas as fases. Nesse contexto, pensa-se na finalização da sprint de desenvolvimento web, incluindo a realização de todos os requisitos funcionais e não funcionais, sem se limitar apenas aos de alta prioridade. Isso garante a robustez da plataforma e o cumprimento integral das expectativas do projeto. Assim, o processo de desenvolvimento segue para as etapas finais, com foco no aperfeiçoamento contínuo do sistema. As próximas etapas incluem:

- **Sprint 7: Testes e Validações** — Realização de testes de integração, unitários e de usabilidade, assegurando que todas as funcionalidades operem conforme o esperado e a experiência do usuário seja satisfatória.
- **Sprint 8: Refinamento e Preparação para Deployment** — Refinamento do sistema com base nos resultados dos testes, ajustando funcionalidades e preparando o sistema para implantação em um ambiente de produção.

A continuidade dessas sprints fechará o ciclo de desenvolvimento inicial do SIAC integrado à MICA, ao mesmo tempo que abre espaço para iterações futuras e melhorias contínuas, garantindo a evolução constante da plataforma.

6.4 Conclusão do Trabalho

O desenvolvimento e a integração entre o sistema SIAC e a MICA foram realizados com sucesso. Nas categorias de requisitos funcionais, todos os itens considerados de alta prioridade foram atendidos. Entre eles, destacam-se os principais de cada categoria, como a possibilidade de o administrador inserir intents aplicáveis a todos os cursos, permitindo que os coordenadores adicionem respostas personalizadas para os usuários poderem consultá-las. Outro requisito de alta prioridade alcançando é a capacidade de usuários não autenticados poderem conversar com o chatbot e consultar o roadmap dos cursos cadastrados. Enquanto também nos requisitos não funcionais, foram realizados os principais, como o desempenho relacionado ao carregamento das telas e entrega das mensagens do chatbot e à responsividade das interfaces.

Deste modo, conforme estabelecido no início deste trabalho, as motivações foram plenamente atingidas por meio dos objetivos delineados, evidenciando o futuro potencial do SIAC em conjunto com a MICA como uma ferramenta valiosa no ambiente acadêmico.

Referências

ADAMOPOULOU, E.; MOUSSIADES, L. An overview of chatbot technology. In: SPRINGER. **IFIP international conference on artificial intelligence applications and innovations**. [S.l.], 2020. p. 373–383. Citado na página 8.

AI, E. **spaCy: Industrial-strength Natural Language Processing in Python**. 2024. <https://spacy.io/>. Acesso em 1 de agosto de 2024. Citado na página 24.

AL., W. C. et. **Manifesto para Desenvolvimento Ágil de Software**. 2001. <https://agilemanifesto.org/iso/ptbr/manifesto.html>. Acesso em 1 de agosto de 2023. Citado na página 18.

ALMEIDA, L. S.; SOARES, A. P.; FERREIRA, J. A. Adaptação, rendimento e desenvolvimento dos estudantes no ensino superior: Construção do questionário de vivências académicas. **Methodus : Revista Científica e Cultural**, 2001. Citado na página 3.

Amazon. **Amazon Lex**. 2023. <https://aws.amazon.com/lex/>. Versão V1. Citado 2 vezes nas páginas 10 e 14.

ARAUJO, E. F. d. S. Solução chatbot no ambiente acadêmico da ufrj. **Pantheon Repositório Institucional da UFRJ**, Universidade Federal do Rio de Janeiro, 2020. Citado 2 vezes nas páginas 16 e 17.

AUTH0. **jsonwebtoken: JSON Web Token implementation for Node.js**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://github.com/auth0/node-jsonwebtoken>. Citado na página 23.

AUTHORS, T. **TensorFlow: An end-to-end open source machine learning platform**. 2024. <https://www.tensorflow.org/>. Acesso em 1 de agosto de 2024. Citado na página 24.

AUTOMATTIC. **Mongoose: MongoDB object modeling for Node.js**. 2024. <https://mongoosejs.com/>. Último acesso em 1 de agosto de 2024. Citado na página 22.

BERNERS-LEE, T. **HTML (Hypertext Markup Language)**. 2023. <https://developer.mozilla.org/en-US/docs/Web/HTML>. Último acesso em 1 de agosto de 2023. Citado na página 21.

BESSA, T.; ARTHAUD, D. D. B. Metodologias ágeis para o desenvolvimento de softwares. **Ciência e Sustentabilidade — CeS**, v. 4, n. 2, p. 173–213, jul./dez. 2018. Citado na página 18.

BIOME. **Biome: A modern linter that can also format and fix your code**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://biomejs.org/>. Citado na página 23.

Botpress. **Botpress**. 2023. <https://www.botpress.com/>. Versão v12.30.9. Citado 2 vezes nas páginas 10 e 14.

CAVALLINI, A. C.; CARAMASCHI, S. Adaptação a universidade. **REPOSITÓRIO INSTITUCIONAL UNESP**, 2012. Citado na página 3.

COLACE, F. et al. Chatbot for e-learning: A case of study. **International Journal of Mechanical Engineering and Robotics Research**, v. 7, n. 5, p. 528–533, 2018. Available at: <https://www.overleaf.com/project/6480b9675aec3d74861e3edb>. Citado na página 7.

COLVIN, S. **Pydantic: Data validation and settings management using Python type annotations**. 2024. <https://pydantic-docs.helpmanual.io/>. Acesso em 1 de agosto de 2024. Citado na página 24.

CONTRIBUTORS, M. L. . A. **Astro 4.15: The web framework for content-driven websites**. 2024. <https://astro.build/>. Acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 21 e 22.

CONTRIBUTORS, N. **Node.js: JavaScript runtime built on Chrome's V8 JavaScript engine**. 2024. <https://nodejs.org/>. Último acesso em 1 de agosto de 2024. Citado na página 22.

DATENSEN. **Moon Modeler - 8.5.1 - Trial Version**. 2024. <https://www.datensen.com/data-modeling/moon-modeler-for-databases.html>. Último acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 21 e 35.

Docker Inc. **Docker: Develop faster. Run anywhere.** 2024. <https://www.docker.com>. Acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 24 e 64.

EICH, B. **JavaScript | MDN**. 2024. <https://developer.mozilla.org/en-US/docs/Web/javascript>. Último acesso em 1 de agosto de 2024. Citado na página 21.

Express.js. **Express**. 2023. <https://expressjs.com/>. Último acesso em 1 de agosto de 2023. Citado na página 22.

FERREIRA, L. P. Desenvolvimento de um chatbot para auxiliar o ensino de espanhol como lıngua estrangeira. **Repositório Institucional UFLA**, 2008. Citado na página 9.

Figma. **Figma 88.1.0 (current) Windows x64**. 2024. <https://www.figma.com/pt-br/>. Último acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 21 e 41.

FILHO, R. L. L. S. et al. A evasão no ensino superior brasileiro. **Cadernos de pesquisa**, SciELO Brasil, v. 37, p. 641–659, 2007. Citado na página 4.

G., J. A. **swagger-autogen: Generate Swagger documentation automatically from your Express routes**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://github.com/dabroek/swagger-autogen>. Citado na página 23.

GALLAGHER, N. **bcrypt: A library to help you hash passwords**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://github.com/kelektiv/node.bcrypt.js>. Citado na página 23.

GitHub. **Axios**. Acesso em 1 de agosto de 2023. <https://github.com/axios/axios>. Citado 2 vezes nas páginas 23 e 69.

GITHUB, I. **GitHub: Repository hosting service**. 2024. <https://github.com/>. Acesso em 1 de agosto de 2024. Citado na página 24.

GOLDBERG, Y. **Neural network methods for natural language processing**. [S.l.]: Springer Nature, 2022. Citado 2 vezes nas páginas 5 e 6.

Google. **Dialogflow ES**. 2023. <https://dialogflow.cloud.google.com/>. Versão ES. Citado 2 vezes nas páginas 10 e 14.

Google. **Dialogflow ES**. 2024. <https://docs.google.com/forms/u/0/>. Último acesso em 6 de agosto de 2024. Citado 2 vezes nas páginas 21 e 26.

HAPKE, H.; HOWARD, C.; LANE, H. **Natural Language Processing in Action: Understanding, analyzing, and generating text with Python**. [S.l.]: Simon and Schuster, 2019. Citado na página 7.

HARRIS, C. R. et al. **Array programming with NumPy**. [S.l.]: Springer Science and Business Media LLC, 2020. 357–362 p. <https://numpy.org/>. Último acesso em 1 de agosto de 2024. Citado na página 27.

HOLANDA, M. et al. Introdução ao ambiente universitário na computação: Uma disciplina para o acolhimento dos estudantes no departamento de ciência da computação da universidade de brasília. **Revista Brasileira de Informática na Educação**, v. 30, p. 494–518, 2022. Citado na página 4.

HUNTER, J. D. Matplotlib: A 2d graphics environment. **Computing in Science & Engineering**, IEEE Computer Society, v. 9, n. 3, p. 90–95, 2007. Último acesso em 1 de agosto de 2024. Citado na página 27.

IBM. **IBM Watson Assistant**. 2023. <https://www.ibm.com/cloud/watson-assistant/>. Versão 4.7.0. Citado 2 vezes nas páginas 9 e 14.

INC., T. L. **tailwindcss v3.4.12: Rapidly build modern websites without ever leaving your HTML.** 2024. <https://tailwindcss.com/>. Acesso em 1 de agosto de 2024. Citado na página 22.

LEONHARDT, M. D. et al. Elektra: Um chatterbot para uso em ambiente educacional. **Revista Novas Tecnologias na Educação**, v. 1, n. 2, 2003. Citado na página 9.

LIE, H. W. **CSS (Cascading Style Sheets) e CSS Modules**. 2023. <https://developer.mozilla.org/en-US/docs/Web/CSS>. Último acesso em 16 de setembro de 2023. Citado na página 22.

LOBO, M. Panorama da evasão no ensino superior brasileiro: aspectos gerais das causas e soluções. **Associação Brasileira de Mantenedoras de Ensino Superior. Cadernos**, v. 25, p. 14, 2012. Citado 2 vezes nas páginas 1 e 4.

LUCCHESI, I. L. et al. Avaliação de um chatbot no contexto educacional: um relato de experiência com metis. **RENOTE: Novas Tecnologias na Educação. Vol. 16, n. 1 (2018), p. 113-122**, 2018. Citado na página 9.

MACIEL, H. B. Ferramentas e criação de chatbot–maciel o robô acadêmico. **Repositório Institucional UFC**, 2019. Citado 2 vezes nas páginas 16 e 17.

MARTIN, J. H. **Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition**. [S.l.]: Pearson/Prentice Hall, 2009. Citado na página 5.

MENDES, L. M.; SIRQUEIRA, T. F. M. Acolhimento de novos estudantes ao ensino superior utilizando chatbots. **Caderno de Estudos em Sistemas de Informação**, v. 7, n. 1, 2022. Citado 2 vezes nas páginas 16 e 17.

Microsoft. **Microsoft Bot Framework**. 2023. <https://dev.botframework.com/>. Versão 2.1.2 SDK v4. Citado 2 vezes nas páginas 10 e 14.

MICROSOFT. **Visual Studio Code: Code Editing. Redefined**. 2024. <https://code.visualstudio.com/>. Acesso em 1 de agosto de 2024. Citado na página 24.

MongoDB. **Mongo 7.0.14 (current) Windows x64**. 2024. <https://www.mongodb.com/pt-br>. Último acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 22 e 34.

MONGODB, I. **MongoDB Compass: The GUI for MongoDB**. 2024. <https://www.mongodb.com/products/compass>. Último acesso em 1 de agosto de 2024. Citado na página 24.

MONGODB, I. **PyMongo: The Python Driver for MongoDB**. 2024. <https://pymongo.readthedocs.io/>. Acesso em 1 de agosto de 2024. Citado na página 24.

OpenAI. **ChatGPT: Conversational AI Model**. 2024. <https://openai.com/chatgpt>. Acesso em 1 de agosto de 2024. Citado na página 48.

Postman. **Postman v11.11 Windows x64**. 2024. <https://www.postman.com/>. Último acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 23 e 43.

PRICKETT, T. et al. Resilience and effective learning in first-year undergraduate computer science. In: **Proceedings of the 2020 acm conference on innovation and technology in computer science education**. [S.l.: s.n.], 2020. p. 19–25. Citado na página 1.

Python Software Foundation. **Python 3.10.11**. 2024. <https://www.python.org/downloads/release/python-31011/>. Último acesso em 1 de agosto de 2024. Citado 2 vezes nas páginas 24 e 44.

Rasa. **Rasa**. 2023. <https://rasa.com/>. Versão 3.6.2. Citado 3 vezes nas páginas 11, 14 e 24.

Rasa. **Rasa Open Source Doc**. 2023. <https://rasa.com/docs/rasa/>. Versão 3.x. Citado 4 vezes nas páginas 12, 13, 14 e 44.

RHYNE, C.; CONTRIBUTORS. **Excalidraw: Online Whiteboard**. 2024. <https://excalidraw.com>. Acesso em 1 de agosto de 2024. Citado na página 21.

RYS, A. **Nodemailer: A module to send emails from Node.js applications**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://nodemailer.com/>. Citado na página 23.

SCOPUS. **Scopus preview—Scopus—Welcome to Scopus, 2020**. 2020. <https://www.scopus.com/standard/marketing.uri>. Acesso em 1 de agosto de 2023. Citado na página 7.

SHARP, R. **nodemon: Monitor for any changes in your node.js application and automatically restart the server**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://nodemon.io/>. Citado na página 23.

SILVA, K. K. C. et al. Desenvolvimento de ferramenta de chatbot como soluçao para a comunicaçao do ifb. In: SBC. **Anais Estendidos do XVII Simpósio Brasileiro de Sistemas de Informação**. [S.l.], 2021. p. 185–188. Citado 2 vezes nas páginas 16 e 17.

SILVA, T. P. d. et al. Desenvolvimento de um sistema do tipo chatbot para o curso de sistemas de informação. **Repositório UFSC**, Florianópolis, SC, 2021. Citado 2 vezes nas páginas 16 e 17.

SOFTWARE, S. **Swagger: Open-source API documentation tool**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://swagger.io/>. Citado na página 23.

SOMMERVILLE, I. Engenharia de software, 9a. **São Palo, SP, Brasil**, p. 63, 2011. Citado na página 18.

STRAUB, S. C. e B. **Git: Version Control System**. 2024. <https://git-scm.com/>. Último acesso em 1 de agosto de 2024. Citado na página 24.

SUTHERLAND, J.; SCHWABER, K. **The Scrum Guide™**. 2017. <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>. Acesso em 1 de agosto de 2023. Citado na página 19.

TECHNOLOGIES, R. **Rasa SDK: Actions and Custom Components for Rasa**. 2024. <https://rasa.com/docs/rasa-sdk/>. Acesso em 1 de agosto de 2024. Citado na página 24.

TEIXEIRA, M. A. P. et al. Adaptação à universidade em jovens calouros. **Psicologia escolar e educacional**, SciELO Brasil, v. 12, p. 185–202, 2008. Citado 2 vezes nas páginas 1 e 4.

TEIXEIRA, R. d. C. P.; MENTGES, M. J.; KAMPFF, A. J. C. Evasão no ensino superior: um estudo sistemático. **Anais do X CIDU – Congresso Ibero-americano de Docência Universitária**, 2019. Citado 2 vezes nas páginas 4 e 5.

The pip developers. **pip 24.2**. 2024. <https://pypi.org/project/pip/>. Último acesso em 1 de agosto de 2024. Citado na página 44.

VAUGHAN, W. **python-dotenv: Reads key-value pairs from a .env file and sets them as environment variables**. 2024. <https://pypi.org/project/python-dotenv/>. Acesso em 1 de agosto de 2024. Citado na página 24.

VISIONMEDIA. **supertest: SuperAgent driven library that allows you to test HTTP servers in Node.js**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://github.com/visionmedia/supertest>. Citado na página 23.

VITE. **Vitest: A blazing-fast unit test framework powered by Vite**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://vitest.dev/>. Citado na página 23.

WILSON, D. C. **morgan: HTTP request logger middleware for node.js**. 2024. Último acesso em 1 de agosto de 2024. Disponível em: <https://github.com/expressjs/morgan>. Citado na página 23.

Wit.ai. **Wit.ai**. 2023. <https://wit.ai/>. Citado 2 vezes nas páginas 10 e 14.