Proposta de Projeto: Plataforma Colaborativa de Pixel Art Distribuída

1. Proposta Geral da Aplicação

O projeto consiste no desenvolvimento de uma plataforma de arte colaborativa em tempo real (inspirada no Wplace.live), onde múltiplos usuários compartilham uma grade de pixels persistente.

Cenário de Uso:

- Ao acessar a aplicação web, o usuário recebe instantaneamente o estado atual do quadro (Canvas), visualizando desenhos feitos previamente por outros participantes.
- O usuário seleciona uma cor em uma paleta predefinida e clica em um pixel na grade.
- A alteração é enviada ao servidor, persistida e propagada em tempo real para a tela de todos os outros usuários conectados via WebSocket, garantindo sincronia visual sem a necessidade de recarregar a página.

2. Tecnologias Adotadas

O sistema adota uma arquitetura de microsserviços containerizados para garantir modularidade e escalabilidade.

- **Linguagem de Backend:** Python
- **Frontend:** React (Vite) + HTML5 Canvas API
- **API Gateway:** FastAPI (Python) atuando como gerenciador de WebSockets e cliente gRPC.
- **Core Service:** Servidor gRPC (Python) contendo a regra de negócios.
- **Protocolos & Comunicação:**
    - **gRPC (Protobuf):** Utilizado para comunicação síncrona e tipada entre o *Gateway* e o *Core Service* (comandos de escrita e leitura de estado).
    - **WebSocket:** Para comunicação bidirecional assíncrona entre Navegador e Servidor.
    - **Redis Pub/Sub:** Para broadcast de eventos (mensageria interna).
- **Persistência:** Redis (In-memory database) para armazenamento da matriz de pixels e controle de concorrência.

- **Infraestrutura:** Docker e Docker Compose.

3. Justificativa

A arquitetura foi projetada para demonstrar conceitos distribuídos da disciplina:

1. **Chamada Remota de Procedimento (RPC):** O uso do gRPC exemplifica como sistemas distribuídos invocam métodos em servidores remotos de forma transparente, utilizando Protocol Buffers para serialização eficiente de dados, desacoplando a interface (Gateway) da lógica de negócios (Core).
2. **Comunicação Orientada a Eventos (Pub/Sub):** A utilização do Redis Pub/Sub demonstra o desacoplamento temporal e espacial. O serviço que altera o pixel (Core) não precisa conhecer quem está conectado; ele apenas publica o evento, e os interessados (Gateways) reagem a ele.
3. **Heterogeneidade e Interoperabilidade:** O sistema conecta um cliente Web (JavaScript) a serviços Backend (Python) através de protocolos padrão (HTTP/WebSocket/gRPC), mostrando como diferentes tecnologias cooperam em um ambiente distribuído.
4. **Consistência e Concorrência:** O projeto aborda o problema de múltiplos agentes tentando alterar o mesmo estado global (o Canvas) simultaneamente, delegando a atomicidade das operações ao Redis.

4. Equipe e Responsabilidades Técnicas

Desenvolvedor Responsável: Thiago Dias Ferreira

Responsável por todo o ciclo de vida da aplicação (End-to-End), com as responsabilidades divididas por camadas:

- **Camada de Infraestrutura & Arquitetura (DevOps):**
    - Definição do contrato de interface (.proto) para tipagem estrita dos dados.
    - Orquestração dos contêineres via Docker Compose.
    - Configuração da rede interna do Docker e exposição de portas.
- **Camada de Backend (Core & Dados):**
    - Implementação do servidor gRPC (Core Service).

    - Modelagem de dados no Redis (uso de Hashes para persistência).
    - Implementação da lógica de publicação de mensagens (Publisher).
- **Camada de Integração (Gateway):**
    - Desenvolvimento do servidor FastAPI para gerenciar conexões persistentes (WebSockets).
    - Implementação do *Stub* gRPC para comunicação com o Core.
    - Lógica de assinatura de eventos (Subscriber) e broadcast para clientes conectados.
- **Camada de Frontend (Cliente):**
    - Desenvolvimento da interface reativa com React.
    - Implementação da lógica de desenho no Canvas HTML5.
    - Integração do WebSocket para envio de comandos e recebimento de atualizações.

5. **Repositório do GitHub**
   thiago9852/pixel-art