Sistema AVD UISA - Especificações Técnicas Completas

Versão: 1.0
Data: 20 de Novembro de 2024
Autor: Equipe de Desenvolvimento AVD UISA

📋 Sumário Executivo

O Sistema AVD UISA é uma plataforma enterprise completa de Avaliação de Desempenho e Gestão de Talentos desenvolvida especificamente para a UISA. O sistema integra **2.889 funcionários reais**, **387 cargos**, **174 departamentos** e oferece funcionalidades avançadas de gestão de pessoas, performance, desenvolvimento e remuneração variável.

Números do Sistema

- **294 arquivos** de código TypeScript/React
- **62+ tabelas** no banco de dados MySQL/TiDB
- **120+ endpoints** backend tRPC
- **50+ páginas** frontend React
- **280 perguntas** de testes psicométricos (7 tipos)
- **481 descrições de cargos** UISA prontas para importação

🏗️ Arquitetura do Sistema

Stack Tecnológico

Frontend

- **Framework**: React 19 com TypeScript
- **Build Tool**: Vite
- **UI Library**: Shadcn/ui (componentes modernos)
- **Styling**: Tailwind CSS 4 (tema UISA customizado)
- **Gráficos**: Chart.js + Recharts
- **Formulários**: React Hook Form + Zod validation
- **Roteamento**: Wouter (leve e performático)
- **State Management**: TanStack Query (via tRPC)

Backend

- **Runtime**: Node.js 22.13.0
- **Framework**: Express 4
- **API**: tRPC 11 (type-safe end-to-end)
- **Validação**: Zod schemas
- **ORM**: Drizzle ORM
- **Autenticação**: JWT + Manus OAuth

Banco de Dados

- **SGBD**: MySQL 8 / TiDB Cloud
- **Tabelas**: 62+ tabelas relacionais
- **Índices**: 30+ índices otimizados
- **Migrations**: Drizzle Kit

Infraestrutura

- **Hospedagem**: Manus Cloud Platform

- **Storage**: AWS S3 (evidências, anexos)
- **E-mail**: Gmail SMTP (avd@uisa.com.br)
- **WebSocket**: Socket.IO (notificações em tempo real)
- **Cron Jobs**: Node-cron (3 jobs automáticos)

📊 Estrutura de Diretórios

```
avd-uisa-sistema-completo/
├── client/                     # Frontend React
│   ├── public/                 # Assets estáticos
│   └── src/
│       ├── pages/              # 50+ páginas do sistema
│       │   ├── Dashboard.tsx
│       │   ├── MetasSMART.tsx
│       │   ├── CriarMetaSMART.tsx
│       │   ├── BonusAprovacoes.tsx
│       │   ├── BonusReport.tsx
│       │   ├── BonusForecast.tsx
│       │   ├── BonusBatchApproval.tsx
│       │   ├── BonusAudit.tsx
│       │   ├── PDIInteligente.tsx
│       │   ├── Avaliacao360Enhanced.tsx
│       │   ├── NineBoxComparativo.tsx
│       │   ├── MapaSucessao.tsx
│       │   ├── TestDISC.tsx
│       │   ├── TestBigFive.tsx
│       │   ├── DescricaoCargosUISA.tsx
│       │   ├── PesquisasPulse.tsx
│       │   └── ... (40+ outras páginas)
│       ├── components/         # Componentes reutilizáveis
│       │   ├── ui/             # Shadcn/ui components
│       │   ├── DashboardLayout.tsx
│       │   ├── NotificationBell.tsx
│       │   ├── GlobalSearch.tsx
│       │   └── Breadcrumbs.tsx
│       ├── contexts/           # React contexts
│       │   └── ThemeContext.tsx
│       ├── hooks/              # Custom hooks
│       │   └── useAuth.tsx
│       ├── lib/                # Utilitários
│       │   ├── trpc.ts         # Cliente tRPC
│       │   └── currency.ts     # Formatação monetária
│       ├── App.tsx             # Rotas principais
│       ├── main.tsx            # Entry point
│       └── index.css           # Tema UISA global
│
├── server/                     # Backend tRPC
│   ├── _core/                  # Infraestrutura
│   │   ├── index.ts            # Servidor Express
```

```
│   │   ├── trpc.ts              # Configuração tRPC
│   │   ├── context.ts           # Context builder
│   │   ├── llm.ts               # Integração IA Gemini
│   │   └── websocket.ts         # Socket.IO
│   ├── routers/                 # 30+ routers tRPC
│   │   ├── bonusRouter.ts       # Sistema de bônus
│   │   ├── bonusWorkflowRouter.ts
│   │   ├── goalsRouter.ts       # Metas SMART
│   │   ├── pdiIntelligentRouter.ts
│   │   ├── evaluation360Router.ts
│   │   ├── nineBoxRouter.ts
│   │   ├── psychometricRouter.ts
│   │   ├── jobDescriptionsRouter.ts
│   │   ├── pulseRouter.ts
│   │   ├── productivityRouter.ts
│   │   ├── alertsRouter.ts
│   │   └── ... (20+ outros routers)
│   ├── utils/                   # Helpers e serviços
│   │   ├── emailService.ts      # Gmail SMTP
│   │   ├── notificationHelper.ts
│   │   ├── pdiRecommendations.ts
│   │   ├── badgeService.ts
│   │   └── exportUtils.ts
│   ├── db.ts                    # Database helpers
│   ├── routers.ts               # App router principal
│   └── cron.ts                  # Cron jobs
│
├── drizzle/                     # Schema e migrations
│   ├── schema.ts                # 62+ tabelas
│   └── migrations/              # SQL migrations
│
├── shared/                      # Tipos compartilhados
│   └── const.ts                 # Constantes
│
├── scripts/                     # Scripts utilitários
│   ├── seed-demo-data.mjs       # Popular dados de teste
│   ├── import-from-excel.ts     # Importar funcionários
│   └── generate-docs.sh         # Gerar documentação
│
└── docs/                        # Documentação
    ├── ESPECIFICACOES-TECNICAS-COMPLETAS.md
    ├── codigo-fonte-completo.txt
    └── TESTES_E2E.md
```

🗄️ Banco de Dados - Schema Completo

1. Gestão de Pessoas (10 tabelas)

employees

Tabela central de colaboradores com 2.889 funcionários UISA.

```
CREATE TABLE employees (
  id INT PRIMARY KEY AUTO_INCREMENT,
  userId INT,                    -- Vínculo com users (autenticação)
  employeeCode VARCHAR(50),      -- Chapa UISA
  name VARCHAR(255) NOT NULL,
  email VARCHAR(320),
  phone VARCHAR(20),
  hireDate DATE,                 -- Data de admissão
  departmentId INT,
  positionId INT,
  managerId INT,                 -- Gestor direto
  costCenter VARCHAR(100),       -- Centro de custos (CC-XXX-DEPTO)
  salary INT,                    -- Salário em centavos
  hierarchyLevel ENUM('diretoria', 'gerencia', 'coordenacao', 'supervisao',
'operacional'),
  status ENUM('ativo', 'inativo', 'ferias', 'afastado'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Índices: employeeCode, userId, departmentId, managerId, costCenter

departments

174 departamentos/seções da UISA.

```sql
CREATE TABLE departments (
  id INT PRIMARY KEY AUTO_INCREMENT,
  code VARCHAR(50) UNIQUE,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  parentId INT,                  -- Hierarquia de departamentos
  managerId INT,
  active BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

positions

387 cargos únicos da UISA.

```sql
CREATE TABLE positions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  code VARCHAR(50) UNIQUE,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  departmentId INT,
  level ENUM('JÚNIOR', 'PLENO', 'SÊNIOR', 'GERENTE', 'DIRETOR'),
  salaryMin INT,                 -- Faixa salarial mínima (centavos)
  salaryMax INT,                 -- Faixa salarial máxima (centavos)
  active BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

costCenters

Centros de custos organizacionais.

```sql
CREATE TABLE costCenters (
  id INT PRIMARY KEY AUTO_INCREMENT,
  code VARCHAR(50) UNIQUE NOT NULL,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  departmentId INT,
  budget DECIMAL(15,2),
  active BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

2. Sistema de Bônus (10 tabelas)

bonusPolicies

Políticas de bônus por cargo/função com multiplicadores de salário.

```sql
CREATE TABLE bonusPolicies (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  positionId INT,                -- Cargo específico (NULL = todos)
  departmentId INT,              -- Departamento específico
  salaryMultiplier DECIMAL(5,2), -- Ex: 1.5 = 1.5x salário
  additionalPercentage DECIMAL(5,2), -- % extra por metas
  eligibilityCriteria JSON,      -- {minPerformance, minTenure, minGoals}
  validFrom DATE,
  validUntil DATE,
  isActive BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

bonusCalculations

Cálculos de bônus individuais por colaborador/mês.

```sql
CREATE TABLE bonusCalculations (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  policyId INT NOT NULL,
  referenceMonth DATE NOT NULL,  -- Mês de referência (YYYY-MM-01)
  baseSalary INT NOT NULL,       -- Salário base (centavos)
  multiplier DECIMAL(5,2),       -- Multiplicador aplicado
  additionalPercentage DECIMAL(5,2),
  goalsCompletionRate DECIMAL(5,2), -- Taxa de conclusão de metas (0-100)
  totalAmount INT NOT NULL,      -- Valor total (centavos)
  status ENUM('calculado', 'aprovado', 'pago', 'rejeitado', 'cancelado'),
  calculatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  approvedAt TIMESTAMP NULL,
  paidAt TIMESTAMP NULL
);
```

bonusApprovalWorkflows

Configuração de workflows de aprovação multinível.

```sql
CREATE TABLE bonusApprovalWorkflows (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  minValue DECIMAL(15,2) DEFAULT 0,  -- Valor mínimo para aplicar
  maxValue DECIMAL(15,2),            -- Valor máximo (NULL = sem limite)
  departmentId INT,                  -- NULL = todos os departamentos
  isActive BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

bonusApprovalLevels

Níveis do workflow (gestor → gerente → diretor → CFO).

```sql
CREATE TABLE bonusApprovalLevels (
  id INT PRIMARY KEY AUTO_INCREMENT,
  workflowId INT NOT NULL,
  levelOrder INT NOT NULL,       -- Ordem do nível (1, 2, 3...)
  approverRole VARCHAR(100) NOT NULL, -- gestor_direto, gerente, diretor,
cfo
  requiresComment BOOLEAN DEFAULT FALSE,
  requiresEvidence BOOLEAN DEFAULT FALSE,
  timeoutDays INT DEFAULT 3,     -- Prazo em dias
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

bonusWorkflowInstances

Execução real de workflows de aprovação.

```sql
CREATE TABLE bonusWorkflowInstances (
  id INT PRIMARY KEY AUTO_INCREMENT,
  bonusCalculationId INT NOT NULL,
  workflowId INT NOT NULL,
  currentLevel INT DEFAULT 1,
  status ENUM('em_andamento', 'aprovado', 'rejeitado', 'cancelado'),
  startedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  completedAt TIMESTAMP NULL
);
```

bonusLevelApprovals

Aprovações individuais por nível.

```sql
CREATE TABLE bonusLevelApprovals (
  id INT PRIMARY KEY AUTO_INCREMENT,
  workflowInstanceId INT NOT NULL,
  levelId INT NOT NULL,
  approverId INT NOT NULL,
  status ENUM('pendente', 'aprovado', 'rejeitado'),
  comments TEXT,
  evidenceUrl VARCHAR(500),      -- URL de evidência (S3)
  decidedAt TIMESTAMP NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

bonusAuditLogs

Histórico completo de auditoria de bônus.

```sql
CREATE TABLE bonusAuditLogs (
  id INT PRIMARY KEY AUTO_INCREMENT,
  bonusCalculationId INT,
  policyId INT,
  userId INT NOT NULL,
  action VARCHAR(100) NOT NULL,  -- created, approved, rejected, paid, etc
  oldValue JSON,
  newValue JSON,
  ipAddress VARCHAR(45),
  userAgent TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

bonusApprovalComments

Sistema de comentários em aprovações.

```
CREATE TABLE bonusApprovalComments (
  id INT PRIMARY KEY AUTO_INCREMENT,
  bonusCalculationId INT NOT NULL,
  userId INT NOT NULL,
  comment TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

3. Metas SMART (5 tabelas)

smartGoals

Metas SMART com validação automática dos 5 critérios.

```sql
CREATE TABLE smartGoals (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  cycleId INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  type ENUM('individual', 'equipe', 'organizacional'),
  category ENUM('financeira', 'comportamental', 'corporativa',
'desenvolvimento'),

  -- Critérios SMART
  specificScore INT,              -- 0-100
  measurableScore INT,
  achievableScore INT,
  relevantScore INT,
  timeBoundScore INT,
  overallSmartScore INT,          -- Média dos 5 critérios

  -- Métricas
  measurementUnit VARCHAR(100),   -- unidade, %, R$, horas, etc
  targetValue DECIMAL(15,2),
  currentValue DECIMAL(15,2) DEFAULT 0,
  progress INT DEFAULT 0,         -- 0-100%

  -- Datas
  startDate DATE,
  dueDate DATE,
  completedAt TIMESTAMP NULL,

  -- Bônus
  isEligibleForBonus BOOLEAN DEFAULT FALSE,
  bonusPercentage DECIMAL(5,2),   -- % do salário
  bonusFixedAmount INT,           -- Valor fixo (centavos)

  -- Status e workflow
  status ENUM('rascunho', 'pendente_aprovacao', 'ativa', 'concluida',
'cancelada'),
  approvalStatus ENUM('pendente', 'aprovado', 'rejeitado'),

  -- Vinculações
  parentGoalId INT,               -- Meta pai (cascata hierárquico)
  pdiPlanId INT,                  -- Vinculação com PDI
  departmentId INT,
  alignmentPercentage INT,        -- % de alinhamento com meta pai
```

```
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

goalMilestones

Marcos intermediários de metas.

```
CREATE TABLE goalMilestones (
  id INT PRIMARY KEY AUTO_INCREMENT,
  goalId INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  dueDate DATE,
  status ENUM('pendente', 'em_andamento', 'concluido', 'atrasado'),
  progress INT DEFAULT 0,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  completedAt TIMESTAMP NULL
);
```

goalApprovals

Workflow de aprovação Gestor → RH.

```
CREATE TABLE goalApprovals (
  id INT PRIMARY KEY AUTO_INCREMENT,
  goalId INT NOT NULL,
  approverId INT NOT NULL,
  approverRole ENUM('manager', 'hr', 'director'),
  status ENUM('pending', 'approved', 'rejected'),
  comments TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  approvedAt TIMESTAMP NULL
);
```

goalComments

Sistema de comentários e acompanhamento.

```
CREATE TABLE goalComments (
  id INT PRIMARY KEY AUTO_INCREMENT,
  goalId INT NOT NULL,
  userId INT NOT NULL,
  comment TEXT NOT NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

goalEvidences

Evidências de progresso (uploads S3).

```
CREATE TABLE goalEvidences (
  id INT PRIMARY KEY AUTO_INCREMENT,
  goalId INT NOT NULL,
  userId INT NOT NULL,
  title VARCHAR(255),
  description TEXT,
  fileUrl VARCHAR(500) NOT NULL, -- URL S3
  fileType VARCHAR(100),
  fileSize INT,
  uploadedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

4. PDI Inteligente (7 tabelas)

pdiPlans

Planos de Desenvolvimento Individual.

```sql
CREATE TABLE pdiPlans (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  targetPositionId INT,          -- Cargo-alvo
  startDate DATE,
  endDate DATE,
  overallProgress INT DEFAULT 0,
  status ENUM('rascunho', 'em_andamento', 'concluido', 'cancelado'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiIntelligentDetails

Detalhes estratégicos do PDI (modelo Nadia).

```sql
CREATE TABLE pdiIntelligentDetails (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL UNIQUE,
  strategicContext JSON,         -- Contexto estratégico
  sponsors JSON,                 -- Sponsors, mentores, guardiões
  currentProfile JSON,           -- Perfil atual (DISC, Big Five)
  targetProfile JSON,            -- Perfil desejado
  keyAreas JSON,                 -- Áreas-chave de desenvolvimento
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiCompetencyGaps

Gaps de competências identificados.

```sql
CREATE TABLE pdiCompetencyGaps (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL,
  competencyName VARCHAR(255) NOT NULL,
  currentLevel INT,              -- 1-5
  targetLevel INT,               -- 1-5
  gap INT,                       -- Diferença
  priority ENUM('alta', 'media', 'baixa'),
  responsibilities TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiActions

Plano de ação 70-20-10.

```sql
CREATE TABLE pdiActions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL,
  type ENUM('70_experiencias', '20_relacionamentos', '10_cursos'),
  title VARCHAR(255) NOT NULL,
  description TEXT,
  metric VARCHAR(255),           -- Métrica de sucesso
  responsible VARCHAR(255),      -- Responsável
  dueDate DATE,
  status ENUM('nao_iniciado', 'em_andamento', 'concluido'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiGovernanceReviews

Feedbacks DGC (Diretoria de Gente e Cultura).

```
CREATE TABLE pdiGovernanceReviews (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL,
  reviewerId INT NOT NULL,
  reviewDate DATE NOT NULL,
  ipsScore INT,                  -- Índice de Prontidão para Sucessão (1-5)
  feedback TEXT,
  nextSteps TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiRisks

Gestão de riscos do PDI.

```
CREATE TABLE pdiRisks (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL,
  type VARCHAR(100) NOT NULL,
  description TEXT NOT NULL,
  impact ENUM('baixo', 'medio', 'alto', 'critico'),
  probability ENUM('baixa', 'media', 'alta'),
  mitigation TEXT,
  responsible VARCHAR(255),
  status ENUM('identificado', 'em_tratamento', 'mitigado', 'ocorrido'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pdiReviews

Acompanhamento RH/Gestor/Sponsors.

```
CREATE TABLE pdiReviews (
  id INT PRIMARY KEY AUTO_INCREMENT,
  pdiPlanId INT NOT NULL,
  reviewerId INT NOT NULL,
  reviewerRole VARCHAR(100),
  reviewDate DATE NOT NULL,
  overallProgress INT,
  strengths TEXT,
  improvements TEXT,
  recommendation TEXT,
  nextSteps TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

5. Avaliação 360° (4 tabelas)

performanceEvaluations

Avaliações de desempenho 360°.

```sql
CREATE TABLE performanceEvaluations (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  cycleId INT NOT NULL,
  evaluatorId INT,
  evaluatorType ENUM('self', 'manager', 'peer', 'subordinate', 'consensus'),
  finalScore DECIMAL(5,2),
  responses JSON,                -- Respostas das perguntas
  comments TEXT,
  managerComments TEXT,
  status ENUM('pendente', 'em_andamento', 'concluida'),

  -- Workflow sequencial
  workflowStatus ENUM('pending_self', 'pending_manager',
'pending_consensus', 'completed'),
  selfCompletedAt TIMESTAMP NULL,
  managerCompletedAt TIMESTAMP NULL,
  consensusCompletedAt TIMESTAMP NULL,

  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  completedAt TIMESTAMP NULL
);
```

evaluation360Questions

Perguntas padrão da avaliação 360° (23 perguntas em 6 categorias).

```sql
CREATE TABLE evaluation360Questions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  category VARCHAR(100) NOT NULL,
  questionText TEXT NOT NULL,
  questionType ENUM('escala', 'texto'),
  orderIndex INT,
  isActive BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

evaluation360Responses

Respostas individuais por pergunta.

```
CREATE TABLE evaluation360Responses (
  id INT PRIMARY KEY AUTO_INCREMENT,
  evaluationId INT NOT NULL,
  questionId INT NOT NULL,
  rating INT,                    -- 1-5 para escala
  textResponse TEXT,             -- Para perguntas abertas
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

calibrationSessions

Sessões de calibração de avaliações.

```
CREATE TABLE calibrationSessions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  cycleId INT NOT NULL,
  departmentId INT,
  facilitatorId INT NOT NULL,
  status ENUM('agendada', 'em_andamento', 'concluida'),
  scheduledDate TIMESTAMP,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

6. Nine Box e Sucessão (8 tabelas)

nineBoxAssessments

Posicionamento na Matriz 9-Box (2.893 colaboradores posicionados).

```sql
CREATE TABLE nineBoxAssessments (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  cycleId INT NOT NULL,
  performance INT NOT NULL,      -- 1-3 (Baixo, Médio, Alto)
  potential INT NOT NULL,        -- 1-3 (Baixo, Médio, Alto)
  position VARCHAR(50),          -- Ex: "Alto Desempenho, Alto Potencial"
  assessorId INT,
  assessmentDate DATE,
  comments TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

9 Categorias da Matriz:

1. Baixo Desempenho, Baixo Potencial (1,1)
2. Baixo Desempenho, Médio Potencial (1,2)
3. Baixo Desempenho, Alto Potencial (1,3)
4. Médio Desempenho, Baixo Potencial (2,1)
5. Médio Desempenho, Médio Potencial (2,2) - “Sólidos”
6. Médio Desempenho, Alto Potencial (2,3)
7. Alto Desempenho, Baixo Potencial (3,1)
8. Alto Desempenho, Médio Potencial (3,2)
9. Alto Desempenho, Alto Potencial (3,3) - “Estrelas”

successionPlans

Planos de sucessão para posições críticas (15 planos UISA).

```sql
CREATE TABLE successionPlans (
  id INT PRIMARY KEY AUTO_INCREMENT,
  positionId INT NOT NULL,
  currentHolderId INT,
  riskLevel ENUM('baixo', 'medio', 'alto', 'critico'),
  impact ENUM('baixo', 'medio', 'alto', 'critico'),
  exitRisk INT,                  -- 0-100%
  competencyGap DECIMAL(5,2),
  preparationTime INT,           -- Meses
  trackingPlan TEXT,
  nextReviewDate DATE,
  status ENUM('ativo', 'inativo'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

successionCandidates

Candidatos a sucessão com integração PDI.

```sql
CREATE TABLE successionCandidates (
  id INT PRIMARY KEY AUTO_INCREMENT,
  successionPlanId INT NOT NULL,
  employeeId INT NOT NULL,
  readinessLevel ENUM('pronto_agora', '1_ano', '2_3_anos', 'mais_3_anos'),
  priority INT,                  -- 1=Principal, 2=Secundário, 3=Backup
  notes TEXT,

  -- Integração PDI
  pdiPlanId INT,
  pdiProgress INT,               -- 0-100%
  pdiCompletedActions INT,
  pdiTotalActions INT,
  readinessScore INT,            -- 0-100 (score unificado)
  competencyGapScore DECIMAL(5,2),
  lastScoreUpdate TIMESTAMP,

  -- Avaliações
  performanceRating ENUM('baixo', 'medio', 'alto', 'excepcional'),
  potentialRating ENUM('baixo', 'medio', 'alto', 'excepcional'),
  nineBoxPosition VARCHAR(100),
  gapAnalysis TEXT,
  developmentActions TEXT,

  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

calibrationMovements

Movimentações na Matriz 9-Box (calibração de diretoria).

```
CREATE TABLE calibrationMovements (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  cycleId INT NOT NULL,
  fromPerformance INT,
  fromPotential INT,
  toPerformance INT NOT NULL,
  toPotential INT NOT NULL,
  justification TEXT NOT NULL,
  requestedById INT NOT NULL,
  requestedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  status ENUM('pendente', 'aprovado', 'rejeitado')
);
```

calibrationApprovals

Workflow de aprovação RH → Diretor Gente → Diretor Área.

```
CREATE TABLE calibrationApprovals (
  id INT PRIMARY KEY AUTO_INCREMENT,
  movementId INT NOT NULL,
  approverRole ENUM('rh', 'diretor_gente', 'diretor_area'),
  approverId INT NOT NULL,
  status ENUM('pendente', 'aprovado', 'rejeitado'),
  comments TEXT,
  evidenceUrl VARCHAR(500),
  decidedAt TIMESTAMP NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

7. Testes Psicométricos (3 tabelas)

psychometricTests

Resultados de testes psicométricos (280 perguntas, 7 tipos).

```sql
CREATE TABLE psychometricTests (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  testType ENUM('disc', 'bigfive', 'mbti', 'ie', 'vark', 'leadership',
'career_anchors'),
  responses JSON,                -- Respostas completas
  results JSON,                  -- Resultados calculados
  discProfile VARCHAR(50),       -- D, I, S, C ou combinações
  bigFiveScores JSON,            -- {O, C, E, A, N}
  mbtiType VARCHAR(4),           -- Ex: INTJ
  completedAt TIMESTAMP NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

7 Tipos de Testes:

1. **DISC** (40 perguntas): Dominância, Influência, Estabilidade, Conformidade
2. **Big Five** (50 perguntas): Abertura, Conscienciosidade, Extroversão, Amabilidade, Neuroticismo
3. **MBTI** (40 perguntas): 16 tipos de personalidade
4. **Inteligência Emocional** (40 perguntas): Modelo Goleman (5 dimensões)
5. **VARK** (40 perguntas): Estilos de aprendizagem (Visual, Auditivo, Leitura, Cinestésico)
6. **Estilos de Liderança** (30 perguntas): Lewin, Bass, Goleman
7. **Âncoras de Carreira** (40 perguntas): Edgar Schein (8 âncoras)

testQuestions

Banco de 280 perguntas.

```
CREATE TABLE testQuestions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  testType ENUM('disc', 'bigfive', 'mbti', 'ie', 'vark', 'leadership',
'career_anchors'),
  dimension VARCHAR(100),        -- Ex: 'D' para DISC, 'Openness' para Big
Five
  questionText TEXT NOT NULL,
  isReversed BOOLEAN DEFAULT FALSE,
  orderIndex INT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

testInvitations

Convites de testes enviados por e-mail.

```
CREATE TABLE testInvitations (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  testType VARCHAR(50) NOT NULL,
  token VARCHAR(255) UNIQUE NOT NULL,
  sentAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  completedAt TIMESTAMP NULL,
  expiresAt TIMESTAMP
);
```

---

8. Descrição de Cargos UISA (7 tabelas)

jobDescriptions

Descrições de cargos completas (template UISA com 8 seções).

```sql
CREATE TABLE jobDescriptions (
  id INT PRIMARY KEY AUTO_INCREMENT,
  positionId INT NOT NULL,
  occupantId INT,                -- Colaborador ocupante

  -- Seção 1: Informações Básicas
  cboCode VARCHAR(50),           -- Código CBO
  division VARCHAR(255),
  reportsTo VARCHAR(255),
  revisionDate DATE,

  -- Seção 2: Objetivo Principal
  mainObjective TEXT,

  -- Seção 7: Qualificação Desejada
  desiredEducation TEXT,
  desiredExperience TEXT,

  -- Seção 8: e-Social
  pcmso TEXT,
  ppra TEXT,

  -- Status e workflow
  status ENUM('rascunho', 'pendente_aprovacao', 'aprovado', 'em_revisao'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

jobResponsibilities

Responsabilidades por categoria (Seção 3).

```sql
CREATE TABLE jobResponsibilities (
  id INT PRIMARY KEY AUTO_INCREMENT,
  jobDescriptionId INT NOT NULL,
  category ENUM('processo', 'analise_kpi', 'planejamento', 'budget',
'resultados', 'outros'),
  description TEXT NOT NULL,
  orderIndex INT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

jobKnowledge

Conhecimentos técnicos com 4 níveis (Seção 4).

```
CREATE TABLE jobKnowledge (
  id INT PRIMARY KEY AUTO_INCREMENT,
  jobDescriptionId INT NOT NULL,
  knowledgeArea VARCHAR(255) NOT NULL,
  level ENUM('basico', 'intermediario', 'avancado', 'obrigatorio'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

jobCompetencies

Competências e habilidades (Seção 6).

```
CREATE TABLE jobCompetencies (
  id INT PRIMARY KEY AUTO_INCREMENT,
  jobDescriptionId INT NOT NULL,
  competencyName VARCHAR(255) NOT NULL,
  description TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

jobDescriptionApprovals

Workflow de aprovação: Ocupante → Superior → RH.

```
CREATE TABLE jobDescriptionApprovals (
  id INT PRIMARY KEY AUTO_INCREMENT,
  jobDescriptionId INT NOT NULL,
  approverRole ENUM('ocupante', 'superior_imediato', 'gerente_rh'),
  approverId INT NOT NULL,
  status ENUM('pendente', 'aprovado', 'rejeitado'),
  comments TEXT,
  decidedAt TIMESTAMP NULL,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

employeeActivities

Registro manual de atividades (para validação da descrição).

```
CREATE TABLE employeeActivities (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  category ENUM('reuniao', 'analise', 'planejamento', 'execucao', 'suporte',
'outros'),
  activityDate DATE NOT NULL,
  startTime TIME,
  endTime TIME,
  durationMinutes INT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

activityLogs

Coleta automática de atividades (futuro - tracking de sistema).

```
CREATE TABLE activityLogs (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  activityType VARCHAR(100),
  application VARCHAR(255),
  durationMinutes INT,
  loggedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

9. Pesquisas de Pulse (3 tabelas)

pulseSurveys

Pesquisas de clima/engajamento/satisfação.

```sql
CREATE TABLE pulseSurveys (
  id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  question TEXT NOT NULL,
  targetAudience ENUM('todos', 'departamento', 'especifico'),
  targetDepartmentId INT,
  targetEmployeeIds JSON,
  startDate DATE,
  endDate DATE,
  status ENUM('rascunho', 'ativa', 'encerrada'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pulseSurveyResponses

Respostas das pesquisas (escala 0-10 + comentário).

```sql
CREATE TABLE pulseSurveyResponses (
  id INT PRIMARY KEY AUTO_INCREMENT,
  surveyId INT NOT NULL,
  employeeId INT,                -- NULL = anônimo
  rating INT NOT NULL,           -- 0-10
  comment TEXT,
  submittedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

pulseSurveyEmailLogs

Histórico de envio de e-mails (cron job a cada 8h).

```
CREATE TABLE pulseSurveyEmailLogs (
  id INT PRIMARY KEY AUTO_INCREMENT,
  surveyId INT NOT NULL,
  employeeId INT NOT NULL,
  emailSent BOOLEAN DEFAULT FALSE,
  sentAt TIMESTAMP NULL,
  failedAt TIMESTAMP NULL,
  errorMessage TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

---

10. Produtividade e Alertas (4 tabelas)

timeClockRecords

Registros de ponto eletrônico importados.

```
CREATE TABLE timeClockRecords (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  recordDate DATE NOT NULL,
  checkIn TIME,
  checkOut TIME,
  lunchStart TIME,
  lunchEnd TIME,
  totalMinutes INT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

timeDiscrepancies

Discrepâncias entre ponto e atividades registradas.

```sql
CREATE TABLE timeDiscrepancies (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT NOT NULL,
  discrepancyDate DATE NOT NULL,
  clockMinutes INT,              -- Minutos de ponto
  activityMinutes INT,           -- Minutos de atividades
  differenceMinutes INT,         -- Diferença
  differencePercentage DECIMAL(5,2),
  classification ENUM('aceitavel', 'over_reported', 'under_reported'),
  justification TEXT,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

alerts

Sistema de alertas automáticos.

```sql
CREATE TABLE alerts (
  id INT PRIMARY KEY AUTO_INCREMENT,
  employeeId INT,
  type VARCHAR(100) NOT NULL,    -- baixa_produtividade,
horas_inconsistentes, etc
  severity ENUM('baixo', 'medio', 'alto', 'critico'),
  message TEXT NOT NULL,
  data JSON,
  status ENUM('pendente', 'resolvido', 'dispensado'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  resolvedAt TIMESTAMP NULL
);
```

---

11. Sistema de Notificações (1 tabela)

notifications

Notificações in-app com WebSocket.

```
CREATE TABLE notifications (
  id INT PRIMARY KEY AUTO_INCREMENT,
  userId INT NOT NULL,
  type VARCHAR(100) NOT NULL,
  title VARCHAR(255) NOT NULL,
  message TEXT NOT NULL,
  link VARCHAR(500),
  isRead BOOLEAN DEFAULT FALSE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Tipos de Notificações:

- bonus_calculated, bonus_approved, bonus_rejected, bonus_paid
- goal_approval, goal_rejected, goal_deadline
- evaluation_pending, evaluation_completed
- pdi_approved, pdi_feedback
- calibration_pending
- test_invitation

---

12. Configurações do Sistema (3 tabelas)

systemSettings

Configurações globais (SMTP, integrações).

```
CREATE TABLE systemSettings (
  id INT PRIMARY KEY AUTO_INCREMENT,
  settingKey VARCHAR(255) UNIQUE NOT NULL,
  settingValue TEXT,
  description TEXT,
  updatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

Configurações SMTP:

- smtp_host: smtp.gmail.com
- smtp_port: 587
- smtp_user: avd@uisa.com.br
- smtp_password: (senha de aplicativo)
- smtp_from_name: Sistema AVD UISA
- smtp_from_email: avd@uisa.com.br

evaluationCycles

Ciclos de avaliação (anual, semestral, trimestral).

```
CREATE TABLE evaluationCycles (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  year INT NOT NULL,
  type ENUM('anual', 'semestral', 'trimestral', 'mensal'),
  startDate DATE NOT NULL,
  endDate DATE NOT NULL,
  description TEXT,
  status ENUM('rascunho', 'em_andamento', 'concluido'),
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

workflows

Workflows genéricos configuráveis.

```
CREATE TABLE workflows (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  type ENUM('bonus', 'ferias', 'promocao', 'transferencia', 'desligamento',
'outros'),
  steps JSON,                    -- [{order, role, requiresApproval}]
  isActive BOOLEAN DEFAULT TRUE,
  createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

🔌 Endpoints Backend - Documentação Completa

bonusRouter (20 endpoints)

```
// CRUD Básico
list(input?: { departmentId?, isActive? })
getById(id: number)
create(input: { name, description, positionId, salaryMultiplier, ... })
update(input: { id, name?, description?, ... })
delete(id: number)

// Cálculo e Gestão
calculateBonus(input: { employeeId, policyId, referenceMonth })
listCalculations(input?: { status?, employeeId?, referenceMonth? })
approveCalculation(input: { id, comments? })
rejectCalculation(input: { id, comments })
markAsPaid(input: { id, paidDate })

// Estatísticas e Relatórios
getStats()  // Total, média, beneficiados, políticas ativas
getMonthlyTrends(input: { months: 3|6|12, departmentId? })
getDepartmentDistribution(input?: { referenceMonth? })

// Aprovação em Lote
approveBatch(input: { calculationIds: number[], comments? })
rejectBatch(input: { calculationIds: number[], comments })

// Comentários
addComment(input: { bonusCalculationId, comment })
getComments(bonusCalculationId: number)

// Auditoria
getAuditLogs(input?: { bonusCalculationId?, userId?, action? })
getApprovalMetrics()  // Aprovações, rejeições, taxa, tempo médio
```

bonusWorkflowRouter (10 endpoints)

```
// CRUD de Workflows
list(input?: { departmentId?, isActive? })
getById(id: number)
create(input: { name, description, minValue, maxValue, levels: [...] })
update(input: { id, name?, description?, isActive? })
delete(id: number)

// Execução de Workflows
startWorkflow(input: { bonusCalculationId })
approveLevel(input: { approvalId, comments?, evidenceUrl? })
rejectLevel(input: { approvalId, comments })

// Consultas
getPendingInstances()  // Instâncias pendentes do usuário logado
getWorkflowHistory(bonusCalculationId: number)
```

goalsRouter (smartGoals) (16 endpoints)

```
// CRUD
list(input?: { cycleId?, status?, category?, employeeId? })
getById(id: number)
create(input: { employeeId, cycleId, title, description, ... })
update(input: { id, title?, description?, targetValue?, ... })
delete(id: number)

// Validação SMART
validateSMART(input: { title, description, measurementUnit, targetValue,
startDate, dueDate })

// Atualização de Progresso
updateProgress(input: { id, currentValue, comments? })

// Workflow de Aprovação
submitForApproval(id: number)
approve(input: { id, comments? })
reject(input: { id, comments })

// Marcos
addMilestone(input: { goalId, title, description, dueDate })
updateMilestone(input: { id, status, progress })

// Vinculações
linkToPDI(input: { goalId, pdiPlanId })

// Comentários e Dashboard
addComment(input: { goalId, comment })
getDashboard(input?: { cycleId?, employeeId? })

// Bônus
calculateBonus(employeeId: number, referenceMonth: string)
```

pdiIntelligentRouter (15 endpoints)

```
// CRUD
list(input?: { employeeId?, status? })
getById(id: number)
create(input: { employeeId, targetPositionId, startDate, endDate, ... })
update(input: { id, overallProgress?, status? })
delete(id: number)

// Gaps de Competências
addGap(input: { pdiPlanId, competencyName, currentLevel, targetLevel,
priority })
updateGap(input: { id, currentLevel?, targetLevel?, priority? })
listGaps(pdiPlanId: number)

// Riscos
addRisk(input: { pdiPlanId, type, description, impact, probability,
mitigation })
updateRisk(input: { id, status?, mitigation? })
listRisks(pdiPlanId: number)

// Reviews
addReview(input: { pdiPlanId, overallProgress, strengths, improvements,
nextSteps })
listReviews(pdiPlanId: number)

// Análise
compareProfiles(input: { employeeId, targetPositionId })

// Ações 70-20-10
addAction(input: { pdiPlanId, type, title, description, metric, responsible,
dueDate })
updateActionStatus(input: { id, status })
getActions(pdiPlanId: number)

// Governança DGC
addGovernanceReview(input: { pdiPlanId, ipsScore, feedback, nextSteps })
getGovernanceReviews(pdiPlanId: number)
getIPSEvolution(pdiPlanId: number)
```

evaluation360Router (10 endpoints)

```
// CRUD
list(input?: { cycleId?, status? })
getById(id: number)
create(input: { employeeId, cycleId })
delete(id: number)

// Perguntas
getQuestions()  // 23 perguntas padrão em 6 categorias

// Workflow Sequencial
submitSelfAssessment(input: { evaluationId, responses: [...] })
submitManagerAssessment(input: { evaluationId, responses: [...], comments?
})
submitConsensus(input: { evaluationId, finalScore, comments })

// Feedback e Detalhes
submitFeedback(input: { evaluationId, comments })
getDetails(id: number)  // Com averages e responses agrupadas
getEvaluationWithWorkflow(id: number)
```

nineBoxRouter (10 endpoints)

```
// CRUD
list(input?: { cycleId?, departmentId? })
getById(id: number)
create(input: { employeeId, cycleId, performance, potential, comments })
update(input: { id, performance?, potential?, comments? })
delete(id: number)

// Ajustes e Calibração
adjust(input: { id, performance, potential, justification })

// Comparações
getComparison(input: { positionId?, departmentId? })

// Hierarquia
getLeaders(input?: { minSubordinates? })
getSubordinates(input: { leaderId, hierarchyLevel? })

// Estatísticas
getDistribution(input?: { cycleId?, departmentId? })
```

psychometricRouter (10 endpoints)

```
// Testes
getQuestions(testType: string)
submitTest(input: { employeeId, testType, responses: [...] })
getTests(input?: { employeeId?, testType? })

// Convites
sendTestInvitation(input: { employeeId, testType })

// Resultados RH
getAllTests(input?: { employeeId?, testType?, departmentId? })

// Integração PDI
getPDIRecommendations(input: { employeeId, testTypes: [...] })
getAggregatedResults(input?: { departmentId?, testType? })

// Públicos (sem autenticação)
getQuestionsPublic(testType: string)
submitTestPublic(input: { email, testType, responses: [...] })
```

jobDescriptionsRouter (12 endpoints)

```
// CRUD
list(input?: { departmentId?, status? })
getById(id: number)
create(input: { positionId, occupantId, mainObjective, ... })
update(input: { id, mainObjective?, desiredEducation?, ... })
delete(id: number)

// Workflow de Aprovação
submitForApproval(id: number)
approve(input: { id, comments? })
reject(input: { id, comments })
getApprovalHistory(id: number)

// Atividades
addActivity(input: { employeeId, title, description, category, activityDate,
... })
getActivities(input: { employeeId, startDate?, endDate? })

// Importação em Massa
importFromDocx(input: { files: File[] })
```

pulseRouter (9 endpoints)

```
// CRUD
list(input?: { status? })
getById(id: number)
create(input: { title, description, question, targetAudience, ... })
update(input: { id, title?, description?, ... })
delete(id: number)

// Ativação e Encerramento
activate(id: number)
close(id: number)

// Respostas
submitResponse(input: { surveyId, rating, comment? })
getResults(id: number)

// Envio de Convites
sendInvitations(id: number)
```

productivityRouter (8 endpoints)

```
// Métricas
getMetrics(input: { employeeId?, departmentId?, startDate?, endDate? })
getTopPerformers(input?: { departmentId?, limit? })

// Atividades
getActivityBreakdown(input: { employeeId, startDate, endDate })
getComparison(input: { employeeIds: number[], startDate, endDate })

// Alertas
getAlerts(input?: { employeeId?, severity? })

// Relatórios
getProductivityReport(input: { departmentId?, startDate, endDate })
exportProductivityData(input: { format: 'excel'|'csv', ... })
```

alertsRouter (7 endpoints)

```
// CRUD
list(input?: { employeeId?, type?, severity?, status? })
getById(id: number)
create(input: { employeeId, type, severity, message, data })

// Ações
resolve(input: { id, resolution })
dismiss(id: number)

// Estatísticas
getStats(input?: { departmentId?, startDate?, endDate? })
```

timeClockRouter (8 endpoints)

```
// Importação
importRecords(input: { records: [...] })  // CSV/API
importFromCSV(file: File)

// Consultas
getRecords(input: { employeeId, startDate, endDate })

// Discrepâncias
calculateDiscrepancies(input: { employeeId?, date? })
getDiscrepancyStats(input?: { departmentId?, startDate?, endDate? })

// Justificativas
addJustification(input: { discrepancyId, justification })
approveJustification(id: number)
rejectJustification(input: { id, comments })
```

executiveRouter (9 endpoints)

```
// KPIs Estratégicos
getKPIs(input?: { costCenter? })
getHeadcountByDepartment(input?: { costCenter? })
getHeadcountTrend(input: { months: 6|12, costCenter? })

// Finanças
getSalaryDistribution(input?: { costCenter? })
getTurnoverRate(input: { months: 6|12, costCenter? })

// Sucessão e Talentos
getSuccessionPipeline(input?: { costCenter? })
getTrainingROI(input?: { costCenter? })

// Performance
getPerformanceDistribution(input?: { costCenter? })
getEngagementMetrics(input?: { costCenter? })
```

⚙️ Regras de Negócio Detalhadas

1. Cálculo de Bônus

Fórmula Base

```
Valor Bônus = (Salário Base × Multiplicador) + (Salário Base × % Adicional × Taxa de Metas)
```

Critérios de Elegibilidade

1. **Performance Mínima**: Score $\geqslant {}^{70}/_{100}$ na avaliação 360°
2. **Tempo de Casa**: Mínimo 6 meses (configurável)
3. **Metas Concluídas**: Mínimo 60% de progresso nas metas ativas
4. **Status**: Colaborador ativo (não em aviso prévio)

Workflow de Aprovação Multinível

Regras de Roteamento por Valor:

- **R**$0 - R$ **5.000**: Nível 1 (Gestor Direto)
- **R**$5.001 - R$ **10.000**: Níveis 1-2 (Gestor + Gerente)
- **R**$10.001 - R$ **20.000**: Níveis 1-3 (Gestor + Gerente + Diretor)
- **R**$20.001 - R$ **50.000**: Níveis 1-4 (+ Diretor de Gente)
- **Acima de R$ 50.000**: Níveis 1-5 (+ CFO)

Prazos de Aprovação:

- Cada nível tem 3 dias úteis para aprovar/rejeitar
- Após timeout, escala automaticamente para o próximo nível
- Notificações automáticas em D-1 (1 dia antes do prazo)

Evidências Obrigatórias:

- Nível 3 (Diretor): Justificativa obrigatória
- Nível 4 (Diretor de Gente): Evidências obrigatórias (upload S3)
- Nível 5 (CFO): Análise de impacto financeiro

---

2. Validação SMART de Metas

Critério S (Specific) - Score 0-100

```
function validateSpecific(title: string, description: string): number {
  let score = 0;

  // Verbo de ação no título (20 pontos)
  const actionVerbs = ['aumentar', 'reduzir', 'implementar', 'desenvolver',
'melhorar', ...];
  if (actionVerbs.some(verb => title.toLowerCase().includes(verb))) {
    score += 20;
  }

  // Título com mais de 10 caracteres (20 pontos)
  if (title.length > 10) score += 20;

  // Descrição com mais de 50 caracteres (30 pontos)
  if (description.length > 50) score += 30;

  // Descrição com contexto (30 pontos)
  if (description.length > 150) score += 30;

  return Math.min(score, 100);
}
```

Critério M (Measurable) - Score 0-100

```
function validateMeasurable(measurementUnit: string, targetValue: number):
number {
  let score = 0;

  // Unidade de medida definida (50 pontos)
  if (measurementUnit && measurementUnit.length > 0) {
    score += 50;
  }

  // Valor alvo numérico (50 pontos)
  if (targetValue && targetValue > 0) {
    score += 50;
  }

  return score;
}
```

Critério A (Achievable) - Score 0-100

```
function validateAchievable(targetValue: number, historicalAverage: number):
number {
  if (!historicalAverage) return 50; // Sem histórico = neutro

  const ratio = targetValue / historicalAverage;

  if (ratio <= 1.5) return 100;      // Até 150% da média = realista
  if (ratio <= 2.0) return 80;       // Até 200% = desafiador
  if (ratio <= 3.0) return 50;       // Até 300% = muito desafiador
  return 20;                         // Acima de 300% = irrealista
}
```

Critério R (Relevant) - Score 0-100

```
function validateRelevant(category: string, impact: string): number {
  let score = 0;

  // Categoria definida (50 pontos)
  const validCategories = ['financeira', 'comportamental', 'corporativa',
'desenvolvimento'];
  if (validCategories.includes(category)) {
    score += 50;
  }

  // Impacto descrito (50 pontos)
  if (impact && impact.length > 20) {
    score += 50;
  }

  return score;
}
```

Critério T (Time-bound) - Score 0-100

```
function validateTimeBound(startDate: Date, dueDate: Date): number {
  const diffMonths = (dueDate.getTime() - startDate.getTime()) / (1000 * 60
* 60 * 24 * 30);

  if (diffMonths < 1) return 20;     // Menos de 1 mês = muito curto
  if (diffMonths <= 3) return 80;    // 1-3 meses = curto prazo
  if (diffMonths <= 6) return 100;   // 3-6 meses = ideal
  if (diffMonths <= 12) return 90;   // 6-12 meses = médio prazo
  if (diffMonths <= 24) return 70;   // 12-24 meses = longo prazo
  return 40;                         // Acima de 24 meses = muito longo
}
```

Score Final SMART

```
overallSmartScore = (
  specificScore +
  measurableScore +
  achievableScore +
  relevantScore +
  timeBoundScore
) / 5;
```

Classificação:

- **90-100**: Excelente (meta muito bem definida)
- **70-89**: Boa (meta adequada)
- **50-69**: Regular (necessita melhorias)
- **Abaixo de 50**: Insuficiente (requer revisão completa)

---

3. Cálculo de Performance 40-30-30

Fórmula Completa

```
Performance Final = (Metas × 0.40) + (Avaliação 360° × 0.30) + (Competências
× 0.30)
```

Componente 1: Metas (40%)

```
function calculateGoalsScore(employeeId: number, cycleId: number): number {
  const goals = getActiveGoals(employeeId, cycleId);

  if (goals.length === 0) return 0;

  // Média ponderada pelo peso de cada meta
  const weightedSum = goals.reduce((sum, goal) => {
    return sum + (goal.progress * goal.weight);
  }, 0);

  const totalWeight = goals.reduce((sum, goal) => sum + goal.weight, 0);

  return (weightedSum / totalWeight) * 100;
}
```

Componente 2: Avaliação 360° (30%)

```
function calculate360Score(employeeId: number, cycleId: number): number {
  const evaluations = get360Evaluations(employeeId, cycleId);

  if (evaluations.length === 0) return 0;

  // Média ponderada por tipo de avaliador
  const weights = {
    self: 0.20,
    manager: 0.40,
    peer: 0.20,
    subordinate: 0.20,
  };

  let weightedSum = 0;
  let totalWeight = 0;

  for (const eval of evaluations) {
    const weight = weights[eval.evaluatorType] || 0;
    weightedSum += eval.finalScore * weight;
    totalWeight += weight;
  }

  return (weightedSum / totalWeight) * 20; // Escala 0-100
}
```

Componente 3: Competências (30%)

```
function calculateCompetenciesScore(employeeId: number): number {
  const gaps = getCompetencyGaps(employeeId);

  if (gaps.length === 0) return 50; // Sem gaps = neutro

  // Média de aderência (quanto menor o gap, maior o score)
  const adherenceSum = gaps.reduce((sum, gap) => {
    const adherence = (gap.targetLevel - gap.gap) / gap.targetLevel;
    return sum + (adherence * 100);
  }, 0);

  return adherenceSum / gaps.length;
}
```

---

4. Sistema de Notificações Automáticas

Eventos de Notificação

Bônus:

- `bonus_calculated`: Quando bônus é calculado (notifica colaborador)
- `bonus_approval`: Quando bônus aguarda aprovação (notifica aprovador)
- `bonus_approved`: Quando bônus é aprovado (notifica colaborador)
- `bonus_rejected`: Quando bônus é rejeitado (notifica colaborador + motivo)
- `bonus_paid`: Quando bônus é pago (notifica colaborador)

Metas:

- `goal_approval`: Meta submetida para aprovação (notifica gestor)
- `goal_approved`: Meta aprovada (notifica colaborador)
- `goal_rejected`: Meta rejeitada (notifica colaborador + motivo)
- `goal_deadline`: Meta vencendo em 7 dias (notifica colaborador + gestor)
- `goal_overdue`: Meta vencida (notifica colaborador + gestor + RH)

Avaliações:

- `evaluation_pending`: Avaliação pendente (notifica avaliador)
- `evaluation_completed`: Avaliação concluída (notifica colaborador)
- `consensus_pending`: Consenso pendente (notifica líder)

PDI:

- `pdi_approved`: PDI aprovado (notifica colaborador)
- `pdi_feedback`: Feedback DGC recebido (notifica colaborador)
- `pdi_action_overdue`: Ação de PDI vencida (notifica colaborador)

Testes Psicométricos:

- `test_invitation`: Convite para teste (notifica colaborador)

- `test_completed`: Teste concluído (notifica RH)

Canais de Notificação

1. In-App (WebSocket)

```
// Servidor envia via Socket.IO
io.to(`user_${userId}`).emit('notification', {
  id: notification.id,
  type: notification.type,
  title: notification.title,
  message: notification.message,
  link: notification.link,
  createdAt: notification.createdAt,
});

// Cliente recebe e exibe badge
socket.on('notification', (data) => {
  updateNotificationBadge(data);
  showToast(data.title, data.message);
});
```

2. E-mail (Gmail SMTP)

```
async function sendEmailNotification(userId: number, notification:
Notification) {
  const user = await getUser(userId);
  const template = getEmailTemplate(notification.type);

  await emailService.sendEmail({
    to: user.email,
    subject: notification.title,
    html: template.render({
      userName: user.name,
      message: notification.message,
      link: `https://avd.uisa.com.br${notification.link}`,
    }),
  });
}
```

---

5. Cron Jobs Automáticos

Job 1: Notificações Automáticas (Diariamente às 9h)

```
cron.schedule('0 9 * * *', async () => {
  console.log('[Cron] Executando job de notificações automáticas...');

  // 1. Metas vencendo em 7 dias
  const goalsDeadline = await getGoalsDeadline(7);
  for (const goal of goalsDeadline) {
    await createNotification({
      userId: goal.employeeId,
      type: 'goal_deadline',
      title: 'Meta Vencendo',
      message: `Sua meta "${goal.title}" vence em 7 dias.`,
      link: `/metas/${goal.id}`,
    });
  }

  // 2. Avaliações 360° pendentes há mais de 3 dias
  const evaluationsPending = await getEvaluationsPending(3);
  for (const eval of evaluationsPending) {
    await createNotification({
      userId: eval.evaluatorId,
      type: 'evaluation_pending',
      title: 'Avaliação Pendente',
      message: `Você tem uma avaliação 360° pendente de
${eval.employeeName}.`,
      link: `/avaliacoes/${eval.id}`,
    });
  }

  // 3. PDIs sem atualização há mais de 30 dias
  const pdisStale = await getPDIsStale(30);
  for (const pdi of pdisStale) {
    await createNotification({
      userId: pdi.employeeId,
      type: 'pdi_action_overdue',
      title: 'PDI Sem Atualização',
      message: 'Seu PDI está sem atualizações há mais de 30 dias.',
      link: `/pdi-inteligente/${pdi.id}`,
    });
  }
});
```

Job 2: Pesquisas de Pulse (A cada 8 horas)

```
cron.schedule('0 */8 * * *', async () => {
  console.log('[Cron] Executando job de envio de pesquisas de pulse...');

  const activeSurveys = await getActivePulseSurveys();

  for (const survey of activeSurveys) {
    const employees = await getTargetEmployees(survey);

    for (const employee of employees) {
      // Verificar se já foi enviado
      const alreadySent = await checkEmailSent(survey.id, employee.id);
      if (alreadySent) continue;

      // Enviar e-mail
      try {
        await emailService.sendEmail({
          to: employee.email,
          subject: `Pesquisa: ${survey.title}`,
          html: pulseSurveyTemplate.render({
            employeeName: employee.name,
            surveyTitle: survey.title,
            surveyLink: `https://avd.uisa.com.br/pesquisa/${survey.id}`,
          }),
        });

        // Registrar envio
        await logEmailSent(survey.id, employee.id, true);
      } catch (error) {
        await logEmailSent(survey.id, employee.id, false, error.message);
      }
    }
  }
});
```

Job 3: Cálculo de Discrepâncias (Diariamente às 6h)

```
cron.schedule('0 6 * * *', async () => {
  console.log('[Cron] Executando job de cálculo de discrepâncias...');

  const yesterday = new Date();
  yesterday.setDate(yesterday.getDate() - 1);

  const employees = await getActiveEmployees();

  for (const employee of employees) {
    // Buscar ponto do dia anterior
    const clockRecord = await getTimeClockRecord(employee.id, yesterday);
    if (!clockRecord) continue;

    // Buscar atividades registradas
    const activities = await getEmployeeActivities(employee.id, yesterday);

    const clockMinutes = clockRecord.totalMinutes;
    const activityMinutes = activities.reduce((sum, a) => sum +
a.durationMinutes, 0);

    const differenceMinutes = clockMinutes - activityMinutes;
    const differencePercentage = (differenceMinutes / clockMinutes) * 100;

    // Classificar discrepância
    let classification = 'aceitavel';
    if (Math.abs(differencePercentage) > 20) {
      classification = differencePercentage > 0 ? 'under_reported' :
'over_reported';
    }

    // Salvar discrepância
    await createTimeDiscrepancy({
      employeeId: employee.id,
      discrepancyDate: yesterday,
      clockMinutes,
      activityMinutes,
      differenceMinutes,
      differencePercentage,
      classification,
    });

    // Criar alerta se discrepância crítica
    if (Math.abs(differencePercentage) > 30) {
```

```
      await createAlert({
        employeeId: employee.id,
        type: 'horas_inconsistentes',
        severity: 'alto',
        message: `Discrepância de ${differencePercentage.toFixed(1)}% entre
ponto e atividades registradas.`,
        data: { clockMinutes, activityMinutes, differencePercentage },
      });
    }
  }
});
```

---

6. Exportações (Excel e PDF)

Exportação Excel com ExcelJS

```
import ExcelJS from 'exceljs';

async function exportBonusToExcel(calculations: BonusCalculation[]) {
  const workbook = new ExcelJS.Workbook();
  const worksheet = workbook.addWorksheet('Relatório de Bônus');

  // Cabeçalho estilizado
  worksheet.columns = [
    { header: 'Colaborador', key: 'employee', width: 30 },
    { header: 'Chapa', key: 'code', width: 15 },
    { header: 'Departamento', key: 'department', width: 25 },
    { header: 'Cargo', key: 'position', width: 25 },
    { header: 'Mês Ref.', key: 'month', width: 12 },
    { header: 'Salário Base', key: 'salary', width: 15 },
    { header: 'Multiplicador', key: 'multiplier', width: 15 },
    { header: 'Valor Bônus', key: 'bonus', width: 15 },
    { header: 'Status', key: 'status', width: 15 },
    { header: 'Data Aprovação', key: 'approvedAt', width: 18 },
  ];

  // Estilo do cabeçalho
  worksheet.getRow(1).font = { bold: true, color: { argb: 'FFFFFFFF' } };
  worksheet.getRow(1).fill = {
    type: 'pattern',
    pattern: 'solid',
    fgColor: { argb: 'FF4472C4' },
  };

  // Dados
  calculations.forEach(calc => {
    worksheet.addRow({
      employee: calc.employeeName,
      code: calc.employeeCode,
      department: calc.departmentName,
      position: calc.positionTitle,
      month: format(calc.referenceMonth, 'MM/yyyy'),
      salary: calc.baseSalary / 100,
      multiplier: calc.multiplier,
      bonus: calc.totalAmount / 100,
      status: calc.status,
      approvedAt: calc.approvedAt ? format(calc.approvedAt, 'dd/MM/yyyy') :
```

```
'-',
    });
  });

  // Formatação monetária
  worksheet.getColumn('salary').numFmt = 'R$ #,##0.00';
  worksheet.getColumn('bonus').numFmt = 'R$ #,##0.00';

  // Linha de totais
  const totalRow = worksheet.addRow({
    employee: 'TOTAL',
    bonus: calculations.reduce((sum, c) => sum + c.totalAmount, 0) / 100,
  });
  totalRow.font = { bold: true };
  totalRow.getCell('bonus').numFmt = 'R$ #,##0.00';

  // Gerar arquivo
  const buffer = await workbook.xlsx.writeBuffer();
  return buffer;
}
```

Exportação PDF com jsPDF + html2canvas

```
import jsPDF from 'jspdf';
import html2canvas from 'html2canvas';

async function exportReportToPDF(elementId: string, filename: string) {
  const element = document.getElementById(elementId);
  if (!element) throw new Error('Elemento não encontrado');

  // Capturar elemento como imagem
  const canvas = await html2canvas(element, {
    scale: 2,
    useCORS: true,
    logging: false,
  });

  const imgData = canvas.toDataURL('image/png');
  const pdf = new jsPDF('p', 'mm', 'a4');

  const imgWidth = 210; // A4 width
  const imgHeight = (canvas.height * imgWidth) / canvas.width;

  let heightLeft = imgHeight;
  let position = 0;

  // Adicionar primeira página
  pdf.addImage(imgData, 'PNG', 0, position, imgWidth, imgHeight);
  heightLeft -= 297; // A4 height

  // Adicionar páginas adicionais se necessário
  while (heightLeft > 0) {
    position = heightLeft - imgHeight;
    pdf.addPage();
    pdf.addImage(imgData, 'PNG', 0, position, imgWidth, imgHeight);
    heightLeft -= 297;
  }

  // Cabeçalho e rodapé
  const pageCount = pdf.internal.pages.length - 1;
  for (let i = 1; i <= pageCount; i++) {
    pdf.setPage(i);

    // Cabeçalho
    pdf.setFontSize(10);
    pdf.setTextColor(100);
```

```
    pdf.text('Sistema AVD UISA', 10, 10);

    // Rodapé
    pdf.text(`Página ${i} de ${pageCount}`, 10, 287);
    pdf.text(format(new Date(), 'dd/MM/yyyy HH:mm'), 150, 287);
  }

  pdf.save(filename);
}
```

🔒 Segurança e Permissões

Controle de Acesso Baseado em Roles

```
// Middleware de permissões
export const permissions = {
  isAdmin: (ctx: Context) => {
    if (ctx.user.role !== 'admin') {
      throw new TRPCError({ code: 'FORBIDDEN', message: 'Acesso negado:
apenas administradores' });
    }
  },

  isLeader: async (ctx: Context) => {
    const employee = await getEmployeeByUserId(ctx.user.id);
    if (!employee) throw new TRPCError({ code: 'NOT_FOUND', message:
'Colaborador não encontrado' });

    const subordinates = await getSubordinates(employee.id);
    if (subordinates.length === 0) {
      throw new TRPCError({ code: 'FORBIDDEN', message: 'Acesso negado:
usuário não é líder' });
    }
  },

  canViewEmployee: async (ctx: Context, targetEmployeeId: number) => {
    if (ctx.user.role === 'admin') return true;

    const employee = await getEmployeeByUserId(ctx.user.id);
    if (!employee) return false;

    // Pode ver a si mesmo
    if (employee.id === targetEmployeeId) return true;

    // Pode ver subordinados diretos e indiretos
    const subordinates = await getAllSubordinates(employee.id);
    return subordinates.some(s => s.id === targetEmployeeId);
  },

  canDoConsensus: async (ctx: Context, evaluationId: number) => {
    const evaluation = await getEvaluation(evaluationId);
    const employee = await getEmployeeByUserId(ctx.user.id);

    // Apenas gestor do colaborador pode fazer consenso
```

```
    const targetEmployee = await getEmployee(evaluation.employeeId);
    return targetEmployee.managerId === employee.id;
  },

  canApproveBonus: async (ctx: Context, bonusCalculationId: number) => {
    const calculation = await getBonusCalculation(bonusCalculationId);
    const instance = await getWorkflowInstance(calculation.id);

    // Verificar se usuário é aprovador do nível atual
    const currentApproval = await getCurrentLevelApproval(instance.id,
instance.currentLevel);
    return currentApproval.approverId === ctx.user.id;
  },
};
```

Hierarquia Organizacional

```
// Buscar todos os subordinados (diretos e indiretos)
async function getAllSubordinates(managerId: number): Promise<Employee[]> {
  const direct = await getDirectSubordinates(managerId);
  const indirect: Employee[] = [];

  for (const subordinate of direct) {
    const children = await getAllSubordinates(subordinate.id);
    indirect.push(...children);
  }

  return [...direct, ...indirect];
}

// Verificar se colaborador está na hierarquia do líder
async function isInHierarchy(leaderId: number, employeeId: number):
Promise<boolean> {
  const subordinates = await getAllSubordinates(leaderId);
  return subordinates.some(s => s.id === employeeId);
}

// Filtrar dados por centro de custos
async function filterByCostCenter(costCenter: string, data: any[]) {
  if (!costCenter) return data;

  const employees = await getEmployeesByCostCenter(costCenter);
  const employeeIds = employees.map(e => e.id);

  return data.filter(item => employeeIds.includes(item.employeeId));
}
```

📈 Performance e Otimizações

Índices de Banco de Dados

```
-- Índices em employees
CREATE INDEX idx_employees_code ON employees(employeeCode);
CREATE INDEX idx_employees_user ON employees(userId);
CREATE INDEX idx_employees_dept ON employees(departmentId);
CREATE INDEX idx_employees_manager ON employees(managerId);
CREATE INDEX idx_employees_cost_center ON employees(costCenter);

-- Índices em smartGoals
CREATE INDEX idx_goals_employee ON smartGoals(employeeId);
CREATE INDEX idx_goals_cycle ON smartGoals(cycleId);
CREATE INDEX idx_goals_status ON smartGoals(status);
CREATE INDEX idx_goals_due_date ON smartGoals(dueDate);

-- Índices em bonusCalculations
CREATE INDEX idx_bonus_employee ON bonusCalculations(employeeId);
CREATE INDEX idx_bonus_ref_month ON bonusCalculations(referenceMonth);
CREATE INDEX idx_bonus_status ON bonusCalculations(status);

-- Índices compostos
CREATE INDEX idx_goals_employee_cycle ON smartGoals(employeeId, cycleId);
CREATE INDEX idx_bonus_employee_month ON bonusCalculations(employeeId,
referenceMonth);
CREATE INDEX idx_evaluations_employee_cycle ON
performanceEvaluations(employeeId, cycleId);
```

Otimização de Queries

```
// Usar Map para lookup O(1) em vez de filter O(n)
const employeesMap = new Map(employees.map(e => [e.id, e]));
const enrichedData = data.map(item => ({
  ...item,
  employeeName: employeesMap.get(item.employeeId)?.name,
}));

// Batch queries em vez de N+1
const employeeIds = data.map(d => d.employeeId);
const employees = await getEmployeesByIds(employeeIds);

// Usar JOINs em vez de múltiplas queries
const result = await db.execute(sql`
  SELECT
    g.*,
    e.name as employeeName,
    d.name as departmentName,
    p.title as positionTitle
  FROM smartGoals g
  JOIN employees e ON g.employeeId = e.id
  LEFT JOIN departments d ON e.departmentId = d.id
  LEFT JOIN positions p ON e.positionId = p.id
  WHERE g.cycleId = ${cycleId}
`);
```

Caching com React Query (tRPC)

```
// Cache de 5 minutos para dados estáticos
const { data: departments } = trpc.departments.list.useQuery(undefined, {
  staleTime: 5 * 60 * 1000,
  cacheTime: 10 * 60 * 1000,
});

// Invalidação automática após mutation
const mutation = trpc.goals.update.useMutation({
  onSuccess: () => {
    trpc.useUtils().goals.list.invalidate();
    trpc.useUtils().goals.getDashboard.invalidate();
  },
});
```

🧪 Testes

Testes Unitários (Vitest)

```
// bonus.test.ts
import { describe, it, expect } from 'vitest';
import { bonusRouter } from './bonusRouter';

describe('bonusRouter', () => {
  it('deve listar políticas ativas', async () => {
    const result = await bonusRouter.list({ isActive: true });
    expect(result).toBeInstanceOf(Array);
    expect(result.every(p => p.isActive)).toBe(true);
  });

  it('deve calcular bônus com multiplicadores', async () => {
    const result = await bonusRouter.calculateBonus({
      employeeId: 1,
      policyId: 1,
      referenceMonth: '2025-01-01',
    });

    expect(result.totalAmount).toBeGreaterThan(0);
    expect(result.status).toBe('calculado');
  });

  it('deve validar elegibilidade por metas', async () => {
    const result = await bonusRouter.calculateBonus({
      employeeId: 999, // Sem metas
      policyId: 1,
      referenceMonth: '2025-01-01',
    });

    expect(result.error).toBe('Colaborador não elegível: metas
insuficientes');
  });
});
```

Testes End-to-End

```
// dashboard.e2e.test.ts
import { test, expect } from '@playwright/test';

test('deve carregar dashboard principal', async ({ page }) => {
  await page.goto('/');

  // Verificar KPIs
  await expect(page.locator('text=Metas Ativas')).toBeVisible();
  await expect(page.locator('text=Avaliações')).toBeVisible();
  await expect(page.locator('text=PDI Ativos')).toBeVisible();

  // Verificar gráficos
  await expect(page.locator('canvas')).toBeVisible();
});

test('deve criar meta SMART', async ({ page }) => {
  await page.goto('/metas/criar');

  await page.fill('input[name="title"]', 'Aumentar vendas em 20%');
  await page.fill('textarea[name="description"]', 'Meta de aumento de
vendas...');
  await page.selectOption('select[name="category"]', 'financeira');
  await page.fill('input[name="targetValue"]', '120');

  await page.click('button[type="submit"]');

  await expect(page.locator('text=Meta criada com sucesso')).toBeVisible();
});
```

📦 Deployment

Variáveis de Ambiente

```
 # Banco de Dados
DATABASE_URL=mysql://user:password@host:port/database

# Autenticação
JWT_SECRET=your_jwt_secret_here
OAUTH_SERVER_URL=https://api.manus.im
VITE_OAUTH_PORTAL_URL=https://portal.manus.im

# E-mail (Gmail SMTP)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=avd@uisa.com.br
SMTP_PASSWORD=your_app_password_here

# Storage (S3)
S3_BUCKET=avd-uisa-storage
S3_REGION=us-east-1
S3_ACCESS_KEY=your_access_key
S3_SECRET_KEY=your_secret_key

# IA (Gemini)
GEMINI_API_KEY=your_gemini_api_key

# Aplicação
VITE_APP_TITLE=Sistema AVD UISA
VITE_APP_LOGO=/logo-uisa.png
NODE_ENV=production
PORT=3000
```

Build e Deploy

```
# Instalar dependências
pnpm install

# Build frontend
pnpm build

# Aplicar migrations
pnpm db:push

# Iniciar servidor
pnpm start
```

Monitoramento

```
// Logs estruturados
console.log('[Auth] User logged in:', { userId, timestamp });
console.error('[Database] Query failed:', { error, query });

// Health check endpoint
app.get('/health', (req, res) => {
  res.json({
    status: 'ok',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    database: db ? 'connected' : 'disconnected',
  });
});
```

📚 Referências e Documentação

Bibliotecas Principais

- **React**: https://react.dev
- **tRPC**: https://trpc.io
- **Drizzle ORM**: https://orm.drizzle.team

- **Shadcn/ui**: https://ui.shadcn.com
- **Tailwind CSS**: https://tailwindcss.com
- **Chart.js**: https://www.chartjs.org
- **ExcelJS**: https://github.com/exceljs/exceljs
- **jsPDF**: https://github.com/parallax/jsPDF

Metodologias e Frameworks

- **Metas SMART**: Specific, Measurable, Achievable, Relevant, Time-bound
- **PDI 70-20-10**: 70% experiências, 20% relacionamentos, 10% cursos formais
- **Matriz 9-Box**: Performance × Potencial (GE/McKinsey)
- **Avaliação 360°**: Feedback multidirecional (self, manager, peers, subordinates)
- **DISC**: Dominância, Influência, Estabilidade, Conformidade
- **Big Five**: Abertura, Conscienciosidade, Extroversão, Amabilidade, Neuroticismo

---

🎯 Próximos Passos Recomendados

Curto Prazo (1-2 meses)

1. Importar 481 descrições de cargos UISA do arquivo ZIP
2. Implementar dashboard de compliance e SLA
3. Integração com folha de pagamento (exportação CSV/XML)
4. Testes de carga e performance
5. Treinamento de usuários (RH, gestores, colaboradores)

Médio Prazo (3-6 meses)

1. Integração com TOTVS RM (sync automático)
2. Integração com Azure AD (SSO)
3. Aplicativo mobile (React Native)
4. Sistema de gamificação expandido

5. Relatórios avançados com BI

Longo Prazo (6-12 meses)

1. IA preditiva para identificação de talentos
2. Chatbot de RH com IA
3. Sistema de recomendação de cursos
4. Análise de sentimento em feedbacks
5. Dashboard executivo em tempo real

Documento gerado automaticamente pelo Sistema AVD UISA
Versão: 1.0 | **Data**: 20/11/2024 | **Total de Páginas**: 50+