📋 PLANO DE HIERARQUIA DE EQUIPES - OrganiZen

🎯 Objetivo

Implementar um sistema hierárquico de equipes onde cada nível de cargo (ADMIN, MANAGER, SUPERVISOR, STAFF) tem suas próprias equipes subordinadas, criando uma estrutura organizacional em cascata.

📊 Estrutura Hierárquica Desejada

```
EMPRESA
│
├── ADMIN (vê e gerencia TODAS as equipes)
│   │
│   ├── MANAGER 1 (Equipe do Manager 1)
│   │   ├── SUPERVISOR 1 (Equipe do Supervisor 1)
│   │   │   ├── STAFF 1
│   │   │   ├── STAFF 2
│   │   │   └── STAFF 3
│   │   │
│   │   └── SUPERVISOR 2 (Equipe do Supervisor 2)
│   │       ├── STAFF 4
│   │       └── STAFF 5
│   │
│   └── MANAGER 2 (Equipe do Manager 2)
│       └── SUPERVISOR 3 (Equipe do Supervisor 3)
│           ├── STAFF 6
│           └── STAFF 7
```

🏗️ Estrutura de Dados Atual

Team (Modelo Atual)

```
model Team {
  id           String     @id @default(cuid())
  name         String
  departmentId String
  createdAt    DateTime   @default(now())
  updatedAt    DateTime   @updatedAt
  department   Department @relation(fields: [departmentId], references: [id])
  users        User[]
}
```

Problemas Identificados:

1. ❌ **Não há hierarquia** - Equipes são planas, sem relação pai-filho

2. ❌ **Sem líder de equipe** - Não sabemos quem lidera cada equipe
3. ❌ **Sem visibilidade controlada** - Não há controle de quem vê o quê

🔧 Estrutura de Dados Proposta

Team (Modelo Modificado)

```
model Team {
  id           String     @id @default(cuid())
  name         String
  description  String?
  departmentId String
  leaderId     String     // Líder da equipe (MANAGER ou SUPERVISOR)
  parentTeamId String?    // Equipe pai (para hierarquia)
  level        TeamLevel  // Nível hierárquico da equipe
  createdAt    DateTime   @default(now())
  updatedAt    DateTime   @updatedAt

  // Relações
  department   Department @relation(fields: [departmentId], references: [id],
onDelete: Cascade)
  leader       User       @relation("TeamLeader", fields: [leaderId], references:
[id])
  parentTeam   Team?      @relation("TeamHierarchy", fields: [parentTeamId], refer-
ences: [id])
  childTeams   Team[]     @relation("TeamHierarchy")
  members      User[]     @relation("TeamMembers")

  @@map("teams")
  @@index([leaderId])
  @@index([parentTeamId])
  @@index([departmentId])
}

enum TeamLevel {
  COMPANY      // Equipe da empresa (Admin)
  MANAGEMENT   // Equipe de gerência (Manager)
  SUPERVISION  // Equipe de supervisão (Supervisor)
  OPERATIONS   // Equipe operacional (Staff)
}
```

User (Modificações)

```
model User {
  // ... campos existentes ...

  // Relações de equipe
  teamId              String?             // Equipe do usuário
  team                Team?               @relation("TeamMembers", fields: [teamId],
references: [id])
  ledTeams            Team[]              @relation("TeamLeader")  // Equipes que lid
era
}
```

🔐 Regras de Permissão e Visibilidade

1. ADMIN (Nível Empresa)

- ✅ **Vê:** TODAS as equipes da empresa
- ✅ **Gerencia:** Pode criar/editar/excluir qualquer equipe
- ✅ **Acesso:** Todos os departamentos e equipes
- ✅ **Pode:** Atribuir Managers a equipes de gerência

2. MANAGER (Nível Gerência)

- ✅ **Vê:**
- Sua própria equipe
- Todas as equipes subordinadas (Supervisores e Staff)
- ✅ **Gerencia:**
- Sua equipe e equipes subordinadas
- Pode criar equipes de supervisão dentro da sua equipe
- ✅ **Pode:**
- Atribuir Supervisores a equipes de supervisão
- Ver relatórios de toda sua hierarquia

3. SUPERVISOR (Nível Supervisão)

- ✅ **Vê:**
- Sua própria equipe
- Apenas os Staff da sua equipe
- ✅ **Gerencia:**
- Apenas sua equipe de Staff
- Turnos e tarefas do seu pessoal
- ✅ **Pode:**
- Adicionar/remover Staff da sua equipe
- Aprovar solicitações dos seus Staff

4. STAFF (Nível Operacional)

- ✅ **Vê:**
- Apenas sua própria equipe
- Membros da mesma equipe
- ❌ **Não gerencia** equipes
- ✅ **Pode:**
- Ver informações da sua equipe
- Interagir com membros da sua equipe

📐 Algoritmo de Hierarquia

Função: Obter Equipes Visíveis

```
function getVisibleTeams(userId: string, userRole: UserRole): Team[] {
  switch(userRole) {
    case 'ADMIN':
      // Admin vê todas as equipes
      return getAllTeams();

    case 'MANAGER':
      // Manager vê sua equipe e todas as subordinadas
      const managerTeam = getUserTeam(userId);
      return [managerTeam, ...getAllDescendantTeams(managerTeam.id)];

    case 'SUPERVISOR':
      // Supervisor vê apenas sua equipe
      return [getUserTeam(userId)];

    case 'STAFF':
      // Staff vê apenas sua equipe
      return [getUserTeam(userId)];
  }
}

// Função recursiva para obter todas as equipes descendentes
function getAllDescendantTeams(teamId: string): Team[] {
  const childTeams = getChildTeams(teamId);
  const descendants = [];

  for (const child of childTeams) {
    descendants.push(child);
    descendants.push(...getAllDescendantTeams(child.id));
  }

  return descendants;
}
```

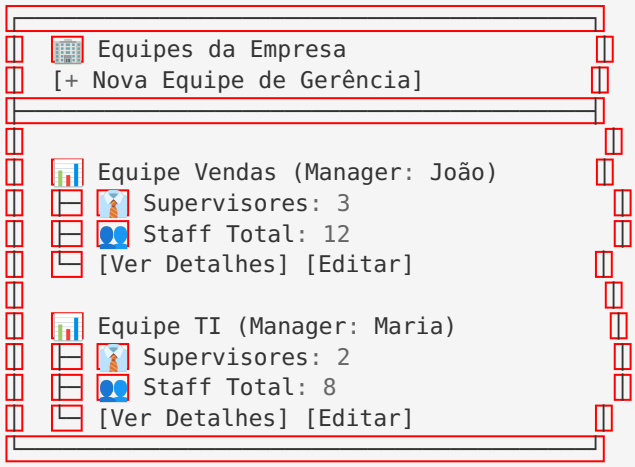
Função: Verificar Permissão

```
function canManageTeam(userId: string, teamId: string): boolean {
  const user = getUser(userId);
  const team = getTeam(teamId);

  // Admin pode gerenciar qualquer equipe
  if (user.role === 'ADMIN') return true;

  // Líder pode gerenciar sua própria equipe
  if (team.leaderId === userId) return true;

  // Manager pode gerenciar equipes subordinadas
  if (user.role === 'MANAGER') {
    const userTeam = getUserTeam(userId);
    return isDescendantTeam(teamId, userTeam.id);
  }

  return false;
}
```

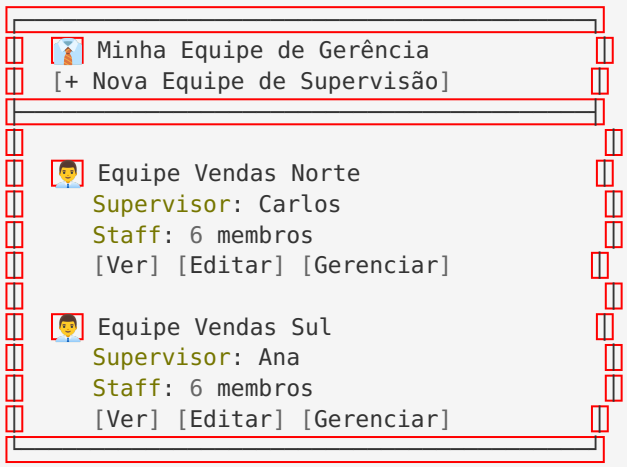
🎨 Interface de Gerenciamento de Equipes

1. Página de Equipes (`/teams`)

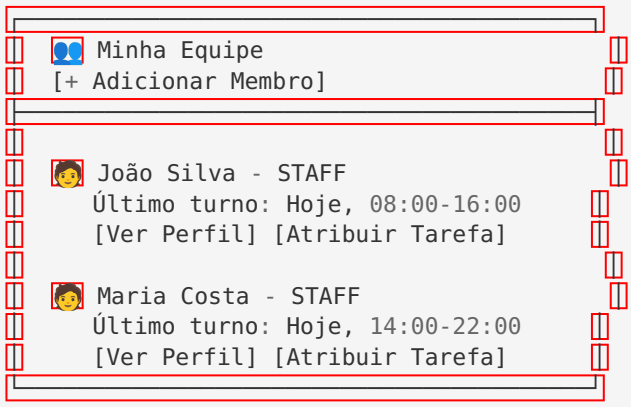
Para ADMIN:



Para MANAGER:



Para SUPERVISOR:



2. Modal de Criação de Equipe

```
interface CreateTeamForm {
  name: string;           // Nome da equipe
  description?: string;   // Descrição
  departmentId: string;   // Departamento
  leaderId: string;       // Líder (Manager ou Supervisor)
  parentTeamId?: string;  // Equipe pai (se aplicável)
  level: TeamLevel;       // Nível automático baseado no líder
}
```

🔄 Fluxo de Criação de Equipe

Cenário 1: Admin cria equipe de Manager

1. Admin acessa `/teams`
2. Clica em “Nova Equipe de Gerência”
3. Preenche:
   - Nome: “Equipe Vendas”
   - Departamento: “Vendas”
   - Líder: João (MANAGER)
4. Sistema define automaticamente:
   - `level`: MANAGEMENT
   - `parentTeamId`: null (é uma equipe de topo)
5. Equipe criada ✅

Cenário 2: Manager cria equipe de Supervisor

1. Manager acessa `/teams`
2. Clica em “Nova Equipe de Supervisão”
3. Preenche:
   - Nome: “Vendas Norte”
   - Líder: Carlos (SUPERVISOR)
4. Sistema define automaticamente:
   - `level`: SUPERVISION
   - `parentTeamId`: ID da equipe do Manager
   - `departmentId`: mesmo do Manager
5. Equipe criada ✅

Cenário 3: Supervisor adiciona Staff

1. Supervisor acessa sua equipe
2. Clica em “Adicionar Membro”
3. Seleciona usuário STAFF
4. Staff é adicionado à equipe ✅

📦 Migração de Dados

Passo 1: Backup

```
# Fazer backup do banco de dados
pg_dump DATABASE_URL > backup_antes_hierarquia.sql
```

Passo 2: Adicionar Campos ao Team

```
// Migração: add_team_hierarchy
- Adicionar: leaderId, parentTeamId, level, description
- Criar índices
```

Passo 3: Migrar Dados Existentes

```
// Script de migração
async function migrateTeamsToHierarchy() {
  const teams = await prisma.team.findMany({ include: { users: true } });

  for (const team of teams) {
    // 1. Encontrar líder (primeiro MANAGER ou SUPERVISOR)
    const leader = team.users.find(u =>
      u.role === 'MANAGER' || u.role === 'SUPERVISOR'
    );

    if (!leader) continue;

    // 2. Definir nível baseado no líder
    const level = leader.role === 'MANAGER'
      ? TeamLevel.MANAGEMENT
      : TeamLevel.SUPERVISION;

    // 3. Atualizar equipe
    await prisma.team.update({
      where: { id: team.id },
      data: {
        leaderId: leader.id,
        level: level,
        parentTeamId: null  // Por enquanto, sem hierarquia
      }
    });
  }
}
```

Passo 4: Criar Equipe do Admin

```
// Criar equipe de nível empresa para cada Admin
async function createAdminTeams() {
  const admins = await prisma.user.findMany({
    where: { role: 'ADMIN' }
  });

  for (const admin of admins) {
    await prisma.team.create({
      data: {
        name: `Equipe ${admin.name}`,
        departmentId: admin.departmentId,
        leaderId: admin.id,
        level: TeamLevel.COMPANY,
        parentTeamId: null
      }
    });
  }
}
```

🎯 API Endpoints Necessários

1. GET `/api/teams` - Listar Equipes

```
// Retorna equipes visíveis baseado no role do usuário
async function GET(req: NextRequest) {
  const session = await getServerSession(authOptions);
  const userId = session.user.id;
  const userRole = session.user.role;

  const teams = await getVisibleTeams(userId, userRole);
  return NextResponse.json(teams);
}
```

2. POST `/api/teams` - Criar Equipe

```
async function POST(req: NextRequest) {
  const session = await getServerSession(authOptions);
  const data = await req.json();

  // Verificar permissão
  if (!canCreateTeam(session.user.id, data.level)) {
    return NextResponse.json({ error: 'Sem permissão' }, { status: 403 });
  }

  // Criar equipe
  const team = await prisma.team.create({ data });
  return NextResponse.json(team);
}
```

3. GET `/api/teams/:id/members` - Membros da Equipe

```
async function GET(req: NextRequest, { params }: { params: { id: string } }) {
  const session = await getServerSession(authOptions);

  // Verificar se pode ver a equipe
  if (!canViewTeam(session.user.id, params.id)) {
    return NextResponse.json({ error: 'Sem permissão' }, { status: 403 });
  }

  const members = await prisma.user.findMany({
    where: { teamId: params.id }
  });

  return NextResponse.json(members);
}
```

4. GET `/api/teams/:id/hierarchy` - Hierarquia da Equipe

```
// Retorna toda a árvore hierárquica de uma equipe
async function GET(req: NextRequest, { params }: { params: { id: string } }) {
  const team = await prisma.team.findUnique({
    where: { id: params.id },
    include: {
      childTeams: {
        include: {
          childTeams: true,  // Recursivo
          members: true,
          leader: true
        }
      },
      members: true,
      leader: true
    }
  });

  return NextResponse.json(team);
}
```

📊 Componentes UI Necessários

1. TeamCard

```
interface TeamCardProps {
  team: Team;
  canManage: boolean;
  onEdit: (id: string) => void;
  onDelete: (id: string) => void;
}
```

2. TeamHierarchyTree

```tsx
// Componente de árvore hierárquica
interface TeamHierarchyTreeProps {
  rootTeam: Team;
  expandedTeams: string[];
  onToggle: (id: string) => void;
}
```

3. CreateTeamModal

```tsx
interface CreateTeamModalProps {
  open: boolean;
  parentTeam?: Team;
  allowedLevels: TeamLevel[];
  onClose: () => void;
  onSuccess: (team: Team) => void;
}
```

4. TeamMembersList

```tsx
interface TeamMembersListProps {
  teamId: string;
  canManage: boolean;
  onAddMember: () => void;
  onRemoveMember: (userId: string) => void;
}
```

🔍 Exemplos de Queries

1. Obter todas as equipes de um Manager

```tsx
const managerTeams = await prisma.team.findMany({
  where: {
    OR: [
      { leaderId: managerId },
      {
        parentTeam: {
          leaderId: managerId
        }
      }
    ]
  },
  include: {
    childTeams: true,
    members: true,
    leader: true
  }
});
```

2. Obter todos os Staff de um Supervisor

```
const supervisorStaff = await prisma.user.findMany({
  where: {
    role: 'STAFF',
    team: {
      leaderId: supervisorId
    }
  }
});
```

3. Verificar se usuário pertence à hierarquia

```
async function isInTeamHierarchy(
  userId: string,
  teamId: string
): Promise<boolean> {
  const user = await prisma.user.findUnique({
    where: { id: userId },
    include: {
      team: {
        include: {
          parentTeam: {
            include: {
              parentTeam: true
            }
          }
        }
      }
    }
  });

  // Verificar se o usuário ou suas equipes ancestrais incluem teamId
  let currentTeam = user.team;
  while (currentTeam) {
    if (currentTeam.id === teamId) return true;
    currentTeam = currentTeam.parentTeam;
  }

  return false;
}
```

✅ Checklist de Implementação

Fase 1: Estrutura de Dados

- [ ] Modificar schema do Prisma (Team model)
- [ ] Adicionar enum TeamLevel
- [ ] Criar migração
- [ ] Testar migração em desenvolvimento
- [ ] Executar script de migração de dados

Fase 2: Backend (APIs)

- [ ] Criar `/api/teams` (GET, POST)

- [ ] Criar `/api/teams/:id` (GET, PUT, DELETE)
- [ ] Criar `/api/teams/:id/members` (GET, POST, DELETE)
- [ ] Criar `/api/teams/:id/hierarchy` (GET)
- [ ] Implementar funções de permissão
- [ ] Criar testes unitários

Fase 3: Frontend (UI)

- [ ] Criar página `/teams`
- [ ] Criar componente TeamCard
- [ ] Criar componente TeamHierarchyTree
- [ ] Criar modal CreateTeamModal
- [ ] Criar modal EditTeamModal
- [ ] Criar componente TeamMembersList
- [ ] Adicionar traduções (PT/EN/FR)

Fase 4: Integração

- [ ] Atualizar filtros de turnos (usar hierarquia)
- [ ] Atualizar filtros de tarefas (usar hierarquia)
- [ ] Atualizar relatórios (considerar hierarquia)
- [ ] Atualizar solicitações (respeitar hierarquia)
- [ ] Atualizar notificações

Fase 5: Testes

- [ ] Testar criação de equipes (Admin)
- [ ] Testar criação de subequipes (Manager)
- [ ] Testar adição de membros (Supervisor)
- [ ] Testar permissões de visualização
- [ ] Testar permissões de edição
- [ ] Testar exclusão em cascata

⚠️ Considerações Importantes

1. Performance

- **Problema:** Queries recursivas podem ser lentas
- **Solução:**
- Usar índices no banco de dados
- Cache de hierarquias frequentes
- Limitar profundidade máxima (ex: 4 níveis)

2. Integridade de Dados

- **Problema:** Ciclos na hierarquia (A pai de B, B pai de A)
- **Solução:**
- Validação antes de salvar
- Constraint no banco de dados

```sql
```

```
-- Prevenir que equipe seja pai de si mesma
ALTER TABLE teams
ADD CONSTRAINT no_self_parent
CHECK (id != parent_team_id);
```

3. Migração de Usuários Existentes

- **Problema:** Usuários sem equipe após migração
- **Solução:**
- Criar equipe “default” para cada departamento
- Atribuir usuários órfãos a essas equipes

4. Exclusão de Equipes

- **Problema:** O que fazer com membros ao excluir equipe?
- **Solução:**
- Não permitir exclusão de equipe com membros
- Ou: mover membros para equipe pai
- Ou: criar equipe “órfãos” temporária

📈 Benefícios da Implementação

✅ **Organização Clara:** Estrutura hierárquica reflete organograma real
✅ **Controle de Acesso:** Permissões baseadas em hierarquia
✅ **Relatórios Melhores:** Métricas por equipe e subequipes
✅ **Escalabilidade:** Sistema cresce com a empresa
✅ **Visibilidade Apropriada:** Cada um vê apenas o que precisa
✅ **Gestão Simplificada:** Managers/Supervisores gerenciam suas equipes

---

📋 Próximos Passos

1. ✅ **Revisar este plano** com o Bruno
2. ⏸️ **Aprovar estrutura de dados**
3. ⏸️ **Implementar Fase 1** (Estrutura de Dados)
4. ⏸️ **Implementar Fase 2** (Backend)
5. ⏸️ **Implementar Fase 3** (Frontend)
6. ⏸️ **Testar em produção**
7. ⏸️ **Documentar para usuários finais**

Documento criado em: 16 de outubro de 2025
Versão: 1.0
Autor: DeepAgent AI
Projeto: OrganiZen - Sistema de Gestão Hierárquica