# Requisitos do Sistema LangNet para Criação de Aplicações Baseadas em Agentes

## 1. Estrutura do Framework LangNet

### 1.1 Fases Principais

1. Leitura e Análise de Documentação
2. Geração de Especificação Funcional
3. Definição de Agentes e Tarefas
4. Geração de Arquivos YAML
5. Modelagem de Redes de Petri
6. Geração de Código Python
7. Integração e Deployment

### 1.2 Fluxo do Processo

* Leitura de documentação → Análise de requisitos → Criação de especificação funcional → Design de agentes e tarefas → Geração de YAMLs → Modelagem de rede de Petri → Geração de código → Integração com frameworks externos → Deployment completo

## 2. Requisitos para Implementação do Sistema

### 2.1 Módulo de Leitura e Análise de Documentação

#### 2.1.1 Tipos de Documentação Suportados

* Documentos estruturados: PDFs, DOCs, apresentações, planilhas
* Documentos não-estruturados: Texto livre, emails, transcrições de reuniões
* Artefatos técnicos: Diagramas, schemas, código-fonte, documentação API

#### 2.1.2 Requisitos de Análise Documental

O sistema deve realizar automaticamente as seguintes análises:

* Extração de entidades relevantes (atores, sistemas, processos, regras)
* Identificação de requisitos funcionais e não-funcionais
* Detecção de relações e dependências entre componentes
* Reconhecimento de fluxos de trabalho implícitos e explícitos
* Identificação de regras de negócio e restrições
* Extração de terminologia específica do domínio

#### 2.1.3 Verificações Complementares

* Consulta a bases de conhecimento para enriquecer compreensão do domínio
* Verificação automática de consistência entre diferentes documentos
* Detecção e resolução de ambiguidades e contradições
* Geração de perguntas para esclarecimento quando necessário
* Priorização de requisitos baseada em análise de impacto e dependências

#### 2.1.4 Verificação de Completude

* Avaliação automática da suficiência das informações disponíveis
* Identificação de lacunas de informação críticas
* Solicitação direcionada de informações complementares
* Verificação de cobertura dos requisitos essenciais
* Avaliação comparativa com padrões de completude por tipo de aplicação

### 2.2 Módulo de Geração de Especificação Funcional

#### 2.2.1 Procedimentos Iniciais

* Consolidação estruturada de todos os requisitos identificados
* Resolução de conflitos e inconsistências detectadas
* Classificação de requisitos por categorias standard (funcional, não-funcional, interface, etc.)
* Priorização baseada em critérios configuráveis (criticidade, dependências, etc.)
* Identificação de casos de uso principais e fluxos de trabalho

#### 2.2.2 Geração de Artefatos

* Criação de documento de especificação funcional completo
* Produção de diagramas ilustrativos (fluxos, entidades, componentes)
* Elaboração de descrições detalhadas para cada funcionalidade
* Definição de critérios de aceitação mensuráveis
* Geração de glossário de termos técnicos e de domínio

#### 2.2.3 Controle de Qualidade da Especificação

* Verificação automática de consistência interna do documento
* Validação cruzada entre requisitos e especificações
* Verificação de viabilidade técnica preliminar
* Detecção de riscos potenciais e complexidades
* Avaliação de completude por funcionalidade

#### 2.2.4 Análise de Arquitetura Preliminar

O sistema deve identificar:

* Componentes e serviços necessários
* Padrões de interação entre componentes
* Requisitos de integração com sistemas externos
* Considerações de performance, segurança e escalabilidade
* Tecnologias e frameworks recomendados

### 2.3 Módulo de Definição de Agentes e Tarefas

#### 2.3.1 Identificação e Design de Agentes

* Análise funcional para determinação de agentes necessários
* Definição de responsabilidades e escopo para cada agente
* Elaboração de perfis detalhados incluindo função, objetivo e backstory
* Atribuição de ferramentas e capacidades específicas
* Definição de parâmetros comportamentais e de interação

#### 2.3.2 Design de Tarefas

* Decomposição de funcionalidades em tarefas atômicas
* Especificação de formato de dados de entrada para cada tarefa
* Detalhamento dos passos de processamento necessários
* Definição clara de saídas esperadas e formatos
* Estabelecimento de critérios de sucesso e validação

#### 2.3.3 Gestão de Dependências

* Identificação de dependências entre tarefas
* Mapeamento de pré-requisitos e pós-condições
* Controle de sequenciamento e paralelismo
* Estabelecimento de pontos de sincronização
* Gestão de recursos compartilhados

#### 2.3.4 Otimização do Conjunto de Agentes e Tarefas

* Análise de cobertura funcional
* Detecção e eliminação de redundâncias
* Balanceamento de carga entre agentes
* Ajuste fino de responsabilidades para maximizar coesão
* Verificação de completude da cobertura de requisitos

### 2.4 Módulo de Geração de Arquivos YAML

#### 2.4.1 Formatação YAML para Agentes

* Conversão de definições de agentes para formato YAML padrão
* Validação sintática e semântica dos arquivos
* Garantia de conformidade com o esquema esperado
* Otimização de formatação para legibilidade e manutenção
* Geração de identificadores únicos e consistentes

#### 2.4.2 Formatação YAML para Tarefas

* Conversão de definições de tarefas para formato YAML padrão
* Estruturação hierárquica de passos e subpassos
* Implementação de referências cruzadas entre arquivos quando necessário
* Validação de tipos de dados e consistência de formatos
* Criação de documentação embutida para clareza

#### 2.4.3 Controle de Versão e Metadados

* Inclusão de informações de versão e timestamp
* Adição de metadados como autor, fonte dos requisitos, etc.
* Rastreabilidade para requisitos originais
* Histórico de alterações e justificativas
* Anotações para facilitar manutenção futura

#### 2.4.4 Validação Integrada

* Verificação cruzada entre definições de agentes e tarefas
* Validação de referências a ferramentas e recursos externos
* Checagem de tipos e formatos de dados
* Teste de carregamento dos arquivos pelo framework
* Detecção proativa de problemas potenciais

### 2.5 Módulo de Modelagem de Redes de Petri

#### 2.5.1 Design da Rede de Petri

* Identificação de lugares (places) baseados em estados do sistema
* Definição de transições baseadas em tarefas e ações
* Estabelecimento de tokens iniciais conforme estado inicial
* Definição de arcos e pesos apropriados
* Verificação de propriedades fundamentais (limitação, vivacidade, etc.)

#### 2.5.2 Mapeamento para Estruturas JSON

* Conversão da rede para formato JSON compatível com editor
* Inclusão de metadados visuais (posições, cores, etc.)
* Armazenamento de propriedades adicionais para simulação
* Validação de conformidade com formato esperado pelo editor
* Otimização para carregamento eficiente

#### 2.5.3 Validação Matemática da Rede

* Análise de cobertura para garantir que todos os estados são alcançáveis
* Verificação de deadlocks e livelocks
* Análise de invariantes de lugar e transição
* Validação de limites de tokens para cada lugar
* Simulação preliminar para validação comportamental

#### 2.5.4 Integração com Requisitos Funcionais

* Mapeamento entre elementos da rede e requisitos originais
* Validação de completude da cobertura funcional
* Análise de caminhos críticos e bottlenecks potenciais
* Verificação de conformidade com fluxos de trabalho esperados
* Rastreabilidade bidirecional entre rede e especificação

### 2.6 Módulo de Geração de Código Python

#### 2.6.1 Implementação da Rede de Petri

* Geração de classes Python representando a estrutura da rede
* Implementação de lógica de execução e processamento de tokens
* Criação de mecanismos de monitoramento de estado
* Implementação de handlers para eventos e transições
* Suporte a checkpoint e persistência de estado

#### 2.6.2 Integração com Framework Existente

* Adaptação do código para interfaces do framework
* Utilização de adaptadores para diferentes implementações de agentes
* Configuração de sistemas de memória conforme necessidade
* Implementação de mecanismos de comunicação entre componentes
* Utilização correta de padrões e convenções do framework

#### 2.6.3 Implementação de Agentes e Tarefas

* Codificação das classes de agentes conforme definições YAML
* Implementação das tarefas com toda lógica de processamento
* Integração com ferramentas externas conforme configurado
* Configuração de parâmetros específicos e comportamentos
* Implementação de tratamento de exceções e recovery

#### 2.6.4 Código de Testes

* Geração de testes unitários para componentes individuais
* Criação de testes de integração para fluxos completos
* Implementação de casos de teste baseados em requisitos
* Configuração de fixtures e mocks necessários
* Suporte a testes automatizados em pipeline CI/CD

### 2.7 Módulo de Integração com MCP via FastMCP

#### 2.7.1 Configuração de Conexão

* Implementação de cliente FastMCP conforme especificação
* Configuração de autenticação e segurança
* Estabelecimento de canais de comunicação
* Implementação de mecanismos de retry e circuit-breaker
* Monitoramento de saúde da conexão

#### 2.7.2 Sincronização de Estados

* Mapeamento entre modelos locais e formatos MCP
* Implementação de protocolos de sincronização
* Resolução de conflitos e consistência eventual
* Otimização de transferência de dados
* Persistência local para operação offline

#### 2.7.3 Consumo e Exposição de Serviços

* Descoberta dinâmica de serviços disponíveis
* Implementação de clientes para APIs remotas
* Exposição de funcionalidades locais como serviços
* Registro e desregistro automático de endpoints
* Versionamento e backward compatibility

### 2.8 Módulo de Monitoramento via Langfuse

#### 2.8.1 Instrumentação de Código

* Implementação de decoradores e wrappers para pontos de observabilidade
* Configuração de níveis de detalhe ajustáveis
* Captura de métricas de performance e utilização
* Rastreamento de fluxos entre componentes
* Monitoramento específico de operações de LLM

#### 2.8.2 Integração com Dashboard

* Configuração de conexão com API Langfuse
* Estruturação de projetos e ambientes
* Implementação de envio de telemetria
* Configuração de retenção e agregação de dados
* Suporte a tags e filtros personalizados

#### 2.8.3 Alertas e Notificações

* Definição de condições de alerta configuráveis
* Implementação de canais de notificação (email, Slack, etc.)
* Suporte a políticas de escalonamento
* Agregação inteligente para redução de ruído
* Integração com sistemas de on-call

### 2.9 Módulo de Frontend

#### 2.9.1 Interfaces Streamlit

* Geração de código Streamlit para interfaces interativas
* Implementação de componentes para visualização de estado
* Criação de formulários para configuração e parametrização
* Suporte a upload e download de artefatos
* Visualização de logs e resultados

#### 2.9.2 Componentes React

* Desenvolvimento de componentes React reutilizáveis
* Implementação de visualizações avançadas (grafos, dashboards)
* Suporte a temas e responsividade
* Integração com APIs backend
* Otimização de performance e acessibilidade

#### 2.9.3 Visualização de Redes de Petri

* Implementação de renderização interativa da rede
* Suporte a zoom, pan e seleção
* Animação de fluxo de tokens em tempo real
* Destaque de estados ativos e transições
* Ferramentas de análise e debugging visual

#### 2.9.4 Interface de Chat e Controle

* Implementação de interface de chat com agentes
* Painéis de controle para monitoramento de agentes
* Visualização de métricas e performance
* Ferramentas de debugging e introspection
* Controles para intervenção manual quando necessário

### 2.10 Módulo de Backend Flask

#### 2.10.1 API REST

* Implementação de endpoints seguindo padrões REST
* Configuração de autenticação e autorização
* Documentação automática via Swagger/OpenAPI
* Versionamento de API
* Implementação de rate limiting e quotas

#### 2.10.2 Orquestração de Agentes

* Implementação de serviço de gerenciamento de agentes
* Suporte a inicialização, monitoramento e finalização
* Balanceamento de carga e pooling
* Isolamento e segurança
* Logging detalhado de atividades

#### 2.10.3 Gestão de Sessões e Filas

* Implementação de sistema de sessões persistentes
* Configuração de filas de processamento assíncrono
* Mecanismos de priorização e scheduling
* Suporte a retry e dead-letter queue
* Persistência e recuperação de estado

#### 2.10.4 Websockets

* Implementação de conexões bidirecionais
* Suporte a canais e rooms para segregação
* Transmissão eficiente de atualizações em tempo real
* Handling de reconexões e backpressure
* Autenticação e autorização específicas

## 3. Requisitos Técnicos e Legais

### 3.1 Base Tecnológica para Implementação

* Python 3.9+ como linguagem principal
* Framework de agentes (CrewAI, LangChain, etc.)
* Suporte a LLMs (OpenAI, Claude, Llama, etc.)
* Frameworks de frontend (React, Streamlit)
* Flask para backend API

### 3.2 Requisitos de Segurança e Privacidade

* Controle de acesso baseado em papéis
* Criptografia em trânsito e em repouso
* Sanitização de prompts e outputs
* Proteção contra injeção de prompt
* Compliance com regulações aplicáveis (GDPR, LGPD, etc.)

### 3.3 Requisitos de Performance e Escalabilidade

* Processamento assíncrono para operações longas
* Caching inteligente de resultados
* Otimização de chamadas para LLMs
* Suporte a processamento distribuído
* Estratégias para lidar com rate limiting de APIs externas

### 3.4 Integrações de Sistema

* Conexão com provedores de LLM (OpenAI, Anthropic, etc.)
* Integração com sistemas de armazenamento (S3, GCS, etc.)
* APIs de ferramentas externas conforme necessidade
* Servidores MCP via FastMCP
* Langfuse para telemetria e observabilidade

## 4. Considerações para Implementação

* Equilíbrio entre automação e supervisão humana
* Transparência nos algoritmos e processamento
* Mecanismos de feedback para melhoria contínua
* Documentação detalhada de todos os componentes
* Extensibilidade para novas funcionalidades
* Testes abrangentes para garantir confiabilidade
* Suporte a diferentes domínios de aplicação
* Mecanismos de fallback em caso de falha

A implementação do sistema LangNet permitirá a criação rápida e confiável de aplicações baseadas em agentes inteligentes, com fluxos de trabalho bem definidos via redes de Petri e interfaces modernas. O sistema deve ser flexível o suficiente para atender a diferentes domínios, mantendo altos padrões de segurança, desempenho e usabilidade.

## Detalhamento do fluxo do processo

## 1. Leitura e Análise de Documentação

### 1.1 Ingestão de Documentos

* Descrição: Capacidade de carregar e processar diversos formatos de documentação (PDF, DOC, TXT, Markdown)
* Detalhes: O sistema deve permitir o upload de múltiplos documentos de requisitos, documentação técnica, descrições de domínio e regras de negócio.
* Comportamento: Analisar a estrutura do documento, identificar seções relevantes, extrair conteúdo formatado.

### 1.2 Compreensão Contextual

* Descrição: Análise semântica profunda para entender o contexto e domínio do problema
* Detalhes: Utilização de LLMs para interpretar requisitos em linguagem natural, identificar ambiguidades e resolver dúvidas com perguntas direcionadas.
* Comportamento: Criar um mapa conceitual do domínio, identificar terminologia específica, abstrair conceitos-chave.

### 1.3 Extração de Requisitos Estruturados

* Descrição: Transformação de texto livre em requisitos estruturados
* Detalhes: Identificar e categorizar requisitos funcionais e não-funcionais, identificar dependências entre requisitos, priorizar informações.
* Comportamento: Gerar uma lista estruturada de requisitos com metadados (prioridade, dependências, origem).

## 2. Geração de Especificação Funcional

### 2.1 Síntese de Especificação

* Descrição: Criação de documento de especificação funcional completo e coerente
* Detalhes: Consolidar requisitos, resolver contradições, preencher lacunas com inferências lógicas, estruturar em formato padronizado.
* Comportamento: Produzir documento detalhado com descrições de funcionalidades, fluxos, entidades e regras.

### 2.2 Modelagem de Dados e Entidades

* Descrição: Identificação das entidades e relacionamentos do sistema
* Detalhes: Extrair modelos de dados implícitos nos requisitos, definir atributos, relacionamentos e restrições.
* Comportamento: Gerar modelos conceituais de dados e diagramas de entidade-relacionamento.

### 2.3 Definição de Fluxos de Trabalho

* Descrição: Mapeamento dos fluxos de processamento necessários
* Detalhes: Identificar sequências de operações, pontos de decisão, loops e condições específicas.
* Comportamento: Produzir descrições detalhadas de fluxos com entradas, saídas e condições de transição.

## 3. Definição de Agentes e Tarefas

### 3.1 Identificação de Agentes Necessários

* Descrição: Determinar quais agentes especializados são necessários para o sistema
* Detalhes: Analisar requisitos e fluxos para identificar áreas de especialização, mapear capacidades necessárias.
* Comportamento: Produzir lista de agentes com funções específicas, objetivos e conhecimento necessário.

### 3.2 Definição de Perfil dos Agentes

* Descrição: Elaborar definições detalhadas para cada agente
* Detalhes: Criar descrições de função (role), objetivo (goal), história de fundo (backstory) e ferramentas (tools).
* Comportamento: Gerar texto em formato compatível com o formato YAML de definição de agentes.

### 3.3 Identificação de Tarefas

* Descrição: Determinar tarefas específicas necessárias para executar os fluxos
* Detalhes: Decompor fluxos em tarefas atômicas, definir entradas e saídas, detalhar passos de processamento.
* Comportamento: Produzir lista de tarefas com descrições, etapas, entradas e saídas esperadas.

### 3.4 Definição Detalhada de Tarefas

* Descrição: Elaborar definições completas para cada tarefa
* Detalhes: Especificar formato de dados de entrada, passos de processamento detalhados e formato de saída esperado.
* Comportamento: Gerar texto em formato compatível com o formato YAML de definição de tarefas.

## 4. Geração de Arquivos YAML

### 4.1 Formatação de Arquivos de Agentes

* Descrição: Criação de arquivo agents.yaml seguindo o formato específico
* Detalhes: Estruturar definições de agentes no formato YAML, garantir sintaxe correta, incluir todos os metadados.
* Comportamento: Produzir arquivo YAML bem-formado contendo todas as definições de agentes.

### 4.2 Formatação de Arquivos de Tarefas

* Descrição: Criação de arquivo tasks.yaml seguindo o formato específico
* Detalhes: Estruturar definições de tarefas no formato YAML, garantir sintaxe correta, validar referências cruzadas.
* Comportamento: Produzir arquivo YAML bem-formado contendo todas as definições de tarefas.

## 5. Modelagem de Redes de Petri

### 5.1 Definição da Estrutura da Rede

* Descrição: Modelar o fluxo de trabalho como uma rede de Petri
* Detalhes: Identificar lugares (places), transições (transitions), tokens iniciais e condições de disparo (firing).
* Comportamento: Criar estrutura matemática completa da rede com todas as conexões e condições.

### 5.2 Mapeamento de Agentes para Transições

* Descrição: Associar agentes às transições da rede
* Detalhes: Determinar qual agente é responsável por cada transição, definir parâmetros de execução.
* Comportamento: Documentar associações entre transições e agentes com configurações específicas.

### 5.3 Geração de Representação JSON

* Descrição: Converter o modelo da rede em formato JSON para o editor
* Detalhes: Estruturar a rede completa no formato esperado pelo editor de redes de Petri.
* Comportamento: Produzir arquivo JSON contendo a definição completa da rede de Petri.

## 6. Geração de Código Python

### 6.1 Implementação da Rede de Petri

* Descrição: Geração de código Python que implementa a rede de Petri
* Detalhes: Criar classes e métodos que representam os elementos da rede, implementar lógica de execução.
* Comportamento: Produzir código Python funcional usando o framework fornecido.

### 6.2 Integração com Framework

* Descrição: Adaptar o código para utilizar o framework existente
* Detalhes: Utilizar as interfaces e classes do framework para implementar a rede, garantir compatibilidade.
* Comportamento: Integrar o código gerado com adaptadores e classes fornecidas pelo framework.

### 6.3 Geração de Código de Testes

* Descrição: Criar testes automatizados para validar o sistema
* Detalhes: Implementar testes unitários e de integração que validam o comportamento esperado.
* Comportamento: Produzir código de teste que verifica o funcionamento correto da aplicação.

## 7. Documentação e Entrega

### 7.1 Geração de Documentação Técnica

* Descrição: Criar documentação detalhada sobre o sistema implementado
* Detalhes: Documentar arquitetura, componentes, fluxos e decisões de design.
* Comportamento: Produzir documentação técnica completa em formato Markdown ou HTML.

### 7.2 Geração de Instruções de Execução

* Descrição: Criar instruções para execução e uso do sistema
* Detalhes: Documentar requisitos, passos de instalação, configuração e exemplos de uso.
* Comportamento: Produzir manual de usuário e guia de instalação detalhados.

## 8. Integração com Servidores MCP via FastMCP

### 8.1 Configuração de Conexão FastMCP

* Descrição: Configurar e estabelecer conexões com servidores MCP utilizando o protocolo FastMCP
* Detalhes: Implementar cliente FastMCP, gerenciar autenticação, estabelecer canais de comunicação seguros
* Comportamento: Criar e manter conexões estáveis, lidar com reconexões automáticas e balanceamento de carga

### 8.2 Sincronização de Estados

* Descrição: Sincronizar estados da rede de Petri com o servidor MCP
* Detalhes: Mapear estados locais para o formato MCP, implementar protocolo de sincronização bidirecional
* Comportamento: Garantir consistência entre estados locais e remotos, resolver conflitos de forma determinística

### 8.3 Consumo de Serviços MCP

* Descrição: Utilizar serviços disponíveis nos servidores MCP
* Detalhes: Descobrir serviços disponíveis, consumir APIs remotas, integrar funcionalidades externas
* Comportamento: Enviar requisições formatadas corretamente, processar respostas e tratar erros específicos do protocolo

### 8.4 Publicação de Serviços para MCP

* Descrição: Expor funcionalidades locais como serviços para o servidor MCP
* Detalhes: Definir interfaces de serviço, implementar endpoints compatíveis, registrar serviços no diretório MCP
* Comportamento: Receber e processar requisições externas, fornecer respostas no formato esperado

## 9. Integração com Monitoramento via Langfuse

### 9.1 Instrumentação de Código

* Descrição: Adicionar instrumentação para monitoramento em pontos-chave do sistema
* Detalhes: Implementar rastreamento de execução em agentes, tarefas e transições da rede de Petri
* Comportamento: Capturar métricas de desempenho, tempo de execução e uso de recursos sem impacto significativo

### 9.2 Configuração de Traços (Traces)

* Descrição: Implementar traços distribuídos para acompanhar fluxos completos
* Detalhes: Criar spans aninhados, propagar contexto entre componentes, adicionar metadados relevantes
* Comportamento: Produzir traços hierárquicos que representam a execução completa de um fluxo de trabalho

### 9.3 Captura de Observabilidade LLM

* Descrição: Monitorar especificamente as operações dos LLMs no sistema
* Detalhes: Capturar prompts, respostas, tokens utilizados, e metadados de cada interação com LLMs
* Comportamento: Registrar todas as chamadas de LLM com contexto completo para análise e otimização

### 9.4 Integração com Dashboard Langfuse

* Descrição: Enviar dados coletados para dashboard Langfuse
* Detalhes: Configurar API keys, implementar cliente de envio, definir estrutura de projetos/ambientes
* Comportamento: Transmitir dados de telemetria em tempo real ou em lotes conforme configuração

### 9.5 Alertas e Notificações

* Descrição: Configurar sistema de alertas baseados em métricas e eventos
* Detalhes: Definir thresholds, condições de alerta, canais de notificação e políticas de escalonamento
* Comportamento: Gerar notificações apropriadas quando condições anômalas forem detectadas

## 10. Geração de Frontend em Streamlit e/ou React

### 10.1 Design de Interface Streamlit

* Descrição: Gerar código para interface de usuário usando Streamlit
* Detalhes: Criar componentes para interação com fluxos, visualização de estados, upload de documentos
* Comportamento: Produzir código Python com widgets Streamlit organizados logicamente por funcionalidade

### 10.2 Componentes React Avançados

* Descrição: Criar componentes React para interfaces mais complexas ou personalizadas
* Detalhes: Desenvolver componentes interativos, visualizações avançadas, e interfaces responsivas
* Comportamento: Gerar código JSX/TSX organizado em componentes reutilizáveis com gerenciamento de estado

### 10.3 Visualização de Rede de Petri

* Descrição: Implementar visualização interativa da rede de Petri
* Detalhes: Criar representação gráfica com nós, arestas, tokens e animações de transição
* Comportamento: Renderizar visualização dinâmica do estado atual da rede, destacando transições ativas

### 10.4 Painel de Controle de Agentes

* Descrição: Desenvolver interface para monitoramento e controle de agentes
* Detalhes: Criar visualizações de status, logs, atividade e métricas de desempenho por agente
* Comportamento: Exibir informações em tempo real sobre cada agente, permitir interação direta

### 10.5 Interface de Chat com Agentes

* Descrição: Criar interface de chat para interação direta com agentes
* Detalhes: Implementar área de mensagens, histórico de conversas, seleção de agentes específicos
* Comportamento: Permitir comunicação natural com agentes, exibir respostas formatadas adequadamente

## 11. Backend de Agentes em Flask

### 11.1 Arquitetura API REST

* Descrição: Desenvolver estrutura de API REST para acesso aos agentes e fluxos
* Detalhes: Definir endpoints, métodos HTTP, formatos de requisição/resposta, autenticação e autorização
* Comportamento: Expor funcionalidades do sistema através de API bem documentada e segura

### 11.2 Orquestração de Agentes

* Descrição: Implementar serviço de orquestração para gerenciar ciclo de vida dos agentes
* Detalhes: Criar mecanismos para inicialização, monitoramento, balanceamento de carga e finalização de agentes
* Comportamento: Gerenciar pool de agentes de forma eficiente, otimizando recursos conforme demanda

### 11.3 Gestão de Sessões

* Descrição: Desenvolver sistema de gestão de sessões para interações persistentes
* Detalhes: Implementar armazenamento de estado de sessão, timeout, recuperação e isolamento
* Comportamento: Manter contexto consistente entre múltiplas interações com o mesmo usuário/sistema

### 11.4 Sistema de Filas

* Descrição: Criar sistema de filas para processamento assíncrono de tarefas
* Detalhes: Implementar produtor/consumidor, priorização, retry e dead-letter queues
* Comportamento: Processar solicitações de forma ordenada e resiliente, especialmente para tarefas longas

### 11.5 Websockets para Atualizações em Tempo Real

* Descrição: Implementar comunicação bidirecional via Websockets
* Detalhes: Configurar canais, gerenciar conexões, implementar protocolos de atualização em tempo real
* Comportamento: Fornecer atualizações instantâneas sobre progresso, estados e resultados parciais

### 11.6 Documentação de API com Swagger/OpenAPI

* Descrição: Gerar documentação interativa da API
* Detalhes: Definir especificação OpenAPI, adicionar descrições, exemplos e schemas
* Comportamento: Produzir documentação navegável que permite teste direto de endpoints

## 12. Integração e Deployment

### 12.1 Containerização com Docker

* Descrição: Gerar configuração para containerização do sistema completo
* Detalhes: Criar Dockerfiles, docker-compose, configurações de rede e volumes
* Comportamento: Produzir ambiente containerizado completo com todos os componentes integrados

### 12.2 Pipeline de CI/CD

* Descrição: Configurar pipeline para integração contínua e deployment
* Detalhes: Implementar workflows para testes, builds, análise estática e deployment automatizado
* Comportamento: Gerar configurações para GitHub Actions, GitLab CI, ou ferramentas similares

### 12.3 Ambiente de Desenvolvimento Local

* Descrição: Criar configuração para ambiente de desenvolvimento local
* Detalhes: Configurar ferramentas de desenvolvimento, dependências, variáveis de ambiente
* Comportamento: Gerar scripts e documentação para setup rápido de ambiente de desenvolvimento

### 12.4 Integração com Serviços de Nuvem

* Descrição: Configurar integração com provedores de nuvem
* Detalhes: Implementar suporte para AWS, Azure, GCP ou outros provedores, configurar serviços específicos
* Comportamento: Gerar templates de infraestrutura como código (Terraform, CloudFormation, etc.)

Estas funcionalidades expandidas complementam o fluxo original, adicionando capacidades de integração externa, monitoramento avançado e interfaces de usuário, além de um backend robusto para suportar o sistema completo.