Manual Técnico e de Implantação - RH ZELLO (Módulo Infra & Dados)

Projeto: RH ZELLO - Backend Database
Versão: 1.0.0
Data: 05/02/2026
Responsável: Victor (Tech Lead IA)
Destinatário: Equipe de Backend (Kaua)

---

1. Resumo Executivo

Este módulo compõe a camada de persistência e infraestrutura de dados do sistema RH ZELLO. Ele fornece um ambiente PostgreSQL containerizado, modelagem de dados relacional otimizada para dashboards e um sistema de migração de esquema robusto.

A arquitetura foi desenhada para garantir:

1. **Segurança:** Criptografia de senhas e isolamento de credenciais.
2. **Performance:** Índices estratégicos para consultas de RH.
3. **Auditabilidade:** Rastreamento histórico de contratos.
4. **Portabilidade:** Ambiente Docker padronizado.

---

2. Stack Tecnológico

Componente	Tecnologia	Versão	Função
Banco de Dados	PostgreSQL	15 (Docker)	Persistência Relacional
Linguagem	Python	3.10+	Scripting e ORM
ORM	SQLAlchemy	2.x	Abstração de Banco de Dados
Migrações	Alembic	Latest	Versionamento de Schema
Segurança	Bcrypt + Passlib	4.0.1	Hashing de Senhas
Ambiente	Docker Compose	3.8	Orquestração de Containers

---

3. Infraestrutura (Docker)

O banco de dados roda isolado em container para evitar conflitos com o ambiente local.

Arquivo: `docker-compose.yml`

```yaml
version: '3.8'

services:
  db:
    image: postgres:15
    container_name: pg_rh_inteligente
    environment:
      POSTGRES_USER: admin_rh
      POSTGRES_PASSWORD: senha_segura_123
      POSTGRES_DB: db_rh
    ports:
      - "5433:5432"  # Mapeia porta 5433 do host para 5432 do container
    volumes:
      - pgdata_rh:/var/lib/postgresql/data

volumes:
  pgdata_rh:
```

Nota Operacional:

- A porta externa é **5433** (não a padrão 5432).
- Comando para iniciar: `docker-compose up -d`
- Comando para parar: `docker-compose down`

---

4. Dicionário de Dados (Modelagem)

O esquema do banco (`db_rh`) é composto por três tabelas principais definidas em `models.py` .

4.1. Tabela `usuarios` (Acesso)

Responsável pela autenticação do sistema.

Coluna	Tipo	Constraints	Descrição
`id`	Integer	PK, Index	Identificador único
`email`	String(255)	Unique, Not Null	Email de login

Coluna	Tipo	Constraints	Descrição
`senha_hash`	String(255)	Not Null	Hash Bcrypt da senha

4.2. Tabela `colaboradores` (Core RH)

Armazena os dados vivos dos funcionários. Otimizada para leitura.

Coluna	Tipo	Descrição
`id`	Integer (PK)	Identificador único
`nome`	String(255)	Nome completo
`sexo`	String(50)	Gênero
`idade`	Integer	Idade calculada
`regiao`	String(100)	Localização do posto de trabalho
`tipo_contrato`	String(50)	CLT, PJ, Temporário
`departamento`	String(100)	Setor de atuação
`data_atualizacao`	DateTime	Timestamp da última alteração

Índices de Performance:

- `idx_departamento`: Acelera filtros por departamento no Dashboard.
- `idx_data_atualizacao`: Acelera relatórios de "últimas alterações".

4.3. Tabela `historico_contratos` (Auditoria)

Registra todas as mudanças de tipo de contrato (ex: Estágio -> CLT).

Coluna	Tipo	Relacionamento
`id`	Integer (PK)	-
`colaborador_id`	Integer	FK -> `colaboradores.id`
`tipo_antigo`	String(50)	Valor anterior
`tipo_novo`	String(50)	Valor novo
`data_mudanca`	DateTime	Data do evento

---

5. Código Fonte de Referência

Para facilitar a integração, seguem os códigos completos dos módulos principais.

5.1. Conexão (`database.py`)

Gerencia a sessão e conexão segura via variáveis de ambiente.

```python
import os
from dotenv import load_dotenv
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker, declarative_base

load_dotenv()

# Usa porta 5433 conforme docker-compose
DATABASE_URL = os.getenv(
    "DATABASE_URL",
    "postgresql://admin_rh:senha_segura_123@localhost:5433/db_rh"
)

engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

5.2. Segurança (`security.py`)

Centraliza a lógica de criptografia.

```python
from passlib.context import CryptContext

pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

def get_password_hash(password: str) -> str:
    return pwd_context.hash(password)

def verify_password(plain_password: str, hashed_password: str) -> bool:
    return pwd_context.verify(plain_password, hashed_password)
```

---

6. Guia de Configuração e Uso

Passo 1: Configurar Variáveis

Crie um arquivo `.env` na raiz `backend_db/` :

```
DATABASE_URL=postgresql://admin_rh:senha_segura_123@localhost:5433/db_rh
```

Passo 2: Instalar Dependências

```
pip install -r requirements.txt
```

Passo 3: Subir o Banco

```
docker-compose up -d
```

Passo 4: Aplicar Migrações (Criar Tabelas)

O Alembic cria as tabelas baseadas no `models.py`.

```
alembic upgrade head
```

Passo 5: Criar Usuário Admin

Script interativo incluído no projeto.

```
python create_user.py
# Siga as instruções na tela para criar email/senha
```

---

7. Manutenção de Schema (Alembic)

Sempre que alterar o arquivo `models.py`, execute:

1. **Gerar Migração:**
   ```
   alembic revision --autogenerate -m "descricao_da_mudanca"
   ```
2. **Aplicar Migração:**
   ```
   alembic upgrade head
   ```

---

8. Check-list de Entrega

- [x] ~~Container Docker configurado (Porta 5433).~~

- [x] ~~Tabelas criadas e indexadas.~~
- [x] ~~Sistema de migração ativo.~~
- [x] ~~Script de criação de usuário funcional.~~
- [x] ~~Documentação técnica gerada.~~