Documento de Arquitetura de Software

| **ChatTCU** | |
| --- | --- |
| **Arquiteto de Software** | **Gerente de Projeto** |
| Luís Felippe Coutinho de Carvalho | Eric Hans Messias da Silva |
| [luis.carvalho@tcu.gov.br](mailto:luis.carvalho@tcu.gov.br) | [erichm@tcu.gov.br](mailto:jessica.costa@saude.gov.br) |
|  |  |

| **Objetivo deste Documento** |
| --- |
| Este documento tem como objetivo descrever as principais decisões de projeto tomadas pela equipe de desenvolvimento e os critérios considerados durante a tomada destas decisões. |

| **Data** | **Autor** | **Descrição** |
| --- | --- | --- |
| 27/05/2024 | Luís Felippe Coutinho de Carvalho | Confecção do diagrama de arquitetura da rotina de conversação |
| 28/05/2024 | Luís Felippe Coutinho de Carvalho | Confecção do diagrama de arquitetura da rotina de upload de arquivos |
| 29/05/2024 | Luís Felippe Coutinho de Carvalho | Redação da introdução e ajustes nos diagramas e confecção do diagrama de pacotes |
| 03/06/2024 | Luís Felippe Coutinho de Carvalho | Redação dos requisitos e restrições e descrição do fluxo de conversação |
| 04/06/2024 | Luís Felippe Coutinho de Carvalho | Redação da descrição do fluxo de upload de arquivos e da descrição dos pacotes |
| 05/06/2024 | Luís Felippe Coutinho de Carvalho | Confecção do diagrama de arquitetura das telas iniciais, sidebar-history, sidebar-upload |
| 06/06/2024 | Luís Felippe Coutinho de Carvalho | Confecção dos diagramas de arquiteturas do chatbox-message-citationbutton, chatbox-input-chipfiles, requisições |
| 07/06/2024 | Luís Felippe Coutinho de Carvalho | Redação da descrição dos componentes e pacotes do frontend |
| 10/06/2024 | Luís Felippe Coutinho de Carvalho | Edição do diagrama sidebar-upload revisão da redação de todo documento |
| 18/09/2024 | Luís Felippe Coutinho de Carvalho | Captura e descrição dos diagramas de pipeline do backend e frontend. |

**Sumário**

[1. INTRODUÇÃO 2](#_heading=h.gjdgxs)

[1.1 Finalidade 3](#_heading=h.30j0zll)

[1.2 Escopo 3](#_heading=h.2et92p0)

[1.3 Definições, Acrônimos e Abreviações 3](#_heading=h.tyjcwt)

[2. REQUISITOS E RESTRIÇÕES ARQUITETURAIS 4](#_heading=h.3dy6vkm)

[3. VISÃO LÓGICA 5](#_heading=h.17dp8vu)

[3.1 Visão Geral - Backend 5](#_heading=)

[3.1.1 Conversação 5](#_heading=)

[3.1.1.1 Descrição do fluxo do caso de uso Conversação 5](#_heading=)

[3.1.2 Upload de arquivos 8](#_heading=)

[3.1.2.1 Descrição do fluxo do caso de uso Upload de Arquivos 8](#_heading=h.yjjbbukk1zh5)

[3.2 Visão de Pacotes - Backend 10](#_heading=h.u9yyfcv92azb)

[3.2.1 Descrição dos pacotes 10](#_heading=h.sfeyxe3s11rx)

[3.3 Visão Geral - Frontend 12](#_heading=h.4od1mg6jk48a)

[3.3.1 Tela Inicial 12](#_heading=h.s0rf483cj1sq)

[3.3.1.1 Descrição dos componentes da tela inicial 12](#_heading=h.m7z0hhnes4ed)

[3.3.2 Sidebar - History 14](#_heading=h.w53jrifmi4r6)

[3.3.2.1 Descrição dos componentes da Sidebar - History 14](#_heading=h.rxpos4jmss65)

[3.3.3 Sidebar - Upload 16](#_heading=h.n18ej537azr0)

[3.3.3.1 Descrição dos componentes da Sidebar - Upload 16](#_heading=h.i90wyx38xtvc)

[3.3.4 ChatBox - Message - CitationButton 19](#_heading=h.w74zd9uoujn2)

[3.3.4.1 Descrição dos componentes do ChatBox - Message - CitationButton 19](#_heading=h.68ku0iqv0aq5)

[3.3.5 ChatBox - Input - ChipFiles 20](#_heading=h.wqahy2usq6b2)

[3.3.5.1 Descrição dos componentes do ChatBox - InputBox - InputFiles 20](#_heading=h.4lyekq74pbtd)

[3.3.6 Diagrama de Arquitetura das Requisições Originadas pelo Frontend 21](#_heading=h.8ueocl4h82j4)

[3.3.6.1 Descrição de Arquitetura das Requisições Originadas pelo Frontend 21](#_heading=h.tda1h4gtkh0j)

[3.4 Visão de Pacotes - Frontend 22](#_heading=h.pos0w0zgni5l)

[3.4.1 Descrição dos pacotes 22](#_heading=h.c5ot3vk97z5q)

# INTRODUÇÃO

## Finalidade

Este documento fornece uma visão arquitetural abrangente do sistema ChatTCU, usando diagramas de arquitetura para **representar** diferentes aspectos do sistema. O objetivo deste documento é capturar e comunicar as decisões arquiteturais significativas que foram tomadas em relação ao sistema.

## Escopo

Este Documento de Arquitetura de Software se aplica ao ChatTCU, que será desenvolvido pelo Núcleo de Inteligência Artificial (NIA) do Tribunal de Contas da União (TCU).

## Definições, Acrônimos e Abreviações

*AI – Artificial Inteligence* ou Inteligência Artificial (IA). A IA é a inteligência similar à humana exibida por mecanismos ou software, além de também ser um campo de estudo acadêmico.

*LLM – Large Language Model*, ou Grande Modelo de Linguagem. Termo utilizado para descrever um modelo de aprendizado de máquina treinado para aprender a partir de enormes bases de dados. Como resultado, a tecnologia consegue gerar uma linguagem para conversar com humanos e desenvolver contexto, permitindo às pessoas obterem respostas rápidas em plataformas de IA generativa.

*NLP – Natural Language Processing*, ou Processamento de Linguagem Natural (PLN). É uma área da ciência da computação que combina computação, linguística e até filosofia da linguagem para estudar como fazer computadores entenderem a linguagem natural humana.

*Embeddings* - Os *embeddings* de texto são uma técnica de PLN que converte dados textuais em vetores numéricos que podem ser processados por algoritmos de aprendizado de máquina, especialmente modelos grandes. Essas representações vetoriais são projetadas para capturar o significado semântico e o contexto das palavras que representam.

# REQUISITOS E RESTRIÇÕES ARQUITETURAIS

Esta seção descreve os requisitos de software e restrições que têm um impacto significativo na arquitetura.

| Requisito | Solução |
| --- | --- |
| Linguagem | *Python* 3.11, utilizando o *framework FastAPI* para a criação de *API RESTful* |
| Plataforma | *Rancher/Kubernetes/Docker* |
| Segurança | Para acesso às rotas protegidas os usuários deverão estar devidamente autenticados pelo SIGA contando com seus respectivos *tokens* e papéis. |
| Persistência | O histórico de conversas com suas mensagens deverão ser persistidas utilizando o *ElasticSearch*. Os metadados dos arquivos enviados por *uploads* deverão ser persistidos no *Azure Cosmos DB* e os arquivos armazenados no *Azure Blob Storage*. Tanto os conteúdos dos arquivos quanto as demais informações utilizadas para contextualização do *LLM* deverão ser indexadas utilizando o *Azure AI Search*, dependendo do caso, utilizando ou não *embeddings*. |

Tabela 1 – Requisitos e restrições do backend

| Requisito | Solução |
| --- | --- |
| Linguagem | *TypeScript* utilizando a biblioteca *ReactJS*, sem *frameworks*. |
| Plataforma | *Rancher/Kubernetes/Docker ou Azure Blob Storage* |
| Segurança | Para que o usuário consiga utilizar a aplicação deverá estar devidamente autenticado no SIGA |
| Persistência | Todas as regras de negócios, que não sejam relacionados a comportamentos de *interface*, deverão constar no *backend*. Assim visando deixar o *frontend*, mais leve e ‘burro’, sem agregar complexidades desnecessárias. |

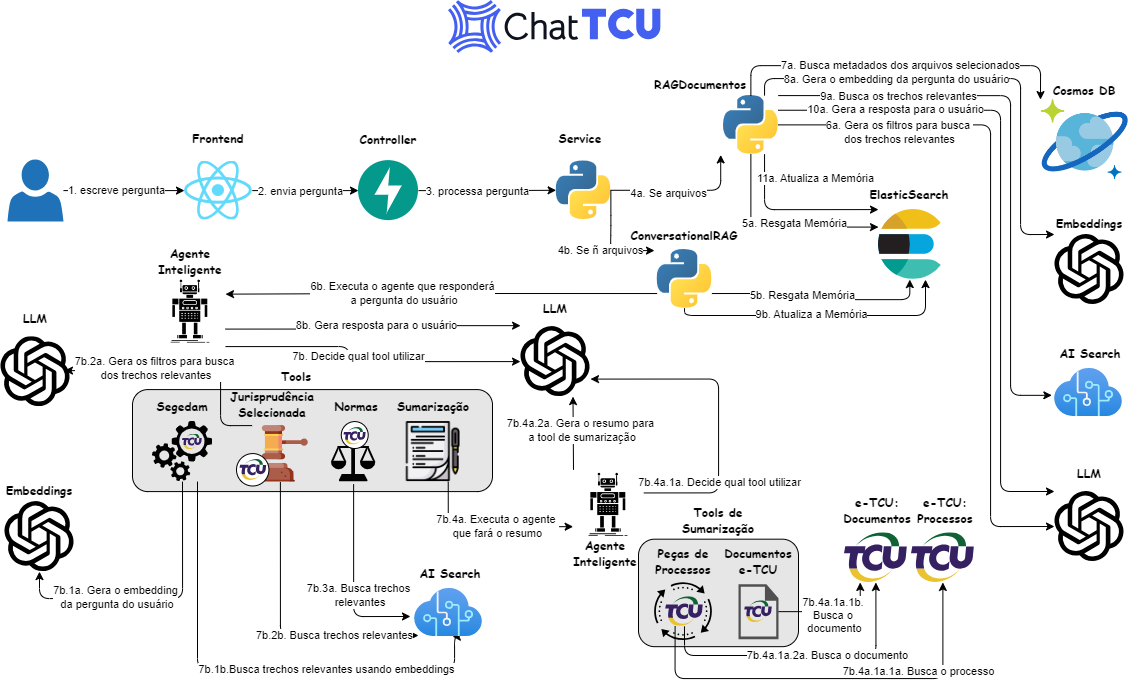
Tabela 2 – Requisitos e restrições do frontend

# VISÃO LÓGICA

Neste capítulo, apresentaremos as principais classes do sistema, a organização do código, os relacionamentos entre elas e o fluxo dos casos de uso mais relevantes.

## Visão Geral - Backend

### Conversação



*Figura 1 – Diagrama de Arquitetura de Software - Conversação*

### *Descrição do fluxo do caso de uso Conversação*

**1 - Escreva a pergunta:** Através do *frontend* o usuário escreve a sua pergunta no campo de *input* apropriado.

**2 - Envia a pergunta:** Através do *frontend* o usuário envia a pergunta ao *backend* ao clicar no botão de envio ou ao pressionar a tecla *“enter”*. Após a ação de envio o *frontend* fica aguardando o início do *stream* da resposta.

**3 - Processa a pergunta:** Após receber a pergunta do usuário através do *controller*, o *backend* envia a pergunta para o devido processamento e geração da resposta.

**4 - Após receber do *controller* a pergunta do usuário, o *service* realizará a validação da pergunta;**

**a - se para a pergunta o usuário tiver selecionado arquivos para a contextualização do *LLM***

**4a - Se arquivos:** definirá como *‘engine’* a classe RAGDocumentos que realizará o devido processamento da pergunta e geração da resposta.

**5a - Resgata a Memória:** Se a pergunta é uma mensagem que continua um *chat* já persistido, resgata todas as mensagens (perguntas/respostas) anteriores a fim de montar a memória do *chat* contextualizando o *LLM*.

**6a - Gera os filtros para a busca dos trechos relevantes:** com base na pergunta do usuário, realiza uma requisição ao *LLM* para extrair possíveis filtros para a busca dos trechos relevantes.

**7a - Busca metadados dos arquivos selecionados:** solicita ao *Azure Cosmos DB* os metadados dos arquivos selecionados pelo usuário para contextualização do *LLM.*

**8a - Gera o *embedding* da pergunta do usuário:** solicita ao modelo de *embeddings* da *OpenAI* o *text embedding* da pergunta do usuário.

**9a - Busca os trechos relevantes:** solicita a *Azure AI Search* os trechos relevantes dos documentos indexados que atendam ao filtro e/ou a similaridade por *embeddings* da pergunta do usuário.

**10a - Gera a resposta para o usuário:** com a contextualização dos trechos relevantes e a memória resgatada, solicita ao *LLM* a geração da resposta à pergunta do usuário.

**11a - Atualiza a memória:** após receber toda a resposta que responde a pergunta do usuário oriunda do *LLM*, persiste as mensagens dessa transação. A mensagem de sistema contém o *prompt* de sistema utilizado e todos os trechos relevantes retornados para a contextualização do *LLM*. A mensagem do usuário contém a pergunta do usuário e a lista de dos identificadores dos arquivos selecionados. A mensagem de resposta contém a resposta do *LLM* à pergunta do usuário, a lista dos identificadores dos arquivos selecionados e todos os trechos utilizados pelo *LLM* para responder a pergunta do usuário.

**b - se para a pergunta o usuário NÃO tiver selecionado arquivos para a contextualização do *LLM***

**4b - Se ñ arquivos:** definirá como *‘engine’* a classe ConversationalRAG que realizará o devido processamento da pergunta e geração da resposta.

**5b - Resgata a memória:** Se a pergunta é uma mensagem que continua um *chat* já persistido, resgata todas as mensagens (perguntas/respostas) anteriores a fim de montar a memória do *chat* contextualizando o *LLM.*

**6b - Executa o agente que responderá a pergunta do usuário:** define o agente inteligente que se encarregará de escolher alguma *tool* a sua disposição para gerar a contextualização do *LLM*, mesmo que essa *tool* seja seu conhecimento geral.

**7b - Decide qual *tool* utilizar:** com base nas descrições das tools que recebeu, realiza uma requisição ao *LLM* a fim de decidir qual *tool* utilizar.

**Segedam:** *Tool* que resgata do índice informações pertinentes às rotinas e processos administrativos do Tribunal.

**7b.1a - Gera o *embedding* da pergunta do usuário:** solicita ao modelo de *embeddings* da *OpenAI* o *text embedding* da pergunta do usuário.

**7b.1b - Busca os trechos relevantes usando *embeddings*:** solicita a *Azure AI Search* os trechos relevantes das informações indexadas que, por similaridade de *embeddings,* sejam semelhantes a pergunta do usuário.

**Jurisprudência Selecionada:** *Tool* que resgata do índice informações pertinentes às Jurisprudências Selecionadas definidas pelo Tribunal.

**7b.2a - Gera os filtros para a busca dos trechos relevantes:** com base na pergunta do usuário, realiza uma requisição ao LLM para extrair possíveis filtros para a busca dos trechos relevantes.

**7b.2b - Busca os trechos relevantes:** solicita a *Azure AI Search* os trechos relevantes das jurisprudências indexadas que atendam ao filtro e a pergunta do usuário.

**Normas:** *Tool* que resgata do índice de informações pertinentes às Normas que orientam o Tribunal.

**7b.3a - Busca os trechos relevantes:** solicita a *Azure AI Search* os trechos relevantes das normas indexadas que atendam a pergunta do usuário.

**Sumarização:** *Tool* encarregada de realizar a sumarização de peças de processo ou de documentos do e-TCU.

**7b.4a - Executa o agente que fará o resumo:** define o agente inteligente que se encarregará de escolher alguma *tool* a sua disposição para gerar a o resumo através do *LLM*.

**7b.4a.1a - Decide qual *tool* utilizar:** com base nas descrições das tools que recebeu, realiza uma requisição ao *LLM* a fim de decidir qual *tool* utilizar.

**Peças de Processos:** *Tool* que sumariza uma determinada peça, documento, associada a um processo.

**7b.4a.1a.1a - Busca o processo:** busca as informações do processo informado na pergunta do usuário, bem como das peças associadas ao processo. Tais informações serão cedidas através de requisições ao serviço que detêm tais informações.

**7b.4a.1a.2a - Busca a peça:** busca o conteúdo da peça associado ao processo, onde ambos foram informados pelo usuário em sua pergunta. Tais informações serão cedidas através de requisições ao serviço que detêm tais informações.

**Documentos e-TCU:** *Tool* que sumariza um determinado documento do e-TCU.

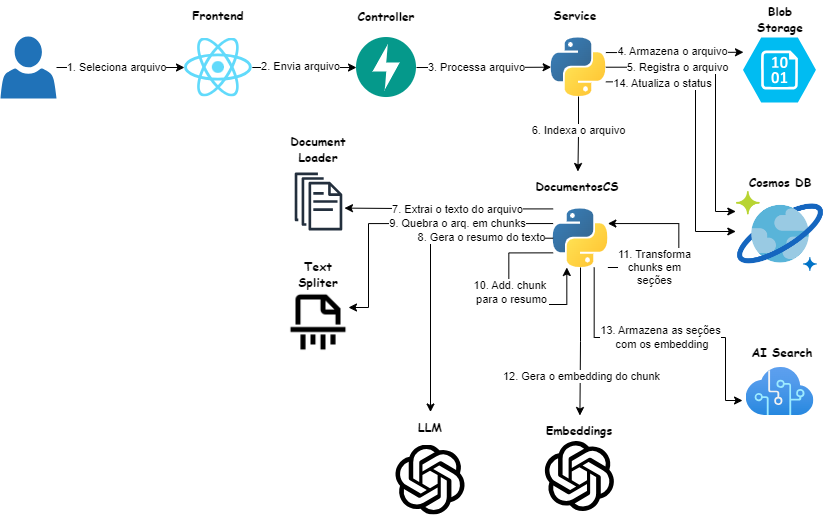
**7b.4a.1a.1b - Busca o documento:** busca o conteúdo do documento informado pelo usuário em sua pergunta. Tais informações serão cedidas através de requisições ao serviço que detêm tais informações.

**7b.4a.2a - Gera o resumo para a *tool* de sumarização:** realiza o resumo intermediário de forma incremental, fazendo requisições ao *LLM* tantas quantas forem necessárias de acordo com o tamanho do texto do documento e janela de contexto do *LLM*.

**8b - Gera a resposta para o usuário:** com a contextualização dos trechos relevantes e a memória resgatada, solicita ao *LLM* a geração da resposta à pergunta do usuário ou o resumo final do documento ou peça do processo indicado pelo usuário em sua pergunta..

**9b - Atualiza a memória:** após receber toda a resposta que responde a pergunta do usuário oriunda do *LLM*, persiste as mensagens dessa transação. A mensagem de sistema contém o *prompt* de sistema utilizado e todos os trechos relevantes retornados para a contextualização do *LLM*. A mensagem do usuário contém a pergunta do usuário. A mensagem de resposta contém a resposta do *LLM* à pergunta do usuário e os trechos utilizados pelo *LLM* para responder a pergunta do usuário.

### Upload de arquivos



*Figura 2 – Diagrama de Arquitetura de Software – Upload de Arquivos*

### Descrição do fluxo do caso de uso Upload de Arquivos

**1 - Seleciona o arquivo:** Através do *frontend* o usuário seleciona o(s) arquivo(s) que deseja enviar via *upload*. O usuário pode escolher o(s) arquivo(s) ao clicar no campo próprio que abrirá uma janela para seleção do(s) arquivo(s) no sistema de arquivos do sistema operacional ou mesmo selecioná-lo(s) no sistema de arquivos do sistema operacional e arrastá-lo(s) ao navegador, até o campo próprio, para realizar o envio.

**2 - Envia o arquivo:** Após o usuário selecionar o(s) arquivos ou soltar o(s) arquivo(s) selecionado(s) no campo próprio, o frontend envia o(s) arquivo(s) ao *backend*. O usuário pode enviar mais de um arquivo, sendo cada um de até *50MB* e de tipos iguais à *PDF, DOCX, XLSX* e *CSV*. O *frontend* envia cada arquivo em sua própria requisição, assim sendo processado individualmente pelo *backend*.

**3 - Processa o arquivo:** O *backend* recebe o arquivo pelo *controller* e encaminha ao *service* para a validação e processamento.

**4 - Armazena o arquivo:** Após a validação do arquivos o *service* enviará o arquivo para o devido armazenamento no *Azure Blob Storage*.

**5 - Registra o arquivo:** Tendo já realizado o armazenamento, o service realiza o devido registro dos metadados do arquivo no *Azure Cosmos DB*. Nesse registro o status do arquivo deverá constar como ARMAZENADO.

**6 - Indexa o arquivo:** Após o registro o *service* instancia a classe *DocumentosCS* que se encarregará da ação de indexar o arquivo para as futuras consultas para as contextualizações do *LLM*.

**7 - Extrai o texto do arquivo:** Realiza o *loader* do arquivo para a extração do texto. Para cada tipo de arquivo há um *loader* próprio. No caso de um arquivo do tipo *docx*, antes de ser extraído o texto, ele é convertido em *PDF* para que a extração do texto ocorra por página, mantendo assim a referência da página de origem ao indexar o arquivo.

**8 - Gera o resumo do texto:** Realiza uma solicitação ao *LLM* para gerar o resumo do texto extraído do arquivo, a fim de que um resumo do arquivo conste entre os trechos do arquivo no índice**.**

**9 - Quebra o arquivo em chunks:** Realiza o *split* do texto do arquivo em *chunks*, trechos que serão indexados como seções do arquivo no índice.

**10 - Add. chunk para resumo:** Adiciona o resumo como um dos *chunks* para ser persistido no índice.

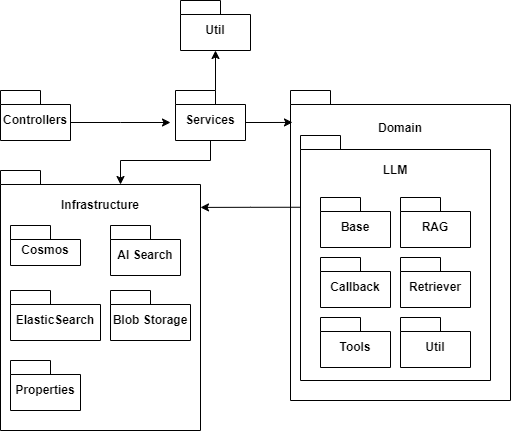
**11 - Transforma chunks em seções:** Dado os *chunks* do arquivo, retorna as seções, estrutura de dados que conterão as informações do *chunk*, como página, nome do arquivo, conteúdo do *chunk*, o respectivo *text embedding* do *chunk*.

**12 - Gera o embedding do chunk:** Para cada *chunk* a ser transformado em seção, solicita o *text embedding* do *chunk* através do modelo de *text embedding* da *OpenAI.*

**13 - Armazena as seções com seus respectivos embeddings:** Após as transformações dos *chunks* em seções, já com seus respectivos *embeddings*, persiste as seções no *Azure AI Search* para futura consulta para contextualização do *LLM* com base na pergunta do usuário.

**14 - Atualiza o status do arquivo:** Tendo concluído a persistência das seções no *Azure AI Search*, processo de indexação, atualiza o *status* do arquivo no *Azure Cosmos DB* definindo o valor como PRONTO.

## Visão de Pacotes - Backend



*Figura 3 – Diagrama de Pacotes*

### Descrição dos pacotes

**1 - *Controllers*:** Pacote que contém os *scripts* que definem as rotas para a *API* do *backend*.

**2 - *Services*:** Pacote que contém os *scripts* que definem os métodos utilizados para processar as requisições recebidas pelos *controllers* e/ou acessar outros serviços durante o processamento das requisições.

**3 - *Domain*:** Pacote que contém os *scripts* que definem as entidades de domínio do sistema.

**3.1 - *LLM*:** Pacote que contém os *scripts* com as classes que definem, principalmente, as ações e comportamentos de uma solicitação ao *LLM*.

**3.1.1 - *Base*:** Pacote que contém os *scripts* que definem as classes que compõem a base de um LLM.

**3.1.2 - *RAG*:** Pacote que contém os *scripts* que definem as classes especializadas em *RAG*

**3.1.3 - *Callback*:** Pacote que contém os *scripts* que definem as classes utilizadas como *callback* durante a requisição ao *LLM*.

**3.1.4 - *Retriever*:** Pacote que contém os *scripts* que definem as classes utilizadas pelas *tools*, principalmente, para recuperação das informações para a contextualização do *LLM*.

**3.1.5 - *Tools*:** Pacote que contém os *scripts* que definem as classes utilizadas como *tools* para que o *LLM* execute tarefas específicas, normalmente de resgate de informações para a devida contextualização do *LLM* ao gerar a resposta à pergunta do usuário.

**3.1.6 - Util:** Pacote que contém os *scripts* que definem as classes e métodos utilitários utilizados pelo *LLM*.

**4 - *Infrastructure*:** Pacote que contém os *scripts* que definem as classes e os métodos relacionados à infraestrutura do sistema.

**4.1 - *Cosmos*:** Pacote que contém os *scripts* que definem as classes e métodos que definem as ações relacionadas às requisições ao *Azure Cosmos DB*.

**4.2 - *AISearch*:** Pacote que contém os *scripts* que definem as classes e métodos que definem as ações relacionadas às requisições ao *Azure AI Search*.

**4.3 - *ElasticSearch*:** Pacote que contém os *scripts* que definem as classes e métodos que definem as ações relacionadas às requisições ao *ElasticSearch*.

**4.4 - *BlobStorage*:** Pacote que contém os *scripts* que definem as classes e métodos que definem as ações relacionadas às requisições ao *Azure Blob Storage*.

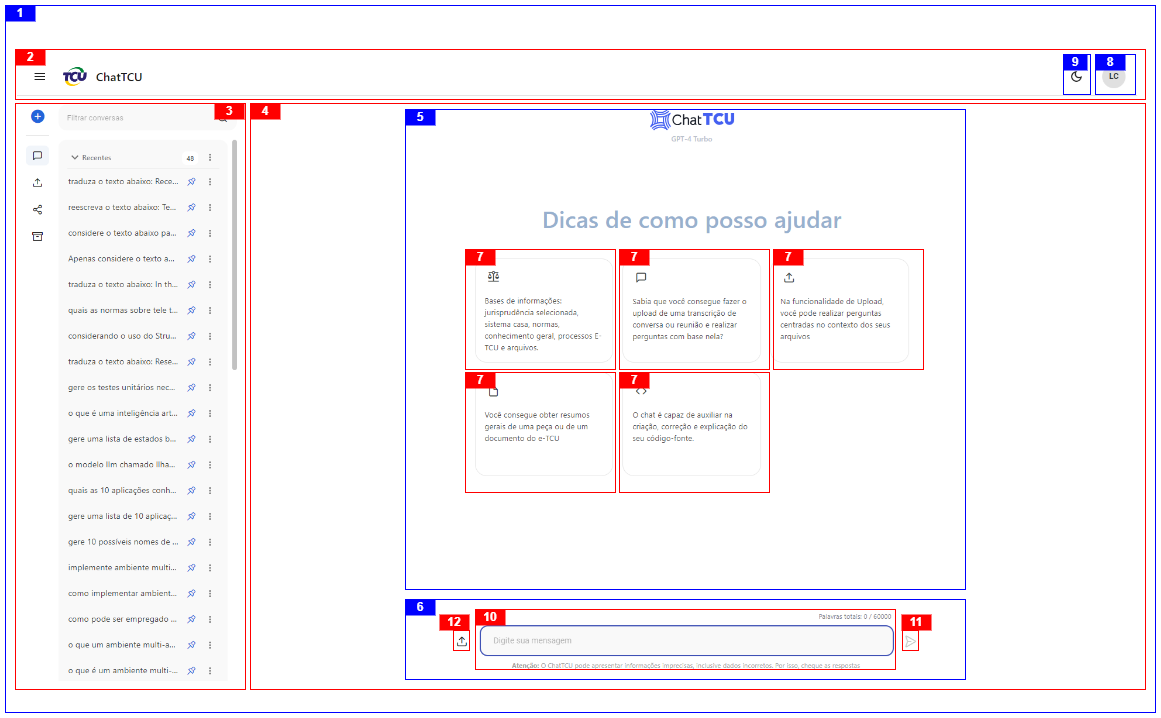
**4.5 - *Properties*:** Pacote que contém os *scripts* que definem as classes e métodos que definem as ações relacionadas às requisições ao *Spring Cloud Properties* e *Azure Key Vault*.

**5 - Util:** Pacote que contém os *scripts* que definem as classes e métodos utilitários utilizados pelo sistema.

## Pipeline CI/CD - Backend

## Visão Geral - Frontend

### Tela Inicial



*Figura 4 – Componentização da Tela Inicial*

### Descrição dos componentes da tela inicial

**1 - Componente - *ChatPage*:** Componente responsável por compor toda a tela principal do *frontend* do *ChatTCU*.

**2 - Componente - *Header*:** Componente responsável por compor o topo da tela do *frontend* do *ChatTCU*.

**3 - Componente - *Sidebar*:** Componente responsável por montar a coluna lateral esquerda da tela principal.

**4 - Componente - *ChatBox*:** Componente responsável por compor a tela principal montando a área central e direita.

**5 - Componente - *Panel*:** Componente responsável por compor o *ChatBox* quando não houver *chat* definido.

**6 - Componente - *InputBox*:** Componente responsável por compor o *ChatBox* com os demais componentes que montam o *input* para que o usuário possa redigir sua pergunta.

**7 - Componente - *EmptyChat*:** Componente responsável por compor o *Panel* exibindo como *card* na tela informações ao usuário sobre o sistema ou mesmo exemplos de perguntas clicáveis que demonstrem a capacidade da aplicação.

**8 - Componente - *AvatarMenu*:** Componente responsável por compor o *Header* onde é exibido duas iniciais do nome do usuário e fornece também um menu que contém a opção de saída do sistema.

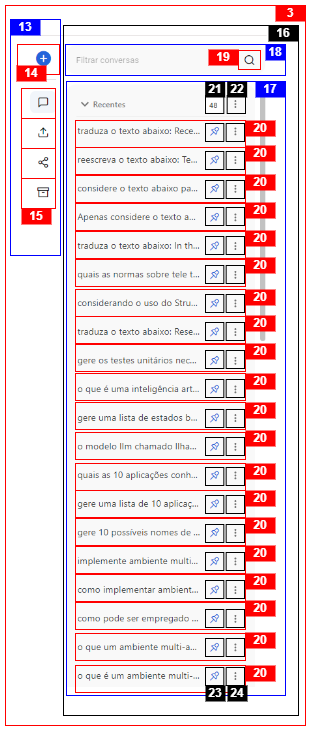
**9 - Componente - DarkmodeToggle:** Componente responsável por implementar a alternância de temas entre o *Dark Mode* e *Light Mode*.

**10 - Componente - TextField:** Componente definido pela biblioteca *MaterialUI,* que no *frontend* tem a função de receber a pergunta digitada pelo usuário.

**11 - Componente - IconButton -** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de enviar a pergunta digitada pelo usuário.

**12 - Componente - IconButton -** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de abrir a aba de *upload***.**

### Sidebar - History



*Figura 5 – Componentização do Sidebar com o History*

### Descrição dos componentes da Sidebar - History

**13 - Componente - *SidebarSwitcher*:** Componente responsável por compor o *Sidebar* implementando a exibição das opções a serem exibidas dentro do *Sidebar*.

**14 - Componente - *IconButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de iniciar um novo *chat***.**

**15 - Componente - *SidebarSwitcherItem*:** Componente responsável por compor o *SidebarSwitcher* implementando cada opção disponível pelo sistema para alternar entre os componentes a ser exibidos no *Sidebar*. As opções são exibidas através do componente *Button* da biblioteca *MaterialUI*, comportando-se como uma aba de uma planilha.

**16 - Componente - *History*:** Componente responsável por compor o *Sidebar* com os componentes que montam o histórico de *chats* do usuário.

**17 - Componente - *HistoryContent*:** Componente responsável por compor o *History* com as listagem dos *chats* em suas categorias. As categorias serão *chats* fixados e recentes.

**17.1 - Componente - *HistoryAccordion*:** Componente responsável por compor o *HistoryContent* com a listagem dos *chats* em uma determinada categoria.

**18 - Componente - *FilterField*:** Componente responsável por compor o *HistoryContent*  com um campo responsável por realizar o filtro dos *chats* constantes no histórico através de seus títulos.

**19 - Componente - *IconButton*:** Componente responsável por compor o *FilterField* com um ícone.

**20 - Componente - *ListItemButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de listar um *chat* do histórico de *chats* do usuário compondo assim o *HistoryAccordion*.

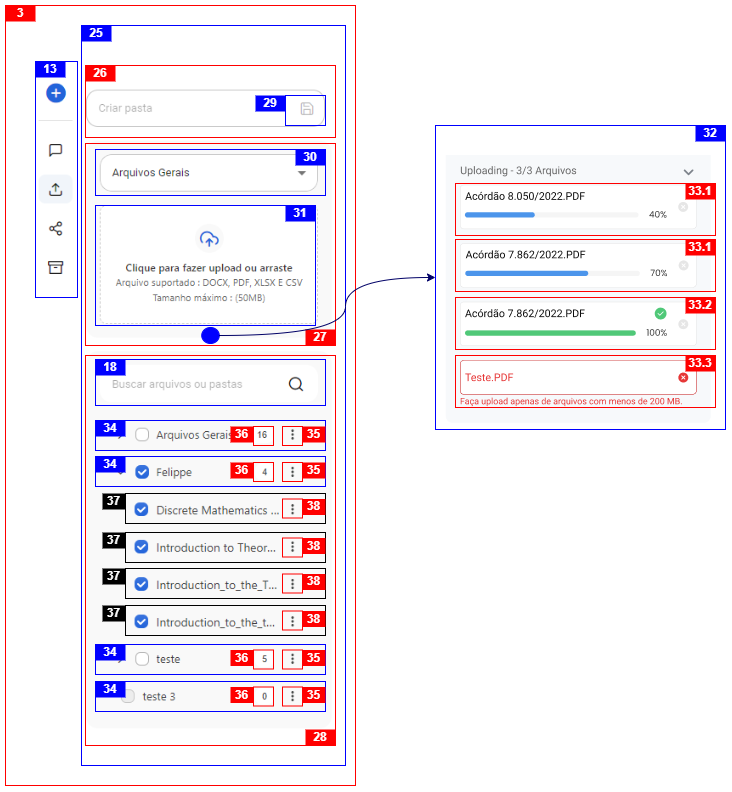
**21 - Componente - *Box*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de exibir a quantidade de *chats* pertencentes àquela categoria, compondo assim o *HistoryAccordion*.

**22 - Componente - *IconButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de acionar o menu pertinente à ações relacionadas a todos os *chats* daquela categoria, compondo assim o *HistoryAccordion*.

**23 - Componente - *IconButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função incluir ou não o *chat* na categoria de fixado. Quando não estiver na categoria fixado, o *chat* consta na categoria recentes. O componente compõe *ListItemButton*.

**24 - Componente - *IconButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* tem a função de acionar o menu pertinente à ações relacionadas ao *chat* em questão, compondo assim o *ListItemButton*.

### Sidebar - Upload



*Figura 6 – Componentização do Sidebar com o Upload*

### Descrição dos componentes da Sidebar - Upload

**25 - Componente -*Upload*:** Componente responsável por compor o *Sidebar* com os componentes que montam o *upload* de arquivos e listagem dos arquivos que foram enviados pelo usuário.

**25.1 - Componente - *FileUpload*:** Componente responsável por compor o *Upload* com a listagem dos *chats* em uma determinada categoria.

**26 - Componente - *FolderField*:** Componente responsável por compor o *FileUpload* com componentes que montam e possibilitam a criação de uma nova pasta para receber *uploads* de arquivos.

**27 - Componente - *UploadDropDown*:** Componente responsável por compor o *FileUpload* com componentes que montam e possibilitam o *upload* de arquivos na pasta de destino.

**28 - Componente - *ListFolderCustom*:** Componente responsável por compor o *FileUpload* com componentes que montam a listagem de pastas e seus arquivos que o usuário realizou o *upload*.

**29 - Componente - *IconButtom*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* que compõe *FolderField* e tem a função de disparar a ação de criar uma nova pasta para receber *upload* de arquivos. Ao ser criada a pasta imediatamente deve constar no *SelectDir* como a opção selecionada.

**30 - Componente - *SelectDir*:** Componente responsável por compor o *UploadDropDown* com componentes que montam uma lista de opções selecionáveis, onde cada opção corresponde a uma pasta que o usuário criou para receber *uploads* de arquivos ou a pasta default *“Arquivos Gerais”*.

**31 - Componente - *UploadDropZone*:** Componente responsável por compor o *UploadDropDown* com componentes que montam a área onde o usuário “largará” os seus arquivos ou clicar para selecionar os arquivos que serão enviados no *upload*.

**32 - Componente - *UploadStatus*:** Componente responsável por compor o *UploadDropDown* com componentes que montem e exibam os *status* de cada arquivo durante o processo de *upload*. Deverá estar visível somente se houver arquivo sendo processado.

**33 - Componentes - *UploadProgress, UploadPreparing* e *UploadError*:**

**33.1 - Componente - *UploadProgress*:** Componente responsável por compor o *UploadStatus* com componentes que montem e exibam os *status* do arquivo enquanto este estiver sendo enviado ao *backend*. Deverá estar visível somente se houver arquivo sendo enviado.

**33.2 - Componente - *UploadPreparing*:** Componente responsável por compor o *UploadStatus* com componentes que montem e exibam os *status* do arquivo após ser enviado e estiver sendo processado pelo *backend*. Deverá estar visível somente após o arquivo for enviado e estiver aguardando a conclusão do processamento.

**33.3 - Componente - *UploadError*:** Componente responsável por compor o *UploadStatus* com componentes que montem e exibam os *status* do arquivo quando houver um erro tanto validação, ainda no frontend, quanto erro oriundo do backend. Deverá estar visível somente quando houver algum erro no envio do arquivo.

**34 - Componente - *Folder*:** Componente responsável por compor o *ListFolderCustom* com componentes que representam a visualização de uma pasta e seus arquivos.

**35 - Componente - *FolderActionsMenu*:** Componente responsável por compor o *Folder* com componentes que representam a visualização de um *menu* com opções de ações relativas a pasta.

**36 - Componente - *IconButton*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* é responsável por compor o *Folder* com componentes que representam a visualização do quantitativo de arquivos presentes na pasta.

**37 - Componente - *File*:** Componente responsável por compor o *Folder* com componentes que representam a visualização de um arquivo contido na pasta.

**38 - Componente - *FileActionsMenu*:** Componente responsável por compor o *File* com componentes que representam a visualização de um *menu* com opções de ações relativas ao arquivo.

### ChatBox - Message - CitationButton



*Figura 7 – Componentização do Message com o CitationButton*

### Descrição dos componentes do ChatBox - Message - CitationButton

**39 - Componente - *MessageBox*:** Componente responsável por compor o *ChatBox* com componentes que montam a listagem de mensagens trocadas em um *chat*.

**40 - Componente - *Message*:** Componente responsável por compor o *MessageBox* com componentes que montam a visualização de uma mensagem do *chat*.

**41 - Componente - *MessageHeader*:** Componente responsável por compor o *Message* com componentes que montam a visualização das informações que montam o topo da mensagem feita pelo assistente como resposta a pergunta do usuário.

**42 - Componente - *MessageContent*:** Componente responsável por compor o *Message* com componentes que montam a visualização do conteúdo da mensagem de um *chat*.

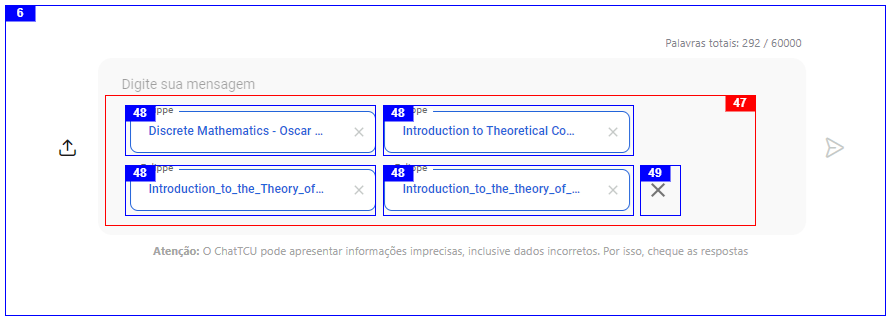
**43 - Componente - *Avatar*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* que compõe *Message* e que tem a função de identificar visualmente o autor da mensagem.

**44 - Componente - *Chip*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* que compõe *MessageHeader* e que tem a função de exibir qual ou quais bases de informação foi utilizada para gerar a resposta.

**45 - Componente - *CitationButton*:** Componente responsável por compor o *Message* com um *Button* com a ação de exibir a citação utilizada para responder.

**46 - Componente - *Actions*:** Componente responsável por compor o *Message* com componentes que montam a visualização de possíveis ações que possam ser realizadas com a mensagem, tais como copiar conteúdo da mensagem, *feedback* positivo e negativo.

### ChatBox - Input - ChipFiles



*Figura 7 – Componentização do ChatBox com Input e ChipFiles*

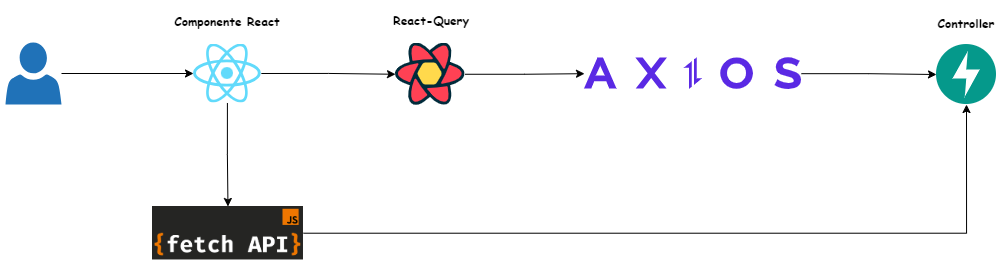
### Descrição dos componentes do ChatBox - InputBox - InputFiles

**47 - Componente - *InputFiles*:** Componente responsável por compor o *InputBox* com componentes que montam a visualização dos arquivos selecionados pelo usuário para contextualizar o assistente para a geração da resposta.

**48 - Componente - *Chip*:** Componente definido pela biblioteca *MaterialUI*, que no *frontend* que compõe *InputFiles* e que tem a função de exibir qual arquivo está selecionado para contextualizar o assistente.

**49 - Componente - *IconButton*:** Componente responsável por compor o *InputFiles* com componentes que montam a visualização de um botão para limpar a seleção de arquivos para contextualização do assistente.

### Diagrama de Arquitetura das Requisições Originadas pelo Frontend



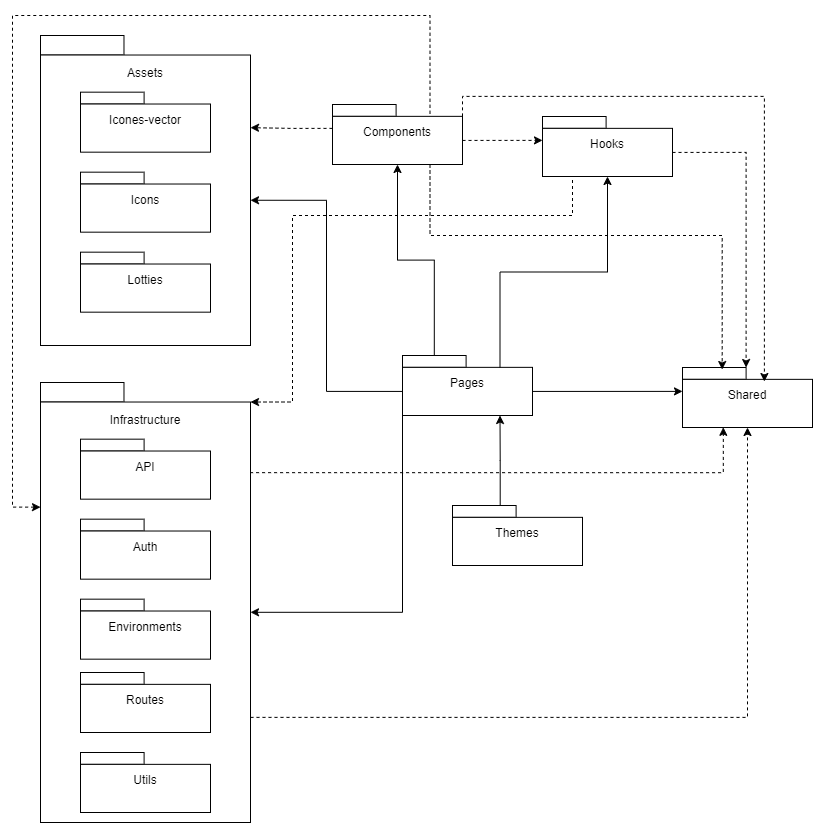
*Figura 8 – Diagrama de arquitetura das requisições realizadas pelo frontend*

### Descrição de Arquitetura das Requisições Originadas pelo Frontend

Como regra geral, todas as requisições são realizadas através da biblioteca *React-Query* da Tanstack. O uso do *React-Query* se dá pela facilidade com que a capacidade de *cache* ou política de *retry* são entregues e implementadas no *frontend*. O *React-Query* apenas gerencia as requisições, que são realizadas através da biblioteca *Axios*.

No caso de envio de prompt, ou seja interação com o assistente, a requisição é realizada através do *Fetch API* por a resposta se tratar de *stream* pelo *SSE (Server Send Events)*, visto que o *Axios*, até então, não conseguia lidar com respostas em *stream*.

## Visão de Pacotes - Frontend



*Figura 9 – Diagrama de pacotes do frontend*

### Descrição dos pacotes

***1 - Assets:*** Pacote que contém diversos recursos utilizáveis para compor a aplicação.

***1.1 - Icones-vector:*** Pacote que contém arquivos *SVG* utilizados como ícones.

***1.2 - Icons:*** Pacote que contém uma biblioteca para integrar ícones ao sistema.

***1.3 - Lotties:*** Pacote que contém arquivos *JSON* utilizados pela biblioteca *Lottie* para ligar com imagens.

***2 - Infrastructure:*** Pacote que contém os arquivos e componentes que montam ou acessam a infraestrutura.

***2.1 - API:*** Pacote que contém os arquivos e definições de acesso à *API*.

***2.2 - Auth:*** Pacote que contém os arquivos e componentes responsáveis para tratar da autenticação do usuário no SIGA.

***2.3 - Environments:*** Pacote que contém os arquivos que definem o ambiente da aplicação, como constantes.

***2.4 - Routes:*** Pacote que contém os arquivos e componentes responsáveis para definir as rotas na aplicação.

***2.5 - Utils:*** Pacote que contém os arquivos e componentes utilitários relacionados à infraestrutura do sistema.

***3 - Components:*** Pacote que contém os arquivos dos componentes da aplicação.

***4 - Pages:*** Pacote que contém os arquivos que montam telas da aplicação, tais telas são montadas com os componentes.

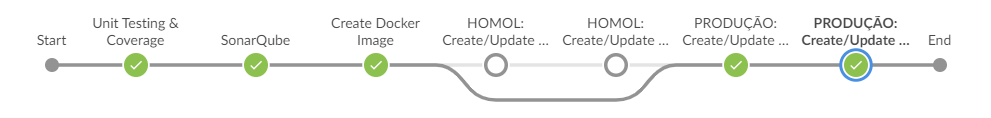
***5 - Themes:*** Pacote que contém os arquivos e componentes que definem temas visuais da aplicação, tais como *light mode* e *dark mode*.

***6 - Hooks:*** Pacote que contém os arquivos e componentes que implementam *Hooks React* que auxiliam no desenvolvimento da aplicação. *Hooks React* são na verdade componentes especiais.

***7 - Shared:*** Pacote que contém os arquivos e componentes que implementam recursos que serão compartilhados entre todos o sistema, tais como *interfaces*, *models*, *types*, *observables*, etc.

# Visão de CI/CD

## Pipeline - Backend



**1 - *Stage* - *Unit Testing & Coverage -*** Nesta fase, são executados os testes unitários para verificar se as funções ou módulos individuais do código estão funcionando corretamente. Também é gerado um relatório de cobertura de testes, indicando o percentual do código que foi testado.

**2 - *Stage* - SonarQube *-*** O código é analisado pela ferramenta SonarQube para identificar potenciais problemas relacionados à qualidade do código, como duplicações, falhas de segurança e violações de boas práticas. É uma etapa crítica para manter a saúde do código em dia.

**3 - *Stage* - *Create* Docker *Image -*** Esta fase cria uma imagem Docker da aplicação, que contém o código-fonte e todas as dependências necessárias. A imagem Docker é utilizada posteriormente para implantar a aplicação em ambientes de contêiner.

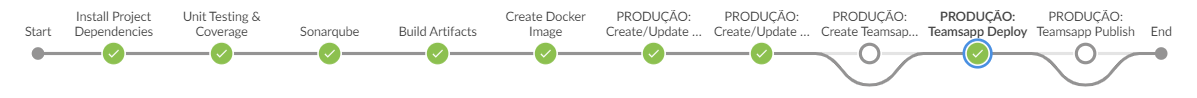
**4 - *Stage* - HOMOL: *Create/Update K8s Namespace -*** Nesta etapa, é criado ou atualizado um "*namespace*" no Kubernetes de Homologação, que é um espaço de nomes isolado onde os recursos da aplicação são organizados e gerenciados dentro de um cluster Kubernetes.

**5 - *Stage* - HOMOL: *Create/Update K8s Objects -*** Aqui, os objetos do Kubernetes de Homologação, como "*deployments*", "*services*" e "*configmaps*", são criados ou atualizados. Esses objetos são essenciais para definir como a aplicação será executada no ambiente de produção.

**6 - *Stage* - PRODUÇÃO: *Create/Update K8s Namespace -*** Nesta etapa, é criado ou atualizado um "*namespace*" no Kubernetes de Produção, que é um espaço de nomes isolado onde os recursos da aplicação são organizados e gerenciados dentro de um cluster Kubernetes.

**7 - *Stage* - PRODUÇÃO: *Create/Update K8s Objects -*** Aqui, os objetos do Kubernetes de Produção, como "*deployments*", "*services*" e "*configmaps*", são criados ou atualizados. Esses objetos são essenciais para definir como a aplicação será executada no ambiente de produção.

## Pipeline - Frontend



**1 - *Stage* - *Install Project Dependencies -*** Esta etapa instala todas as dependências do projeto, como pacotes de bibliotecas e frameworks. No caso, por se tratar de um projeto ReactJS, é executado o comando *npm ci* para garantir que todos os pacotes necessários para o projeto estejam disponíveis.

**2 - *Stage* - *Unit Testing & Coverage -*** Nesta fase, são executados os testes unitários para verificar se as funções ou módulos individuais do código estão funcionando corretamente. Também é gerado um relatório de cobertura de testes, indicando o percentual do código que foi testado. Para isso é executado o comando *npm test:ci.*

**3 - *Stage* - *SonarQube -*** O código é analisado pela ferramenta SonarQube para identificar potenciais problemas relacionados à qualidade do código, como duplicações, falhas de segurança e violações de boas práticas. É uma etapa crítica para manter a saúde do código em dia.

**4 - *Stage* - *Build Artifacts -*** Aqui, os artefatos de build são gerados. Ou seja, é a execução da compilação do projeto, onde são criados os arquivos otimizados para a produção.

**5 - *Stage* - *Create* Docker *Image -*** Esta fase cria uma imagem Docker da aplicação, que contém os códigos otimizados com todas as suas dependências. A imagem Docker é utilizada posteriormente para implantar a aplicação em ambientes de contêiner.

**6 - *Stage* - PRODUÇÃO: *Create/Update K8s Namespace -*** Nesta etapa, é criado ou atualizado um "*namespace*" no Kubernetes, que é um espaço de nomes isolado onde os recursos da aplicação são organizados e gerenciados dentro de um *cluster* Kubernetes.

**7 - *Stage* - PRODUÇÃO: *Create/Update K8s Objects -*** Aqui, os objetos do Kubernetes, como "*deployments*", "*services*" e "*configmaps*", são criados ou atualizados. Esses objetos são essenciais para definir como a aplicação será executada no ambiente de produção.

**8 - *Stage* - PRODUÇÃO: *Create* Teamsapp Docker *Image -*** Uma imagem Docker específica para o aplicativo Microsoft Teams é criada. Ela inclui a aplicação configurada para funcionar dentro do ambiente do Teams.

**9 - *Stage* - PRODUÇÃO: Teamsapp *Deploy -*** A imagem Docker gerada na etapa anterior é implantada no ambiente de produção do Microsoft Teams, tornando a aplicação disponível para uso.

**10 - *Stage* - PRODUÇÃO: Teamsapp *Publish -*** Esta é a última etapa, onde a aplicação é publicada oficialmente para os usuários dentro do Microsoft Teams, permitindo o acesso e uso da aplicação no ambiente de produção.