📚 DOCUMENTAÇÃO TÉCNICA COMPLETA - RecruitAI

🎯 Visão Geral do Sistema

Nome: RecruitAI - Plataforma de Recrutamento e Seleção Inteligente
Versão: 2.0.0
Data de Criação: Novembro 2025
Última Atualização: 26 de Novembro de 2025

🏗️ ARQUITETURA DO SISTEMA

Stack Tecnológico

Frontend

- **Framework:** Next.js 14.2.28 (App Router)
- **React:** 18.2.0
- **TypeScript:** 5.2.2
- **Estilização:** Tailwind CSS 3.3.3 + Shadcn/ui
- **Gerenciamento de Estado:** Zustand 5.0.3, Jotai 2.6.0
- **Formulários:** React Hook Form 7.53.0 + Zod 3.23.8
- **Data Fetching:** SWR 2.2.4, TanStack Query 5.0.0
- **Gráficos:** Chart.js 4.4.9 + React-Chartjs-2 5.3.0, Plotly.js 2.35.3
- **Animações:** Framer Motion 10.18.0

Backend

- **Runtime:** Node.js 22.x
- **API:** Next.js API Routes (App Router)
- **Autenticação:** NextAuth.js 4.24.11
- **ORM:** Prisma 6.7.0
- **Validação:** Zod 3.23.8
- **Criptografia:** bcryptjs 2.4.3

Banco de Dados

- **Tipo:** PostgreSQL 15+
- **ORM:** Prisma ORM
- **Provider Atual:** HostedDB (Abacus.AI)

Armazenamento

- **Tipo:** AWS S3
- **SDK:** @aws-sdk/client-s3 v3
- **Uso:** Armazenamento de currículos e logos de empresas

Pagamentos

- **Gateway:** Stripe
- **Integração:** Stripe Node.js SDK
- **Webhooks:** Configurados para todos os eventos de assinatura

IA/ML

- **Provider:** Abacus.AI (Gemini)
- **Uso:** Análise automática de currículos e matching

📁 ESTRUTURA DE DIRETÓRIOS

```
ats_platform/
├── nextjs_space/                    # Aplicação Next.js principal
│   ├── app/                         # App Router
│   │   ├── admin/                   # Painel administrativo
│   │   │   ├── applications/        # Gestão de candidaturas
│   │   │   ├── companies/           # Gestão de empresas
│   │   │   ├── jobs/                # Gestão de vagas
│   │   │   └── profile/             # Perfil do admin
│   │   ├── api/                     # Endpoints da API
│   │   │   ├── admin/               # APIs administrativas
│   │   │   ├── ai/                  # APIs de IA
│   │   │   ├── applications/        # APIs de candidaturas
│   │   │   ├── auth/                # APIs de autenticação
│   │   │   ├── calendar/            # APIs de calendário
│   │   │   ├── candidates/          # APIs de candidatos
│   │   │   ├── checkout/            # APIs de pagamento
│   │   │   ├── company-users/       # APIs de membros da equipe
│   │   │   ├── dashboard/           # APIs de dashboard
│   │   │   ├── jobs/                # APIs de vagas
│   │   │   ├── locations/           # APIs de localização
│   │   │   ├── notifications/       # APIs de notificações
│   │   │   ├── permissions/         # APIs de permissões
│   │   │   ├── plans/               # APIs de planos
│   │   │   ├── subscriptions/       # APIs de assinaturas
│   │   │   ├── tasks/               # APIs de tarefas
│   │   │   ├── team-groups/         # APIs de grupos
│   │   │   └── webhooks/            # Webhooks (Stripe)
│   │   ├── auth/                    # Páginas de autenticação
│   │   ├── candidate/               # Área do candidato
│   │   ├── dashboard/               # Área da empresa
│   │   ├── pricing/                 # Página de planos
│   │   └── vagas/                   # Páginas públicas de vagas
│   ├── components/                  # Componentes React
│   │   ├── applications/            # Componentes de candidaturas
│   │   ├── notifications/           # Componentes de notificação
│   │   └── ui/                      # Componentes UI (Shadcn)
│   ├── hooks/                       # Custom React Hooks
│   ├── lib/                         # Bibliotecas e utilitários
│   │   ├── admin.ts                 # Helpers admin
│   │   ├── auth.ts                  # Configuração NextAuth
│   │   ├── aws-config.ts            # Configuração AWS
│   │   ├── db.ts                    # Cliente Prisma
│   │   ├── email.ts                 # Utilitários de email
│   │   ├── locations.ts             # Dados de localização
│   │   ├── professions.ts           # Lista de profissões
│   │   ├── s3.ts                    # Utilitários S3
│   │   ├── stripe.ts                # Cliente Stripe
│   │   ├── types.ts                 # Definições de tipos
│   │   ├── utils.ts                 # Utilitários gerais
│   │   └── verification.ts          # Verificação de email
│   ├── prisma/                      # Configuração Prisma
│   │   └── schema.prisma            # Schema do banco
│   ├── public/                      # Arquivos estáticos
│   └── scripts/                     # Scripts utilitários
│       ├── seed.ts                  # Seed do banco
│       └── cleanup_*.ts             # Scripts de limpeza
├── DOCUMENTACAO_TECNICA.md          # Este arquivo
├── DEPLOY_VERCEL.md                 # Guia de deploy
└── STRIPE_SETUP.md                  # Guia Stripe
```

💾 MODELO DE DADOS (DATABASE SCHEMA)

Entidades Principais

1. User (Usuário Principal)

```
model User {
  id             String    @id @default(cuid())
  name           String?
  email          String    @unique
  emailVerified DateTime?
  password       String?
  image          String?
  role           String    @default("candidate") // candidate, company, superadmin

  // Campos específicos de empresa
  companyName    String?   // Razão social
  tradeName      String?   // Nome fantasia
  cnpj           String?
  phone          String?
  address        String?
  city           String?
  state          String?
  logoUrl        String?

  // Relações
  accounts            Account[]
  sessions            Session[]
  jobs                Job[]
  subscriptions       Subscription[]
  notifications       Notification[]
  companyUsers        CompanyUser[] @relation("CompanyUsers")
  teamGroups          TeamGroup[] @relation("CompanyTeamGroups")
  memberPermissions  MemberPermission[] @relation("CompanyPermissions")
  events              CalendarEvent[] @relation("UserEvents")
  tasks               Task[] @relation("UserTasks")
}
```

2. CandidateProfile (Perfil do Candidato)

```
model CandidateProfile {
  id              String   @id @default(cuid())
  email           String   @unique
  fullName        String?
  phone           String?
  dateOfBirth     DateTime?
  address         String?
  street          String?
  neighborhood    String?
  city            String?
  state           String?
  country         String?
  zipCode         String?
  profession      String?
  photoUrl        String?
  linkedinUrl     String?
  instagramUrl    String?
  resumeUrl       String?  // S3 path
  hasNoExperience Boolean @default(false)

  // Relações
  applications    Application[]
  education       Education[]
  experiences     Experience[]
  skills          Skill[]
  courses         Course[]
  certifications  Certification[]
}
```

3. Job (Vaga)

```
model Job {
  id          String   @id @default(cuid())
  userId      String
  title       String
  description String   @db.Text
  requirements String? @db.Text
  location    String
  country     String   @default("BR")
  state       String?
  city        String?
  type        String   // full-time, part-time, contract, internship
  workMode    String   @default("presencial") // remoto, hibrido, presencial
  status      String   @default("active") // active, closed, paused

  // Relações
  user          User          @relation(fields: [userId], references: [id], onDelete:
Cascade)
  applications  Application[]
  criteria      JobCriteria[]
  stages        JobStage[]
}
```

4. Application (Candidatura)

```
model Application {
  id                  String    @id @default(cuid())
  candidateId         String
  jobId               String
  status              String    @default("pending") // pending, approved, rejected, h
ired

  // Análise de IA
  resumeAnalysis      Json?
  compatibilityScore  Int?
  aiClassification    String?   // 🟢🔵🟡🔴

  // Processo de seleção
  currentStageId      String?
  invitedForInterview Boolean   @default(false)
  invitedAt           DateTime?
  interviewDate       DateTime?
  interviewLink       String?
  attendedInterview   Boolean?
  interviewNotes      String?   @db.Text

  // Contratação
  isHired             Boolean   @default(false)
  hiredAt             DateTime?
  hiredBy             String?

  // Relações
  candidate    CandidateProfile @relation(fields: [candidateId], references: [id])
  job          Job              @relation(fields: [jobId], references: [id])
  currentStage JobStage?        @relation(fields: [currentStageId], references: [id])
}
```

5. Plan (Plano de Assinatura)

```
model Plan {
  id            String   @id @default(cuid())
  name          String   @unique // free, bronze, prata, ouro, personalizado
  displayName   String
  price         Float
  jobLimit      Int      // Número de vagas por mês
  memberLimit   Int      @default(1) // Limite de membros da equipe
  features      String[]
  isActive      Boolean  @default(true)
  stripePriceId String?

  subscriptions Subscription[]
}
```

6. Subscription (Assinatura)

```
model Subscription {
  id                    String    @id @default(cuid())
  userId                String
  planId                String
  status                String    // trial, active, past_due, canceled, expired, grace
_period
  startDate             DateTime  @default(now())
  endDate               DateTime?
  trialEndDate          DateTime?
  gracePeriodEndDate    DateTime? // Data limite do período de graça
  gracePeriodDays       Int?      // Número de dias de graça concedidos
  suspensionReason      String?   // Motivo da suspensão/problema
  jobsCreatedThisMonth  Int       @default(0)
  lastResetDate         DateTime  @default(now())

  // Stripe fields
  stripeCustomerId      String?   @unique
  stripeSubscriptionId  String?   @unique
  stripeCheckoutSessionId String? @unique

  user User @relation(fields: [userId], references: [id])
  plan Plan @relation(fields: [planId], references: [id])
}
```

7. MemberPermission (Permissões de Membros)

```
model MemberPermission {
  id                      String   @id @default(cuid())
  companyId               String
  name                    String   // Nome do perfil (ex: "Gerente", "Recrutador")

  // Permissões de visualização
  canViewAllJobs          Boolean  @default(true)
  canViewOwnGroupJobs     Boolean  @default(false)
  canViewAllProfiles      Boolean  @default(true)
  canViewAllApplications  Boolean  @default(true)
  canViewOwnApplications  Boolean  @default(false)
  canViewTeamMembers      Boolean  @default(true)
  canViewAllGroups        Boolean  @default(true)
  canViewOwnGroup         Boolean  @default(false)

  // Permissões de ação
  canCreateJobs           Boolean  @default(false)
  canEditJobs             Boolean  @default(false)
  canDeleteJobs           Boolean  @default(false)
  canManageApplications   Boolean  @default(false)
  canHireCandidates       Boolean  @default(false)
  canInviteInterviews     Boolean  @default(false)
  canManageTeam           Boolean  @default(false)
  canManageGroups         Boolean  @default(false)
  canAccessReports        Boolean  @default(false)

  company                 User     @relation("CompanyPermissions", fields:
[companyId], references: [id])
  members                 CompanyUser[]
}
```

8. CompanyUser (Membro da Equipe)

```
model CompanyUser {
  id            String   @id @default(cuid())
  companyId     String
  groupId       String?
  permissionId  String?
  name          String
  email         String   @unique
  password      String
  role          String   @default("member") // admin, member
  isActive      Boolean  @default(true)

  company       User             @relation("CompanyUsers", fields: [companyId], ref-
erences: [id])
  group         TeamGroup?       @relation(fields: [groupId], references: [id])
  permission    MemberPermission? @relation(fields: [permissionId], references: [id])
}
```

9. TeamGroup (Grupo de Equipe)

```
model TeamGroup {
  id          String   @id @default(cuid())
  companyId   String
  name        String
  description String?
  color       String?  // Para identificação visual

  company     User          @relation("CompanyTeamGroups", fields: [companyId], refer-
ences: [id])
  members     CompanyUser[]
}
```

10. SupportTicket (Ticket de Suporte)

```
model SupportTicket {
  id          String   @id @default(cuid())
  companyId   String
  subject     String
  status      String   @default("open") // open, in_progress, waiting_company, closed
  priority    String   @default("medium") // low, medium, high
  closedAt    DateTime?
  createdAt   DateTime @default(now())
  updatedAt   DateTime @updatedAt

  company     User             @relation(fields: [companyId], references: [id])
  messages    SupportMessage[]
}
```

11. SupportMessage (Mensagem de Suporte)

```
model SupportMessage {
  id        String   @id @default(cuid())
  ticketId  String
  senderId  String
  message   String   @db.Text
  isAdmin   Boolean  @default(false)
  createdAt DateTime @default(now())

  ticket    SupportTicket @relation(fields: [ticketId], references: [id])
  sender    User          @relation(fields: [senderId], references: [id])
}
```

12. MemberInvitation (Convite de Membro)

```
model MemberInvitation {
  id            String   @id @default(cuid())
  companyId     String
  email         String
  name          String
  groupId       String?
  permissionId  String?
  token         String   @unique
  status        String   @default("pending") // pending, accepted, expired
  expiresAt     DateTime
  createdAt     DateTime @default(now())
  acceptedAt    DateTime?

  company       User     @relation(fields: [companyId], references: [id])
}
```

13. CompanyInvitation (Convite de Empresa - Plano Personalizado)

```
model CompanyInvitation {
  id                  String   @id @default(cuid())
  email               String
  companyName         String   // Razão social
  tradeName           String?  // Nome fantasia
  cnpj                String?
  phone               String?
  token               String   @unique

  // Detalhes do Plano Personalizado
  customPlanName      String
  customJobLimit      Int
  customPrice         Float
  customFeatures      String[]

  // Stripe
  stripeCustomerId    String?
  stripeCheckoutUrl   String?

  // Status
  status              String   @default("pending") // pending, password_set, pay-
ment_pending, completed, expired
  expiresAt           DateTime
  createdAt           DateTime @default(now())
  completedAt         DateTime?

  // Admin
  createdBy           String
  notes               String?  @db.Text

  admin               User     @relation(fields: [createdBy], references: [id])
}
```

14. MaintenanceLog (Log de Manutenção)

```
model MaintenanceLog {
  id          String   @id @default(cuid())
  action      String
  params      Json?
  result      Json?
  status      String   // success, error
  executedAt  DateTime @default(now())
}
```

15. PasswordResetToken (Token de Recuperação de Senha)

```
model PasswordResetToken {
  id        String   @id @default(cuid())
  email     String
  token     String   @unique
  expiresAt DateTime
  createdAt DateTime @default(now())
}
```

16. CustomPlanRequest (Solicitação de Plano Personalizado)

```
model CustomPlanRequest {
  id        String   @id @default(cuid())
  name      String
  email     String
  phone     String?
  message   String?  @db.Text
  status    String   @default("pending") // pending, contacted, converted, rejected
  notes     String?  @db.Text
  createdAt DateTime @default(now())
  updatedAt DateTime @updatedAt
}
```

Outras Entidades

- **Account** - OAuth accounts (NextAuth)
- **Session** - User sessions (NextAuth)
- **VerificationToken** - Email verification tokens
- **Education** - Formação acadêmica do candidato
- **Experience** - Experiência profissional
- **Skill** - Habilidades
- **Course** - Cursos
- **Certification** - Certificações
- **JobCriteria** - Critérios de avaliação da vaga
- **JobStage** - Fases de seleção da vaga
- **Notification** - Notificações do sistema
- **CalendarEvent** - Eventos do calendário
- **Task** - Tarefas (to-do list)

🔐 SISTEMA DE AUTENTICAÇÃO

NextAuth.js Configuration

Arquivo: `lib/auth.ts`

Providers Configurados

1. **Credentials Provider** - Login com email/senha
2. **Google Provider** - OAuth Google
3. **LinkedIn Provider** - OAuth LinkedIn

Verificação de Email

- Tokens de verificação com validade de 24 horas
- Email obrigatório antes do primeiro login
- Processo: Cadastro → Email → Verificação → Login

Roles do Sistema

- **candidate** - Candidato a vagas
- **company** - Empresa/Recrutador
- **superadmin** - Administrador da plataforma

Session Management

```
callbacks: {
  jwt: async ({ token, user, trigger, session }) => {
    // Adicionar role, companyName e logoUrl ao token
    if (user) {
      token.role = user.role;
      token.companyName = user.companyName;
      token.logoUrl = user.logoUrl;
    }
    // Suportar atualizações dinâmicas
    if (trigger === "update" && session?.logoUrl) {
      token.logoUrl = session.logoUrl;
    }
    return token;
  },
  session: async ({ session, token }) => {
    // Expor dados no cliente
    session.user.id = token.sub;
    session.user.role = token.role;
    session.user.companyName = token.companyName;
    session.user.logoUrl = token.logoUrl;
    return session;
  }
}
```

💳 SISTEMA DE PAGAMENTOS E ASSINATURAS

Integração Stripe

Arquivo de Configuração: `lib/stripe.ts`

Planos Disponíveis

Plano	Preço	Vagas/Mês	Membros	Features
Teste Grátis	R$ 0	5 (7 dias)	1	Todos os recursos por 7 dias
Bronze	R$ 300	25	4	IA, Dashboard, Permissões
Prata	R$ 500	50	15	Bronze + Job Boards, Relatórios
Ouro	R$ 800	100	30	Prata + API, Suporte 24/7
Personalizado	Custom	Ilimitado	Ilimitado	Tudo + Customizações

Fluxo de Assinatura

1. **Cadastro**
   - Empresa seleciona plano
   - Se FREE → Cria trial de 7 dias automaticamente
   - Se PAGO → Redireciona para Stripe Checkout

2. **Checkout**
   - API: `/api/checkout/create-session`
   - Métodos: Cartão, PIX, Boleto
   - Metadados: userId, planId, planName

3. **Webhooks**
   - Endpoint: `/api/webhooks/stripe`
   - Eventos monitorados:
     - `checkout.session.completed`
     - `customer.subscription.created`
     - `customer.subscription.updated`
     - `customer.subscription.deleted`
     - `invoice.paid`
     - `invoice.payment_failed`

4. **SubscriptionGuard**
   - Componente: `components/ui/subscription-guard.tsx`
   - Protege rotas que requerem assinatura ativa
   - Bloqueia acesso quando limite de vagas é atingido

Controle de Limites

Vagas por Mês:

```
// Verificação ao criar vaga
const subscription = await db.subscription.findFirst({
  where: { userId, status: { in: ['trial', 'active'] } },
  include: { plan: true }
});

if (subscription.jobsCreatedThisMonth >= subscription.plan.jobLimit) {
  // Bloquear criação
}
```

Membros por Plano:

```
// Verificação ao adicionar membro
const currentMemberCount = await db.companyUser.count({
  where: { companyId, isActive: true }
});

if (currentMemberCount >= subscription.plan.memberLimit) {
  // Bloquear adição
}
```

Validação de Domínio:

```
// Todos os membros devem ter o mesmo domínio do email da empresa
const companyDomain = companyUser.email.split('@')[1];
const memberDomain = newMemberEmail.split('@')[1];

if (companyDomain !== memberDomain) {
  // Rejeitar
}
```

🤖 SISTEMA DE INTELIGÊNCIA ARTIFICIAL

Análise Automática de Currículos

Endpoint: `/api/ai/analyze-resume`

Processo de Análise

1. **Entrada:**
   - Currículo (PDF/DOC/DOCX) armazenado no S3
   - Critérios da vaga (peso de cada critério)
   - Informações da vaga

2. **Processamento:**
   ```typescript
   // Extração de dados do currículo
   const extractedData = await analyzeResumeWithAI(resumeContent, {
   name, email, education, experience, skills
   });

// Cálculo de compatibilidade
const score = calculateCompatibility(extractedData, jobCriteria);

// Classificação visual
const classification = getClassification(score);
// 🟢 Excelente (80-100%)
// 🔵 Boa (60-79%)
// 🟡 Média (40-59%)
// 🔴 Baixa (0-39%)
```

1. **Saída:**
   ```json
   {
     "compatibilityScore": 85,
     "aiClassification": "🟢",
     "resumeAnalysis": {
       "summary": "Candidato altamente qualificado...",
       "strengths": ["5+ anos de experiência", "..."],
       "concerns": ["Sem certificação X", "..."],
       "criteriaMatch": [
         { "criterion": "Experiência", "match": 90, "explanation": "..." }
       ]
   ```

```
    }
  }
```

Provider de IA

Atual: Abacus.AI (Gemini 1.5)
- Endpoint: API da Abacus.AI
- Modelo: Gemini 1.5 Flash
- API Key: Configurada em `.env`

📦 SISTEMA DE ARMAZENAMENTO (AWS S3)

Configuração

Arquivos:
- `lib/aws-config.ts` - Configuração do S3 Client
- `lib/s3.ts` - Utilitários de upload/download

Funções Disponíveis

```
// Upload de arquivo
const s3Key = await uploadFile(fileBuffer, fileName);
// Retorna: "uploads/1234567890-resume.pdf"

// Download de arquivo (URL assinada)
const signedUrl = await downloadFile(s3Key);
// Retorna URL válida por 1 hora

// Deletar arquivo
await deleteFile(s3Key);
```

Estrutura de Pastas no S3

```
bucket-name/
├── uploads/
│   ├── TIMESTAMP-resume.pdf
│   └── TIMESTAMP-outro.pdf
└── logos/
    ├── TIMESTAMP-company-logo.png
    └── TIMESTAMP-outro-logo.jpg
```

Segurança

- URLs assinadas com validade de 1 hora
- Apenas o S3 key é armazenado no banco (nunca URLs)
- Geração de URLs sob demanda

🔔 SISTEMA DE NOTIFICAÇÕES

Tipos de Notificações

1. **Aplicação Recebida** - Empresa recebe nova candidatura

2. **Mudança de Status** - Candidato é movido de fase
3. **Convite para Entrevista** - Candidato recebe convite
4. **Contratação** - Candidato é contratado
5. **Sistema** - Avisos gerais

Implementação

API: `/api/notifications`

```
// Criar notificação
await db.notification.create({
  data: {
    userId,
    type: 'application',
    title: 'Nova candidatura',
    message: 'João Silva se candidatou para Desenvolvedor',
    link: '/dashboard/jobs/123'
  }
});

// Marcar como lida
await db.notification.update({
  where: { id },
  data: { isRead: true }
});
```

Componente UI

Arquivo: `components/notifications/notification-bell.tsx`

- Bell icon com badge de não lidas
- Popover com lista de notificações
- Auto-refresh a cada 30 segundos
- Links diretos para páginas relacionadas

🎨 SISTEMA DE PERMISSÕES

Perfis de Permissão

Criados pela empresa para controlar acesso dos membros.

API: `/api/permissions`

Permissões Disponíveis

Visualização:
- Ver todas as vagas / apenas do próprio grupo
- Ver perfis de candidatos
- Ver todas as candidaturas / apenas das próprias vagas
- Ver membros da equipe
- Ver todos os grupos / apenas o próprio

Ações:
- Criar vagas
- Editar vagas

- Deletar vagas
- Gerenciar candidaturas (avançar fases)
- Contratar candidatos
- Convidar para entrevistas
- Gerenciar equipe
- Gerenciar grupos
- Acessar relatórios

Uso

```
// Criar perfil de permissão
POST /api/permissions
{
  "name": "Recrutador",
  "canViewAllJobs": true,
  "canCreateJobs": true,
  "canManageApplications": true,
  "canInviteInterviews": true,
  ...
}

// Atribuir a um membro
PATCH /api/company-users/[id]
{
  "permissionId": "perm_123"
}
```

🔑 SISTEMA DE RECUPERAÇÃO DE SENHA

Fluxo de Recuperação

Arquivos: `lib/password-reset.ts`, `/api/auth/forgot-password`, `/api/auth/reset-password`

Passo 1: Solicitar Recuperação

```
POST /api/auth/forgot-password
{
  "email": "usuario@exemplo.com"
}
```

- Gera token de recuperação com validade de 1 hora
- Envia email com link de redefinição
- Retorna sempre sucesso (anti-enumeração de emails)

Passo 2: Redefinir Senha

```
POST /api/auth/reset-password
{
  "token": "abc123...",
  "password": "novaSenha123",
  "confirmPassword": "novaSenha123"
}
```

- Valida token e expiração
- Hash da nova senha (bcrypt)
- Atualiza senha no banco
- Invalida token usado

🎫 SISTEMA DE SUPORTE

Para Empresas

Área de Acesso: `/dashboard/support`

Criar Ticket

```
POST /api/support
{
  "subject": "Problema com upload de currículos",
  "message": "Descrição detalhada do problema...",
  "priority": "high" // low, medium, high
}
```

Adicionar Mensagem ao Ticket

```
POST /api/support/[ticketId]
{
  "message": "Nova informação sobre o problema..."
}
```

Para Superadmins

Área de Acesso: `/admin/support`

Listar Todos os Tickets

```
GET /api/support
// Retorna todos os tickets de todas as empresas
```

Responder Ticket

```
POST /api/support/[ticketId]
{
  "message": "Resposta do suporte..."
}
// Marca automaticamente isAdmin: true
```

Atualizar Status/Prioridade

```
PATCH /api/support/[ticketId]
{
  "status": "in_progress", // open, in_progress, waiting_company, closed
  "priority": "high"
}
```

Notificações Automáticas

- **Empresa cria ticket** → Notifica todos os admins
- **Admin responde** → Notifica empresa (status muda para waiting_company)
- **Empresa responde** → Notifica todos os admins
- **Ticket fechado** → Registra data de fechamento

👥 SISTEMA DE CONVITES DE MEMBROS

Fluxo de Convite

API: `/api/member-invitations`

Passo 1: Empresa Convida Membro

```
POST /api/member-invitations
{
  "email": "membro@empresa.com",
  "name": "João Silva",
  "groupId": "group_123",      // Opcional
  "permissionId": "perm_456"   // Opcional
}
```

Validações Automáticas:
- ✅ Email deve ter o mesmo domínio da empresa
- ✅ Email não pode já existir como usuário ou em outro convite pendente
- ✅ Respeita limite de membros do plano da empresa

Ação:
- Gera token único
- Define validade de 7 dias
- Envia email com link de aceitação

Passo 2: Membro Aceita Convite

Página: `/member-invite/[token]`

```
// Verificar convite
GET /api/member-invitations/[token]
// Retorna detalhes (email, nome, companyName)

// Aceitar e criar senha
POST /api/member-invitations/[token]
{
  "password": "senha123",
  "confirmPassword": "senha123"
}
```

Ação:
- Cria conta `CompanyUser`
- Marca convite como aceito
- Notifica empresa
- Redireciona para login

🏢 SISTEMA DE EMPRESAS COM PLANO PERSONALIZADO

Fluxo Completo (Superadmin → Empresa)

Passo 1: Superadmin Cria Convite

Área: `/admin/custom-companies`

```
POST /api/admin/companies/create-with-custom-plan
{
  "email": "empresa@exemplo.com",
  "companyName": "Empresa XYZ Ltda",
  "tradeName": "XYZ Tech",           // Opcional
  "cnpj": "12345678000190",          // Opcional
  "phone": "+5511999999999",         // Opcional

  // Plano Personalizado
  "customJobLimit": 150,
  "customPrice": 1200.00,
  "customFeatures": [
    "150 vagas por mês",
    "Membros ilimitados",
    "Suporte prioritário 24/7",
    "API dedicada"
  ],

  "notes": "Cliente VIP - atender com prioridade"  // Interno
}
```

Ação:
- Cria `CompanyInvitation`
- Gera token com validade de 7 dias
- Envia email para empresa com detalhes do plano e link de configuração

Passo 2: Empresa Configura Conta

Página: `/company-setup/[token]`

2.1 - Criar Senha:

```
POST /api/company-setup/create-password
{
  "token": "abc123...",
  "password": "senha123",
  "confirmPassword": "senha123"
}
```

Ação:
- Cria usuário `company` no banco
- Usa dados do convite (email, companyName, cnpj, phone, etc.)
- Atualiza status do convite para `password_set`

2.2 - Realizar Pagamento:

```
POST /api/company-setup/create-checkout
{
  "token": "abc123..."
}
```

Ação:
- Cria/recupera Stripe Customer
- Cria Checkout Session com plano personalizado
- Atualiza convite com `stripeCheckoutUrl`
- Status muda para `payment_pending`
- Redireciona para Stripe

Passo 3: Webhook Stripe Processa Pagamento

Endpoint: `/api/webhooks/stripe`

Quando `checkout.session.completed` é recebido:
- Verifica se é plano personalizado (metadata)
- **Cria novo `Plan` no banco** com dados do convite
- Marca `plan.isCustom = true` e `plan.customCompanyId = userId`
- Cria `Subscription` para a empresa usando o novo plano
- Atualiza `CompanyInvitation` para status `completed`
- Envia email de confirmação

Resultado Final:
- Empresa tem conta ativa
- Plano personalizado exclusivo criado
- Assinatura ativa por 30 dias
- Pronta para usar a plataforma

Gestão de Convites (Superadmin)

```
// Listar convites
GET /api/admin/company-invitations?status=pending

// Ver detalhes
GET /api/admin/company-invitations/[id]

// Atualizar notas/status
PATCH /api/admin/company-invitations/[id]
{
  "notes": "Cliente pagou, liberar acesso",
  "status": "completed"
}

// Deletar convite
DELETE /api/admin/company-invitations/[id]
```

🔧 API DE MANUTENÇÃO REMOTA

Visão Geral

Permite que a equipe Abacus.AI realize manutenções sem acesso SSH ao servidor.

Documentação Completa: `API_MANUTENCAO.md`

Autenticação

Todas as requisições exigem:

```
Authorization: Bearer <MAINTENANCE_SECRET>
```

Endpoints Principais

1. Status do Sistema

```
GET /api/maintenance/status
// Retorna: banco, servidor, memória, estatísticas
```

2. Executar Ações

```
POST /api/maintenance/execute
{
  "action": "restart_server" | "clear_cache" | "prisma_generate" |
            "prisma_push" | "run_seed" | "cleanup_orphans" | "get_logs",
  "params": { ... }  // Opcional
}
```

3. Ver Histórico

```
GET /api/maintenance/logs?limit=50&status=success
// Retorna log de todas as manutenções executadas
```

Ações Disponíveis

Ação	Descrição	Quando Usar
`restart_server`	Reinicia Next.js	Sistema travado
`clear_cache`	Limpa .next e .build	Após mudanças
`check_database`	Verifica conectividade	Diagnóstico
`prisma_generate`	Gera Prisma Client	Erro de Prisma
`prisma_push`	Aplica schema	Mudanças no schema
`run_seed`	Popula banco	Inicialização
`cleanup_orphans`	Remove órfãos	Limpeza
`get_logs`	Obtém logs	Debug

Cenário de Uso: Sistema Parou

```
# 1. Verificar status
curl -X GET "https://app.com/api/maintenance/status" \
  -H "Authorization: Bearer <TOKEN>"

# 2. Ver logs
curl -X POST "https://app.com/api/maintenance/execute" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{"action": "get_logs", "params": {"lines": 50}}'

# 3. Reiniciar
curl -X POST "https://app.com/api/maintenance/execute" \
  -H "Authorization: Bearer <TOKEN>" \
  -H "Content-Type: application/json" \
  -d '{"action": "restart_server"}'
```

💌 SISTEMA DE EMAILS

Configuração SMTP

Arquivo: `lib/email.ts`

```
SMTP_HOST=smtp.zoho.com
SMTP_PORT=587
SMTP_USER=noreply@fcmtech.com.br
SMTP_PASS=<senha_do_email>
SMTP_FROM_NAME=RecruitAI
```

Templates Disponíveis

```
emailTemplates = {
  // Candidato
  applicationReceived: { subject, html, text },
  applicationStatusUpdate: { subject, html, text },
  interviewInvite: { subject, html, text },

  // Empresa
  trialExpiry: { subject, html, text },
  subscriptionConfirmed: { subject, html, text },

  // Verificação
  emailVerification: { subject, html, text },
  passwordReset: { subject, html, text },

  // Convites
  memberInvitation: { subject, html, text },
  companyInvitation: { subject, html, text },

  // Suporte
  supportTicketCreated: { subject, html, text },
  supportTicketReply: { subject, html, text }
}
```

Envio de Email

```
import { sendEmail, emailTemplates } from '@/lib/email';

await sendEmail({
  to: user.email,
  subject: emailTemplates.passwordReset.subject,
  html: emailTemplates.passwordReset.html(resetUrl),
  text: emailTemplates.passwordReset.text(resetUrl)
});
```

📊 PAINEL ADMINISTRATIVO

Recursos do Superadmin

Acesso: `/admin`

1. Gestão de Empresas

- Listar todas as empresas
- Ver detalhes e métricas
- Editar informações
- Deletar empresas (com confirmação)

2. Gestão de Assinaturas

- Listar todas as assinaturas
- Filtrar por status
- **Conceder período de graça:**
  `typescript`

```
POST /api/admin/subscriptions/[id]/grace-period
{ "days": 10, "reason": "Problema no pagamento" }
```

- **Suspender assinatura:**

```typescript
POST /api/admin/subscriptions/[id]/suspend
{ "reason": "Violação dos termos" }
```

- **Reativar assinatura:**

```typescript
POST /api/admin/subscriptions/[id]/reactivate
```

3. Gestão de Administradores

- Listar administradores
- **Adicionar novo admin:**

```typescript
POST /api/admin/users
{ "name": "Novo Admin", "email": "...", "password": "..." }
```

- Remover administrador (mantém pelo menos 1)
- **Trocar senha:**

```typescript
POST /api/admin/change-password
{
  "currentPassword": "...",
  "newPassword": "...",
  "confirmPassword": "..."
}
```

4. Dashboard Global

- Total de empresas, vagas, candidaturas
- Estatísticas de status
- Atividade recente
- Gráficos de tendências

🚀 VARIÁVEIS DE AMBIENTE

Arquivo `.env`

```
# Database
DATABASE_URL="postgresql://user:pass@host:5432/db"

# NextAuth
NEXTAUTH_SECRET="secret_forte_aqui"
NEXTAUTH_URL="http://localhost:3000" # Produção: https://seudominio.com

# Stripe (Pagamentos)
STRIPE_SECRET_KEY="sk_test_..."
STRIPE_PUBLISHABLE_KEY="pk_test_..."
STRIPE_WEBHOOK_SECRET="whsec_..."

# AWS S3 (Armazenamento)
AWS_REGION="us-west-2"
AWS_S3_REGION="us-west-2"
AWS_BUCKET_NAME="seu-bucket"
AWS_S3_BUCKET_NAME="seu-bucket"
AWS_FOLDER_PREFIX="resumes/"
AWS_S3_FOLDER_PREFIX="resumes/"
AWS_ACCESS_KEY_ID="..."         # Opcional se usar IAM
AWS_SECRET_ACCESS_KEY="..."     # Opcional se usar IAM

# Abacus.AI (IA)
ABACUSAI_API_KEY="5bb8032f287b4b89bfcae4529b50a199"

# Email SMTP (Obrigatório)
SMTP_HOST="smtp.zoho.com"
SMTP_PORT="587"
SMTP_USER="noreply@fcmtech.com.br"
SMTP_PASS="senha_do_email"
SMTP_FROM_NAME="RecruitAI"

# API de Manutenção (Obrigatório)
MAINTENANCE_SECRET="3977aa7046e9bf25ce7e91d535177b4c00794ec8fd29b98b5fc5a2697a455c1e"

# Sistema de Teste (Opcional)
TEST_MODE_EMAIL="teste@fcmtech.com.br"

# OAuth (Opcional)
GOOGLE_CLIENT_ID="..."
GOOGLE_CLIENT_SECRET="..."
LINKEDIN_CLIENT_ID="..."
LINKEDIN_CLIENT_SECRET="..."
```

🛠️ SCRIPTS ÚTEIS

Package.json Scripts

```
{
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  }
}
```

Comandos Prisma

```
# Sincronizar schema com banco (desenvolvimento)
yarn prisma db push

# Gerar Prisma Client
yarn prisma generate

# Abrir Prisma Studio (GUI)
yarn prisma studio

# Rodar seed
yarn tsx --require dotenv/config scripts/seed.ts
```

Scripts de Manutenção

```
# Verificar orphan jobs
yarn tsx scripts/check_orphan_jobs.ts

# Limpar dados órfãos
yarn tsx scripts/cleanup_orphans_db.ts

# Testar senha de usuário
yarn tsx scripts/test_password.ts
```

📝 BOAS PRÁTICAS IMPLEMENTADAS

Segurança

- ✅ Senhas hasheadas com bcrypt (salt rounds: 12)
- ✅ Validação de sessão em todas as APIs
- ✅ Verificação de role (RBAC)
- ✅ Proteção contra CSRF (NextAuth)
- ✅ SQL Injection protection (Prisma ORM)
- ✅ XSS protection (React)
- ✅ URLs assinadas para S3

Performance

- ✅ Server-side rendering (SSR)
- ✅ Static generation onde possível
- ✅ Image optimization (Next.js Image)
- ✅ Code splitting automático
- ✅ Cache de queries (SWR)
- ✅ Lazy loading de componentes
- ✅ **Polling otimizado** - Notificações: 2 minutos (antes 30s)
- ✅ **Emails assíncronos** - Não bloqueiam APIs
- ✅ **Timeouts configurados** - 15s para operações críticas

Código Limpo

- ✅ TypeScript para type safety
- ✅ Componentes reutilizáveis (Shadcn/ui)
- ✅ Separação de concerns
- ✅ Comentários em funções complexas
- ✅ Tratamento de erros consistente
- ✅ Logging estruturado

UX/UI

- ✅ Design responsivo (mobile-first)
- ✅ Loading states
- ✅ Error states
- ✅ Toast notifications
- ✅ Confirmações para ações destrutivas
- ✅ Validação de formulários em tempo real

🔍 DEBUGGING E LOGS

Logs do Sistema

- Console.log para desenvolvimento
- Logs estruturados em produção
- Erros capturados e registrados
- Prisma query logging (opcional)

Ferramentas de Debug

```
// Debug de sessão
console.log('Session:', await getServerSession(authOptions));

// Debug de banco
// Habilitar em schema.prisma:
generator client {
  provider = "prisma-client-js"
  log = ["query", "info", "warn", "error"]
}

// Debug de Stripe
// Usar Stripe CLI para testar webhooks localmente
stripe listen --forward-to localhost:3000/api/webhooks/stripe
```

📞 CONTATOS E SUPORTE

Equipe de Desenvolvimento

- **Desenvolvedor Principal:** DeepAgent (Abacus.AI)
- **Data de Criação:** Novembro 2025

Recursos

- **Documentação Next.js:** https://nextjs.org/docs
- **Prisma Docs:** https://www.prisma.io/docs
- **Stripe Docs:** https://stripe.com/docs
- **NextAuth Docs:** https://next-auth.js.org/

Fim da Documentação Técnica