*Com base no "**Briefing**" , preparei um script detalhado para a criação do seu aplicativo "Indica NCF Seguro". Siga este guia passo a passo para transformar sua visão em realidade.*

Script Detalhado: Aplicativo "**NCFSegurosIndico**" **(31/01/2025)**

Título: Guia Detalhado para Desenvolvimento de PWA de Indicação de Seguros

1. Planejamento e Arquitetura

- **1.1. Análise de Requisitos:**
    - **Requisitos Funcionais:**
        - **Indicação de Amigos:**
            - Usuário deve poder indicar amigos através do PWA.
            - O sistema deve coletar informações básicas do amigo indicado (nome, telefone, e-mail).
            - O sistema deve validar se o amigo indicado já foi cadastrado anteriormente.
        - **Descontos Progressivos:**
            - Usuário deve receber 1% de desconto por indicação válida.
            - Usuário deve receber mais 1% de desconto quando o amigo indicado contratar o seguro.
            - Desconto máximo acumulativo de 10% ao ano na apólice.

- **Painel Administrativo:**
    - Administrador deve receber notificações push de novas indicações.
    - Administrador deve poder visualizar os dados da indicação (quem indicou e quem foi indicado).
    - Administrador deve poder confirmar a validade da indicação.
    - Administrador deve poder enviar um link da proposta para o indicado.
    - Administrador deve poder registrar a contratação do seguro pelo indicado.
- **Notificações Push:**
    - Usuário que indicou deve receber notificação push quando a indicação for confirmada e o desconto concedido.
    - Administrador deve receber notificação push quando uma nova indicação for realizada.
- **Autenticação:**
    - Usuários (clientes da seguradora) devem poder se autenticar no PWA.
    - Administrador deve ter um login separado com permissões elevadas.
- **Relatórios:**
    - O administrador pode gerar relatórios sobre o total de indicações, conversões e descontos concedidos.

- **Requisitos Não Funcionais:**
    - **Performance:** O PWA deve ser rápido e responsivo.
    - **Disponibilidade:** O PWA deve estar disponível 24/7.
    - **Segurança:** Os dados dos usuários devem ser protegidos.
    - **Escalabilidade:** O PWA deve ser escalável para suportar um grande número de usuários.
    - **Usabilidade:** O PWA deve ser fácil de usar e intuitivo.
    - **Acessibilidade:** O PWA deve ser acessível a usuários com deficiência.
    - **Offline:** O PWA deve funcionar mesmo quando o usuário estiver offline (pelo menos parcialmente).

- **1.2. Escolha de Tecnologias:**
    - **Frontend:**
        - **React:** Biblioteca JavaScript para construção de interfaces de usuário interativas e reativas.
            - *Justificativa:* Componentização, grande comunidade, ecossistema rico (Redux, Material-UI, etc.), facilita a criação de interfaces complexas e escaláveis. Ótima opção para PWAs.
        - **Alternativas:** Vue.js (mais fácil de aprender, bom para projetos menores), Angular (mais robusto, para projetos complexos).
    - **Backend:**
        - **Node.js com Express:** Ambiente de execução JavaScript no servidor e framework para criação de APIs REST.
            - *Justificativa:* JavaScript no frontend e backend, facilita o desenvolvimento full-stack. Express é leve e flexível.
        - **Alternativas:** Python com Django/Flask (se você tiver experiência com Python), Firebase (se quiser uma solução NoSQL e serverless).
    - **Banco de Dados:**
        - **PostgreSQL:** Banco de dados relacional robusto e escalável.
            - *Justificativa:* Confiável, ACID-compliant, bom suporte a transações, ideal para dados estruturados.
        - **Alternativas:** MongoDB (se você preferir NoSQL), Firebase Firestore (se usar Firebase como backend).
    - **Notificações Push:**
        - **Firebase Cloud Messaging (FCM):** Serviço de mensagens push do Google.
            - *Justificativa:* Gratuito, fácil de integrar com PWAs, confiável.
        - **Alternativas:** OneSignal.

    - **Outras Ferramentas:**
        - **npm ou Yarn:** Gerenciadores de pacotes para JavaScript.
        - **Git:** Sistema de controle de versão.
        - **ESLint:** Linter para JavaScript.
        - **Prettier:** Formatador de código.
        - **Webpack ou Parcel:** Bundlers para JavaScript.

- **1.3. Arquitetura da Aplicação:**
    - **Frontend (React):**
        - `components/`: Componentes da UI (ex: `IndicationForm`, `ConfirmationModal`, `AdminDashboard`, etc.).
        - `pages/`: Páginas da aplicação (ex: `Home`, `Login`, `Admin`, etc.).
        - `services/`: Serviços para comunicação com o backend (ex: `api.js`).
        - `context/`: Contextos para gerenciamento de estado global (ex: `AuthContext`, `DiscountContext`).
        - `utils/`: Funções utilitárias.
        - `App.js`: Componente raiz da aplicação.
        - `index.js`: Ponto de entrada da aplicação.
        - `service-worker.js`: Service Worker para caching e notificações push.
        - `manifest.json`: Manifesto da Web App.

    - **Backend (Node.js/Express):**
        - `controllers/`: Lógica de controle das rotas (ex: `indicationController.js`, `authController.js`).
        - `models/`: Modelos de dados (ex: `Indication.js`, `User.js`).
        - `routes/`: Definição das rotas da API (ex: `indicationRoutes.js`, `authRoutes.js`).
        - `middleware/`: Middlewares para autenticação, autorização, etc.
        - `config/`: Arquivos de configuração (ex: conexão com o banco de dados).
        - `server.js`: Ponto de entrada do servidor.

- **1.4. Fluxo de Dados:**
    - Usuário preenche o formulário de indicação no frontend.
    - Frontend envia os dados para o backend (API Node.js/Express).
    - Backend valida os dados e salva no banco de dados (PostgreSQL).
    - Backend envia uma notificação push para o administrador via FCM.
    - Administrador confirma a indicação no painel administrativo.
    - Backend registra a confirmação no banco de dados.
    - Backend calcula o desconto e envia uma notificação push para o usuário que indicou.
    - Frontend exibe o desconto atualizado para o usuário.

- **1.5. Estratégia de Caching:**
    - **Cache-First:** Serve os recursos do cache, e atualiza o cache em segundo plano. Bom para assets estáticos (CSS, JavaScript, imagens).
    - **Network-First:** Tenta obter os recursos da rede, e se falhar, usa o cache. Bom para dados dinâmicos que precisam estar atualizados.
    - **Cache then Network:** Mostra os dados cacheados imediatamente e atualiza com os dados mais recentes da rede. Ideal para uma boa experiência do usuário, mostrando algo rápido e atualizando depois.

- **1.6. Estratégia de Notificações Push:**
    - Usar Firebase Cloud Messaging (FCM).
    - Frontend se inscreve para receber notificações push.
    - Backend envia notificações push quando:
        - Nova indicação é realizada.
        - Indicação é confirmada.
        - Amigo indicado contrata o seguro.

- **1.7. Considerações de Segurança:**
    - **HTTPS:** Essencial para PWAs.
    - **Validação de Dados:** Validar todos os dados recebidos do frontend e do backend.
    - **Autenticação e Autorização:** Implementar autenticação robusta para usuários e administrador. Usar JWT (JSON Web Tokens) para autorização.
    - **Proteção contra CSRF e XSS:** Usar mecanismos de proteção contra ataques CSRF e XSS.
    - **Sanitização de Dados:** Sanitizar os dados antes de salvar no banco de dados.
    - **Limitar Taxa de Requisições (Rate Limiting):** Proteger a API contra ataques de força bruta.

2. Configuração do Ambiente de Desenvolvimento

- **2.1. Ferramentas Necessárias:**
    1. **Node.js:** Ambiente de execução JavaScript no servidor.
        1. https://nodejs.org/
    2. **npm:** Gerenciador de pacotes para JavaScript (instalado com Node.js).
    3. **Yarn:** Outro gerenciador de pacotes para JavaScript (opcional).

        1. https://yarnpkg.com/
    4. **Editor de Código:** Visual Studio Code (recomendado), Sublime Text, Atom, etc.
        1. https://code.visualstudio.com/
    5. **Git:** Sistema de controle de versão.
        1. https://git-scm.com/
    6. **PostgreSQL:** Banco de dados relacional.
        1. https://www.postgresql.org/
    7. **Postman ou Insomnia:** Ferramentas para testar APIs.

- **2.2. Configuração do Projeto:**
    1. **Frontend (React):**
        1. Crie um novo diretório para o projeto: `mkdir seguro-indicacao-pwa`
        2. Navegue para o diretório: `cd seguro-indicacao-pwa`
        3. Crie um novo projeto React usando Create React App: `npx create-react-app frontend`
        4. Navegue para o diretório do frontend: `cd frontend`
        5. Inicie o servidor de desenvolvimento: `npm start` ou `yarn start`
    2. **Backend (Node.js/Express):**
        1. Crie um novo diretório para o backend: `mkdir backend`
        2. Navegue para o diretório: `cd backend`
        3. Inicialize um novo projeto Node.js: `npm init -y` ou `yarn init -y`
        4. Instale as dependências: `npm install express cors pg bcrypt jsonwebtoken dotenv firebase-admin` ou `yarn add express cors pg bcrypt jsonwebtoken dotenv firebase-admin`
        5. Crie o arquivo `server.js` com o código básico do servidor Express.
        6. Crie a estrutura de pastas (controllers, models, routes, middleware, config).

- **2.3. Configuração do Service Worker:**
    1. Crie o arquivo `service-worker.js` na pasta `public` do frontend.
    2. Registre o Service Worker no arquivo `index.js` do frontend:

```
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then((registration) => {
        console.log('Service Worker registrado com sucesso:', registration);
      })
      .catch((error) => {
```

```
      console.log('Erro ao registrar o Service Worker:', error);
    });
});

        }
```

3. Configure o Service Worker para caching e notificações push (ver exemplos de código abaixo).

- **2.4. Configuração do Web App Manifest:**
  1. Crie o arquivo `manifest.json` na pasta `public` do frontend.
  2. Adicione o seguinte conteúdo ao arquivo `manifest.json`:

```
{
  "name": "Seguro Indicação",
  "short_name": "Indicação Segura",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#007bff",
  "icons": [
    {
      "src": "icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]

        }
```

  3. Substitua os valores com os detalhes do seu PWA.
  4. Crie os ícones (icon-192x192.png e icon-512x512.png) e coloque-os na pasta `public`.
  5. Adicione a tag `<link rel="manifest" href="/manifest.json">` no `<head>` do arquivo `public/index.html`.

3. Desenvolvimento do Frontend (React)

- **3.1. Estrutura de Componentes:**
  - `IndicationForm`: Formulário para indicar amigos.
  - `ConfirmationModal`: Modal para confirmar a indicação.
  - `AdminDashboard`: Painel administrativo para o administrador.
  - `Login`: Formulário de login.
  - `Home`: Página inicial do usuário.
  - `Header`: Cabeçalho da aplicação.
  - `Footer`: Rodapé da aplicação.

- **3.2. Criação de Componentes:**
  - Exemplo de componente `IndicationForm`:

```
import React, { useState } from 'react';

function IndicationForm() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [phone, setPhone] = useState('');

  const handleSubmit = (event) => {
    event.preventDefault();
    // Lógica para enviar os dados para o backend
    console.log('Dados do formulário:', { name, email, phone });
  };

  return (
    <form onSubmit={handleSubmit}>
      <label>
        Nome:
        <input type="text" value={name} onChange={(e) => setName(e.target.value)} />
      </label>
      <label>
        Email:
        <input type="email" value={email} onChange={(e) => setEmail(e.target.value)} />
      </label>
      <label>
        Telefone:
        <input type="tel" value={phone} onChange={(e) => setPhone(e.target.value)} />
      </label>
      <button type="submit">Indicar Amigo</button>
    </form>
  );
}
```

- **3.3. Estilização:**
  - **CSS Modules:** Recomendado para evitar conflitos de nomes de classes.
    - Crie arquivos `.module.css` para cada componente.
    - Importe o arquivo CSS no componente: `import styles from './IndicationForm.module.css';`
    - Use as classes CSS: `<button className={styles.button}>Indicar Amigo</button>`
  - **Alternativas:** Styled Components, Tailwind CSS, Bootstrap.

- **3.4. Gerenciamento de Estado:**
  - **useState:** Para estados locais de componentes.
  - **useReducer:** Para estados mais complexos dentro de um componente.
  - **Context API:** Para compartilhar estado entre componentes sem precisar passar props manualmente.
  - **Redux:** Para gerenciamento de estado global em aplicações maiores.

- **3.5. Roteamento:**
  - Use `react-router-dom` para implementar o roteamento.
    - Instale: `npm install react-router-dom` ou `yarn add react-router-dom`
    - Exemplo:

```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './pages/Home';
import Login from './pages/Login';
import Admin from './pages/Admin';

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/login" component={Login} />
        <Route path="/admin" component={Admin} />
      </Switch>
    </Router>
  );
}
```

- **3.6. Formulários:**
    - Use componentes controlados para gerenciar os valores dos campos do formulário.
    - Use bibliotecas de validação como `formik` ou `react-hook-form` para validar os formulários.

- **3.7. Comunicação com o Backend:**
    - Use `fetch` ou `axios` para fazer requisições HTTP para o backend.
        - Instale `axios`: `npm install axios` ou `yarn add axios`
        - Exemplo:

```
import axios from 'axios';

const handleSubmit = async (event) => {
  event.preventDefault();
  try {
    const response = await axios.post('/api/indications', { name, email, phone });
    console.log('Resposta do backend:', response.data);
  } catch (error) {
    console.error('Erro ao enviar os dados:', error);
  }

        };
```

- **3.8. Implementação das Funcionalidades Essenciais:**
    - **Indicação de Amigos:**
        - Crie o componente `IndicationForm` para coletar os dados do amigo indicado.
        - Envie os dados para o backend (rota `/api/indications`).
        - Valide os dados no backend e salve no banco de dados.
        - Envie uma notificação push para o administrador via FCM.

    - **Painel Administrativo:**
        - Crie o componente `AdminDashboard` para o painel administrativo.
        - Liste as indicações pendentes.
        - Permita que o administrador confirme ou rejeite as indicações.
        - Implemente a lógica para enviar o link da proposta para o indicado.
        - Registre a contratação do seguro pelo indicado.

    - **Cálculo de Descontos:**
        - Crie uma função no backend para calcular o desconto com base no número de indicações e contratações.
        - Atualize o desconto do usuário no banco de dados.

    - **Notificações Push:**
        - Implemente a lógica para enviar notificações push para o usuário que indicou quando a indicação for confirmada e o desconto concedido.

- **3.9. Implementação das Funcionalidades Desejáveis:**
    - **Relatórios:** Crie telas para gerar relatórios sobre indicações, conversões e descontos concedidos.
    - **Integração com o Sistema de CRM da Seguradora:** Integre o PWA com o sistema de CRM da seguradora para atualizar os dados dos clientes.

4. Desenvolvimento do Backend (Node.js/Express)

- **4.1. Escolha de Tecnologia:** Node.js com Express (justificativa acima).
- **4.2. Estrutura do Backend:** (descrita acima).
- **4.3. API:**
    - **Rotas:**
        1. `POST /api/indications`: Recebe os dados da indicação.
        2. `GET /api/indications`: Lista as indicações (requer autenticação de administrador).
        3. `PUT /api/indications/:id/confirm`: Confirma uma indicação (requer autenticação de administrador).
        4. `POST /api/auth/login`: Autentica o usuário (cliente ou administrador).
        5. `GET /api/users/me`: Retorna os dados do usuário autenticado.
        6. `GET /api/discounts/me`: Retorna o desconto do usuário autenticado.

- **4.4. Autenticação e Autorização:**
    - Use JWT (JSON Web Tokens) para autenticação e autorização.
    - Crie um middleware para verificar o JWT nas rotas protegidas.
    - Crie um middleware para verificar se o usuário é administrador.

- **4.5. Banco de Dados:**
    - PostgreSQL (justificativa acima).
    - Modelagem de Dados:
        1. `users`: `id`, `email`, `password`, `role` (admin ou user), `discount`
        2. `indications`: `id`, `user_id` (quem indicou), `name`, `email`, `phone`, `status` (pending, confirmed, rejected), `created_at`

- **4.6. Implementação das Funcionalidades do Backend:**
    - **Rota POST /api/indications:**
        1. Recebe os dados da indicação.
        2. Valida os dados.
        3. Cria um novo registro na tabela `indications`.
        4. Envia uma notificação push para o administrador via FCM.

    - **Rota PUT /api/indications/:id/confirm:**
        1. Recebe o ID da indicação.
        2. Verifica se o usuário é administrador.
        3. Atualiza o status da indicação para "confirmed".
        4. Calcula o desconto do usuário que indicou.
        5. Atualiza o desconto do usuário na tabela `users`.
        6. Envia uma notificação push para o usuário que indicou.

    - **Implementação do FCM:**
        1. Crie um projeto no Firebase Console.
        2. Obtenha as credenciais do servidor do Firebase.
        3. Use o pacote `firebase-admin` para enviar notificações push.

5. Testes

- **5.1. Tipos de Testes:**
    - **Testes Unitários:** Testam as unidades de código isoladamente (ex: funções, componentes).

    - **Testes de Integração:** Testam a interação entre diferentes partes do código (ex: componentes que se comunicam, frontend e backend).
    - **Testes End-to-End (E2E):** Testam o fluxo completo da aplicação, simulando a interação do usuário.

- **5.2. Ferramentas de Teste:**
    - **Jest:** Framework de teste para JavaScript (ótimo para testes unitários e de integração).
    - **React Testing Library:** Biblioteca para testar componentes React (foco na experiência do usuário).
    - **Cypress:** Framework para testes End-to-End (fácil de usar e configurar).

- **5.3. Exemplos de Testes:**
    - **Teste Unitário (Jest):**

```
// sum.test.js
const sum = require('./sum');

test('adds 1 + 2 to equal 3', () => {
  expect(sum(1, 2)).toBe(3);

        });
```

    - **Teste de Componente React (React Testing Library):**

```
import React from 'react';
import { render, screen } from '@testing-library/react';
import IndicationForm from './IndicationForm';

test('renders indication form', () => {
  render(<IndicationForm />);
  const nameInput = screen.getByLabelText('Nome:');
  expect(nameInput).toBeInTheDocument();

        });
```

6. Publicação (Deployment)

- **6.1. Opções de Hospedagem:**
    - **Netlify:** Gratuito para projetos pequenos, fácil de usar, integração com Git.
    - **Vercel:** Similar ao Netlify, focado em performance.

    - **Firebase Hosting:** Gratuito para projetos pequenos, integração com Firebase.
    - **Heroku:** Plataforma como serviço (PaaS), mais flexível, mas pode ser mais caro.
    - **AWS (Amazon Web Services):** Mais complexo, mas oferece mais controle e escalabilidade.
- **6.2. Passos para Publicação (Netlify):**
    - Crie uma conta no Netlify.
    - Conecte o seu repositório Git ao Netlify.
    - Configure as configurações de build (ex: comando de build, diretório de publicação).
    - O Netlify irá automaticamente construir e publicar o seu PWA.
- **6.3. Configuração de Domínio Personalizado:**
    - Compre um domínio personalizado (ex: `seuroseguro.com.br`).
    - Configure os registros DNS do seu domínio para apontar para o Netlify/Vercel/Firebase.
- **6.4. Configuração de HTTPS:**
    - Netlify, Vercel e Firebase Hosting fornecem HTTPS automaticamente.

7. Otimização e Performance

- **7.1. Melhores Práticas:**
    - **Minificação de Arquivos:** Minimize os arquivos CSS e JavaScript.
    - **Compressão de Imagens:** Comprima as imagens para reduzir o tamanho dos arquivos.
    - **Lazy Loading:** Carregue as imagens e outros recursos sob demanda.
    - **Code Splitting:** Divida o código em chunks menores para carregamento mais rápido.
    - **Caching:** Use caching para armazenar os recursos no navegador do usuário.
    - **Otimização de Fontes:** Use fontes web otimizadas.

- **7.2. Ferramentas de Otimização:**
    - **Google PageSpeed Insights:** Analisa a performance do seu PWA e sugere otimizações.
    - **Lighthouse:** Ferramenta de auditoria do Google Chrome.
    - **Webpack Bundle Analyzer:** Analisa o tamanho dos seus bundles JavaScript.

8. Manutenção e Evolução

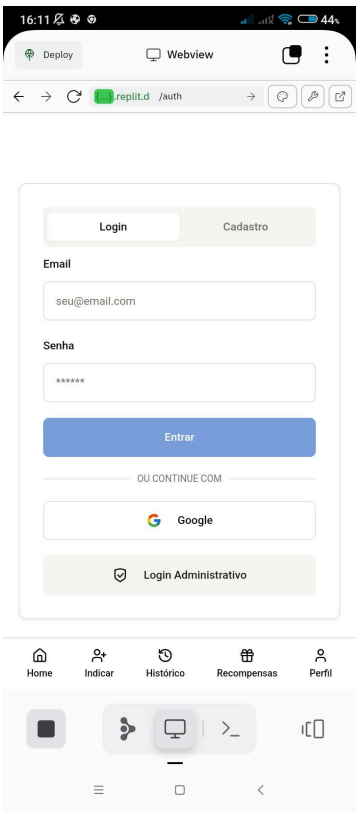
- **8.1. Estratégias:**
    - **Monitoramento:** Monitore a performance do PWA e os erros que ocorrem.
    - **Atualizações:** Mantenha as dependências atualizadas.
    - **Refatoração:** Refatore o código regularmente para melhorar a qualidade e a manutenibilidade.
    - **Testes:** Escreva testes para garantir que as novas funcionalidades não quebrem as funcionalidades existentes.
    - **Feedback:** Colete feedback dos usuários para identificar áreas de melhoria.

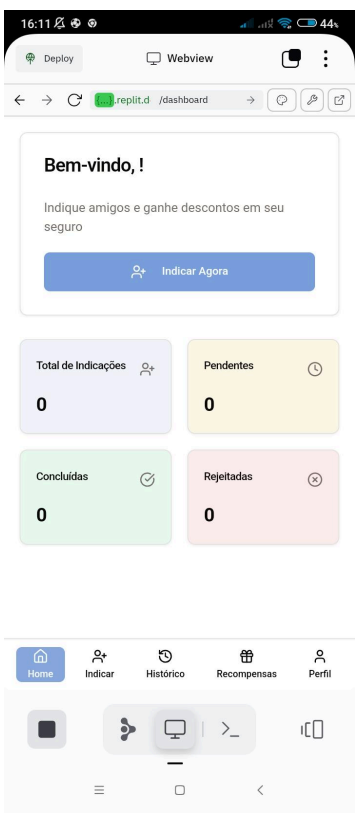
Recursos Adicionais:

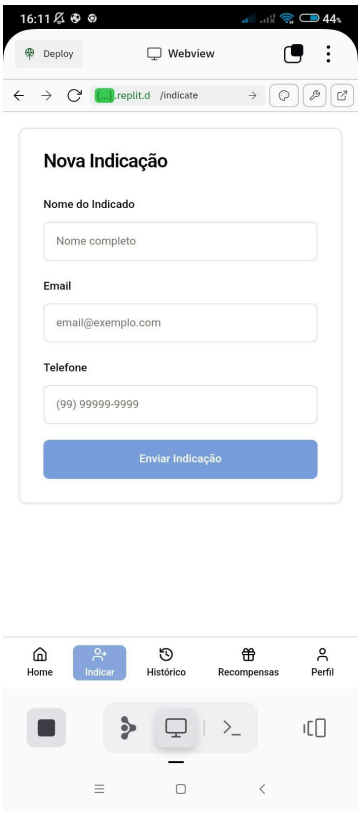
- **Documentação React:** https://reactjs.org/
- **Documentação Node.js:** https://nodejs.org/
- **Documentação Express:** https://expressjs.com/
- **Documentação PostgreSQL:** https://www.postgresql.org/docs/
- **Documentação Firebase Cloud Messaging:** https://firebase.google.com/docs/cloud-messaging
- **Guia PWA do Google:** https://web.dev/progressive-web-apps/

https://replit.com/@rogeriomatos751/AndroidUICompose#workspace.rogeriomatos751.repl.co


Deploy
Webview
.replit.d /
NCF
SEGUROS
Home
Indicar
Histórico
Recompensas
Perfil

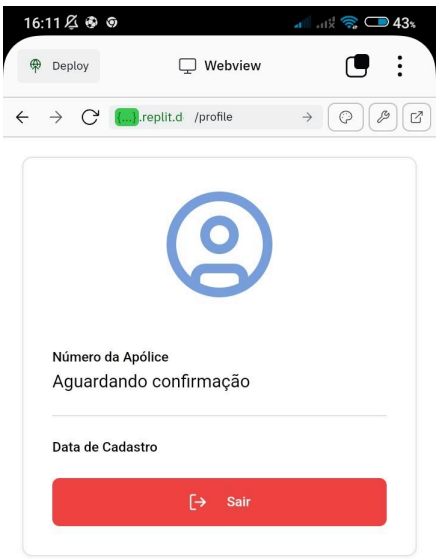

Deploy
Webview
.replit.d /auth
Login
Cadastro
Email
seu@email.com
Senha
******
Entrar
OU CONTINUE COM
Google
Login Administrativo
Home
Indicar
Histórico
Recompensas
Perfil

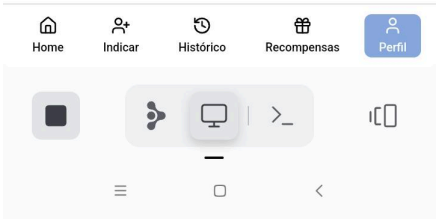

Deploy
Webview
.replit.d /dashboard
Bem-vindo, !
Indique amigos e ganhe descontos em seu seguro
Indicar Agora
Total de Indicações
0
Pendentes
0
Concluídas
0
Rejeitadas
0
Home
Indicar
Histórico
Recompensas
Perfil


Deploy
Webview
.replit.d /indicate
Nova Indicação
Nome do Indicado
Nome completo
Email
email@exemplo.com
Telefone
(99) 99999-9999
Enviar Indicação
Home
Indicar
Histórico
Recompensas
Perfil


Deploy
Webview
.replit.d /history
Histórico de Indicações
Home
Indicar
Histórico
Recompensas
Perfil


Deploy
Webview
.replit.d /rewards
Seu Desconto Atual
0%
Progresso para 5%
0/3 indicações
Níveis de Desconto
5% de desconto
3 indicações concluídas
10% de desconto
5 indicações concluídas
15% de desconto
10 indicações concluídas
Home
Indicar
Histórico
Recompensas
Perfil


16:11
43%
Deploy
Webview
.replit.d
/profile
Número da Apólice
Aguardando confirmação
Data de Cadastro
Sair


Home
Indicar
Histórico
Recompensas
Perfil