DIAGNÓSTICO - Arquitetura Multi-Tenant com RLS

Data: 20/02/2026

---

1. Análise do Cliente Supabase (`supabase.ts`)

Localização: `/supabase.ts`

```
import { createClient } from '@supabase/supabase-js'

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL
const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY

export const supabase = createClient(supabaseUrl, supabaseAnonKey)
```

Problemas identificados:
- ✅ Configuração correta via variáveis de ambiente
- ⚠️ Também existe uma duplicação em `App.tsx` (linhas 13-16) com credenciais hardcoded
- ⚠️ Não existe tipagem para o cliente Supabase (Database types)

---

2. Análise dos Componentes de Autenticação

Login.tsx

Fluxo atual:
1. Usa `supabase.auth.signInWithPassword()`
2. Busca perfil na tabela `profiles` por `id`
3. Se não existe, cria perfil mínimo
4. Faz cast direto `profile as any` para o tipo User

Problemas:
- ⚠️ Import de `supabase` vem de `../App` em vez de `../supabase`
- ⚠️ Recebe `users: User[]` (mock array) como prop mas não usa efetivamente após auth
- ⚠️ Não há verificação de organização/tenant
- ⚠️ Campos do `profiles` não correspondem ao tipo `User`

Register.tsx

Fluxo atual:
1. Usa `supabase.auth.signUp()`
2. Insere na tabela `profiles` com campos: `id`, `nome`, `email`, `role`
3. Cria objeto `User` local com todos os campos
4. Atualiza estado local `setUsers(prev => [...prev, newUser])`
5. Mostra alert e redireciona para login

Problemas:
- ⚠️ Usa campo `nome` mas Login.tsx espera `nome_completo`
- ⚠️ Role é string `'CLIENTE'` mas tipo espera enum `UserRole`
- ⚠️ Sem vínculo com organização
- ⚠️ Dados extras (documentId, taxId, etc.) só existem no estado local

---

3. Análise dos Dashboards

UserDashboard.tsx

Props: `{ currentUser: User; onLogout: () => void }`

Campos usados de `currentUser`:
- `name`, `registrationDate`, `status`, `serviceManager`, `notes`
- `protocol`, `unit`, `documentId`, `taxId`, `country`, `phone`, `maritalStatus`, `address`
- `deadline`, `lastUpdate`, `processNumber`

AdminDashboard.tsx

Props: `{ currentUser: User; users: User[]; setUsers: ...; onLogout: () => void }`

Funcionalidades:
- Lista todos os usuários (do array mock `users`)
- Filtra por nome, protocolo, email
- Atualiza status, deadline, notes, serviceManager
- Cria/edita usuários administrativos
- Deleta usuários

Problemas críticos:
- ⚠️ Opera 100% sobre array local `users` (localStorage)
- ⚠️ Não faz nenhuma operação no Supabase
- ⚠️ Não há isolamento por organização

---

4. Tabelas Referenciadas no Código

Tabela	Arquivo	Operação
`profiles`	Login.tsx	SELECT, INSERT
`profiles`	Register.tsx	INSERT
`auth.users`	Login.tsx, Register.tsx	signIn, signUp

Tabelas necessárias (não existem):
- `organizations` - Multi-tenant
- `org_members` - Vínculo usuário-organização
- `servicos` - Catálogo de serviços
- `processos` - Processos/casos

5. Dependências do Array Mock `users[]`

App.tsx

```
const [users, setUsers] = useState<User[]>(() => {
  const saved = localStorage.getItem('sgi_users');
  return saved ? JSON.parse(saved) : INITIAL_MOCK_USERS;
});
```

Locais que dependem de `users`:

Arquivo	Uso	Refatoração Necessária
`App.tsx`	Estado principal	Remover, usar Supabase
`Login.tsx`	Prop (não usado efetiva-mente)	Remover prop
`AdminDashboard.tsx`	Lista/CRUD de usuários	Migrar para Supabase queries
`constants.ts`	`INITIAL_MOCK_USERS`	Manter apenas para dev/seed

6. Estrutura do Tipo `User` Atual

```
export interface User {
  id: string;
  name: string;           // → nome_completo
  email: string;
  password?: string;      // ❌ NUNCA armazenar
  role: UserRole;         // → mover para org_members
  documentId: string;     // → documento_identidade
  taxId: string;          // → nif_cpf
  address: string;        // → endereco
  maritalStatus: string;  // → estado_civil
  country: string;        // → pais
  phone: string;          // OK
  processNumber?: string; // → processo_numero (separar para tabela processos)
  unit: ServiceUnit;      // → mover para processo/serviço
  status: ProcessStatus;  // → mover para processo
  protocol: string;       // → mover para processo
  registrationDate: string;
  lastUpdate?: string;
  hierarchy?: Hierarchy;  // → mover para org_members
  notes?: string;         // → mover para processo
  deadline?: string;      // → mover para processo
  serviceManager?: string; // → mover para processo
}
```

7. Plano de Migração

Fase 1 - Schema Multi-Tenant

- [x] Criar tabela `organizations`
- [x] Criar tabela `org_members`
- [x] Ajustar tabela `profiles`

Fase 2 - RLS

- [x] Habilitar RLS em todas as tabelas
- [x] Criar funções helper (`is_org_member` , `is_org_admin`)
- [x] Criar policies de acesso

Fase 3 - Frontend

- [ ] Criar `/src/lib/tenant.ts`
- [ ] Atualizar `types.ts`
- [ ] Remover dependência de `users[]` mock
- [ ] Atualizar `Login.tsx` e `Register.tsx`
- [ ] Adaptar dashboards para queries Supabase

---

8. Variáveis de Ambiente Necessárias

```
VITE_SUPABASE_URL=https://ktrrrqaqaljdcmxqdcff.supabase.co
VITE_SUPABASE_ANON_KEY=sb_publishable_ZcEU2_K18A4NU43hO4zPmA_N5SkuqO_
VITE_ORG_SLUG=default  # Novo: slug da organização padrão
```

---

9. Riscos e Mitigações

Risco	Mitigação
Perda de dados localStorage	Script de migração para importar dados existentes
Quebra de funcionalidade	Manter compatibilidade gradual
RLS mal configurado	Testar exaustivamente com diferentes roles
Organização inexistente	Criar org “default” no seed