# 🎯 Prompt para Construção do Aplicativo de Controle de Territórios (Pregação TJ)

## 1. Objetivo e Contexto

Desenvolver um aplicativo web completo e responsivo (MVP - Produto Mínimo Viável) para gerenciar e visualizar o progresso da pregação Testemunhas de Jeová em territórios. **O sistema deve ler e escrever (fazer commit) diretamente no arquivo de dados YAML no repositório GitHub, utilizando a API.**

## 2. Especificações Técnicas (Stack)

- **Frontend:** React (com Hooks e Functional Components)
- **Styling:** Tailwind CSS (garantir responsividade total para mobile e desktop)
- **Backend/Database:** Arquivo de dados **YAML** (data/db.yml) mantido no repositório GitHub.
- **Persistência de Dados (CRÍTICO!):**
  - **Leitura (Sincronização Inicial):** A aplicação deve, ao iniciar, fazer uma requisição GET para a API de Conteúdo do GitHub para carregar o conteúdo atual do data/db.yml.
  - **Escrita (CRUD) - INTEGRAÇÃO COM GITHUB API:**
    1. Toda ação que modifique o estado (mudança de status, CRUD) deve aplicar a alteração no estado de memória local.
    2. Imediatamente, a aplicação deve converter o estado local em uma nova string YAML.
    3. Uma função assíncrona (commitToGitHub) deve ser acionada, utilizando a **API de Conteúdo do GitHub** (PUT request), para atualizar o arquivo data/db.yml no repositório.
    4. A aplicação utiliza um PAT (Personal Access Token) armazenado localmente para autenticar o commit (ver 5.4).
- **Output:** O código deve ser entregue em **um único arquivo React (.jsx ou .tsx)**.

## 3. Modelo de Dados (Estrutura YAML)

Todo o estado da aplicação deve ser representado em um único objeto JavaScript/YAML, conforme o exemplo abaixo:

```
appConfig:
  superAdminEmail: "zico.josias@gmail.com" # E-mail do Super Admin (acesso total)
  # Simulação de base de usuários para checagem de perfil (Substitui Firebase Auth)
```

```
  userRoles:
    zico.josias@gmail.com: "Super Admin"
    capitao.exemplo@email.com: "Capitão"
    admin.exemplo@email.com: "Admin"

territories:
  "01":
    name: "Território Central"
    status: "Em Uso"
    lastUse: "2025-10-15T10:00:00Z"
    blocks:
      "1":
        summary: { total: 10, verde: 5, amarelo: 2, laranja: 2, vermelho: 1 }
        imageUrl: "placeholder-url"
        houses:
          "1": { status: 1, chalkNumber: "1", officialNumber: "123A", notes: "", lastVisit: null }
          "2": { status: 2, chalkNumber: "2", officialNumber: "123B", notes: "Voltar depois das 18h",
lastVisit: "2025-10-15T09:30:00Z" }
          # ... demais casas
```

### Status da Casa (houses.status) - Fluxo de Visita:

1. **Verde (1):** Não Visitada (Ponto de partida).
2. **Amarelo (2):** Visita 1 (Sem Contato).
3. **Laranja (3):** Visita 2 (Sem Contato).
4. **Vermelho (4):** Contato Estabelecido.

# 4. Requisitos de Funcionalidade por Perfil

O aplicativo deve implementar os seguintes perfis, sendo a identificação baseada na entrada de e-mail e conferência com appConfig.userRoles.

## 4.0. Autenticação e Inicialização (Substituto do Google Login)

- A tela inicial deve ter um campo para o usuário inserir seu e-mail e um botão "Entrar".
- Ao entrar, o aplicativo deve consultar o campo userRoles no YAML para definir o role do usuário. Se o e-mail não estiver na lista, o papel padrão é **Comum**.

## 4.1. Usuário Comum (Comum)

- **Visualização:** Deve ver os carrosséis de **Territórios** e **Quadras** (se o Território estiver ativo).
- **Interação de Status:**

    - Ao selecionar uma quadra, carregar e exibir o **Mosaico de Edificações**, mostrando o **chalkNumber**.
    - Clicar em uma casa deve ciclar o status no seguinte fluxo: **Verde (1) -> Amarelo (2) -> Laranja (3) -> Vermelho (4) -> Verde (1)**.
    - **Após a mudança de status, a função commitToGitHub deve ser chamada AUTOMATICAMENTE.**
- **Interação de Numeração:**
    - Deve ser capaz de atualizar a **Numeração Oficial/Fachada (houses.officialNumber)** de uma residência (e.g., através de um modal).
    - **Após a mudança, a função commitToGitHub deve ser chamada AUTOMATICAMENTE.**

### 4.2. Capitão (Capitão)

- **Tudo o que o Usuário Comum faz, mais:**
- **Numeração de GIZ/Sequencial (chalkNumber):**
    - Deve ser capaz de definir/atualizar a numeração sequencial (1 a N) de uma quadra.
    - **Lógica de Numeração Automática:** Uma função deve reordenar e, se necessário, criar/remover casas na quadra com base em um valor máximo de 'N' fornecido, atualizando o campo chalkNumber de cada casa para o valor sequencial exato (1, 2, 3, ..., N). Novas casas devem ser criadas com status **Verde (1)**.
- **Proposta de Mudança:** Toda operação de mudança de dados **deve acionar automaticamente a função commitToGitHub**.

### 4.3. Administrador (Admin)

- **Tudo o que o Capitão faz, mais:**
- **Gerenciamento Total de Territórios (CRUD):**
    - Criar, Editar (Nome, Número), Excluir Territórios.
    - Criar, Editar (Número, Imagem do Mapa), Excluir Quadras dentro de um Território.
- **Gerenciamento de Usuários: NÃO** possui permissão para CRUD de Usuários.
- **Proposta de Mudança:** Toda operação de CRUD **deve acionar automaticamente a função commitToGitHub**.

### 4.4. Super Administrador (Super Admin)

- **Perfil Chave:** Associado ao e-mail appConfig.superAdminEmail.
- **Acesso Total:** Possui todas as funcionalidades do Capitão e do Administrador, mais:
- **Gerenciamento de Usuários (CRUD):**
    - Atualizar a lista appConfig.userRoles para incluir ou remover e-mails e alterar o papel (role) associado.
- **Proposta de Mudança:** Toda operação de CRUD **deve acionar automaticamente a função commitToGitHub**.

## 5. Requisitos de Interface e Experiência do Usuário

# (UI/UX)

- **Layout:** Limpo e moderno, usando Tailwind.
- **Responsividade:** Total para mobile e desktop.
- **Acessibilidade de CRUD:** Botões de CRUD devem ser visíveis apenas para os perfis autorizados (Admin/Super Admin para Territórios; apenas Super Admin para Usuários).
- **UI para CRUD:** Utilizar modais/diálogos para formulários. Usar **lucide-react** para ícones.
- **Componente de Login:** Deve ser o primeiro componente a ser renderizado, solicitando o e-mail do usuário.

## 5.4. Integração GitHub API (Autenticação PAT) - CRÍTICO!

- **Dados Necessários:** O código deve conter constantes para o repositório, que o usuário real substituirá:
  const REPO_OWNER = 'seu_usuario_github';
  const REPO_NAME = 'nome_do_repositorio';
  const FILE_PATH = 'data/db.yml';
  const COMMIT_MESSAGE = 'Dados atualizados pelo App de Territórios';

- **Fluxo Transparente de PAT (CRÍTICO):** O aplicativo **NÃO** deve ter o PAT hardcoded.
  - **APENAS o Super Admin** deve ser solicitado a inserir o PAT em um modal de configuração **na primeira vez que acessar a aplicação** (ou quando o token não estiver armazenado).
  - O PAT deve ser armazenado localmente (localStorage ou sessionStorage) para uso em commits futuros (simulando persistência segura).
  - O PAT armazenado será usado por *qualquer* usuário logado (Capitão, Admin, Super Admin) para fazer o commit, mas **SÓ** o Super Admin pode configurá-lo inicialmente.
- **Lógica da Função commitToGitHub:**
  1. A função deve primeiro fazer uma requisição GET para obter o SHA (hash) atual do arquivo data/db.yml.
  2. Em seguida, deve fazer uma requisição PUT para o endpoint de conteúdo do GitHub, autenticando-se com o token armazenado, incluindo:
     - O novo conteúdo YAML (codificado em Base64).
     - A mensagem de commit.
     - O SHA do arquivo obtido na etapa 1 (obrigatório para commits).
  3. Exibir uma notificação de sucesso ou erro na UI após a tentativa de commit.

## 5.5. Sincronização Automática (Invisível)

- **Polling:** O aplicativo deve configurar um setInterval para verificar a API do GitHub a cada **30 segundos** (ou em um intervalo razoável) para identificar mudanças no SHA do arquivo data/db.yml. Se o SHA mudou, a função de leitura deve ser acionada para buscar e aplicar os novos dados.
- **Re-fetch on Focus:** O aplicativo deve utilizar o evento window.onfocus para acionar

imediatamente a função de leitura, garantindo que o usuário veja os dados mais recentes assim que retornar à aba do navegador.
- **Leitura Pós-Commit:** Após um commit bem-sucedido, o aplicativo **DEVE** acionar imediatamente a função de leitura para garantir que o novo estado local esteja sincronizado com o estado canônico do GitHub (em caso de conflito, o estado do GitHub sempre prevalece).

# 6. Versionamento e Deployment

- **Versionamento (Git/GitHub):** Todo o código-fonte deve ser versionado no GitHub.
- **Deployment (CI/CD Automático):** O aplicativo **DEVE** implementar um fluxo de trabalho **GitHub Actions** (CI/CD) para:
  1. **Build Automático:** Compilar o projeto React (e.g., usando npm run build).
  2. **Deploy Contínuo:** Publicar automaticamente os arquivos estáticos gerados na etapa de build para **GitHub Pages** (usando o gh-pages ou ação similar).
  3. **Gatilho (Trigger):** Este pipeline deve ser executado automaticamente **em cada push (commit) na branch principal (main ou master)**.
     - **Objetivo:** Garantir que qualquer alteração de dados no data/db.yml (feita via API do app) ou qualquer alteração no código (feita pelo desenvolvedor) resulte em uma nova versão online e acessível no GitHub Pages de forma **totalmente automática e invisível** para os usuários finais.