**Universidade Federal de Goiás**

**Instituto de Informática**

**Software Concorrente e Distribuído**

# Documento de Design de Software - Artistic Trivia

**Alany Gabriely**
**André Lopes**
**Artur Lapot**
**Cauã Rebelo**
**Bruno Milioli**
**Filipe Paço**
**Tayna Crisllen**

**Goiânia**

**Julho de 2024**

# 1. Overview do Projeto

## 1.1. Contextualização

Este projeto visa combinar elementos de aprendizado e diversão com a criatividade proporcionada por tecnologias avançadas de inteligência artificial. Utilizando um jogo de perguntas de múltipla escolha, onde os jogadores podem testar seus conhecimentos e competir entre si, juntamente com a capacidade de gerar imagens personalizadas a partir de descrições textuais, criaremos uma plataforma dinâmica e envolvente.

## 1.2. Objetivos do Projeto

Desenvolver uma plataforma interativa que combine um jogo de perguntas de múltipla escolha, integrado ao Telegram. No jogo os usuários poderão testar seus conhecimentos e diversão ao jogar com um oponente onde os dois competem para ser o primeiro a alcançar 5 pontos.

## 1.3. Discussões relevantes

Como ocorreu um atraso muito grande no desenvolvimento das imagens, houve uma discussão entre os membros do grupo sobre a implementação dessa funcionalidade. Outra questão que foi muito discutida era como íamos aplicar a concorrência e distribuição, ou seja , a matéria ministrada na disciplina na construção da plataforma .

## 1.4. Tecnologias utilizadas

API do Telegram Bot ,para integração com o Telegram .

Use Django/Flask ou Node.js/Express, para construir o servidor do aplicativo.

MongoDB/PostgreSQL, para o armazenamento dos dados.

### 1.5. Trabalhos relacionados

https://github.com/Kekko01/Trivia-Bot-Telegram

https://quiz.directory/quiz/JEUXoEuZ

https://github.com/ptkdev/quizquickanswer-telegram-game-bot

## 2. Requisitos

### 2.1. Requisitos de usuário

RU 01 - Usuários podem iniciar um jogo de múltipla escolha e a plataforma escolher o oponente.

RU 02 - Usuários vão responder perguntas com quatro opções de resposta e apenas uma é a resposta correta.

RU 03 - Usuários devem receber feedback imediato se a resposta está correta ou não.

RU 04 - Usuários devem ser notificados quando um jogador alcançar 5 pontos e for declarado vencedor.

### 2.2. Requisitos funcionais

RF 01 - O sistema deve selecionar perguntas aleatórias de um banco de dados e apresentá-las aos jogadores.

RF 02 - O sistema deve validar as respostas dos jogadores e atualizar suas pontuações.

RF 03 - O sistema deve receber e processar comandos de texto dos usuários através do Telegram.

RF 04 - O sistema deve enviar perguntas e receber respostas dos usuários através do Telegram.

## 2.3. Requisitos não funcionais

RNF 01 - O sistema deve ser capaz de lidar com um grande número de usuários simultâneos sem sofrer queda de desempenho.

RNF 02 -O sistema deve garantir alta disponibilidade (pelo menos 99,9%) para que os usuários possam acessar a plataforma a qualquer momento.

RNF 03 - O sistema deve ter uma interface fácil de usar com informações claras e uma navegação intuitiva e clara.

RNF 04 - O sistema deve ser capaz de se recuperar rapidamente de erros e manter a integridade dos dados.

RNF 05 - O sistema deve garantir que somente usuários autenticados possam acessar o jogo.

RNF 06 - O sistema deve proteger os dados dos usuários contra acessos não autorizados.

RNF 07 - O sistema deve evitar fraudes, como a manipulação do jogo da pontuação do jogo.

RNF 08 - O sistema deve operar perfeitamente dentro do ambiente do Telegram, aproveitando ao máximo todas as suas funcionalidades.

RNF 09 - O sistema deve ser compatível com uma variedade de dispositivos (computadores, tablets e celulares).

# 3. Fundamentos de SD relacionados ao projeto

## 3.1. Princípios de Sistemas Distribuídos

Para garantir escalabilidade, flexibilidade e eficiência, o sistema precisa aplicar os conceitos arquitetônicos e de sistemas distribuídos por fundamentos.

- Escalabilidade

O sistema deverá utilizar uma arquitetura baseada em microsserviços onde cada função (autenticação, tratamento de dúvidas, pontuação de etc.) pode ser dimensionada de forma independente e utilizar o equilíbrio de carga para dividir as atividades entre diferentes servidores.

- Disponibilidade

O sistema deverá usar servidores redundantes e técnicas de replicação de dados para garantir que o sistema continue funcionando mesmo em caso de falha de um componente.

- Desempenho e Latência

O sistema deve obter dados acessados com frequência em cache usando Redis ou Memcached, como perguntas e respostas e deve aprimorar consultas do banco de dados e distribuir conteúdo estático através do CDN.

- Consistência

Para atividades críticas, como atualização de resultados, o sistema deverá utilizar transações no banco de dados e implementar sistemas de sincronização para preservar a coerência das informações entre vários servidores.

- Tolerância a Falhas

O sistema deve implementar contingências automáticas de failover e recuperação, utilizando serviços de monitoramento para identificar erros rapidamente e tomar medidas corretivas.

- Segurança

Dados sensibilizados em trânsito e respostas podem ser criptografados logo o sistema deve empregar autorizações e autenticações robustas e incorporar defesa contra-ataques DDoS e outras vulnerabilidades comuns.

- Durabilidade

O sistema deve irá utilizar DevOps e práticas de desenvolvimento ágil e utilizar ferramentas de automação e infraestruturas como código (IaC) para gerenciamento de implantações e atualizações.

- Cooperação

O sistema deve utilizar de APIs RESTful APIs para integração com Telegram e outras plataformas possíveis e garantir compatibilidade com vários dispositivos e navegadores.

## 3.2. Fundamentos de arquitetura de Sistemas Distribuídos e dos estilos arquiteturais

O Frontend (Telegram Bot) será responsável por interagir com os usuários, enviar e receber mensagens deles, exibir perguntas de múltipla escolha.

O Backend (Servidor de Aplicação), é responsável por processar a lógica do jogo, manter o estado do jogo e progressão do jogador.

O Banco de Dados é encarregado de guardar as informações, dados avaliados e histórico dos jogos.

- Estilos Arquiteturais

O sistema deve separar uma aplicação de microsserviços em pequenos serviços independentes, como classificação e perguntas. Cada serviço pode ser desenvolvido, implementado, de forma independente, desenvolvido, implementado, dimensionado e dimensionado.

O sistema deve empregar os eventos para comunicação entre serviços conforme a Arquitetura Orientada a Eventos, principalmente para atualizar os estados do jogo e notificar novos pontos.

Serão implementadas APIs RESTful para a comunicação entre o servidor de aplicação e o Telegram Bot.

## 3.3. Fundamentos de paradigmas de comunicação em

# Sistemas Distribuídos

- Paradigmas de Comunicação

1. Comunicação por passagem de mensagens: A comunicação por passagem de mensagens envolve a troca de mensagens entre processos ou sistemas, em vez de chamadas de procedimentos ou funções. Esse paradigma é especialmente útil em ambientes distribuídos onde os componentes do sistema não compartilham memória comum. Exemplos incluem sistemas de microserviços, onde diferentes serviços se comunicam através de mensagens para realizar tarefas de forma assíncrona ou síncrona. Pode ser implementado usando filas de mensagens, brokers, ou middleware de mensagens.

2. Chamadas de Procedimento Remoto (RPCs): RPC é um paradigma que permite que um programa execute uma função ou procedimento em outro endereço de memória ou sistema, como se fosse uma chamada de função local. Esse tipo de comunicação facilita a interação entre sistemas distribuídos, permitindo que um sistema solicite serviços de outro sistema de forma transparente. RPCs podem ser síncronas ou assíncronas e geralmente envolvem a serialização de dados para transferência pela rede.

3. Comunicação assíncrona: Na comunicação assíncrona, o emissor e o receptor da mensagem não precisam estar sincronizados no tempo. O emissor pode enviar uma mensagem e continuar com outras tarefas, sem esperar por uma resposta imediata. O receptor processa a mensagem em seu próprio tempo. Este tipo de comunicação é útil para sistemas onde a latência ou a disponibilidade não são garantidas, como em sistemas de mensagens e filas de trabalho.

4. Comunicação Síncrona: Na comunicação síncrona, o emissor e o receptor estão sincronizados no tempo, ou seja, o emissor espera até que o receptor processe a mensagem e responda. Esse tipo de comunicação é comum em sistemas onde a resposta imediata é necessária, como em chamadas de função locais ou em algumas implementações de RPCs. A principal desvantagem é que o emissor fica bloqueado esperando pela resposta, o que pode levar a problemas de desempenho se o receptor for lento ou não estiver disponível.

5. Pub/Sub: No modelo Pub/Sub, os produtores de mensagens (publicadores) enviam mensagens a um sistema de mensagens central, que distribui essas mensagens a consumidores (assinantes) interessados. Os assinantes expressam interesse em mensagens de certos tópicos, e o sistema de mensagens garante que eles recebam as mensagens relevantes. Esse paradigma é útil para sistemas onde a distribuição de informações precisa ser escalável e desacoplada, como em sistemas de eventos, notificações ou integração de sistemas distribuídos.

- Aplicação na Plataforma

1. Comunicação por Mensagens

Para executar ações, os componentes trocam mensagens, sem acesso direto à memória.

Entre o servidor de aplicação e o bot do Telegram: O bot avisa o servidor de aplicação da resposta quando um usuário responde a uma pergunta.

2. RPC (Remote Procedure Call)

A função de um programa pode ser realizada em outro espaço de endereço, normalmente em outro servidor, como o uso do RPC (Remote Procedure Call).

3. Comunicação Assíncrona

O sistema continua a operar enquanto a resposta é enviada, permitindo que o sistema continue funcionando enquanto espera.

Bot do Telegram e o Servidor de Aplicação: as respostas do usuário podem ser processadas de forma assíncrona, com atualizações enviadas ao bot quando estiverem disponíveis.

4. Comunicação Síncrona

O remetente espera pela resposta do destinatário antes de prosseguir com o processamento. Entre o servidor de aplicação e o bot do Telegram: para atividades críticas que exigem feedback rápido, como validação instantânea dos resultados dos usuários.

5. Pub/Sub (Publish/Subscribe)

Uma comunicação de estratégia, em que os publicadores enviam mensagens para um canal sem saber quem são os assinantes. Os receptores registram o interesse nos canais.

Notificações de Pontuação e Atualizações de Jogo: quando um usuário alcança um novo ponto ou termina um jogo, o servidor de aplicação publica um evento de atualização de jogo, e o bot do Telegram, que está inscrito nesses eventos, envia notificações aos usuários.

## 3.4. Robustez em sistemas distribuídos, nomeação, coordenação, consenso, consistência, replicação e tolerância a falhas

- Nomeação

Identificadores Únicos: atribuir identificadores únicos a cada usuário, sessão de jogos.

Serviço de Nomeação: para mapear nomes lógicos para endereços físicos, como DNS para serviços internos.

- Coordenação

Zookeeper/etcd: para gerenciar a configuração distribuída, bloqueios de exclusão mútua e sincronização entre componentes durante a coordenação de serviços.

Gerenciamento de Sessão: coordenar entre os vários jogadores usando um coordenador central que garante que as respostas sejam processadas na ordem correta.

- Consenso

Algoritmos de Consenso**:** Raft ou Paxos para garantir que todos os participantes do sistema participem em estados críticos, tais como a pontuação do jogo ou o estado atual da partida.

Coordenação de Pontuação: garantir que a pontuação dos jogadores seja consistente e acordada por todos os nós envolvidos no processamento do jogo.

- Consistência

Consistência Eventual: operações não críticas onde a latência é mais importante, como atualizações de perfil de usuário.

Consistência Forte: para dados críticos como pontuação e estado do jogo, é importante que tenhamos a mesma visão do estado entre todos.

- Replicação

Banco de Dados Replicado**:** para armazenar questões, respostas e pontuações.

- Tolerância a Falhas

Detecção de falhas: implementar mecanismos para identificar falhas de componentes usando ferramentas, como Heartbeat ou outros sistemas de monitoramento.

Failover Automático: para que quando um de nós falhar, o tráfego seja redirecionado para nós com segurança.

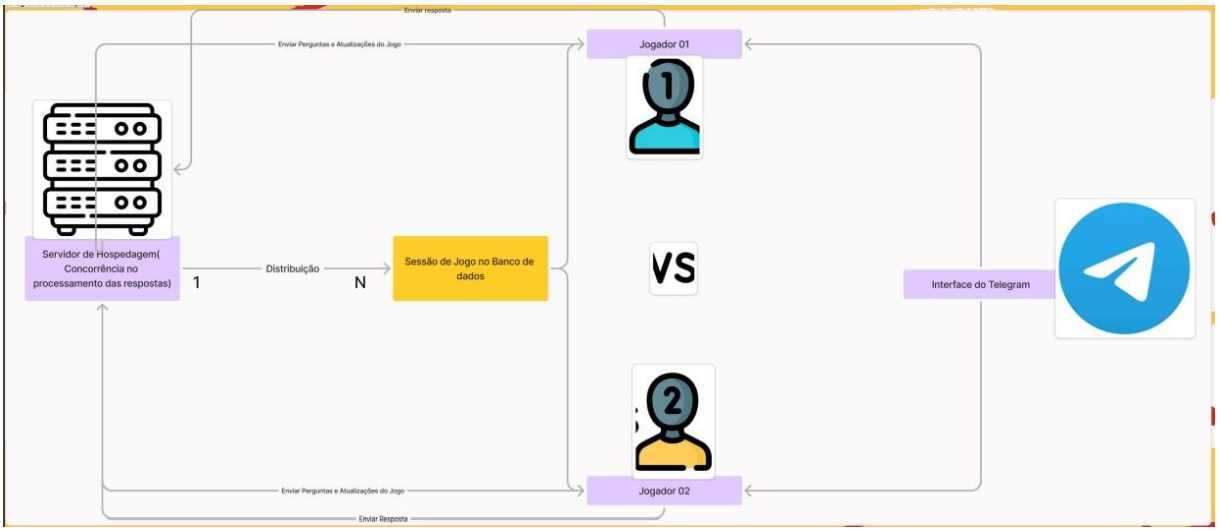
Retry Mechanism: implementar mecanismos de tentativa e erro para operações críticas que podem falhar devido a problemas temporários.

# 4. Resultados

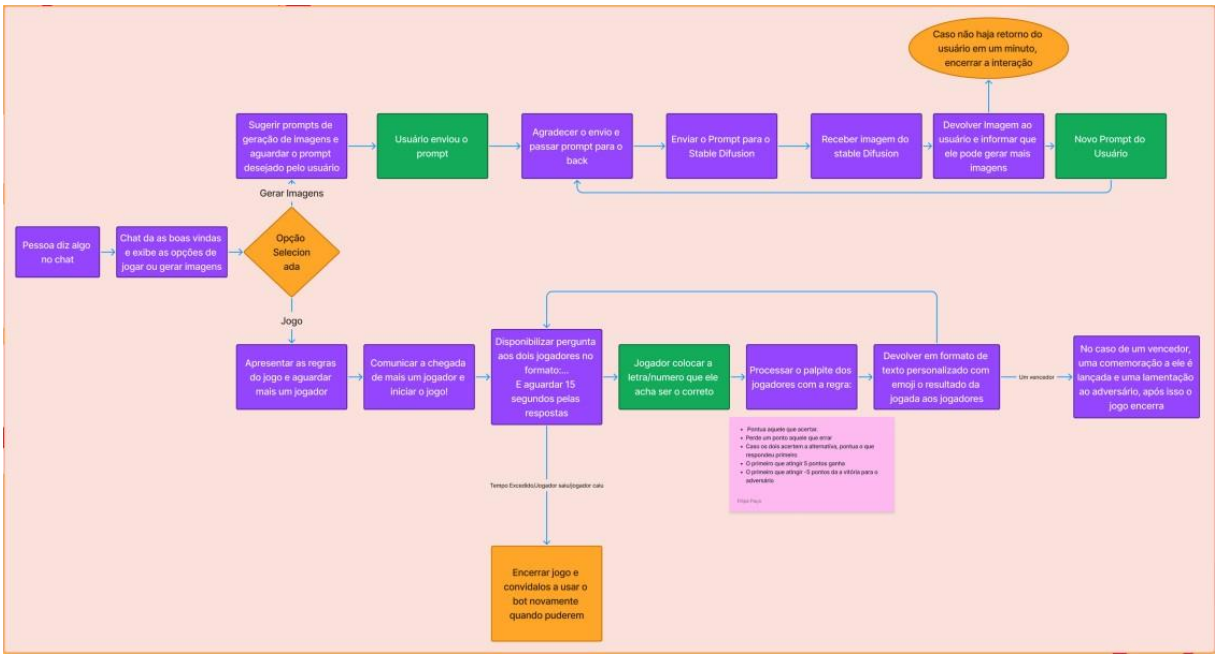
## 4.1. Design arquitetural

Telegram Bot (Frontend)
Servidor de Aplicação (Backend)
Banco de Dados

## 4.2. Diagrama de Iteração




## 4.3. Diagrama de Fluxo



# 5. Limitações, trabalhos futuros e perspectiva do projeto

## 5.1. Limitações

As limitações e desafios que ao desenvolver um sistema que combina um jogo de perguntas de múltipla escolha integrado ao Telegram são:

- Limitações Técnicas

Limites de taxa na API do Telegram podem afetar a capacidade de resposta do bot quando muitos usuários estão ativos ao mesmo tempo.

- Escalabilidade

Manter o estado do jogo para vários usuários simultâneos pode ser desafiador e exigir uma arquitetura de distribuição de dados eficaz. É fundamental dividir a carga de trabalho entre os servidores de maneira eficiente, mas isso pode ser difícil, principalmente se houver picos de usos inesperados.

- Consistência e Sincronização

Garantir que o estado jogo e a pontuação sejam mantidos de forma consistente ao longo de várias sessões e dispositivos pode ser desafiador, principalmente em um ambiente distribuído. Um desafio técnico importante é fornecer atualizações em tempo real para ambos os jogadores sem atrasos perceptíveis.

- Segurança e Privacidade

Proteger os dados pessoais e as atividades dos usuários contra acessos não autorizados.

Custos Operacionais

Manter o sistema operando e fornecer suporte ao usuário pode consumir muito tempo e exigir uma equipe dedicada o que pode gerar um alto custo.

- Dependência de Terceiros

O sistema é altamente dependente da API Telegrama, e quaisquer alterações ou limitações na API podem afetar a funcionalidade do sistema.

Continuar acompanhando as versões de melhorias dos modelos de IA pode ser desafiador e exigir investimento.

- Legalidade e Conformidade

Garantir que o sistema esteja em conformidade com os regulamentos de proteção de dados (como o GDPR) e outras leis pertinentes é crucial.

Confiabilidade e Disponibilidade

Garantir a disponibilidade do sistema durante a manutenção e atualizações pode ser um desafio.

## 5.2. Trabalhos futuros

Tem muitas áreas com potencial para expandir o uso da plataforma, como por exemplo: a educação que pode utilizar para reforço escolar, aulas de arte e cursos de idiomas, na publicidade com promoções e concursos, na saúde com treinamentos cognitivos, , entre outras áreas que vão proporcionar vários benefícios à sociedade.