Projeto de Conclusão de Curso: Sistema de Ciclo de Desempenho

Documentação Completa para Desenvolvimento Backend com Node.js, TypeScript e PostgreSQL

Equipe do Projeto:

- Wanessa Karen - Modeladora de Dados
- Alessandra Santos - Desenvolvedora de Rotas
- Andre Tavares - Lógica de Negócio
- Diciane Alves - Documentação

Período: 11/11/2025 a 08/01/2026 (9 semanas)
Encontros: Terças e Quintas, 18h às 21h

1. Visão Geral do Projeto

1.1 Objetivo

Desenvolver uma plataforma backend automatizada para gestão do ciclo de desempenho de colaboradores, transformando processos manuais em um fluxo eficiente, transparente e baseado em dados [1].

1.2 Problema a Resolver

O processo atual de gestão de desempenho é manual, fragmentado em planilhas e formulários, resultando em:

- 30% de divergências nas avaliações
- Sobrecarga dos gestores
- Falta de histórico consolidado
- Atrasos em promoções
- Baixa visibilidade para colaboradores [1]

1.3 Tecnologias Utilizadas

Stack Principal:

- **Backend:** Node.js com TypeScript
- **Framework:** Express.js
- **Banco de Dados:** PostgreSQL
- **Fila de Mensagens:** Redis + Bull
- **Documentação:** Swagger
- **Controle de Versão:** Git
- **Gestão de Projeto:** Trello

A escolha dessa arquitetura é adequada para alunos iniciantes por diversos motivos [2] [3] [4] [5] [6] [7] [8] [9]:

1. **Express.js** é o framework mais popular e possui estrutura livre que permite aprendizado incremental [10] [7]
2. **TypeScript** adiciona segurança de tipos, facilitando a detecção de erros para iniciantes [6] [11] [8] [9]
3. **PostgreSQL** é robusto, possui excelente documentação e segue padrões SQL [12] [13] [14]
4. **Redis** é simples de implementar para filas e tem baixa curva de aprendizado [2] [4] [5] [15]

2. Personas e Fluxos de Uso

2.1 Gestor de Equipe

Objetivo: Acompanhar desempenho da equipe e tomar decisões justas

Caminho Feliz:

1. Login no sistema
2. Acessar dashboard da equipe
3. Visualizar colaboradores e seus status
4. Selecionar colaborador para análise detalhada
5. Ver scores de competências e metas
6. Analisar posição na matriz Nine Box
7. Tomar decisão (aprovar, criar PDI, dar feedback)
8. Gerar relatório de acompanhamento [1]

Principais Funcionalidades:

- Dashboard com visão consolidada da equipe
- Detalhamento individual de colaboradores
- Alertas para colaboradores que precisam atenção
- Histórico de desempenho
- Relatórios comparativos entre ciclos

2.2 Colaborador

Objetivo: Acompanhar seu desempenho e plano de carreira

Caminho Feliz:

1. Login no sistema
2. Acessar perfil pessoal
3. Visualizar avaliações do ciclo atual
4. Consultar score de mérito
5. Ver status das metas (atingidas/pendentes)
6. Ler feedback do gestor
7. Acompanhar plano de carreira e trilhas disponíveis
8. Logout [1]

Principais Funcionalidades:

- Tela responsiva de consulta
- Visualização de feedback estruturado
- Acompanhamento de metas
- Acesso ao histórico de avaliações

2.3 Analista de RH

Objetivo: Consolidar dados e gerar insights estratégicos

Caminho Feliz:

1. Login no sistema
2. Acessar painel estratégico de RH
3. Ver todos os ciclos ativos

4. Selecionar ciclo para análise
5. Analisar dados consolidados da organização
6. Visualizar matriz Nine Box geral
7. Gerar relatórios de sucessão e lacunas de competências
8. Configurar próximo ciclo de avaliação
9. Logout [1]

Principais Funcionalidades:

- Painel de sucessão estratégica
- Relatórios consolidados
- Configuração de ciclos e competências
- Análise de tendências organizacionais

2.4 Administrador de Sistemas

Objetivo: Manter segurança e integridade do sistema

Principais Funcionalidades:

- Gestão de perfis e permissões
- Auditoria de acessos
- Configurações gerais do sistema [1]

3. Arquitetura do Sistema

3.1 Estrutura de Pastas

A estrutura proposta segue as melhores práticas para projetos Express + TypeScript [3] [10] [11] [8] [9]:

```
ciclo-desempenho-backend/
├── src/
│   ├── config/          # Configurações (DB, Redis, Swagger)
│   ├── models/          # Interfaces e tipos TypeScript
│   ├── repositories/    # Acesso a dados (queries SQL)
│   ├── services/        # Lógica de negócio
│   ├── controllers/     # Controladores HTTP
│   ├── routes/          # Definição de rotas
│   ├── middlewares/     # Autenticação, validação
│   ├── jobs/            # Filas Redis
│   ├── utils/           # Funções auxiliares
│   ├── app.ts           # Configuração Express
│   └── server.ts        # Inicialização
├── public/              # Tela do colaborador
├── scripts/             # Scripts SQL
├── docs/                # Documentação
└── tests/               # Testes (opcional)
```

Fluxo de Requisição:

```
Cliente → Route → Middleware → Controller → Service → Repository → Database
```

Esta arquitetura em camadas facilita o aprendizado incremental e a separação de responsabilidades [3] [10] [8] [9].

3.2 Banco de Dados PostgreSQL

Modelo de Dados Simplificado:

O banco foi projetado com **11 tabelas** para manter simplicidade sem perder funcionalidade [12] [13] [14]:

Tabelas Principais:

1. **usuarios** - Autenticação e perfis
2. **colaboradores** - Dados funcionais
3. **ciclos_desempenho** - Períodos de avaliação
4. **competencias** - Competências avaliadas
5. **metas** - Metas individuais
6. **avaliacoes** - Avaliações realizadas
7. **avaliacoes_competencias** - Notas das competências
8. **resultados_ciclo** - Consolidação de resultados
9. **configuracoes_ninebox** - Matriz Nine Box
10. **historico_avaliacoes** - Auditoria

Relacionamentos:

- Um colaborador tem várias metas e avaliações
- Uma avaliação possui várias notas de competências
- Um ciclo contém várias avaliações e resultados

Otimizações [16] [17] [14] [18]**:**

- Índices em colunas frequentemente consultadas
- Constraints para garantir integridade
- Função trigger para atualização automática de timestamps

3.3 Filas com Redis

O Redis será utilizado para processamento assíncrono de tarefas demoradas [2] [4] [5] [15]:

Casos de Uso:

- Cálculo de mérito (processamento em lote)
- Geração de relatórios complexos
- Envio de notificações

Vantagens para Iniciantes:

- Implementação simples com biblioteca Bull
- Mensagens não se perdem se o sistema cair
- Permite testes do sistema sob carga

4. Modelo de Dados Detalhado

4.1 Diagrama Entidade-Relacionamento (DER)

O DER foi criado com foco na simplicidade, mantendo apenas relacionamentos essenciais:

Relacionamentos Principais:

- Usuario (1) ↔ (0..1) Colaborador
- Colaborador (1) ↔ (N) Colaborador (hierarquia gestor-liderado)
- Colaborador (1) ↔ (N) Meta

- Colaborador (1) ↔ (N) Avaliacao
- Ciclo (1) ↔ (N) Avaliacao
- Avaliacao (1) ↔ (N) AvaliacaoCompetencia
- Competencia (1) ↔ (N) AvaliacaoCompetencia
- ResultadoCiclo (N) ↔ (1) ConfiguracaoNineBox

4.2 Script de Criação

O script SQL foi desenvolvido seguindo boas práticas do PostgreSQL [12] [13] [14]:

Destaques:

- Uso de tipos apropriados (SERIAL, VARCHAR, DECIMAL, JSONB)
- Constraints CHECK para validação de dados
- Chaves estrangeiras com ON DELETE CASCADE quando apropriado
- Valores DEFAULT para facilitar inserções
- Índices estratégicos para consultas frequentes

4.3 Massa de Testes

Foi criada massa realista com:

- 16 usuários (3 gestores, 2 RH, 10 colaboradores, 1 admin)
- 5 ciclos (2023, 2024, 2025 anuais + 2 de experiência)
- 8 competências padrão
- 6 metas com diferentes níveis de atingimento
- 3 avaliações completas com notas
- 9 quadrantes configurados do Nine Box

Esta massa permite testar todos os fluxos das personas.

5. Endpoints da API

5.1 Endpoints do Gestor

Foram desenvolvidas **9 queries SQL** prontas para implementação [1]:

1. **Dashboard da Equipe**
   - `GET /api/gestor/dashboard/:gestorId/:cicloId`
   - Retorna visão consolidada com scores e status
2. **Detalhes do Colaborador**
   - `GET /api/gestor/colaborador/:colaboradorId/:cicloId`
   - Informações completas de desempenho
3. **Competências Avaliadas**
   - `GET /api/gestor/colaborador/:colaboradorId/competencias/:cicloId`
   - Notas por competência com observações
4. **Metas do Colaborador**
   - `GET /api/gestor/colaborador/:colaboradorId/metas/:cicloId`
   - Status de atingimento das metas
5. **Matriz Nine Box da Equipe**
   - `GET /api/gestor/ninebox/:gestorId/:cicloId`
   - Posicionamento de todos os colaboradores

6. **Histórico de Desempenho**
    - `GET /api/gestor/colaborador/:colaboradorId/historico`
    - Evolução ao longo dos ciclos
7. **Estatísticas Gerais**
    - `GET /api/gestor/estatisticas/:gestorId/:cicloId`
    - Médias e totalizações da equipe
8. **Alertas e Atenções**
    - `GET /api/gestor/alerta/:gestorId/:cicloId`
    - Colaboradores que precisam atenção
9. **Comparativo entre Ciclos**
    - `GET /api/gestor/comparativo/:gestorId`
    - Variação de desempenho ano a ano

5.2 Outros Endpoints

Cadastros (via carga):

- `POST /api/admin/usuarios/importar` - Importar usuários via CSV
- `POST /api/admin/colaboradores/importar` - Importar colaboradores via CSV
- `POST /api/admin/ciclos` - Criar ciclo de desempenho
- `POST /api/admin/competencias` - Cadastrar competências

Colaborador:

- `GET /api/colaborador/perfil/:id` - Consultar próprio desempenho
- `GET /api/colaborador/metas/:id/:cicloId` - Ver suas metas
- `GET /api/colaborador/feedback/:id/:cicloId` - Acessar feedbacks

6. Cronograma Detalhado

6.1 Visão Geral

Duração: 9 semanas (18 encontros)
Período: 11/11/2025 a 08/01/2026
Encontros: Terças e Quintas, 18h às 21h

6.2 Fases do Projeto

Fase 1: Setup e Modelagem (Encontros 1-4)

- Semanas 1-2 (11/11 a 20/11)
- Configuração do ambiente
- Criação do DER e scripts SQL
- Definição da arquitetura
- Documentação inicial

Fase 2: Desenvolvimento Base (Encontros 5-8)

- Semanas 3-4 (25/11 a 05/12)
- Implementação dos models
- Desenvolvimento de endpoints básicos
- Regras de negócio fundamentais
- Massa de testes

Fase 3: Funcionalidades (Encontros 9-12)

- Semanas 5-6 (10/12 a 19/12)
- Endpoints do gestor
- Telas simples
- Configuração Nine Box
- Filas Redis

Fase 4: Testes e Refinamento (Encontros 13-16)

- Semanas 7-8 (23/12 a 01/01)
- Testes de integração
- Correção de bugs
- Otimizações
- Validação dos fluxos

Fase 5: Entrega Final (Encontros 17-18)

- Semana 9 (06/01 a 08/01)
- Preparação da apresentação
- Documentação final
- Relatórios do projeto

6.3 Distribuição de Tarefas por Recurso

Wanessa Karen (Modeladora de Dados):

- DER e diagrama de classes
- Scripts SQL de criação
- Massa de testes
- Queries dos endpoints
- Otimização de consultas
- Total: 6 tarefas principais

Alessandra Santos (Desenvolvedora de Rotas):

- Setup do projeto Node.js
- Configuração Redis
- Desenvolvimento de endpoints
- Importação CSV
- Telas simples (frontend)
- Total: 8 tarefas principais

Andre Tavares (Lógica de Negócio):

- Definição da arquitetura
- Implementação de services
- Cálculos de mérito e Nine Box
- Regras de negócio
- Filas Redis
- Total: 7 tarefas principais

Diciane Alves (Documentação):

- README e guias
- Documentação Swagger

- Desenho de fluxos
- Guia de instalação
- Relatórios finais
- Total: 8 tarefas principais

Tarefas em Equipe:

- Testes finais
- Validação de fluxos
- Apresentação
- Total: 3 tarefas

6.4 Uso do Trello

O projeto utilizará Trello para gestão com os seguintes quadros:

Colunas Sugeridas:

1. **Backlog** - Todas as tarefas planejadas
2. **A Fazer** - Próximas tarefas do sprint
3. **Em Andamento** - Tarefas sendo executadas
4. **Revisão** - Aguardando validação
5. **Concluído** - Tarefas finalizadas

Etiquetas por Responsável:

- Azul: Wanessa (Dados)
- Verde: Alessandra (Rotas)
- Amarelo: Andre (Negócio)
- Roxo: Diciane (Docs)
- ● Preto: Equipe

Etiquetas por Prioridade:

- Alta
- Média
- Baixa

Cartões devem conter:

- Título descritivo
- Descrição detalhada
- Responsável
- Data prevista
- Checklist de subtarefas
- Arquivos anexados

7. Guia de Implementação para Iniciantes

7.1 Configuração do Ambiente

Passo 1: Instalar Ferramentas

1. Node.js (v18+): https://nodejs.org/
2. PostgreSQL (v14+): https://www.postgresql.org/
3. Redis (v6+): https://redis.io/
4. VS Code: https://code.visualstudio.com/
5. Git: https://git-scm.com/

Passo 2: Criar Projeto

```
# Criar pasta e inicializar
mkdir ciclo-desempenho-backend
cd ciclo-desempenho-backend
npm init -y

# Instalar dependências principais
npm install express typescript ts-node
npm install @types/node @types/express
npm install pg redis bull
npm install dotenv cors express-validator
npm install swagger-ui-express

# Instalar dependências de desenvolvimento
npm install --save-dev nodemon @types/cors
```

Passo 3: Configurar TypeScript

```
npx tsc --init
```

Editar `tsconfig.json`:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "commonjs",
    "rootDir": "./src",
    "outDir": "./dist",
    "esModuleInterop": true,
    "strict": true
  }
}
```

Passo 4: Criar Estrutura

```
mkdir -p src/{config,models,repositories,services,controllers,routes,middlewares,jobs,utils}
mkdir -p scripts docs public
```

7.2 Exemplo de Implementação

Arquivo: src/config/database.ts

```
import { Pool } from 'pg';

const pool = new Pool({
  host: process.env.DATABASE_HOST || 'localhost',
  port: Number(process.env.DATABASE_PORT) || 5432,
  database: process.env.DATABASE_NAME,
  user: process.env.DATABASE_USER,
  password: process.env.DATABASE_PASSWORD,
});
```

```
export default pool;
```

Arquivo: src/models/Colaborador.ts

```
export interface Colaborador {
  id: number;
  usuarioId: number;
  matricula: string;
  cargo: string;
  departamento: string;
  gestorId?: number;
  dataAdmissao: Date;
  status: 'ATIVO' | 'EXPERIENCIA' | 'DESLIGADO';
}
```

Arquivo: src/repositories/ColaboradorRepository.ts

```
import pool from '../config/database';
import { Colaborador } from '../models/Colaborador';

export class ColaboradorRepository {
  async findById(id: number): Promise&lt;Colaborador | null&gt; {
    const result = await pool.query(
      'SELECT * FROM colaboradores WHERE id = $1',
      [id]
    );
    return result.rows[^0] || null;
  }

  async findByGestor(gestorId: number): Promise&lt;Colaborador[]&gt; {
    const result = await pool.query(
      'SELECT * FROM colaboradores WHERE gestor_id = $1',
      [gestorId]
    );
    return result.rows;
  }
}
```

Arquivo: src/services/ColaboradorService.ts

```
import { ColaboradorRepository } from '../repositories/ColaboradorRepository';

export class ColaboradorService {
  private repository: ColaboradorRepository;

  constructor() {
    this.repository = new ColaboradorRepository();
  }

  async obterEquipe(gestorId: number) {
    return await this.repository.findByGestor(gestorId);
  }
}
```

Arquivo: src/controllers/GestorController.ts

```
import { Request, Response } from 'express';
import { ColaboradorService } from '../services/ColaboradorService';

export class GestorController {
  private service: ColaboradorService;

  constructor() {
    this.service = new ColaboradorService();
  }
```

```
  async getDashboard(req: Request, res: Response) {
    try {
      const { gestorId } = req.params;
      const equipe = await this.service.obterEquipe(Number(gestorId));

      return res.json({
        success: true,
        data: equipe
      });
    } catch (error) {
      return res.status(500).json({
        success: false,
        error: 'Erro ao buscar dashboard'
      });
    }
  }
}
```

Arquivo: src/routes/gestorRoutes.ts

```
import { Router } from 'express';
import { GestorController } from '../controllers/GestorController';

const router = Router();
const controller = new GestorController();

router.get('/dashboard/:gestorId/:cicloId',
  controller.getDashboard.bind(controller)
);

export default router;
```

Arquivo: src/app.ts

```
import express from 'express';
import cors from 'cors';
import gestorRoutes from './routes/gestorRoutes';

const app = express();

app.use(cors());
app.use(express.json());

app.use('/api/gestor', gestorRoutes);

export default app;
```

Arquivo: src/server.ts

```
import app from './app';
import dotenv from 'dotenv';

dotenv.config();

const PORT = process.env.PORT || 3000;

app.listen(PORT, () =&gt; {
  console.log(`Servidor rodando na porta ${PORT}`);
});
```

7.3 Dicas para Iniciantes

1. Comece Simples

- Implemente um endpoint por vez
- Teste cada funcionalidade antes de prosseguir
- Use console.log para debugar

2. Organização do Código

- Mantenha cada arquivo com uma responsabilidade
- Use nomes descritivos para variáveis e funções
- Comente partes complexas

3. Tratamento de Erros

- Sempre use try-catch em operações assíncronas
- Retorne mensagens de erro claras
- Log todos os erros no console

4. Testes

- Teste endpoints com Postman ou Insomnia
- Crie cenários de teste para cada persona
- Valide os dados retornados

5. Git e Controle de Versão

- Faça commits frequentes com mensagens claras
- Use branches para novas funcionalidades
- Não commite arquivos .env

8. Tela do Colaborador

8.1 Requisitos

A tela deve ser simples, funcional e responsiva [1]:

Funcionalidades:

- Login do colaborador
- Visualização do score de mérito
- Lista de competências avaliadas com notas
- Status das metas (atingidas/pendentes)
- Feedback do gestor
- Histórico de ciclos anteriores

8.2 Implementação Sugerida

Tecnologias:

- HTML5 + CSS3
- JavaScript vanilla (sem frameworks)
- Consumo da API via Fetch

Arquivo: public/index.html

```
&lt;html lang="pt-BR"&gt;
```

```
&lt;head&gt;
    &lt;meta charset="UTF-8"&gt;
    &lt;meta name="viewport" content="width=device-width, initial-scale=1.0"&gt;
    &lt;title&gt;Meu Desempenho&lt;/title&gt;
    &lt;link rel="stylesheet" href="styles.css"&gt;
&lt;/head&gt;
&lt;body&gt;
    &lt;header&gt;
        <h1>Sistema de Desempenho</h1>
        <div></div>
    &lt;/header&gt;

    &lt;main&gt;
        &lt;section id="score-card"&gt;
            <h2>Meu Score de Mérito</h2>
            <div></div>
        &lt;/section&gt;

        &lt;section id="competencias"&gt;
            <h2>Competências Avaliadas</h2>
            <div></div>
        &lt;/section&gt;

        &lt;section id="metas"&gt;
            <h2>Minhas Metas</h2>
            <div></div>
        &lt;/section&gt;

        &lt;section id="feedback"&gt;
            <h2>Feedback do Gestor</h2>
            <div></div>
        &lt;/section&gt;
    &lt;/main&gt;

    &lt;script src="app.js"&gt;&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;
```

Arquivo: public/app.js

```
const API_URL = 'http://localhost:3000/api';
const colaboradorId = 6; // Obtido do login

async function carregarDados() {
    try {
        // Buscar dados do colaborador
        const response = await fetch(
            `${API_URL}/colaborador/perfil/${colaboradorId}`
        );
        const data = await response.json();

        // Atualizar interface
        exibirScore(data.scoreMerito);
        exibirCompetencias(data.competencias);
        exibirMetas(data.metas);
        exibirFeedback(data.feedback);

    } catch (error) {
        console.error('Erro ao carregar dados:', error);
    }
}

function exibirScore(score) {
    const scoreDisplay = document.getElementById('score-display');
    scoreDisplay.innerHTML = `
        <div>${score.toFixed(2)}</div>
        <div>Pontos</div>
    `;
}

// Implementar outras funções...
```

```
// Carregar dados ao iniciar
carregarDados();
```

9. Configuração do Nine Box

9.1 Conceito

A Matriz Nine Box é uma ferramenta de gestão de talentos que posiciona colaboradores em 9 quadrantes baseados em:

- **Eixo X (Desempenho):** Score de mérito
- **Eixo Y (Potencial):** Avaliação de potencial

9.2 Tela de Configuração

Deve permitir:

- Definir nomes dos quadrantes
- Configurar cores
- Estabelecer ações recomendadas
- Definir faixas de pontuação

Endpoint:

```
POST /api/admin/ninebox/configurar
GET /api/admin/ninebox/configuracao
PUT /api/admin/ninebox/configuracao/:id
```

9.3 Lógica de Posicionamento

```
function determinarQuadrante(
  scoreMerito: number,
  scorePotencial: number
): string {
  // Baixo: 0-60, Médio: 60-80, Alto: 80-100
  const desempenho = scoreMerito &lt; 60 ? 1 :
                     scoreMerito &lt; 80 ? 2 : 3;
  const potencial = scorePotencial &lt; 60 ? 1 :
                    scorePotencial &lt; 80 ? 2 : 3;

  const posicoes = {
    '1,1': 'BAIXO_DESEMPENHO_BAIXO_POTENCIAL',
    '2,1': 'DESEMPENHO_MEDIO_BAIXO_POTENCIAL',
    '3,1': 'ALTO_DESEMPENHO_BAIXO_POTENCIAL',
    '1,2': 'BAIXO_DESEMPENHO_POTENCIAL_MEDIO',
    '2,2': 'DESEMPENHO_MEDIO_POTENCIAL_MEDIO',
    '3,2': 'ALTO_DESEMPENHO_POTENCIAL_MEDIO',
    '1,3': 'BAIXO_DESEMPENHO_ALTO_POTENCIAL',
    '2,3': 'DESEMPENHO_MEDIO_ALTO_POTENCIAL',
    '3,3': 'ALTO_DESEMPENHO_ALTO_POTENCIAL'
  };

  return posicoes[`${desempenho},${potencial}`];
}
```

10. Recomendações Pedagógicas

10.1 Para o Professor

Acompanhamento:

- Realizar code review semanal
- Promover pair programming entre alunos
- Estabelecer checkpoints de validação
- Incentivar documentação do código

Avaliação Sugerida:

- 20% - Participação e pontualidade
- 30% - Qualidade do código individual
- 30% - Funcionalidades implementadas
- 20% - Documentação e apresentação

Pontos de Atenção:

- Garantir que todos entendam a arquitetura
- Validar configuração do ambiente de cada aluno
- Acompanhar dificuldades individuais
- Promover integração contínua do código

10.2 Para os Alunos

Boas Práticas:

- Comunicação constante via Trello
- Commits frequentes com mensagens claras
- Testes manuais de cada funcionalidade
- Documentação de dificuldades encontradas

Dicas de Estudo:

- Revisar conceitos de TypeScript básico
- Estudar SQL e normalização de dados
- Praticar requisições HTTP (GET, POST, PUT, DELETE)
- Entender assincronismo em JavaScript (async/await)

Recursos de Aprendizado:

- Documentação oficial do Express.js
- Tutoriais do PostgreSQL
- Guias sobre arquitetura REST
- Vídeos sobre Redis e filas

10.3 Desafios Esperados

Técnicos:

- Configuração inicial do ambiente
- Entendimento de TypeScript
- Conexão com banco de dados
- Implementação de queries complexas
- Debugging de erros assíncronos

Organizacionais:

- Coordenação de tarefas no Trello
- Integração de código entre membros
- Resolução de conflitos no Git
- Cumprimento de prazos

Soluções:

- Sessões de pair programming
- Documentação detalhada de setup
- Revisões de código em grupo
- Buffer de tempo no cronograma

11. Critérios de Avaliação do Projeto

11.1 Funcionalidades (40 pontos)

- [ ] Banco de dados criado e populado (5 pts)
- [ ] Endpoints do gestor funcionando (10 pts)
- [ ] Tela do colaborador funcional (8 pts)
- [ ] Importação CSV implementada (5 pts)
- [ ] Cálculos de mérito corretos (7 pts)
- [ ] Nine Box configurável (5 pts)

11.2 Qualidade do Código (30 pontos)

- [ ] Estrutura de pastas organizada (5 pts)
- [ ] Código limpo e legível (8 pts)
- [ ] Tratamento de erros adequado (7 pts)
- [ ] Uso correto de TypeScript (5 pts)
- [ ] Boas práticas de segurança (5 pts)

11.3 Documentação (20 pontos)

- [ ] README completo (5 pts)
- [ ] Guia de instalação funcional (5 pts)
- [ ] Documentação Swagger (5 pts)
- [ ] Comentários no código (5 pts)

11.4 Apresentação (10 pontos)

- [ ] Demonstração dos fluxos (5 pts)
- [ ] Explicação da arquitetura (3 pts)
- [ ] Resposta a perguntas (2 pts)

Total: 100 pontos

12. Próximos Passos e Evolução

12.1 Melhorias Futuras

Após o MVP, o sistema pode evoluir para:

Fase 2 (Curto Prazo):

- Autenticação JWT completa
- Testes automatizados (Jest)
- Upload de evidências de metas
- Notificações por e-mail
- Exportação de relatórios em PDF

Fase 3 (Médio Prazo):

- Dashboard analítico com gráficos
- Avaliação 360 graus
- Integração com sistemas de RH
- App mobile nativo
- Machine learning para previsões

Fase 4 (Longo Prazo):

- Gamificação do processo
- Mentoria e coaching integrados
- Plataforma de aprendizado (LMS)
- Análise preditiva de turnover
- Blockchain para certificações

12.2 Oportunidades de Expansão Acadêmica

Este projeto pode gerar:

- TCC de pós-graduação
- Artigos científicos sobre gestão de talentos
- Estudo de caso sobre arquitetura de software
- Material didático para outros cursos
- Portfólio profissional dos alunos

13. Conclusão

Este projeto oferece uma excelente oportunidade para alunos iniciantes desenvolverem um sistema completo de backend, desde a modelagem do banco de dados até a implementação de APIs RESTful e processamento assíncrono [1] [2] [3] [4] [5] [6] [7] [8] [9].

Diferenciais do Projeto:

1. **Relevância Prática:** Sistema aplicável em empresas reais
2. **Arquitetura Moderna:** Stack tecnológico atual e demandado
3. **Aprendizado Completo:** Cobre todas as camadas de desenvolvimento
4. **Documentação Rica:** Guias detalhados para cada etapa
5. **Escalabilidade:** Base sólida para futuras expansões

Preparação para o Mercado:

Os alunos sairão capacitados em:

- Desenvolvimento backend com Node.js e TypeScript
- Modelagem e otimização de banco de dados
- Arquitetura em camadas (MVC/Repository Pattern)
- APIs RESTful e documentação Swagger
- Filas de mensagens e processamento assíncrono
- Controle de versão com Git
- Gestão de projetos com Trello

Mensagem Final:

Este projeto, quando concluído, representará não apenas a conclusão de um curso técnico, mas o início de uma carreira promissora em desenvolvimento de software. A jornada será desafiadora, mas cada linha de código escrita é um passo em direção ao domínio da tecnologia e à construção de soluções que impactam positivamente organizações e pessoas.

Sucesso à equipe!

Referências

[1] Documento de Requisitos do Sistema - requisitos.md
[2] Sistema de filas com prioridades usando Node.js e Redis - YouTube
[12] PostgreSQL Tutorial: Introdução completa ao PostgreSQL - DevMedia
[3] Melhores Práticas para Estruturar Projetos Express.js - Blog MOC Soluções
[4] Arquitetura assíncrona PubSub com Redis e Node.js - LuizTools
[16] Práticas recomendadas gerais | Cloud SQL for PostgreSQL - Google Cloud
[10] API Rest, Node e Typescript: Estrutura de pastas - YouTube
[5] Background jobs (filas) no Node.js com Redis - YouTube
[13] Dominando o Postgres: Da Criação do Banco de Dados - Kinsta
[6] Iniciando um projeto NodeJs, Express com Typescript - Dev.to
[15] Redis em Aplicações Node.js - Dev.to
[17] Otimizando o desempenho de consultas do PostgreSQL - AWS
[11] node-typescript-structure - GitHub
[7] Introdução Express/Node - MDN Web Docs
[14] Índices no PostgreSQL: estratégias para otimização - Alura
[8] Como criar uma WebApi com Node.js, Express e TypeScript - LuizTools
[9] Como criar uma WebApi com Node.js, Express e TypeScript - iMasters

[19] [20] [21]

⁂

1. requisitos.md
2. https://www.youtube.com/watch?v=GOISMrNzYwQ
3. https://blog.mocsolucoes.com.br/desenvolvimento-web/melhores-praticas-estrutura-projetos-expressjs/
4. https://www.luiztools.com.br/post/arquitetura-assincrona-pubsub-com-redis-nodejs/
5. https://www.youtube.com/watch?v=uonKHztGhko
6. https://dev.to/rogeriorioli/iniciando-um-projeto-nodejs-express-com-typescript-4bfl
7. https://developer.mozilla.org/pt-BR/docs/Learn_web_development/Extensions/Server-side/Express_Nodejs/Introduction
8. https://www.luiztools.com.br/post/como-criar-uma-webapi-com-node-js-express-e-typescript/
9. https://imasters.com.br/typescript/como-criar-uma-webapi-com-node-js-express-e-typescript
10. https://www.youtube.com/watch?v=0NCnwiXCks4
11. https://github.com/pedroksty/node-typescript-structure
12. https://www.devmedia.com.br/postgresql-tutorial/33025
13. https://kinsta.com/pt/blog/dominando-bancos-de-dados-postgres/
14. https://www.alura.com.br/artigos/indices-no-postgresql
15. https://dev.to/vitorrios1001/redis-em-aplicacoes-nodejs-31ed

16. https://cloud.google.com/sql/docs/postgres/best-practices?hl=pt-br
17. https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/postgresql-query-tuning/postgresql-query-tuning.pdf
18. https://translate.google.com/translate?u=https%3A%2F%2Fwww.tigerdata.com%2Flearn%2Fpostgresql-database-operations&hl=pt&sl=en&tl=pt&client=srp
19. https://translate.google.com/translate?u=https%3A%2F%2Fwww.freecodecamp.org%2Fnews%2Fhow-to-use-queues-in-web-applications%2F&hl=pt&sl=en&tl=pt&client=srp
20. https://www.reddit.com/r/node/comments/139w29v/how_do_you_handle_queues_in_nodejs_have_you_ever/
21. https://translate.google.com/translate?u=https%3A%2F%2Fwww.instaclustr.com%2Feducation%2Fpostgresql%2Fcomplete-guide-to-postgresql-features-use-cases-and-tutorial%2F&hl=pt&sl=en&tl=pt&client=srp