ChatGPT

Plano de Ação

Para organizar o desenvolvimento do projeto de microfrontends, dividimos em fases sequenciais. Cada fase contém tarefas detalhadas de configuração, implementação e aprendizado prático, seguindo a arquitetura proposta.

Fase 1: Setup do Monorepo e Ferramentas Básicas

- **Criar monorepo:** inicializar repositório Git e estruturar um monorepo (por exemplo, com Yarn Workspaces ou PNPM). Criar pastas iniciais para cada parte (host, MFEs, backend, biblioteca compartilhada, infra).
- **Configurar ferramentas de desenvolvimento:** instalar Node.js (>=16), Yarn/npm e Python (para FastAPI). Configurar ESLint/Prettier globais e TypeScript se for usado.
- **Configurar build tools:** criar package.json principal com *workspaces* apontando para subprojetos (host, MFEs, libs, backends). Garantir que cada pacote tenha seu próprio package.json. Estudar conceitos de *monorepo* e *workspaces* (monorepos facilitam compartilhar dependências e scripts entre projetos).

Fase 2: Configuração do Host Next.js + Module Federation

- **Criar container host:** iniciar aplicação Next.js (`npx create-next-app host-container`). Este será o shell principal que carrega os microfrontends.
- **Instalar Module Federation:** adicionar dependência `@module-federation/nextjs-mf` e configurar o plugin `NextFederationPlugin` em `next.config.js` do host (conforme exemplo do guia) [1]. Isso habilita o carregamento dinâmico dos remotos no Next.js.
- **Definir exportações do host:** no `NextFederationPlugin`, definir `name: 'host'`, apontar `filename` e (mesmo que vazio) listar módulos a expor, além de declarar `remotes` dos próximos MFEs a serem implementados.
- **Testar configuração inicial:** rodar `npm run dev` no host e verificar que o Webpack está carregando sem erros de Module Federation. Criar uma rota ou componente básico no host para servir de container.
- **Integração de autenticação básica:** instalar NextAuth.js e adicionar provider inicial (ex: GitHub para teste) para familiarizar com fluxos de login. Isso prepara o terreno para integrar Cognito depois.

Fase 3: Biblioteca de Componentes Compartilhados (UI Library)

- **Criar biblioteca de UI:** dentro do monorepo, criar um pacote (ex: `shared-ui`) para componentes React reutilizáveis. Inicializar projeto com `package.json`, React e TypeScript/ configurações comuns.
- **Configurar Storybook:** adicionar Storybook ao `shared-ui`, definindo pastas de stories para cada componente. O Storybook permite desenvolver componentes de forma isolada [2].
- **Integrar Chromatic e Changesets:** configurar publicação de Storybook no Chromatic (para testes visuais) e estabelecer Changesets para versionamento semântico. Criar fluxo de PR ou CI que atualize o Storybook automaticamente.

- **Publicar componente básico:** criar e documentar um botão estilizado como exemplo. Executar Storybook (`npm run storybook`) e verificar renderização. Esse processo ensina a criar componentes reutilizáveis e documentá-los.

Fase 4: Microfrontend GeoGrid (Jogo de Bandeiras)

- **Criar app GeoGrid:** gerar nova aplicação React usando Vite (`pnpm create vite mfe-geogrid --template react`). Configurar TypeScript, se desejado.
- **Implementar lógica do jogo:** desenvolver a interface de um jogo de bandeiras em grade 3x3. Exibir bandeiras (imagens ou emojis) e permitir seleção com regras por linha/coluna. Essa parte foca em React, estado e lógica de jogo.
- **Configurar Module Federation no MFE:** instalar `vite-plugin-federation` (ou configurar outra forma) para expor o componente principal do jogo ao host. Em `vite.config.js`, usar a federação para expor o módulo (ex: `./GeoGridComponent`).
- **Consumir no host:** no `next.config.js` do host, adicionar referência ao remote GeoGrid (endereço local, ex: `geogrid@http://localhost:3001/assets/remoteEntry.js`). No host, importar dinamicamente o componente do GeoGrid e renderizá-lo em uma rota ou página.
- **Testar integração:** executar host e GeoGrid em portas diferentes (ex: 3000 e 3001) e verificar que o componente do jogo carrega no host via Module Federation. Ajustar compartilhamento de dependências (React, etc) para evitar duplicação.

Fase 5: Microfrontend Removedor de Background

- **Criar app de upload:** gerar aplicação React com Vite (`pnpm create vite mfe-bg-remover --template react`).
- **Implementar UI de upload:** criar interface simples para o usuário enviar uma imagem (input file) e exibir o resultado com fundo removido. Incluir área de pré-visualização.
- **Integrar serviço de IA:** no frontend, ao enviar imagem, fazer requisição para o backend Python (FastAPI) que processará a imagem. Usar fetch/axios para `POST /remove-background`.
- **Configurar Module Federation:** expor o componente principal do Removedor ao host (similar ao passo anterior). Consumir via host em rota dedicada.
- **Testar fluxo completo:** garantir que o upload do front chama o FastAPI corretamente e exibe resultado. Aprender a comunicar front React com backend Python e manipular dados binários (imagens).

Fase 6: Microfrontend Ambiente de Foco

- **Clonar app de produtividade:** criar aplicação React via Vite (`pnpm create vite mfe-focus --template react`). Implementar ou refatorar um projeto de produtividade existente (ex: um Pomodoro clone).
- **Desenvolver funcionalidades básicas:** focar em tempo de foco, listas de tarefas e qualquer recurso chave do projeto escolhido. Esse MFE serve para praticar integração de estados e rotas internas.
- **Configurar Module Federation:** expor módulos/rotas relevantes ao host. No host, adicionar acesso a esse ambiente (por exemplo, via botão de menu).
- **Validar integração:** testar a navegação do host para o MFE de Foco, mantendo estado ou autenticação compartilhada se aplicável.

Fase 7: Backend NestJS (API Principal)

- **Iniciar projeto NestJS:** usar CLI (`nest new backend-api`) para criar esqueleto do servidor. Aprender a estrutura modular do Nest [3].
- **Configurar banco de dados:** no Nest, instalar e configurar TypeORM (ou Prisma) com PostgreSQL. Criar entidades iniciais (usuário, pontuações do GeoGrid, tarefas, etc). Estabelecer conexão e migrations iniciais.
- **Criar endpoints essenciais:** implementar rotas REST (ou GraphQL) para autenticação (login, registro), salvar dados do jogo GeoGrid, gerenciar tarefas do foco, etc. Usar decorators e serviços do Nest para modularidade.
- **Integrar Redis/Mongo:** definir propósito de Redis (ex: cache de sessão ou fila) e MongoDB (ex: guardar documentos não-relacionais). Configurar conexões e testar. Esse passo ensina sobre arquitetura de microserviços de dados.
- **Testar com Postman:** garantir que a API responde corretamente. Expor endpoints no host Next.js (API Routes) ou comunicar diretamente pelo domínio do Nest.

Fase 8: Backend FastAPI (Processamento de Imagens)

- **Configurar projeto FastAPI:** criar venv Python, instalar FastAPI e Uvicorn. Estruturar um app básico em `main.py`.
- **Implementar lógica de remoção de fundo:** usar bibliotecas como rembg ou OpenCV/DNN para processar a imagem recebida. Criar rota `POST /remove-background` que recebe um arquivo e retorna imagem PNG com fundo removido.
- **Gerenciar dependências:** verificar performance (Uvicorn com Workers) e limites de tamanho de upload. Documentar com docs interativos do FastAPI (Swagger/UI).
- **Containerizar serviço:** escrever Dockerfile Python para FastAPI. Aprender como isolar o serviço de ML. Testar execução local e API.

Fase 9: Banco de Dados e Cache

- **Dockerizar bancos:** criar containers Docker (via Compose ou k8s) para PostgreSQL, MongoDB e Redis. Predefinir usuários e senhas (armazenar em `env`).
- **Configurar NestJS e serviços:** ajustar conexões no NestJS para usar os containers locais. Verificar acesso ao Redis (p.ex. para sessões ou filas).
- **Rodar migrates e semear dados:** executar migrações do TypeORM (ou Prisma) para criar tabelas. Inserir dados iniciais de teste (ex: bandeiras cadastradas, usuários).
- **Testar conexões:** garantir que todos os serviços conseguem ler/escrever no respectivo DB. Aprender sobre orquestração de múltiplos DBs e suas utilidades (relacional vs doc vs cache).

Fase 10: Autenticação (NextAuth.js, JWT, Cognito)

- **Configurar AWS Cognito:** criar User Pool e App Client no console AWS Cognito. Configurar domínio de Hosted UI e redirecionamentos.
- **Integrar NextAuth.js:** no host Next.js, instalar NextAuth e configurar `providers: [CognitoProvider({...})]` [4]. Ajustar variáveis de ambiente (`COGNITO_CLIENT_ID`, `COGNITO_ISSUER`, `COGNITO_CLIENT_SECRET`).
- **Fluxo de login:** criar páginas de login/callback do NextAuth. Proteger rotas sensíveis (usando middleware ou Higher-Order Components).

- **JWT e sessão:** usar JWTs para manter sessão. Configurar o `jwt` e `session` callbacks no NextAuth para incluir informações do usuário. Autenticar requisições do frontend no backend Nest com esses tokens.
- **Testes de segurança:** garantir que somente usuários autenticados acessem certos endpoints (por ex. salvar pontuação no GeoGrid). Entender ciclo de vida da autenticação em SPAs e SSR.

Fase 11: Docker + Kubernetes (Desenvolvimento Local)

- **Criar Dockerfiles:** escrever Dockerfile para cada serviço (host, cada MFE, NestJS, FastAPI). Usar multi-stage builds para frontends, instalar apenas dependências de produção.
- **Configurar Kubernetes local:** escolher ferramenta (minikube, kind ou k3d). Criar manifests para **Deployments**, **Services** e **Ingress** (ou Ingress Nginx) para todos os serviços.
- **Definir rede e portas:** mapear contêineres para portas (ex: host:80, geogrid:3001, etc). Configurar env vars nos pods (ex. variáveis de conexão de DB e Cognito).
- **Testar cluster local:** aplicar `kubectl apply -f infra/` e verificar pods rodando. Acessar pelo navegador via `localhost`. Resolver eventuais problemas de configuração e aprender orquestração de contêineres [5] .

Fase 12: CI/CD com GitHub Actions

- **Configurar pipelines:** criar workflows no GitHub Actions que executem lint, testes unitários e de integração a cada push.
- **Deploy automático:** configurar Actions para buildar as imagens Docker e publicá-las num container registry (ex: GitHub Container Registry ou AWS ECR).
- **Testes de UI:** integrar execução do Storybook/Chromatic. Toda vez que componentes mudarem, executar testes visuais no Chromatic para detectar diferenças.
- **Versão e releases:** usar Changesets para atualizar versão da biblioteca UI e demais pacotes. Criar workflow para gerar release notes automaticamente a partir de PRs.
- **Notificações:** configurar badges em README para status do build, cobertura de testes e versão do pacote.

Fase 13: Deploy em Produção (AWS ECS/Fargate)

- **Preparar AWS:** configurar repositório ECR e criar cluster ECS com Fargate. Definir Task Definitions para host, backends (NestJS/ FastAPI) e possivelmente sites estáticos.
- **Banco de dados na nuvem:** provisionar RDS (PostgreSQL) e instâncias MongoDB (p. ex. DocumentDB ou Mongo Atlas) e Redis (Amazon ElastiCache). Atualizar strings de conexão.
- **Load Balancing e Domínio:** criar Service no ECS com ALB (Application Load Balancer). Configurar domínio customizado e SSL (via ACM).
- **Parâmetros de produção:** ajustar escala (vCPUs/memória), auto-scaling no Fargate e políticas de deployment (rolling updates). Testar migração de dados para o ambiente prod.
- **Monitoramento:** integrar CloudWatch para logs e métricas. Configurar alertas básicos (e.g. para erros ou alta latência). Este passo foca em entender deploy em escala e “infra como código”.

Fase 14: Documentação Final e Portfólio

- **Escrever README.md:** elaborar o README principal (como abaixo) com visão geral, estrutura, instruções de instalação e arquitetura. Certificar-se de que ele reflita todo o aprendizado e decisões.

- **Documentar arquitetura:** incluir diagramas ASCII ou links para docs no repo sobre como os serviços se comunicam.
- **Roadmap do projeto:** descrever funcionalidades futuras (e.g. melhorias no jogo, novas integrações, testes automáticos adicionais). Essa seção demonstra planejamento de evolução.
- **Apresentação no portfólio:** preparar um ambiente demo (ex: hospedado ou vídeo) e destacar o projeto em portfólio, ressaltando o uso de microfrontends, CI/CD e cloud. Atualizar links no README (Storybook, DockerHub, etc).

README.md Inicial

```
# Projeto MicroFrontends (Portfólio)

## Visão Geral
Este projeto demonstra uma arquitetura de **Micro-Frontends** integrada, com um *host* em Next.js que carrega três micro-aplicações independentes em React/Vite. As MFEs são: **GeoGrid** (jogo de bandeiras em grade 3x3), **Removedor de Background** (upload de imagem para remoção de fundo via IA) e **Ambiente de Foco** (app de produtividade inspirado em projeto existente). Há também uma biblioteca compartilhada de componentes React documentada com Storybook, promovendo código reaproveitável e testes visuais [2].

**Características principais:**
- Arquitetura baseada em [Module Federation](https://webpack.js.org/concepts/module-federation/) do Webpack 5, permitindo carregar dinamicamente componentes de outras aplicações [6] [7].
- Container host em **Next.js** (SSR/SSG) orquestrando as MFEs.
- Cada MFE em **React 18** usando **Vite** para bundling e desenvolvimento rápido.
- Backend principal em **NestJS** (API em Node.js/TypeScript) [3] e serviço de remoção de fundo em **FastAPI** (Python de alta performance) [8].
- Persistência de dados com **PostgreSQL** (dados relacionais), **MongoDB** (dados documentais) e **Redis** (cache/mensageria).
- Autenticação centralizada com **NextAuth.js**, JWT e Amazon **Cognito** (OAuth) [4].
- Deploy local com **Docker + Kubernetes** e produção em **AWS ECS/Fargate** (computação serverless de containers) [9] [5].
- CI/CD automatizado com **GitHub Actions** para build, testes, deploy e versionamento (Changesets) da biblioteca UI.

## Estrutura de Diretórios
- `/host` – Aplicação Next.js (host) com configuração de Module Federation.
- `/mfes/geogrid` – Microfrontend React/Vite do jogo de bandeiras (GeoGrid).
- `/mfes/bg-remover` – Microfrontend React/Vite do removedor de background.
- `/mfes/focus` – Microfrontend React/Vite do ambiente de foco (produtividade).
- `/shared-ui` – Biblioteca de componentes React compartilhados + Storybook.
- `/backend/nestjs` – Backend NestJS (API principal).
- `/backend/fastapi` – Backend FastAPI (serviço de IA de remoção de fundo).
- `/infra` – Arquivos de infraestrutura (Docker, manifests Kubernetes,
```

```
configs AWS, etc).
- `/docs` – Documentação adicional (ex: diagramas, especificações).

## Tecnologias Utilizadas
- **Front-end:** Next.js, React 18, Vite, Module Federation (Webpack 5), Storybook, Chromatic, CSS-in-JS ou Tailwind.
- **Back-end:** NestJS (TypeScript/Node.js) [3], FastAPI (Python) [8], PostgreSQL, MongoDB, Redis.
- **Autenticação:** NextAuth.js, JWT, Amazon Cognito, OAuth 2.0.
- **Infraestrutura:** Docker, Kubernetes (local), AWS ECS Fargate [9], AWS RDS/DocumentDB/ElastiCache.
- **DevOps:** GitHub Actions (CI/CD), Changesets (versionamento), Yarn Workspaces/PNPM (monorepo), AWS CLI.

## Instalação e Execução Local
1. **Requisitos:** instale Node.js (>=16) e Yarn ou PNPM, além de Docker (para bancos de dados e serviços) e kubectl/kind para orquestração local.
2. **Clonar repositório:** `git clone <URL-do-repo> && cd <repo>`.
3. **Instalar dependências:** executar `yarn install` na raiz para instalar todos os pacotes.
4. **Configurar variáveis de ambiente:** criar arquivos `.env` conforme exemplos, definindo strings de conexão dos bancos, chaves JWT e credenciais do Cognito (COGNITO_CLIENT_ID, COGNITO_ISSUER, etc).
5. **Rodar serviços localmente:**
   - Inicie bancos de dados e serviços auxiliares via Docker/Kubernetes (`docker-compose up` ou `kubectl apply -f infra/`).
   - Em terminais separados, rode `yarn dev` em `/host`, `/mfes/geogrid`, `/mfes/bg-remover`, `/mfes/focus`, `/backend/nestjs` e `/backend/fastapi`. Cada frontend abrirá numa porta (ex: host em 3000, geogrid em 3001, etc).
6. **Acessar a aplicação:** abra `http://localhost:3000` para ver o host e navegar entre os microfrontends via menus. Os dados usados nos jogos e funcionalidades estarão salvos no banco local (PostgreSQL/Mongo).
7. **Executar testes:** rode `yarn test` nos pacotes que contêm testes (por exemplo, componentes do `shared-ui`).

## Arquitetura
A aplicação segue uma arquitetura de microfrontends **baseada em containers independentes**. O *host* em Next.js utiliza o Module Federation para carregar as MFEs React/Vite como remotos em tempo de execução, compartilhando bibliotecas comuns (como React) para evitar duplicação [7] [6]. Os MFEs rodam em seus próprios domínios internos e expõem componentes/rotas para o host usar. O backend é composto por dois serviços: um servidor NestJS monolítico (API centralizada, gerencia lógica de negócios e banco de dados relacional) e um serviço de FastAPI em Python para processamento de imagem (remoção de fundo) usando IA. Os dados são distribuídos entre PostgreSQL (dados relacionais), MongoDB (dados não-relacionais) e Redis (cache/filas). Para o ambiente local, usamos **Docker e Kubernetes** para orquestrar todos os serviços juntos. Em produção, cada serviço é executado em containers no **AWS ECS/Fargate** (serverless) [9], garantindo escalabilidade. A autenticação é gerenciada por NextAuth.js (node) usando JWT, integrado ao Amazon Cognito [4].
```

```
## Como Executar Localmente
- **Via scripts:** com todos os serviços em execução (veja seção de instalação), pode-se executar `yarn dev` no host e em cada MFE. O host acessa automaticamente os remotos.
- **Via Docker/Kubernetes:** ajustar o `docker-compose.yaml` (ou manifests K8s) na pasta `/infra` para subir todos de uma vez. Usar `docker compose up --build` ou `kubectl apply -f infra/`.
- **Database:** caso prefira, é possível conectar serviços a instâncias externas de banco (ex: RDS/Postgres, Mongo Atlas). Ajustar variáveis de ambiente para isso.
- **Storybook:** navegar até `/shared-ui` e rodar `yarn storybook`. O Storybook estará disponível em `http://localhost:6006`, exibindo componentes documentados.
- **Testes visuais:** caso tenha Chromatic configurado, ele irá gerar snapshots das histórias a cada mudança e enviar relatórios.

## Links Relevantes
- [Documentação oficial do Next.js](https://nextjs.org/docs) e [Module Federation](https://webpack.js.org/concepts/module-federation/) [1].
- [NextAuth.js com Cognito](https://next-auth.js.org/providers/cognito) [4].
- [FastAPI (Python) – documentação oficial](https://fastapi.tiangolo.com/) [8].
- [Documentação NestJS](https://docs.nestjs.com/) [3].
- [Storybook](https://storybook.js.org/) e [Chromatic](https://www.chromatic.com/) para integração visual [2].
- [AWS Fargate](https://aws.amazon.com/fargate/) e [Kubernetes](https://kubernetes.io/) para deploy.

## Roadmap
- Adicionar testes end-to-end (Cypress/Playwright) integrando front e API.
- Implementar internacionalização (i18n) no host e MFEs.
- Melhorar acessibilidade (a11y) em todos os componentes.
- Expandir o jogo GeoGrid com placar global e modos de dificuldade.
- Otimizar serviço de remoção de fundo (ex.: usar GPU ou modelos avançados).
- Migrar banco de dados de desenvolvimento para ambientes cloud (e.g. RDS, DocumentDB, ElastiCache).
- Refatorar o ambiente de foco adicionando notificações e estatísticas de produtividade.
- Publicar site estático de documentação e demo do projeto.
```

Este README inicial resume a arquitetura, instalação e uso do projeto. À medida que o desenvolvimento avança, deve-se atualizar detalhes específicos (como URLs de serviços, chaves de API) e expandir a documentação conforme necessário.

---

[1] [7] NextJS Module Federation Quickstart Guide
https://nextjsstarter.com/blog/nextjs-module-federation-quickstart-guide/

[2] Setup • Chromatic docs
https://www.chromatic.com/docs/storybook/

3 Documentation | NestJS - A progressive Node.js framework
https://docs.nestjs.com/

4 Amazon Cognito | NextAuth.js
https://next-auth.js.org/providers/cognito

5 Is Kubernetes an Operating System? Demystifying the Confusion
https://botkube.io/learn/is-kubernetes-an-operating-system

6 Micro-frontend with Module Federations [Part 1] - Vite + React - DEV Community
https://dev.to/kevin-uehara/micro-frontend-with-module-federations-part-1-vite-33nd

8 FastAPI
https://fastapi.tiangolo.com/

9 Serverless Compute Engine – AWS Fargate – AWS
https://aws.amazon.com/fargate/