

RPG.AI

Equipe rpg.ai

<https://github.com/Dhayson/rpg.ai>

Matheus Pessoa (mop2) - GP

Diego Rocha (djaar)

Recife, 13/08/2025

Sumário

1. Introdução	
2. Metodologia	
3. Documentação da Execução do Projeto	2
3.1 Imersão	
3.2 Ideação	
3.3 Produção	
3.4 Validação	
4. Discussões Técnicas e Estratégicas	
5. Considerações Éticas	
6. Lições Aprendidas e Reflexões Finais	
7. Perspectivas Futuras	
8. Referências	
9. Apêndices	

1. Introdução

Como ponto de partida, buscamos fazer uma aplicação, principalmente, que fosse divertida e interessante para nós e que gostaríamos de usar

Assim, dentro da temática de RPG, investigamos as dores, barreiras e desafios enfrentados, principalmente por iniciantes ou pessoas querendo entrar nesse mundo. Dessa forma, buscamos criar um sistema para que pessoas possam ter aventuras de RPG com baixíssima barreira de entrada.

Por isso, optamos por criar um agente especialista em RPG, integrado na plataforma Discord, que é onde se encontra em grande parte nosso público alvo.

- **Objetivos Gerais:** Democratizar o acesso ao RPG, capacitando jogadores a mestrear suas próprias partidas de forma intuitiva, sem a necessidade de conhecimento prévio aprofundado. Nossa solução visa eliminar a barreira de entrada, permitindo que mais pessoas desfrutem da experiência de jogo.
- **Objetivos Específicos:**
 - Desenvolver um agente de IA co-mestre para o Discord, que ofereça suporte em tempo real para a condução de partidas.
 - Capacitar o agente a responder a dúvidas, gerar ideias para campanhas e gerenciar a narrativa de forma coerente.
 - Utilizar técnicas de engenharia de prompts e arquitetura de agentes para garantir a consistência do contexto da campanha e mitigar alucinações do modelo."

Partindo da abordagem AIDesign, alinhamos nossa estratégia para a construção de uma prova de conceito simples, porém funcional e que valide nossa tese sobre o valor de nosso produto: desejável, viável, factível e defensável.

Com isso, construímos um agente especialista em RPG e capaz de responder às interações dos jogadores e, juntamente com eles, desenvolver uma aventura. Dessa forma, o sistema é melhor utilizado como um co-piloto para os jogadores evoluírem sua aventura de forma rápida, acessível e dinâmica.

Justificamos o uso de LLM pois essa é a tecnologia ideal para a geração de textos e, aliado ao sistema de agente, ser capaz de gerenciar uma partida de RPG. De fato, o modelo de linguagem é fundamental para que nossa solução automatizada seja factível e viável.

2. Metodologia

A metodologia de trabalho adotada pela equipe foi uma abordagem colaborativa e adaptativa, orientada pela comunicação constante e flexibilidade. Em vez de seguir um framework formal (como Scrum ou Kanban), o processo foi construído de forma orgânica, priorizando o progresso contínuo e a resolução de desafios à medida que surgiam.

A principal característica dessa metodologia foi a divisão dinâmica de tarefas, onde os integrantes se revezavam nas responsabilidades, garantindo que o conhecimento fosse compartilhado e que o trabalho avançasse de forma equilibrada.

Aplicamos a metodologia AI Design no sentido de focar no aspecto de geração de valor de nossa solução.

Assim, podemos resumir as quatro fases:

- **Exposição**

Buscando entender e empatizar com as dores dos usuários, desenhamos uma solução com foco principal na geração de valor dentro do domínio de RPG.

Desse modo, levantamos os princípios chave do Sinfonia:

- **Desejável:** Buscamos resolver as dores de jogadores novatos ou inexperientes de RPG, em especial na dificuldade de encontrar alguém para mestrar partidas de forma acessível e a entender esse ramo que em geral exige muito tempo, dedicação e conhecimento aprofundado. Por isso, um agente especialista é o ideal para auxiliar esses jogadores a ter uma experiência acessível e divertida. Disponibilizado na plataforma Discord para ter o mínimo de barreira de entrada para nosso público alvo.
- **Viável:** Nossa solução integra diretamente com provedores de LLMs, logo é favorecido pela eficiência em escala desses grandes provedores, o que permite custos reduzidos e, assim, viabiliza economicamente nossa solução.
- **Factível:** Os *Lange Language Models* (LLMs) no estado-da-arte possibilitam a chamada de funções, de modo a possibilitar a construção de sistemas agentes. Nesse sentido, o Gemini Flash 2.5, o modelo em que focamos para validar nossa solução, oferece o melhor equilíbrio entre custo, velocidade, praticidade e performance.

- **Defensável:** As conversas são restritas a um canal do servidor, de forma que não há vazamentos de informações confidenciais entre servidores. Os dados utilizados são fornecidos pelo sistema RAG e pelos usuários, sendo compartilhados apenas conosco e com o provedor da API de LLM. Os usuários são informados e consentem com isso, segundo as normas da LGPD.

- **Composição**

Desenhemos componentes de IA tendo como foco obter o produto que queremos usar, de forma simples e concisa. Assim, identificamos a necessidade de ter um sistema de agente para gerenciar os diversos sistemas dentro do RPG e um sistema RAG para retornar informações do sistema D&D conforme necessário.

Finalmente, decidimos utilizar o Redis para manutenção do contexto.

- **Ensaio**

Experimentamos com engenharia de prompt, engenharia de contexto e com a arquitetura do sistema de agente. Dessa forma, buscamos implementar as features e ferramentas essenciais para validar nossa ideia de solução.

Demos ao agente as capacidades especiais de gerenciar a história, rolar dados e gerenciar o início da partida. Além de seu comportamento usual de dar continuidade a conversa, evoluindo a partida de RPG junto dos jogadores.

Também experimentamos com um sistema RAG para responder perguntas dos usuários sobre D&D, o que é de grande valor para usuários novatos buscando entender pela primeira vez como seguir a partida.

- **Ressonância**

Aprendemos muito sobre a criação de agentes e as capacidades, e limitações, dos modelos generativos de linguagem atuais. Foi necessária bastante experimentação e tentativa e erro para obter os resultados que desejávamos.

Gestão do projeto

Buscamos alinhar a gestão com nossas necessidades de desenvolvimento e exploração e experimentação na construção da solução.

Como ferramentas de apoio, utilizamos um grupo do Whatsapp para comunicação e o sistema de versionamento Git, dividindo as features em desenvolvimento em branches na abordagem Git Flow.

A gestão do projeto foi pautada por uma comunicação síncrona e assíncrona robusta e direta.

Comunicação Assíncrona (WhatsApp): Utilizada para trocas rápidas de informações, compartilhamento de ideias e alinhamento de tarefas fora do horário de aula.

Comunicação Síncrona (Presencial): Momentos de discussão e planejamento face a face na universidade, essenciais para a tomada de decisões estratégicas e para a resolução de problemas mais complexos.

Essa estratégia permitiu um ciclo rápido de feedback e decisão, mantendo o projeto sempre em movimento.

Etapas da execução do projeto

- **Imersão**
 - Discussão para o domínio do projeto
 - Identificação das principais dores dos jogadores de RPG
- **Ideação**
 - Brainstorm e conversas para elaboração do que é essencial na solução
 - Pesquisa por tecnologias: modelos de linguagem, agentes, RAG
 - Estudo das APIs, engenharia de prompt, engenharia de contexto e arquitetura de agentes
- **Produção**
 - Criação do repositório Git
 - Obtenção das chaves de API do Discord e do Google AI
 - Criar chat bot básico no Discord
 - Elaborar prompts para geração de texto na temática RPG
 - Criar funcionalidades de RPG
 - Desenvolver sistema RAG para retornar informações importantes sobre D&D
 - Desenvolvimento de sistema agente para se conectar as funcionalidades de RPG
 - Engenharia de prompt para precisão nas respostas
 - Engenharia de contexto do agente para um melhor funcionamento, corrigindo bugs e alucinações
 - Criação de comandos específicos e ajuda ao usuário
 - Integração do agente com o sistema RAG
 - Uso do Redis para manutenção do contexto
- **Validação**
 - Interação com o sistema pelo Discord
 - Teste de todas as funcionalidades em variados contextos



3. Documentação da Execução do Projeto

3.1 Imersão

Canvas de Identificação do Domínio (resumo)

O domínio de aplicação da solução está focado no nicho de jogos de RPG de mesa. O escopo foi delimitado para atender jogadores e mestres iniciantes, oferecendo suporte para a criação e condução de campanhas em um formato de texto interativo. Conhecendo o público, sabemos que o uso de Discord é bastante comum, então decidimos usá-lo como FEaaS (Frontend as a Service)

Persona Principal:

Nome: Lucas Martins

Idade: 29 anos

Nível de conhecimento em RPG: Iniciante

Plataformas favoritas: YouTube, Discord, Instagram

Dispositivos: Notebook com Windows e celular Android

Motivações

- Quer explorar mundos imaginários e viver histórias épicas.
 - Sempre teve curiosidade sobre D&D, mas acha que é complexo demais para começar.
 - Gosta da ideia de jogar com amigos online, mas não conhece mestres disponíveis.
 - Já assistiu vídeos do Cellbit e Critical Role, mas sente que falta alguém para guiar o grupo dele.
 - Tem vontade de usar IA para facilitar as coisas, mas não tem habilidades técnicas avançadas.
-

Dores e Problemas

- Não sabe como começar uma campanha ou criar um personagem.
 - Acha os livros de regras difíceis de ler e muito longos/chatos.
 - Tem receio de fazer algo errado como jogador.
 - Falta de tempo para estudar as regras e para preparar sessões.
 - Dificuldade em manter os amigos engajados em sessões semanais.
-

Objetivos

- Jogar D&D de maneira divertida, leve e sem complicações.
 - Ser capaz de organizar uma sessão com os amigos sem depender de um mestre humano.
 - Criar histórias e personagens sem precisar decorar regras.
 - Usar ferramentas tecnológicas para automatizar o que for possível.
-

Comportamentos

- Prefere aprender jogando, não estudando.
 - Valoriza experiências bem desenhadas (UX/UI), é sensível ao visual. Gosta de soluções que funcionam out-of-the-box, sem configurações complicadas.
 - É ativo no Discord.
 - Costuma assinar ferramentas digitais se sentir que agregam valor real.
-

Frases que ele diria

- “Sempre quis jogar RPG, mas nunca tive um mestre.”
“Tenho vontade, mas parece que você precisa estudar um livro inteiro.”
 - “Se tivesse um jeito automático de narrar, eu jogava toda semana com a galera.”
 - “Nenhum dos meus amigos sabe mexer.”
-



Como o rpg.ai pode ajudar

Função do Produto

Valor Entregue para Lucas

IA gera enredo com base em sugestões simples	Ele não precisa criar toda a narrativa sozinho
Criação automática de personagens com fichas simplificadas	Poupa tempo e reduz ansiedade de "fazer errado"
Narração fluida com texto	Traz imersão sem precisar de um mestre experiente
Explicação contextual das regras durante o jogo	Ajuda ele a aprender jogando, sem atrito
Tirar dúvidas de todo o grupo	Ajuda a tirar não só as dúvidas do mestre, mas também dos players
Integração com Discord	Se encaixa nas ferramentas que ele já usa

Por isso, acreditamos que essa Persona captura as dores que queremos solucionar e, logo, naturalmente trás o contexto de nosso público alvo.

Mapeamento de fontes de dados

Usamos como fontes de dados

- O sistema RAG: Manual do Jogador D&D
- O contexto a partir do input dos usuários em um canal do Discord
- O modelo Gemini Flash 2.5 disponibilizado pela API do Google AI

O uso de dados respeita a segurança e privacidade dos usuários. O usuário é informado sobre quais dados são coletados e com quem são compartilhados, conosco e com o provedor de LLM. Dessa forma, respeitamos as normas da LGPD quanto ao uso dos dados.

Canvas de Objetivos de Projeto

Objetivo Estratégico Geral:

- Sistema especialista em RPG integrado com Discord, usando de ferramentas para gerenciar o contexto da sessão de RPG, supervisionando a tomada de decisões ao ministrar.

Objetivos Estratégicos Secundários:

- Boa formatação das mensagens lidas e escritas no canal do Discord. O modelo deve entender o contexto de um histórico de mensagens.
- Conversas fluidas e criação de aventuras divertidas e engajantes. Os usuários devem passar vários minutos em uma aventura.
- Isolamento e segurança das informações dos usuários.
- Uso de estratégias para recuperar informações importantes e gerenciar o contexto para uma boa performance e eficiência. A história deve se manter coerente pela aventura.
- Sistema de perguntas e respostas para o jogador sobre D&D.
- Tutorial, ajuda e suporte aos usuários.

Objetivos Terciários:

- Elaborar cenários e seções pré prontos para serem jogados.
- Uso do chat de voz para realização da partida.

Matriz de Impacto/Esforço:

Impacto/Esforço	Baixo	Médio	Alto Esforço
Alto	-	A: Criar app no discord para co-mestrar as partidas RPG	C: Plataforma completa com integracao com voz para gerenciar e mestrar a partida
Médio	-	-	-
Baixo	B: GPT (no proprio chatgpt) especialista em RPG	-	-

3. Resultados-Chave Esperados (Quantitativos)

- **Aumento da taxa de retenção de usuários:** Elevar a taxa para 60% em 6 meses, garantindo que a maioria dos jogadores continue a usar o bot semanalmente.
- **Redução no tempo de consulta de regras:** Diminuir o tempo médio de resposta do Sistema RAG de 5 segundos para 2 segundos em 3 meses, para um fluxo de jogo mais ágil.
- **Aumento no uso de ferramentas:** Elevar o uso das ferramentas agentic, como **JogarD20** e **ExpandirHistoria**, em 40% em 4 meses para tornar as sessões de RPG mais interativas e dinâmicas.

4. Indicadores-Chave de Sucesso (KPIs)

- Taxa de Retenção de Usuários: Mede o engajamento e a utilidade a longo prazo.
- Tempo Médio de Resposta do Sistema RAG: Avalia a eficiência na entrega de informações sobre regras.
- Taxa de Sucesso das Ferramentas: Mede a precisão e a confiança do agente na execução de tarefas.
- Taxa de Feedback Positivo: Indica a satisfação geral do usuário com as interações do bot.

.

5. Requisitos Estratégicos e Restrições

- A solução deve estar em **conformidade com os termos de serviço da Discord e as políticas da API do Google Gemini**.
- O projeto deve ter um **orçamento de consumo de API sustentável**, com monitoramento constante para evitar custos inesperados.
- É necessária a **manutenção da licença do conteúdo de D&D** (no caso, a OGL) para garantir o uso legal das regras.
- A **arquitetura modular** deve ser mantida para permitir a fácil integração de novos modelos de LLM e ferramentas no futuro.

6. Priorização de Objetivos

- **Alta Prioridade:** Reduzir o tempo de consulta de regras para manter o fluxo do jogo ininterrupto.
- **Média Prioridade:** Aumentar o uso das ferramentas agentic para tornar as sessões de RPG mais interativas e ricas.
- **Baixa Prioridade:** Adicionar suporte a outros sistemas de RPG, como Pathfinder, para expandir a base de usuários.

7. Ações e Recursos Necessários

- **Ação:** Configurar um sistema de monitoramento de custos da API do Google Gemini.
- **Ação:** Desenvolver a ferramenta `ExpandirHistoria` para criar um protótipo inicial de criação de mundo.
- **Recurso:** Equipe de desenvolvimento (2 engenheiros) com conhecimento em Python e APIs de LLMs.
- **Recurso:** Orçamento para licenciamento e consumo de API de IA.

3.2 Ideação

Canvas de Ideação de Soluções

1. Problema a Ser Resolvido e Solução

- Problema: Falta de acesso a partidas de RPG devido ao alto nível de complexidade.
- Solução: Agente de IA vai ser um co-mestre da partida, ajudando o jogador mestre a desenvolver cenários, falas, tirar dúvidas e de maneira geral, possibilitar o desenvolvimento da partida de maneira fluída.

2. Ideias de Solução (Brainstorming)

**A: Criar app no discord para co-mestrar/mestrar as partidas RPG.
(escolhido)**

B: GPT (no proprio chatgpt) especialista em RPG

C: Plataforma completa com integracao com voz para gerenciar e mestrar a partida

Impacto/Esforço	Baixo	Médio	Alto Esforço
Alto	-	A: Criar app no discord para co-mestrar as partidas RPG	C: Plataforma completa com integracao com voz para gerenciar e mestrar a partida
Médio	-	-	-
Baixo	B: GPT (no proprio chatgpt) especialista em RPG	-	-

4. Solução Priorizada para Prototipagem

- Escolhemos **Criar app no discord para co-mestrar/mestrar as partidas RPG** devido a parecer fazer mais sentido, de acordo com a persona e também, retorno maior na relação esforço x impacto

Canvas de Design de Prompts

Nosso propósito é gerenciar sessões de RPG de mesa via Discord, integrando sistema RAG para consultas D&D 5e com agente conversacional inteligente e sistema de contexto persistente via Redis.

Assim, utilizamos como modelo o Gemini Flash 2.5 integrado com esses vários módulos e funcionalidades por meio do agente.

Exemplo de prompt de conversa:

Situação: Uma aventura no mundo da Terra Média. Os jogadores, em sua expedição ao anoitecer, encontraram uma caverna que pode servir de abrigo.

Input: Thorin: parece que essa caverna foi escavada recentemente. Melhor termos cuidado! @LLM

Contexto de Entrada Necessário: Lista de mensagens do chat, nome do bot, contexto Redis da sessão

Para isso, em um contexto de conversa, o agente une várias informações na construção do prompt:

```
req = prompts.preinit + \

self.tool_settings.get_conversation_tools_explanation() + \

    self.world_history.GetHistory() + \

    (f"\n📋 **CONTEXTO DA
SESSÃO:**\n{context_summary}\n" if context_summary else "") + \

    prompts.chatBuild(chat.messages) + \

    prompts.postinit()
```

Parte por parte:

- Preinit se trata das informações básicas a serem passadas para o modelo, sobre o que ele é, o que ele pode fazer e como fazer.

```
"""Você é um modelo de linguagem conversando num chat de Discord
Seu objetivo principal é auxiliar a gerenciar uma sessão de RPG
Cite outros usuários utilizando @[nome-do-usuário]
Responda continuando a conversa de forma natural, continuando e
contribuindo para a aventura com os jogadores

"""
```

- A história se trata de um espaço para o modelo escrever um resumo das principais informações da aventura atual.
- O contexto da sessão são informações retornadas pelo sistema redis, previamente salvas.
- chatBuild são as mensagens do canal do Discord, formatadas em um formato entendido pelo modelo

```
def chatBuild(messages: list[ChatMessage]):
```

```

chat_text = map(lambda x: f"$ Mensagem de {x.username} às
{x.time}: " + parse_message(x.discord_message) + "\n$$$",
messages)

chat_text = "\n".join(chat_text)

final_text = f"""As mensagens anteriores são sinalizadas com [
$ Mensagem de x: ] e [ $ Mensagem do modelo: ] terminando com
$$$

Agora seguem as mensagens:

{chat_text}

```

- Postinit é uma formatação para instruir o modelo a modelar sua mensagem ou chamar uma função, conforme a necessidade

```

postinit = lambda: f"\n\n$ Mensagem de {LLM_NAME} às
{datetime.datetime.now()}: "

```

Com isso, o modelo tem todas as informações necessárias para responder a um mensagem de conversa, enviando uma mensagem no canal e interagindo com os usuários de forma fluida.

Intenção da Resposta: Resposta conversacional natural que continua a narrativa RPG, citando usuários com @ e mantendo o contexto da aventura

Exemplo de Saída: "Perfeito @Thorin! Como guerreiro anão, você consegue identificar que a caverna tem marcas de escavação recente. @Gandalf, você detecta uma aura mágica sutil vindo das profundezas. O que vocês querem fazer?"

CrITÉRIOS de qualidade: DEVE estar de acordo com o contexto da conversa e da história. DEVE mencionar os usuários (com @) sempre que pertinente. DEVE dar continuidade a aventura de forma interessante e engajante. NÃO DEVE entrar em loop na resposta.

Exemplo de prompt de inicialização de partida:

Situação: Os jogadores desejam iniciar uma partida de RPG, com a temática Cyberpunk

Input: Dhay: Quero algo no estilo de Cyberpunk, em uma cidade futurista, perigosa, porém cheia de oportunidades e aventura @LLM

Contexto de Entrada Necessário: Mensagens descrevendo como será a história e a ambientação da partida de RPG

Formato do prompt:

```
req = prompts.preinit_create_history + WorldHistoryTool_explanation + \
    (f"\n📋 **CONTEXTO DA SESSÃO:**\n{context_summary}\n"
    if context_summary else "") + \
    prompts.chatBuild(chat.messages) +
    prompts.postinit_alt() + prompts.postinit_create_history
```

Onde:

- Preinit create history instrui o agente a criar uma história interessante.

```
preinit_create_history = """Você é um modelo de linguagem conversando
num chat de Discord

Seu objetivo principal é auxiliar a gerenciar uma sessão de RPG

Agora, você deverá criar a história do mundo com base nas informações
dadas pelos usuários.

Utilize a criatividade, visando criar uma partida de RPG engajante.

"""
```

- World History Tool explanation explica a ferramenta a ser utilizada

```
WorldHistoryTool_explanation = """Agora, você deve chamar a função
InicializarHistória com um texto sobre a história \

do mundo de RPG da partida que está sendo inicializada.

"""
```

- chatBuild são as mensagens do canal do Discord, formatadas em um formato entendido pelo modelo
- O restante modela o final do request para o modelo utilizar a função fornecida para inserir a história produzida:

```
WorldHistoryTool_glm = glm.FunctionDeclaration(  
    name = "InicializarHistoria",  
    # OBSERVAÇÃO: Isso não está no formato correto  
    description = "Escreve o esboço inicial da história da partida de  
rpg",  
    parameters = glm.Schema(  
        type = glm.Type.OBJECT,  
        properties = {  
            "history_text": glm.Schema(  
                type = glm.Type.STRING,  
                description = "Corpo principal da história. É essencial  
que possua todos os detalhes principais do mundo da aventura de RPG."  
            )  
        },  
        required = ["history_text"],  
    ),  
)
```

Intenção da Resposta: Utilizar a função para construir a história do mundo, a salvando para ser referenciada depois

Exemplo de Saída: \ História do mundo: Bem-vindos a Neo-Veridia, uma metrópole cyberpunk pulsante, onde arranha-céus espelhados perfuram um céu perpetuamente nublado por poluição luminosa. A cidade é um labirinto de neon e aço, dividida em distritos que variam de luxuosos enclaves corporativos a favelas sombrias e perigosas. A tecnologia avançou a passos largos, com cibernética,

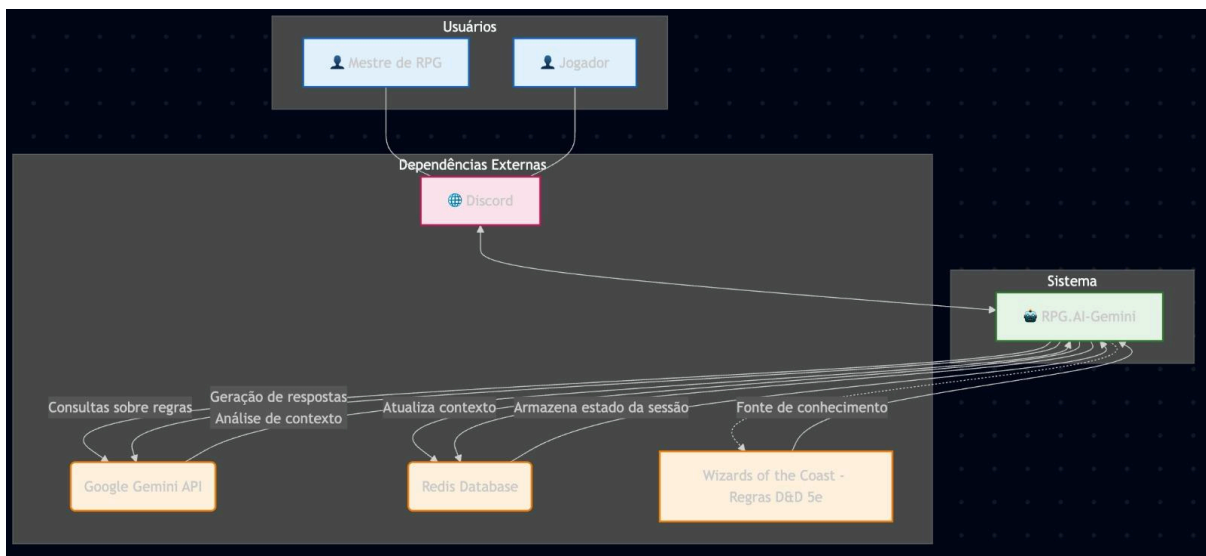
inteligência artificial e realidade aumentada entrelaçadas no tecido da vida cotidiana. No entanto, sob o brilho ofuscante, a desigualdade social é gritante. Megacorporações impérios detêm o poder absoluto, enquanto a maioria da população luta pela sobrevivência nas ruas, dominadas por gangues ciberneticamente aprimoradas e um mercado negro florescente. A aventura em Neo-Veridia oferece oportunidades infinitas para aqueles que ousam desafiar o sistema, seja como hackers habilidosos, mercenários de aluguel, ou rebeldes em busca de um futuro melhor. Cada esquina pode esconder um perigo mortal ou uma chance de glória e fortuna.

& A aventura foi inicializada

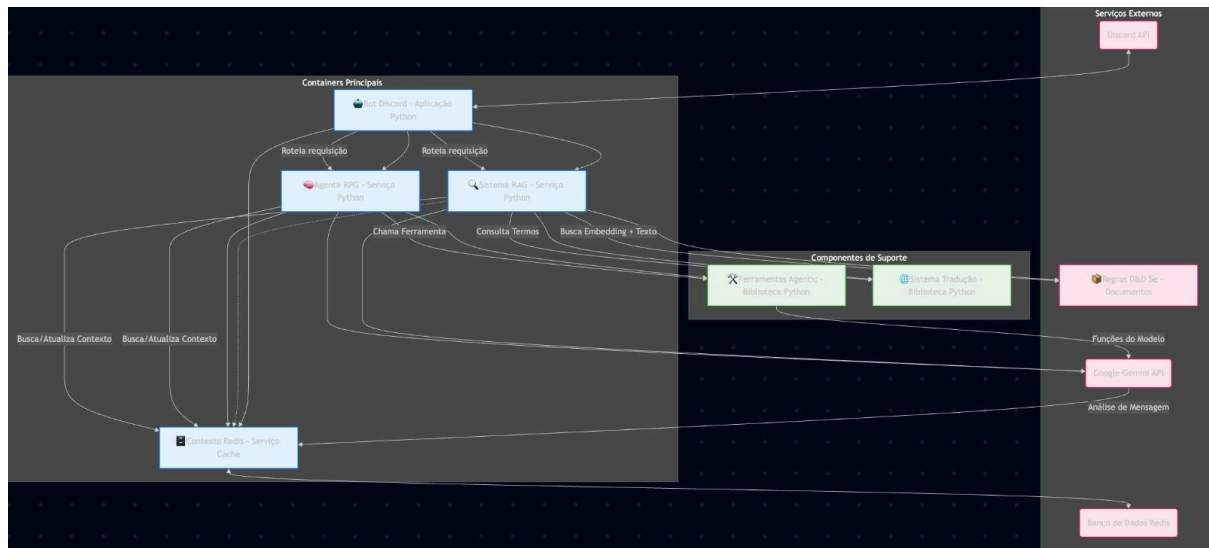
Critérios de qualidade: DEVE criar uma história baseado inteiramente no que os usuários inseriram. DEVE mencionar os usuários (com @) sempre que pertinente. DEVE ser interessante e engajante. NÃO DEVE entrar em loop na resposta. NÃO DEVE ser chamado com a aventura em execução

3.3 Produção

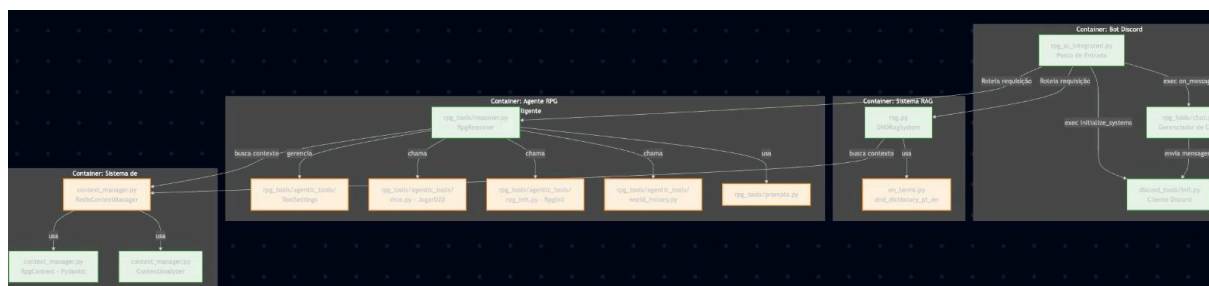
C4 Nível 1



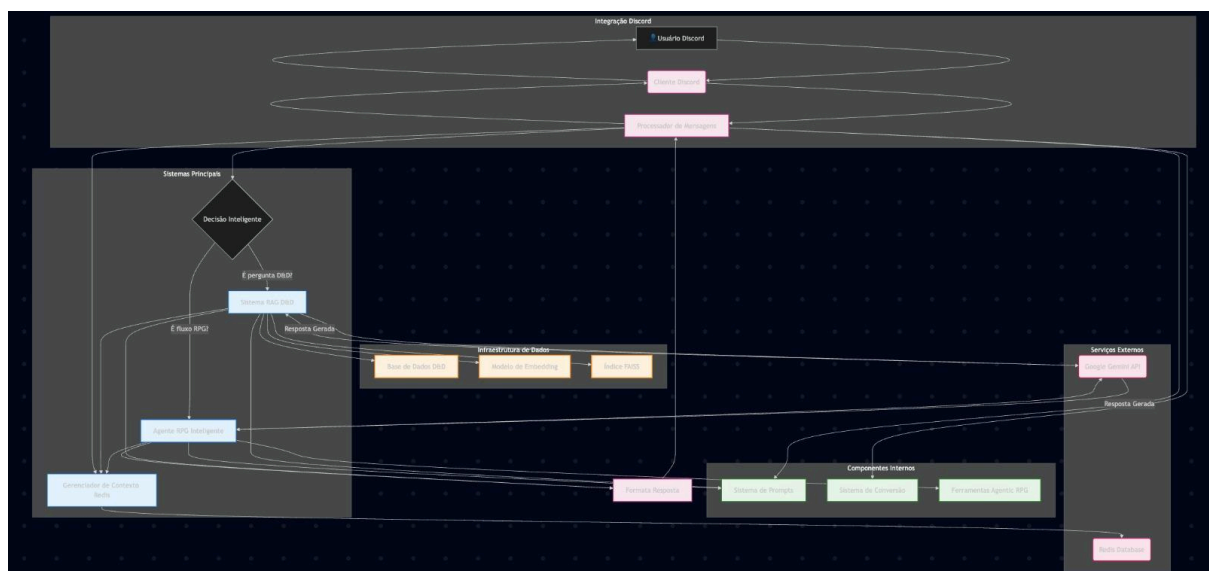
C4 Nível 2



C4 Nível 3



C4 Nível 4



Estratégia de desenvolvimento e tecnologias adotadas

Containers e Componentes Internos

Esta seção detalha os componentes internos de cada container, mostrando como eles são implementados e como se relacionam entre si.

Container: Sistema RAG D&D

O **Sistema RAG D&D** é o serviço responsável por buscar e gerar respostas baseadas nas regras de D&D. Ele contém os seguintes componentes:

- **DNDragSystem (rag.py):** Este é o componente principal que orquestra a busca de informações. Ele é o núcleo do sistema RAG e utiliza o dicionário de termos para refinar suas buscas.
 - **dnd_dictionary_pt_en (en_terms.py):** Um dicionário de tradução que mapeia termos de D&D de português para inglês. O **DNDragSystem** utiliza este componente para traduzir consultas e buscar chunks mais relevantes nas regras em inglês.
-

Container: Agente RPG Inteligente

O **Agente RPG Inteligente** é o coração do sistema para o gerenciamento de sessões de jogo, contendo o núcleo de raciocínio e as ferramentas agentic.

- **RpgReasoner (rpg_tools/reasoner.py):** O cérebro do agente. Ele gerencia o estado da sessão, decide qual ferramenta utilizar e orquestra a interação com o modelo de linguagem e os outros componentes.
 - **ToolSettings (rpg_tools/agentic_tools/__init__.py):** Um componente que gerencia de forma dinâmica quais ferramentas estão disponíveis para o agente em um dado momento, controlando a lista de ferramentas ativas.
 - **JogarD20 (rpg_tools/agentic_tools/dice.py):** A ferramenta para rolagem de dados. É um componente chamado pelo **RpgReasoner** para simular um dado de 20 faces.
 - **RpgInit (rpg_tools/agentic_tools/rpg_init.py):** A ferramenta de inicialização de RPG. Atua como um gatilho para iniciar uma nova sessão de jogo, instruindo o agente sobre o fluxo de trabalho de criação.
 - **world_history.py:** Um componente para gerenciar a história do mundo. Permite ao agente escrever ou expandir a narrativa da campanha.
 - **prompts.py:** Um conjunto de componentes que fornece os prompts otimizados para o agente, garantindo que ele tenha o contexto e as instruções corretas para cada tarefa.
-

Container: Sistema de Contexto Redis

Este container lida com a persistência e a análise do contexto da sessão de RPG, usando o Redis como banco de dados.

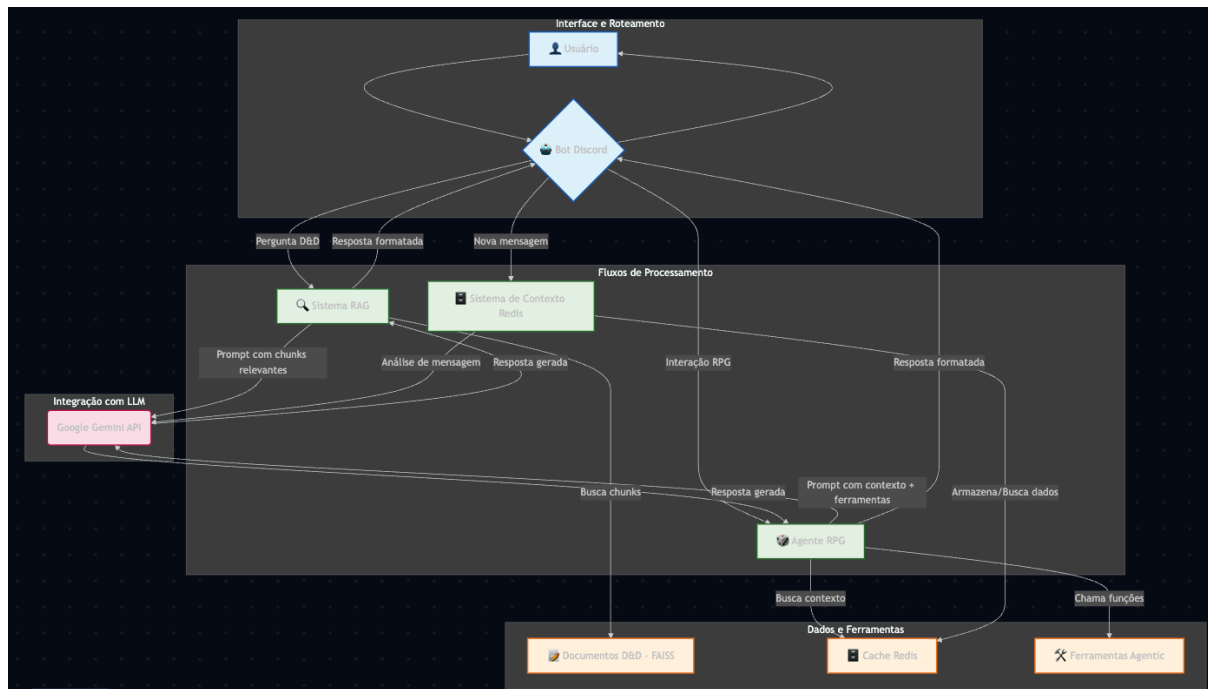
- **RpgContext (context_manager.py):** Um modelo de dados estruturado (criado com Pydantic) que define a estrutura do contexto de uma sessão de RPG, incluindo personagens, locais e eventos.
 - **ContextAnalyzer (context_manager.py):** Um componente que utiliza o Gemini para analisar as mensagens do chat e extrair informações relevantes, como menções de personagens ou mudanças de localização.
 - **RedisContextManager (context_manager.py):** O componente principal para gerenciar o cache. Ele utiliza o cliente Redis para criar, recuperar e atualizar o contexto de sessões, garantindo que as informações sejam persistentes.
-

Interações entre Containers

O fluxo de dados entre os containers de alto nível funciona da seguinte forma:

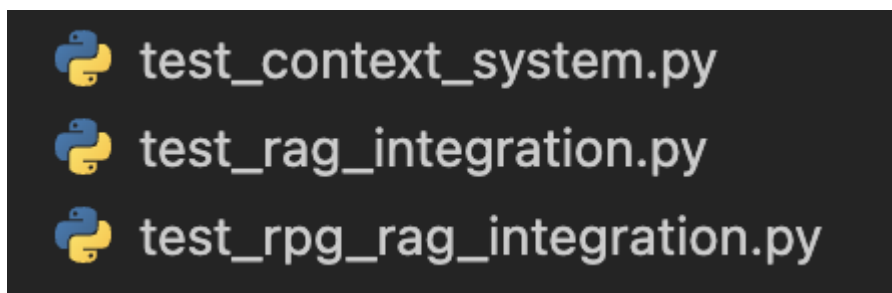
- **Roteamento:** O **Bot Discord** recebe uma requisição e a roteia para o **Sistema RAG** (se for uma pergunta sobre D&D) ou para o **Agente RPG** (se for uma interação de jogo).
- **Contexto:** Tanto o **Agente RPG** quanto o **Sistema RAG** buscam e atualizam informações no **Sistema de Contexto Redis** para garantir que suas respostas sejam sempre contextuais e relevantes para a sessão atual.

Fluxo de integração com LLMs



Evidências de testes e validação da solução

A cada grande mudança, com o auxílio de IA, geramos um script de testes. O mais atualizado é o `test_rpg_rag_integration.py`, entretanto, já houveram algumas atualizações para refinamento da aplicação após ele e concentramos os nossos testes sendo diretamente pelo Discord em nosso servidor que contém o bot.



Vídeo da aplicação em funcionamento:

[!\[\]\(3cb60d42b10e53f9522bb0b392c1c4cd_img.jpg\) CLIQUE AQUI! LINK DO VÍDEO](#)

Link do Projeto:

Nossa solução usa o Discord como frontend as a service (FEaaS), então não faz sentido o envio de um link de acesso, mas se o senhor nos passar o seu discord, a gente pode te adicionar no server pra você testar!

1. Objetivo da Escalabilidade

O principal objetivo da escalabilidade do RPG.AI-Gemini é garantir que o sistema possa suportar um aumento significativo no número de usuários e interações simultâneas sem degradar o desempenho. O foco é manter a baixa latência e a alta disponibilidade, permitindo que a plataforma cresça de forma sustentável e atenda a uma comunidade de RPG em expansão.

2. Volume Esperado de Interações

- **Cenário Atual (Piloto):**
 - 1 a 5 servidores Discord
 - 10 a 50 usuários ativos por dia
 - ~500 interações (mensagens processadas) por dia
 - 1-3 sessões de RPG ativas simultaneamente
- **Cenário Futuro (Escalado):**
 - +50 servidores Discord
 - +500 usuários ativos por dia
 - +5.000 interações por dia
 - 10-20 sessões de RPG ativas simultaneamente
 - +200 novos usuários por mês

3. Requisitos de Infraestrutura

- **Orquestração:** Migração de Heroku para **Docker e Kubernetes** para orquestração de containers, permitindo escalabilidade horizontal.
- **Serviços Distribuídos:** Cada container (Bot Discord, Sistema RAG, Agente RPG) deve ser executado como um microserviço separado.
- **Memória:** Aumento de RAM por instância do Sistema RAG para acomodar o índice FAISS (~5.3MB) e os embeddings em memória.
- **Cache:** Um cluster **Redis** para alta disponibilidade e replicação, garantindo que o cache de contexto da sessão seja acessível e resiliente.
- **Armazenamento:** Volume de armazenamento persistente (como S3 ou Google Cloud Storage) para o índice FAISS pré-processado, evitando o reprocessamento em cada nova instância.
- **Rede:** Balanceador de carga para distribuir as requisições da Discord API entre as instâncias do bot.

4. Estratégias de Escalabilidade

- **Escalabilidade Horizontal:** Adicionar novas instâncias de cada container (Bot, RAG, Agente) conforme a demanda aumenta. O Kubernetes se encarregará de gerenciar e provisionar esses containers automaticamente.

- **Sharding:** Implementar o sharding do bot do Discord, dividindo os servidores entre diferentes instâncias para otimizar o uso da memória e evitar o limite de `rate limit`.
- **Cache Distribuído:** Utilizar o Redis Cluster para garantir que o contexto de uma sessão de RPG esteja disponível para qualquer instância do Agente RPG, independentemente de qual instância a atendeu.
- **Processamento Assíncrono:** Utilizar filas de mensagens (RabbitMQ ou Celery) para lidar com requisições intensivas (como a geração de respostas longas), liberando o bot para processar novas interações imediatamente.
- **Otimização de Consultas:** Refinar a busca em índices FAISS e o dicionário de tradução para garantir que a latência não aumente com o volume de dados.

5. Custo Estimado

- **Infraestrutura:** Custo de servidores em nuvem (GCP, AWS) para os clusters de Kubernetes e Redis. Uma estimativa inicial para um ambiente de produção pode variar de **R\$ 200 a R\$ 800 por mês**, dependendo do volume de interações.
- **APIs:** Aumento de custos com a **API do Google Gemini** conforme o volume de tokens gerados e analisados cresce. É esperado que o custo da API seja a principal despesa operacional, podendo variar de **R\$ 150 a R\$ 1.000 por mês** em um cenário escalado.
- **Manutenção:** Custo de horas de trabalho de um desenvolvedor para gerenciar a infraestrutura e monitorar os sistemas.

6. Riscos e Mitigação

- **Risco: Latência da API do Google Gemini.**
 - **Mitigação:** Implementar cache de respostas frequentes, usar o modelo mais rápido (Gemini 2.0 Flash Lite) para tarefas menos complexas (análise de contexto) e utilizar backoff exponencial em caso de erros.
- **Risco: Rate limits do Discord.**
 - **Mitigação:** Implementar o sharding do bot e usar um gerenciador de `rate limit` para garantir a conformidade com as políticas da API.
- **Risco: Degradação de desempenho com o aumento de sessões ativas.**
 - **Mitigação:** Monitorar o uso de memória e CPU por container e configurar regras de auto-scaling no Kubernetes para adicionar novas instâncias automaticamente.
- **Risco: Instabilidade do Redis.**
 - **Mitigação:** Usar um cluster Redis gerenciado com replicação para garantir alta disponibilidade e persistência dos dados.

7. Monitoramento de Escalabilidade

- Monitoramento contínuo de métricas de desempenho em cada container (CPU, memória, latência).
- Alertas configurados para picos de uso que possam indicar a necessidade de adicionar novas instâncias.
- Uso de ferramentas de observabilidade como **Datadog** ou **Prometheus/Grafana** para visualizar a performance do sistema em tempo real e monitorar a saúde de cada microsserviço.

8. Plano de Teste em Ambiente Escalado

- **Teste de Carga:** Simular o volume de interações esperado em um ambiente de staging para identificar gargalos de desempenho antes do deploy em produção.
- **Teste de Estresse:** Enviar um volume de requisições acima do esperado para verificar o ponto de falha do sistema e como ele se comporta sob pressão.
- **Teste de Resiliência:** Simular a falha de um container ou serviço externo para garantir que o sistema se recupere automaticamente.
- **Teste de A/B:** Se possível, testar diferentes configurações de escalabilidade (ex.: mais RAM vs. mais instâncias) para encontrar a configuração mais otimizada em termos de desempenho e custo.

Plano de Monitoramento: Canvas de Métricas de Escala e Impacto

1. Objetivo do Monitoramento

O monitoramento do RPG.AI-Gemini tem como objetivo principal avaliar o desempenho técnico do sistema e o impacto estratégico na comunidade de RPG. O foco é garantir que o assistente opere com alta eficiência, ofereça uma experiência de usuário positiva e esteja preparado para um crescimento contínuo.

2. Métricas de Uso

- **Interações diárias:** Quantidade de mensagens processadas pelo bot por dia.
- **Usuários ativos:** Contagem de usuários únicos que interagiram com o assistente por semana/mês.
- **Sessões ativas:** Número de sessões de RPG gerenciadas ativamente pelo agente a cada hora.
- **Taxa de uso do RAG:** Porcentagem de interações que acionaram o Sistema RAG versus o Agente RPG.

- **Taxa de uso de ferramentas:** Quantidade de vezes que cada ferramenta (JogarD20, InicializarRPG, etc.) foi chamada.

3. Métricas de Desempenho

- **Tempo médio de resposta (TMR):** Latência média para gerar uma resposta.
 - TMR Sistema RAG (incluindo busca): 2-5 segundos.
 - TMR Agente RPG (incluindo `function calls`): 3-8 segundos.
- **Taxa de erros:** Porcentagem de interações que resultam em uma falha, como um timeout ou um erro na API.
- **Latência da API:** Tempo de resposta das APIs externas (Gemini, Discord, Redis).
- **Uso de recursos:** Porcentagem de CPU e memória por container.
- **Hit rate do cache Redis:** Porcentagem de vezes que o contexto foi recuperado do cache.

4. Métricas de Impacto no Negócio

- **Adoção em servidores:** Número de servidores Discord que adotaram o bot.
- **Retenção de usuários:** Porcentagem de usuários que retornam ao bot após a primeira semana.
- **Custo por interação:** Custo médio da API do Google Gemini por mensagem processada.
- **Redução de tempo de consulta:** Estimativa do tempo economizado por mestres de RPG ao usar o assistente em vez de manuais.

5. Métricas de Satisfação do Usuário

- **Feedback explícito:** Contagem de feedbacks positivos (+1, muito bom) e negativos (erro, não funcionou) nas respostas.
- **Taxa de retenção do Agente RPG:** Porcentagem de sessões de RPG que continuam ativas por mais de uma hora.
- **Taxa de abandono:** Número de usuários que desativaram o bot em seus servidores.

6. Ferramentas de Monitoramento

- **Uso e Desempenho:** **Datadog** ou **Prometheus/Grafana** para coletar métricas dos containers.
- **Logs:** **ELK Stack** (Elasticsearch, Logstash, Kibana) ou **Datadog Log Management** para análise estruturada de logs.

- **APIs:** Ferramentas de monitoramento de API, como **Postman Monitors**, para testar a saúde de endpoints externos.
- **Feedback:** Análise de feedback explícito via scripts no bot, além de ferramentas de pesquisa de satisfação se implementadas.

7. Benchmarks

- **Tempo de Resposta:** ≤ 5 segundos para 95% das interações.
- **Satisfação:** $\geq 90\%$ de feedback positivo explícito nas interações de regras (RAG).
- **Taxa de Erros:** $\leq 1\%$ de falhas durante as interações.
- **Custo da API:** $\leq R\$ 0,05$ por interação.

8. Acompanhamento de Tendências

- **Relatórios semanais** de uso e desempenho para identificar picos de tráfego, sazonalidades e possíveis problemas.
- **Análise comparativa** de métricas do sistema com benchmarks de desempenho para avaliar a performance.
- **Monitoramento de tendências** de uso para prever o crescimento e planejar a escalabilidade.

9. Ações Baseadas nas Métricas

- **Otimizar prompts** se o custo por interação exceder o benchmark.
- **Revisar a lógica do Agente RPG** se a taxa de sucesso de `function calls` cair abaixo de 90%.
- **Ajustar a configuração do bot** se a latência de resposta exceder 5 segundos.
- **Lançar campanhas de divulgação** para aumentar a taxa de adoção se os números de usuários ativos não estiverem crescendo.

10. Relatórios e Compartilhamento

- **Painel interativo** em uma ferramenta como Grafana para a equipe de desenvolvimento e gestão.
- **Relatórios mensais** automatizados com as principais métricas e tendências para a liderança.
- **Reuniões de revisão trimestrais** para ajustar a estratégia com base nos dados e no feedback.

Canvas de Feedback e Iteração

1. Objetivo do Ciclo de Feedback

O principal objetivo do ciclo de feedback é criar um processo contínuo de aprendizado e melhoria para o RPG.AI-Gemini. A pergunta central a ser respondida é: **"Como podemos garantir que o assistente seja mais útil e imersivo para a comunidade de RPG?"**

2. Fontes de Feedback

- **Feedback Explícito:** Reações diretas dos usuários (ex.: +1, obrigado, "não funcionou") em respostas do bot.
- **Feedback Implícito:**
 - **Taxa de Retenção:** Quantos usuários voltam a usar o bot.
 - **Padrões de Uso:** Qual a frequência de uso das ferramentas ou do Sistema RAG.
 - **Abandono:** Taxa de servidores que desativaram o bot.
- **Análise de Conversa:**
 - **Logs de Chat:** Análise de prompts e respostas para identificar erros de raciocínio ou falhas na geração de texto.
 - **Mensagens Mal Interpretadas:** Análise de inputs que o bot não conseguiu processar corretamente.
- **Comunicação Direta:**
 - **Canal de Suporte:** Um canal de Discord dedicado para feedback e sugestões.
 - **Reuniões com Usuários:** Encontros com mestres de RPG e jogadores para entender as necessidades e pontos problemáticos.

3. Análise do Feedback

- **Classificação:** Categorizar o feedback como bug, sugestão de recurso, elogio ou erro de conteúdo.
- **Priorização:** Atribuir uma pontuação de prioridade a cada item de feedback com base na frequência e no impacto potencial.
- **Análise de Causa-Raiz:** Investigar por que um bug ocorreu ou por que uma funcionalidade não está sendo usada.
- **Validação de Hipóteses:** Usar o feedback para validar ou invalidar hipóteses sobre o uso do assistente.

4. Ações e Iteração

- **Planejamento:** Adicionar os itens de feedback priorizados ao backlog de desenvolvimento.

- **Design de Solução:** Propor soluções para os problemas identificados, como otimização de prompts, ajuste na lógica do agente ou criação de uma nova ferramenta.
- **Implementação:** Desenvolver e testar as melhorias em um ambiente de staging.
- **Deploy:** Lançar as novas funcionalidades ou correções para a comunidade.
- **Monitoramento:** Acompanhar as métricas (ex.: taxa de erro, tempo de resposta) após o lançamento para garantir que a melhoria teve o impacto desejado.

5. Conclusão do Ciclo

Após cada ciclo de feedback, é crucial comunicar as mudanças aos usuários e repetir o processo. O objetivo é criar um loop virtuoso onde o feedback da comunidade alimenta diretamente a evolução do assistente, garantindo que ele se adapte continuamente às necessidades dos jogadores e mestres de RPG.

4. Discussões Técnicas e Estratégicas

Tomadas de decisão

Buscamos inicialmente implementar nossa prova de conceito de forma simplificada, porém aberta a extensão de novas funcionalidades. Isso garantiu flexibilidade, agilidade e capacidade de experimentar com diferentes configurações.

Por isso, decidimos implementar um agente de LLM de forma que podemos adicionar e remover funções e parâmetros dinamicamente. Como o sistema RAG, as ferramentas fornecidas para a API e os prompts de inicialização.

Desenhamos nossa arquitetura para desacoplar na medida do possível do modelo utilizado e do mecanismo de mensagens. Porém, nosso foco em questão de validação da prova de conceito foi o uso do Discord com a API do Gemini.

Decisões Arquiteturais Justificadas

1. Arquitetura em Camadas com C4 Model

Decisão Implementada Adoção do C4 Model para documentação da arquitetura em 4 níveis: Contexto, Containers, Componentes e Código.

Justificativa Técnica

- **Comunicação:** *Facilita a comunicação entre times técnicos e não-técnicos.*
- **Manutenibilidade:** *A documentação estruturada permite uma evolução controlada.*
- **Padrão da Indústria:** *Metodologia amplamente reconhecida e validada.*
- **Escalabilidade:** *Suporte a um crescimento sustentável da arquitetura.*

Alternativas Consideradas

- **UML:** *Muito complexo para stakeholders sem conhecimento técnico.*
 - **Documentação Informal:** *Difícil de manter e evoluir.*
 - **Arquitetura de Referência:** *Genérica demais para decisões específicas do projeto.*
-

2. Arquitetura Modular com Containers Independentes

Decisão Implementada O sistema é composto por containers independentes (Bot Discord, Sistema RAG, Agente RPG e Contexto Redis) que se comunicam através de interfaces bem definidas.

Justificativa Técnica

- **Desenvolvimento Paralelo:** *Equipes podem trabalhar de forma independente.*
- **Testabilidade:** *Cada componente pode ser testado isoladamente.*
- **Escalabilidade:** *Permite escalar os containers de forma independente.*
- **Manutenibilidade:** *Mudanças em um container não afetam os demais.*

Alternativas Consideradas

- **Monolito:** *Dificulta a escalabilidade e a manutenção.*
 - **Microserviços:** *Apresenta uma complexidade excessiva para o tamanho atual do projeto.*
 - **Arquitetura em Camadas:** *Gera um forte acoplamento entre as camadas.*
-

3. Sistema de Roteamento Inteligente

Decisão Implementada Um sistema que decide automaticamente entre usar o RAG ou o Agente RPG com base no conteúdo da mensagem e no contexto.

Justificativa Técnica

- **Eficiência:** Utiliza o sistema mais apropriado para cada requisição.
- **Experiência do Usuário:** Oferece respostas mais relevantes e contextualizadas.
- **Otimização de Recursos:** Evita processamentos desnecessários.
- **Flexibilidade:** Permite a evolução independente dos sistemas.

Alternativas Consideradas

- **Roteamento Manual:** Complexo para o usuário, que teria que escolher o sistema.
 - **Sistema Único:** Combinar todas as funcionalidades geraria uma implementação muito complexa.
 - **Load Balancing:** Distribui a carga, mas não considera o contexto da mensagem.
-

4. Sistema de Contexto Persistente com Redis

Decisão Implementada Utilização do Redis como um sistema de cache persistente para o contexto das sessões de RPG, com TTL (Time-To-Live) automático.

Justificativa Técnica


- **Performance:** Acesso rápido aos dados de contexto.
- **Persistência:** Os dados sobrevivem a reinicializações.
- **TTL Automático:** Limpeza automática de dados expirados.
- **Escalabilidade:** Suporte a múltiplas instâncias.

Alternativas Consideradas

- **Memória Local:** Perda de dados em caso de reinicializações.
 - **Banco Relacional:** Alto overhead para dados temporários.
 - **Arquivos Locais:** Complexo para sincronizar entre processos.
 - **MongoDB:** Complexidade excessiva para dados simples.
-


Avaliação de Alternativas Técnicas




1. Seleção de Modelo de Linguagem

Modelo	Vantagens	Desvantagens	Score	Decisão
Google Gemini	Function Calling nativo, bom desempenho, preço competitivo	Dependência da Google, rate limits	9.2	 ESCOLHIDO
OpenAI GPT-4	Qualidade excepcional, ecossistema maduro	Preço alto, sem tier gratuito, sem Function Calling nativo	8.5	 Rejeitado
Claude (Anthropic)	Qualidade alta, ética	API menos madura, preço alto	7.8	 Rejeitado
Modelos Locais	Privacidade total, sem custos de API	Qualidade inferior, recursos computacionais	6.5	 Rejeitado

Exportar para as Planilhas




2. Sistema de Busca Semântica


Tecnologia	Vantagens	Desvantagens	Score	Decisão
FAISS	Desempenho excepcional, otimizado para produção	Complexidade de configuração	9.0	 ESCOLHIDO

<i>Elasticsearch</i>	<i>Funcionalidades avançadas, ecossistema rico</i>	<i>Overhead para busca simples, complexidade</i>	7.5	 Rejeitado
<i>Pinecone</i>	<i>SaaS gerenciado, fácil implementação</i>	<i>Custos altos, dependência externa</i>	7.0	 Rejeitado
<i>Scikit-learn</i>	<i>Familiaridade, fácil implementação</i>	<i>Desempenho inferior para grandes volumes</i>	6.0	 Rejeitado

Exportar para as Planilhas

3. Sistema de Cache e Persistência

<i>Tecnologia</i>	<i>Vantagens</i>	<i>Desvantagens</i>	<i>Score</i>	<i>Decisão</i>
Redis	<i>Desempenho alto, persistência, TTL automático</i>	<i>Dependência externa, configuração</i>	9.0	 ESCOLHIDO
<i>Memcached</i>	<i>Desempenho alto, simplicidade</i>	<i>Sem persistência, funcionalidades limitadas</i>	7.0	 Rejeitado
<i>PostgreSQL</i>	<i>ACID, funcionalidades avançadas</i>	<i>Overhead para cache, complexidade</i>	6.5	 Rejeitado

Arquivos Locais	Simplicidade, sem dependências	Desempenho baixo, sincronização	5.0	 Rejeitado
-----------------	--------------------------------	---------------------------------	-----	---

Exportar para as Planilhas

4. Framework de Comunicação

Tecnologia	Vantagens	Desvantagens	Score	Decisão
Discord.py	Estabilidade, funcionalidades ricas, comunidade, boa documentação, presença do público alvo	Limitações da plataforma Discord, rate limit	9.0	 ESCOLHIDO
Telegram Bot API	Funcionalidades avançadas, multiplataforma, presença do público alvo	Comunidade menor, documentação	8.0	 Rejeitado
Slack API	Integração empresarial, funcionalidades	Custos para funcionalidades avançadas	7.5	 Rejeitado
WebSocket Custom	Controle total, flexibilidade	Complexidade de implementação	6.0	 Rejeitado

Exportar para as Planilhas

Integração com Bases de Dados, APIs e Ferramentas Externas

1. Google Generative AI API

- **Configuração de Integração:**
`genai.configure(api_key=G00GLE_API_TOKEN)`
 - **Funcionalidades Integradas:** Geração de texto, Function Calling, Análise de Contexto, Processamento de Linguagem.
 - **Configurações de Segurança:** API Keys restritas, Rate Limiting, monitoramento de quotas e Audit Logs.
 - **Tratamento de Erros:** Retry automático, fila de requisições e fallback para respostas básicas.
-

2. Discord API

- **Configuração de Integração:** `intents = Intents.default(), client = Client(intents=intents)`
 - **Funcionalidades Integradas:** WebSocket Gateway, REST API, Event Handling.
 - **Configurações de Segurança:** Bot Tokens com escopos limitados e Intents restritas às permissões necessárias.
 - **Tratamento de Erros:** Reconexão automática e fila de mensagens.
-

3. Redis

- **Configuração de Integração:** `redis_client = redis.from_url(redis_url)`
 - **Funcionalidades Integradas:** Cache persistente para sessões de RPG e TTL automático.
 - **Configurações de Segurança:** Autenticação com senha configurável e acesso restrito por IP.
 - **Tratamento de Erros:** Retry automático e políticas de despejo (eviction policies).
-

4. Sistema de Arquivos Local

- **Configuração de Integração:** Arquivos `dnd.txt`, `dnd_index.pkl`, `dnd_index.faiss` e `en_terms.py`.
 - **Funcionalidades Integradas:** Regras de D&D, busca semântica otimizada e um sistema de tradução com mais de 1800 mapeamentos.
 - **Configurações de Segurança:** Permissões de arquivo restritas e backups regulares com controle de versão Git.
-

Registro de Estratégia e Inteligência - OBRIGATÓRIO

1. Análise de Requisitos Funcionais





- **Requisitos Identificados:** Consulta de regras de D&D, gerenciamento de sessões, ferramentas agentic, suporte multilíngue e integração com o Discord.
 - **Priorização:** A consulta de regras e o gerenciamento de sessões foram definidos como **alta prioridade**, enquanto as ferramentas agentic e o suporte multilíngue foram classificados como **média prioridade**.
-

2. Análise de Requisitos Não-Funcionais

- **Performance:** Latência de menos de 5 segundos, suporte a mais de 100 usuários simultâneos e escalabilidade horizontal.
 - **Disponibilidade:** Uptime de 99,5%, recuperação automática de falhas e backup de dados.
 - **Segurança:** Autenticação OAuth2, princípio do menor privilégio e auditoria de logs.
 - **Manutenibilidade:** Código limpo, documentação técnica, cobertura de testes acima de 80% e deploy automatizado.
-

3. Análise de Riscos Técnicos

Risco	Probabilidade	Impacto	Mitigação	Status
-------	---------------	---------	-----------	--------

Indisponibilidade da API Gemini	Média	Alto	Fallback para respostas básicas	 Parcialmente Mitigado
Rate Limiting Discord	Alta	Médio	Fila de mensagens, retry automático	 Mitigado
Falha do Redis	Baixa	Alto	Fallback para memória local	 Mitigado
Degradação de Performance	Média	Médio	Monitoring, alerting, scaling automático	 Em Mitigação

Exportar para as Planilhas

4. Análise de Custos e ROI

- **Custos de Desenvolvimento:** 10-14 meses de desenvolvimento, \$50-100/mês em infraestrutura e 20-30% do esforço em manutenção.
- **Benefícios Esperados:** Redução de 70-80% no tempo de consulta de regras, respostas consistentes, escalabilidade e extensibilidade.
- **Análise de ROI:** Payback de 6-12 meses, ROI anual de 300-500% e valor presente líquido positivo após 18 meses.

5. Roadmap Estratégico

- **Fase 1 (1-2 meses): Estabilização** - Estabilizar o sistema, implementar CI/CD e testes abrangentes.
 - **Fase 2 (2-4 meses): Expansão** - Tornar o sistema multiplataforma e adicionar suporte a outros sistemas de RPG.
 - **Fase 3 (4-8 meses): Inovação** - Integrar IA avançada e criar uma plataforma de colaboração.
-

Conclusões e Recomendações

1. Decisões Arquiteturais Validadas

- **Pontos Fortes:** Arquitetura modular, integração inteligente, escalabilidade e resiliência.
 - **Áreas de Melhoria:** Implementação de CI/CD, monitoramento com APM, aumento da cobertura de testes e manutenção da documentação.
-

2. Recomendações Estratégicas

- **Prioridades Imediatas:** Implementar CI/CD, containerizar o sistema com Docker, adicionar monitoramento e expandir a cobertura de testes.
 - **Investimentos de Médio Prazo:** Expandir para outras plataformas (Telegram, Slack) e sistemas de RPG, e implementar personalização e analytics.
 - **Visão de Longo Prazo:** Criar uma plataforma de colaboração, usar IA avançada (multimodal e preditiva) e desenvolver um marketplace de conteúdo.
-

3. Próximos Passos

- **Sprint Atual:** Finalizar o monitoramento, implementar testes de integração, documentar APIs internas e preparar para a containerização.
- **Próximo Sprint:** Implementar o pipeline de CI/CD, containerizar o sistema com Docker, implementar health checks e preparar o deploy em Kubernetes.
- **Sprint Futuro:** Implementar multi-tenancy, adicionar um sistema de feedback, implementar a personalização e preparar a expansão multi-plataforma.

Desafios enfrentados e Soluções encontradas

Precisamos escolher um bom modelo generativo de linguagem, equilibrando custo, velocidade, praticidade e performance. Optamos pelo Gemini 2.5 Flash oferecido pela API do Google, com o qual concluímos ter o melhor equilíbrio.

Lidar com bugs e “alucinações”. Especialmente, no caso do agente chamar uma função ou produzir um mesmo texto em loop. Foi necessário gerenciar as ferramentas dadas ao agente e os prompts de inicialização com cuidado.

Desenvolver de código limpo e uma arquitetura bem fundamentada em boas práticas de engenharia de software, adaptada ao contexto de agente de IA. Assim, utilizamos uma abordagem modular para isolar e desacoplar cada componente e, assim, facilitar o desenvolvimento.

Foi necessária muita tentativa e erro, testando e validando cada iteração do agente para avaliar se está realizando o comportamento esperado.

Um desafio técnico foi a integração com sistema RAG, para acessar informações sobre o sistema de D&D conforme necessário. Precisamos encontrar o equilíbrio entre ter esse sistema para ajudar e ele não se tornar intromissivo demais.

5. Considerações Éticas

Possíveis riscos ou vieses e Estratégias de mitigação

Os riscos e vieses dessa solução dependem principalmente do modelo base adotado (Gemini Flash 2.5). No entanto, por se tratar de algo com foco lúdico, não é uma solução de alto risco.

O principal risco para os usuários é o compartilhamento de mensagens do servidor conosco e com o provedor de LLM. Assim, garantimos informar os usuários sobre o compartilhamento de dados, em conformidade com a LGPD, e o isolamento do contexto de conversa e do agente a cada canal do servidor, para impedir vazamentos.

Avaliação de impacto social, humano e ambiental

Temos como propósito principal auxiliar usuários novatos a criarem suas histórias e aventuras. Dessa forma, almejamos impactar positivamente no cenário de RPG trazendo novos usuários.

Nossa solução promove a interação virtual entre as pessoas nas jogatinas de RPG, propiciando momentos lúdicos, interativos e de socialização. Assim, tem potencialmente um impacto social muito positivo ao trazer esse jogo fortemente social para mais pessoas.

6. Lições Aprendidas e Reflexões Finais

Diego: Foi uma experiência muito gratificante, de muito aprendizado e também fundamental para o desenvolvimento de nossa solução o uso da metodologia AI Design.

A metodologia foi importante porque ela exigiu para nós darmos respostas a perguntas fundamentais na criação de uma solução robusta e bem fundamentada focada na geração de valor.

Com isso, conseguimos entregar uma solução funcional e que gostamos de utilizar para criar aventuras. Concluo, logo, que a proposta de valor foi entregue com sucesso.

Para projetos futuros, é interessante estudar mais a literatura sobre o funcionamento na prática de modelos de linguagem e de agentes de IA. Principalmente, lições de engenharia de prompt e engenharia de contexto foram fundamentais para nossa solução ganhar corpo. No entanto, desenvolver um sistema na prática foi excelente para aprender com a experiência.

Na prática, aprendi que muito do uso desses sistemas ainda é bastante tentativa e erro com o uso de prompts e da engenharia de contexto. Não há uma receita de bolo pronta para a criação de um agente. Um exemplo prático é que é importante gerenciar o contexto para evitar bugs e alucinações do modelo, como gerar texto em loop ou chamar uma função repetidamente, então não é trivial o que vai funcionar em cada caso.

Matheus: O desenvolvimento do sistema RPG.AI foi uma jornada de aprendizado contínuo, com percepções valiosas em diversas frentes, desde as decisões técnicas até a gestão de processos e a compreensão do usuário.

Na parte técnica, a equipe aprendeu que a modularidade foi essencial. A criação de componentes independentes permitiu o desenvolvimento em paralelo, facilitou a manutenção e isolou falhas, em vez de criar um monolito complexo. A abstração das LLMs foi outra decisão crucial, garantindo que o sistema não ficasse refém de um único provedor. Ferramentas específicas, como o FAISS para a busca semântica, o chunking inteligente e o connection pooling, foram decisivas para alcançar a performance e a escalabilidade necessárias. Em relação às integrações, o Function Calling nativo do Google Gemini se destacou como uma vantagem para a orquestração de ferramentas, e o roteamento inteligente validou que a resposta mais eficaz é a que melhor se adequa ao contexto, e não necessariamente a mais complexa.

Do ponto de vista de processo e metodologia, a equipe se beneficiou do desenvolvimento iterativo, que permitiu a entrega de valor contínuo e a incorporação rápida de feedback. A gestão proativa de dependências, monitorando o rate limiting e o versionamento das APIs externas, foi fundamental para evitar interrupções.

As lições de negócio e usuário confirmaram que o sucesso do projeto está em resolver um problema real com simplicidade de uso. O suporte multilíngue ampliou o alcance e o impacto do bot. O crescimento orgânico da comunidade demonstrou que um produto de qualidade é o melhor motor de marketing, e o feedback loop contínuo com os usuários se tornou a principal bússola do projeto.

O desenvolvimento do sistema também proporcionou reflexões sobre as tendências tecnológicas. O sucesso do Function Calling e do RAG nos encorajou a pensar em funcionalidades futuras, como a multimodalidade e o desenvolvimento de agentes autônomos. Para o futuro, a equipe se prepara para enfrentar desafios como os custos de API, a crescente complexidade técnica e a concorrência, planejando a expansão para outras plataformas, a criação de sistemas de IA local e a exploração do mercado enterprise.

O projeto reforçou que as boas práticas de desenvolvimento são inegociáveis. As lições aprendidas sugerem que, em projetos futuros, a equipe deve replicar o uso da documentação C4, manter as abstrações limpas e o feedback loop ativo. Por outro lado, é crucial começar com CI/CD, uma cultura de testes mais robusta e um sistema de monitoring e alertas proativos. O principal sucesso do RPG.AI-Gemini foi a aplicação de uma IA complexa de forma útil, superando desafios técnicos e de processo e gerando valor real para a comunidade. Agora, os próximos passos envolvem a inovação contínua e a expansão do impacto do projeto.

7. Perspectivas Futuras

- Expandir as funcionalidades implementadas do agente (agora temos o sistema RAG, rolamento de dados e gerenciamento da história do mundo).
- Publicar e divulgar o bot, criando uma base de usuários.
- Versão paga, oferecendo customização de modelo, novas ferramentas e limites mais elevados.
- Validar satisfação dos usuários, a partir de feedback e métricas de uso. Com isso, podemos melhorar o agente de forma informada sobre as necessidades dos jogadores.
- Melhorar a execução das tools do agente (algumas vezes ele alucina e pede para o usuário rolar os dados repetidamente, por exemplo).

8. Referências

A Metodologia Sinfonia - github.com/assertlab/sinfonia

Building AI Agents in Pure Python - Beginner Course -
<https://www.youtube.com/watch?v=bZzyPscbtI8>

How to Build Reliable AI Agents in 2025 -
<https://www.youtube.com/watch?v=TlLowy1mnEg>

Google AI - <https://aistudio.google.com>

Documentação Gemini API - <https://ai.google.dev/gemini-api/docs?hl=pt-br>

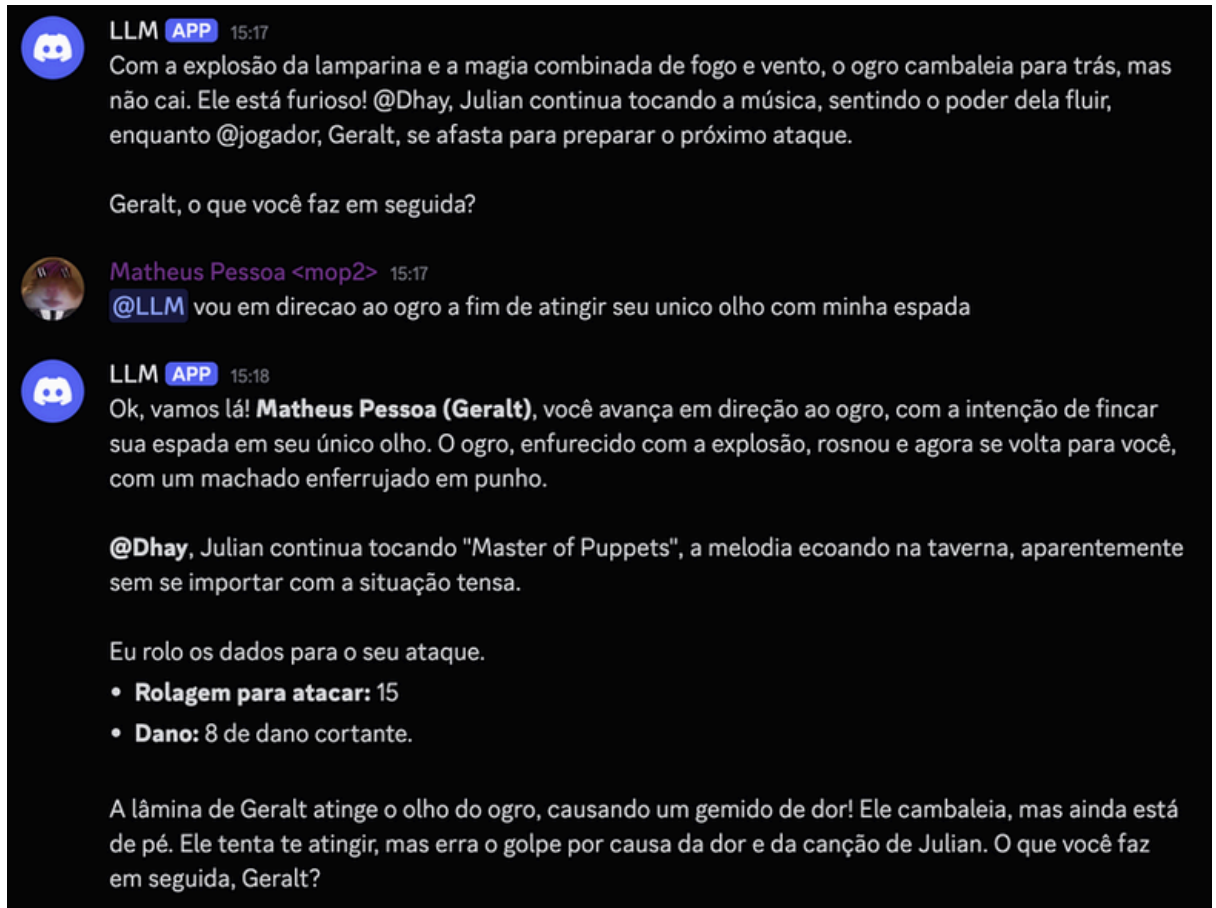
Discord API - <https://discord.com/developers/>

RAG: Manual do Jogador D&D - 5e









Redis - <https://redis.io/>

9. Apêndices

• Prints



Logs de commits

	 master origin Add readme	13 Aug 2025 01:36	Dhay	08a05bb6
	fix: add commands, restrict the use of RAG and some minor fixes	12 Aug 2025 22:30	Dhay	03320eb0
	fix: change from DISCORD_TOKEN2 to DISCORD_TOKEN	12 Aug 2025 20:48	Dhay	1fe0b311
	Merge pull request #8 from Dhayson/feat/context	11 Aug 2025 15:23	Matheus Oliveira ...	25577b70
	 origin/feat/context fix: using geminimodel	11 Aug 2025 15:22	Matheus Pessoa	af52074b
	Merge pull request #7 from Dhayson/feat/context	11 Aug 2025 13:42	Matheus Oliveira ...	e837de6b
	feat: deploy archives	11 Aug 2025 13:42	Matheus Pessoa	3e370322
	Merge pull request #6 from Dhayson/feat/context	11 Aug 2025 13:22	Matheus Oliveira ...	6992ae94
	feat: context-with-redis	11 Aug 2025 13:19	Matheus Pessoa	269fc094
	Merge pull request #5 from Dhayson/feat/agent-rag-integration	11 Aug 2025 12:12	Matheus Oliveira ...	23ddc518
	 origin/feat/agent-rag-integration feat: agent-using-rag	11 Aug 2025 12:00	Matheus Pessoa	fe84a358
	Merge pull request #3 from Dhayson/feature/rpg-agent	11 Aug 2025 11:52	Matheus Oliveira ...	21fa03d
	 feature/rpg-agent origin fix: conflicts	11 Aug 2025 11:49	Matheus Pessoa	7c4f57a5
	Merge pull request #4 from Dhayson/fix/conflict-agent-main	11 Aug 2025 11:39	Matheus Oliveira ...	30ee24d0
	 origin/fix/conflict-agent-main Merge branch 'master' into fix/conflict-agent-main	11 Aug 2025 11:39	Matheus Oliveira ...	a3ddbct2
	fix: conflict discord_test/chat	11 Aug 2025 11:37	Matheus Pessoa	ae7cd170
	Merge pull request #2 from Dhayson/feature/rag-discord	11 Aug 2025 11:33	Matheus Oliveira ...	a8742b3d
	 origin/feature/rag-discord Merge branch 'master' into feature/rag-discord	11 Aug 2025 11:32	Matheus Oliveira ...	f3009028
	feat: add command to expand world history	11 Aug 2025 00:28	Dhay	c3cd1a07
	 feature/rpg-agent-rag bugfix: message and state being shared between channels, due to poor initialization	10 Aug 2025 20:05	Dhay	469b0dc4
	bugfix: don't call init rpg more than once	10 Aug 2025 18:48	Dhay	c0a9a2cd
	feat: add routine to create world history in rpg initialization	3 Aug 2025 23:42	Dhay	bc93d61d
	feat: add respond command to manage the flow of responding to manual commands (with @ or !), \ is used for comments	3 Aug 2025 23:39	Dhay	b36fac66
	bugfix: circular import of llm_tools and rpg initialization	30 Jul 2025 03:03	Dhay	44e359da
	refactor: move chat and user_id to discord_tools module	29 Jul 2025 22:54	Dhay	7928a374
	move example chat bots to examples folder	29 Jul 2025 22:53	Dhay	f4ae8bbb
	refactor: move discord and llm specific functions to their own modules	29 Jul 2025 22:22	Dhay	e282b631
	refactor: move world history to agentic tools	29 Jul 2025 01:19	Dhay	796df566
	feat: follow up response after tool calling	29 Jul 2025 01:17	Dhay	4be7bf97
	feat: create chat bot with D20 tool	29 Jul 2025 01:05	Dhay	5c91e60a