PLANO DE DESENVOLVIMENTO – PORTFÓLIO SPA (React, TS, Firebase)

Contexto
- Aplicação SPA para portfólio.
- Tecnologias: React, TypeScript, Context API, styled-components, i18n, Firebase (Auth + Firestore).
- Já implementado: NavBar completa, botão de troca de idioma, botão de tema dark/light.

Objetivo do plano
Organizar o desenvolvimento em fases para evitar retrabalho, mantendo:
- Boa arquitetura
- Uso correto de i18n
- Separação de UI e conteúdo
- Possibilidade de expansão futura
- Uso dos recursos gratuitos do Firebase

VISÃO GERAL DAS FASES

1. Arquitetura de pastas e rotas base
2. i18n bem estruturado (conteúdo estático)
3. Modelagem de domínio (types) + dados mockados
4. Configuração do Firebase
5. Camada de serviços (Firestore + Auth)
6. Leitura de conteúdo dinâmico (projetos/skills) a partir do Firestore
7. Internacionalização do conteúdo dinâmico
8. Autenticação + rotas protegidas
9. Dashboard (CRUD de projetos/skills)
10. Regras de segurança, testes e polimentos finais

FASE 1 – ARQUITETURA DE PASTAS E ROTAS BASE

Objetivo
Ter o esqueleto da aplicação pronto, com páginas e layout base, sem lógica de dados.

Tarefas
- Adicionar React Router (se ainda não estiver configurado).
- Organizar estrutura de pastas de forma mais orientada a features, por exemplo:

```
src/
  assets/
  components/
  features/
    home/
    projects/
    skills/
    contact/
    dashboard/
      pages/
      components/
  context/
  hooks/
```

services/
styles/
types/
utils/
i18n/
App.tsx
main.tsx

- Definir as rotas principais:
  - "/" → Home
  - "/projects"
  - "/skills"
  - "/contact"
  - "/dashboard/login"
  - "/dashboard" (protegida – placeholder no início)

- Criar páginas simples com conteúdo estático (ex.: “Em construção”) apenas para garantir:
  - Funcionamento da NavBar existente
  - Navegação entre rotas
  - Integração do tema dark/light

Não fazer ainda
- Nada de Firebase.
- Nada de dashboard complexo.
- Nada de i18n para conteúdo dinâmico.

FASE 2 – I18N ESTRUTURADO (CONTEÚDO ESTÁTICO)

Objetivo
Deixar o sistema de tradução da interface bem organizado antes de integrar com dados dinâmicos.

Tarefas
- Criar pasta src/i18n com:
  - index.ts (configuração do react-i18next)
  - Arquivos de tradução por namespace, por exemplo:
    - common.json (labels gerais: botões, mensagens genéricas)
    - navbar.json
    - home.json

- Mover conteúdos atuais de tradução (pasta translateFile/en e pt) para essa estrutura, se fizer sentido.
- Garantir que:
  - O botão de idioma altera i18n.language.
  - As páginas usam t("chave") para textos fixos (títulos, menus, labels, etc.).

Exemplo de uso
- Em uma página Home:

```
const { t } = useTranslation("home");
<h1>{t("hero.title")}</h1>
<p>{t("hero.subtitle")}</p>
```

Não fazer ainda
- Não colocar textos dinâmicos (projetos, skills, etc.) no i18n.
- Não misturar conteúdo gerenciado no dashboard com arquivos de tradução.

FASE 3 – MODELAGEM DE DOMÍNIO (TYPES) + DADOS MOCKADOS

Objetivo
Definir o formato dos dados (projetos, skills, etc.) antes de configurar Firestore, garantindo forte tipagem e flexibilidade para i18n.

Tarefas
- Criar interfaces em src/types, por exemplo:

```
type LanguageCode = "pt" | "en";

interface LocalizedContent {
  pt: string;
  en: string;
}

interface Project {
  id: string;
  slug: string;
  coverImage: string;
  techs: string[];
  order: number;
  published: boolean;
  title: LocalizedContent;
  description: LocalizedContent;
  details: LocalizedContent;
}
```

- Criar mocks de projetos e skills (src/features/projects/mock/projectsMock.ts, etc.), por exemplo:

```
projectsMock: Project[] = [
  {
    id: "1",
    slug: "uptime-task-manager",
    coverImage: "/assets/uptime.png",
    techs: ["React", "Firebase", "TypeScript"],
    order: 1,
    published: true,
    title: { pt: "UpTime Task Manager", en: "UpTime Task Manager" },
    description: {
      pt: "Sistema de gestão de tarefas com controle de tempo.",
      en: "Task management system with time tracking."
    },
    details: {
      pt: "Projeto pessoal focado em produtividade...",
      en: "Personal project focused on productivity..."
    }
```

```
  }
];
```

- Usar esses mocks nas páginas públicas (projects, skills) para construir:
  - Layout dos cards
  - Responsividade
  - Integração com tema dark/light

Não fazer ainda
- Não integrar com Firebase.
- Não criar dashboard antes de a parte pública estar visualmente consistente.

FASE 4 – CONFIGURAÇÃO DO FIREBASE (INFRA MÍNIMA)

Objetivo
Ter o projeto Firebase pronto, mas ainda desacoplado da UI.

Tarefas
- Criar projeto no Firebase Console.
- Habilitar:
  - Firestore
  - Authentication (e-mail/senha ou Google, escolhido conforme preferência).
- Criar arquivo src/config/firebase.ts com inicialização do app, Firestore e Auth.
- Configurar variáveis de ambiente (.env) e carregar usando import.meta.env.

Não fazer ainda
- Não chamar Firestore diretamente em componentes.
- Não implementar tela de login ainda.

FASE 5 – CAMADA DE SERVIÇOS (FIRESTORE + AUTH)

Objetivo
Criar uma camada de acesso a dados seguindo princípios de responsabilidade única (SRP), sem acoplamento com componentes.

Tarefas
- Criar pasta src/services.
- Criar services específicos:

  - projectService.ts → CRUD de projetos
  - skillService.ts → CRUD de habilidades
  - authService.ts → login, logout, obter usuário atual

- Exemplo de função em projectService.ts:

```
async function getAllProjects(): Promise<Project[]> {
  const snapshot = await getDocs(collection(db, "projects"));
  return snapshot.docs.map(doc => ({
    id: doc.id,
    ...(doc.data() as Omit<Project, "id">)
  }));
```

```
}
```

- Criar hooks finos em src/hooks para consumir os serviços, por exemplo useProjects().

Não fazer ainda
- Não acessar Firestore diretamente dentro de componentes de UI.
- Não conectar o dashboard aos serviços ainda.

FASE 6 – LEITURA DE CONTEÚDO DINÂMICO NA UI

Objetivo
Substituir gradualmente os mocks pelos dados reais do Firestore sem alterar a arquitetura.

Tarefas
- Atualizar ProjectsSection e outras seções para usar useProjects() em vez de dados mockados.
- Garantir que os documentos no Firestore seguem exatamente a estrutura tipada em Project.
- Implementar estados de loading e erro simples.

Resultado
- A parte pública do portfólio passa a exibir dados reais do Firestore, ainda que eventualmente apenas em português.

FASE 7 – INTERNACIONALIZAÇÃO DO CONTEÚDO DINÂMICO

Objetivo
Conectar o sistema de idiomas existente (i18n) aos campos localizados armazenados no Firestore.

Tarefas
- Garantir que os documentos no Firestore possuam campos localizados, por exemplo:

```
{
  "title": { "pt": "Título PT", "en": "Title EN" },
  "description": { "pt": "Descrição...", "en": "Description..." }
}
```

- Nos componentes, usar o idioma atual do i18n para escolher o conteúdo certo:

```
const lang = i18n.language as "pt" | "en";
project.title[lang];
project.description[lang];
```

- Implementar fallback opcional, ex.: se não houver tradução em en, usar pt.

Resultado
- Todos os projetos e skills são exibidos no idioma escolhido pelo usuário, usando a mesma fonte de verdade (Firestore) para o conteúdo e i18n apenas para a interface.

FASE 8 – AUTENTICAÇÃO + ROTAS PROTEGIDAS

Objetivo

Restringir o acesso ao dashboard apenas ao dono da aplicação.

Tarefas
- Criar AuthContext com:
  - usuário atual
  - estado de loading
  - funções login e logout

- Usar onAuthStateChanged para monitorar o estado de autenticação.
- Criar componente ProtectedRoute que redireciona usuários não autenticados para /dashboard/login.
- Proteger a rota /dashboard usando ProtectedRoute.

Resultado
- Estrutura de login funcional e rotas sensíveis protegidas.

FASE 9 – DASHBOARD (CRUD DE PROJETOS/SKILLS)

Objetivo
Permitir que o proprietário gerencie o conteúdo dinâmico (projetos, habilidades, etc.) via interface.

Tarefas
- Em features/dashboard, criar:
  - DashboardLayout
  - ProjectListPage (lista, editar, excluir)
  - ProjectFormPage (criar/editar)
  - SkillListPage
  - SkillFormPage

- Formular os campos necessários:
  - slug
  - coverImage
  - techs
  - order
  - published
  - title.pt, title.en
  - description.pt, description.en
  - details.pt, details.en (se aplicável)

- Reutilizar serviços de Firestore:
  - createProject
  - updateProject
  - deleteProject
  - etc.

Resultado
- Conteúdo mostrado na parte pública pode ser totalmente gerenciado pelo dashboard.

FASE 10 – REGRAS DE SEGURANÇA, TESTES E POLIMENTOS

Objetivo
Aprimorar o projeto para um nível próximo de produção, focando em segurança e qualidade.

Tarefas
- Firestore Security Rules:
  - Permitir leitura pública em collections de conteúdo (projects, skills).
  - Restringir escrita apenas ao UID do proprietário.

  Exemplo simplificado:
  ```
  match /databases/{database}/documents {
    match /projects/{docId} {
      allow read: if true;
      allow write: if request.auth != null && request.auth.uid == "SEU_UID";
    }
  }
  ```

- Testes básicos:
  - Components puros (Navbar, ProjectCard, etc.).
  - Serviços (mockando o Firestore).
- Melhorias de UX:
  - Loadings e mensagens de erro amigáveis.
  - Acessibilidade (aria-labels, contraste).
  - Pequenas animações, se desejado.

RESUMO DA ORDEM RECOMENDADA

1. Rotas + layout geral (SPA funcionando com NavBar e temas).
2. i18n da interface (textos estáticos organizados).
3. Types + mocks + layout das seções públicas.
4. Configuração do Firebase (sem acoplar à UI).
5. Camada de serviços (Firestore + Auth) + hooks.
6. Integração da parte pública com Firestore (dados reais).
7. Internacionalização do conteúdo dinâmico (projetos/skills).
8. Autenticação + proteção das rotas do dashboard.
9. Dashboard completo com CRUD de conteúdo.
10. Regras de segurança, testes e refinamentos finais.

Com esse plano, você evita desenvolver partes fora de ordem, mantém o foco em boas práticas (SOLID, separação de responsabilidades, i18n bem estruturado) e usa Firebase de forma eficiente e gratuita.