**🚀 Guia de Configuração para Produção**

**Sistema de Agendamento - Salão de Beleza**

**📋 Checklist Geral**

* **1. Configurar Variáveis de Ambiente**
* **2. Preparar PostgreSQL para Produção**
* **3. Adicionar Dependências de Produção**
* **4. Configurar CI/CD (GitHub Actions)**
* **5. Integrar Sentry para Monitoramento**
* **6. Scripts de Backup do Banco**
* **7. Documentação Final**

**1️⃣ CONFIGURAR VARIÁVEIS DE AMBIENTE**

**📁 Criar arquivo .env.example na raiz do projeto**

env

# DATABASE

DATABASE\_URL=sqlite:///database.db

# SECURITY

SECRET\_KEY=sua-chave-secreta-super-segura-aqui-mude-isso

ALGORITHM=HS256

ACCESS\_TOKEN\_EXPIRE\_MINUTES=30

# API CONFIG

API\_HOST=0.0.0.0

API\_PORT=8000

CORS\_ORIGINS=http://localhost:5173,http://localhost:3000

# EMAIL (SendGrid)

SENDGRID\_API\_KEY=sua-chave-sendgrid-aqui

FROM\_EMAIL=noreply@seusalao.com

# ENVIRONMENT

ENVIRONMENT=development

DEBUG=True

**📁 Criar arquivo .env.production.example**

env

# DATABASE (PostgreSQL em produção)

DATABASE\_URL=postgresql://user:password@host:5432/database\_name

# SECURITY (GERAR NOVA CHAVE!)

SECRET\_KEY=GERAR\_NOVA\_CHAVE\_FORTE\_AQUI

ALGORITHM=HS256

ACCESS\_TOKEN\_EXPIRE\_MINUTES=1440

# API CONFIG

API\_HOST=0.0.0.0

API\_PORT=8000

CORS\_ORIGINS=https://seudominio.com,https://www.seudominio.com

# EMAIL

SENDGRID\_API\_KEY=sua-chave-sendgrid-producao

FROM\_EMAIL=contato@seusalao.com

# SENTRY (Monitoramento)

SENTRY\_DSN=sua-dsn-do-sentry-aqui

# ENVIRONMENT

ENVIRONMENT=production

DEBUG=False

# BACKUP

BACKUP\_ENABLED=True

BACKUP\_FREQUENCY=daily

BACKUP\_RETENTION\_DAYS=30

**🔑 Como gerar SECRET\_KEY forte**

Execute no terminal Python:

python

import secrets

print(secrets.token\_urlsafe(32))

**⚠️ Importante!**

* Adicione .env e .env.production no .gitignore
* Nunca commite suas chaves no Git!

**2️⃣ PREPARAR POSTGRESQL PARA PRODUÇÃO**

**📦 Adicionar dependência PostgreSQL**

No requirements.txt, adicione:

txt

psycopg2-binary==2.9.9

**🔧 Atualizar database.py**

python

import os

from sqlmodel import Session, create\_engine, SQLModel

from sqlalchemy.pool import QueuePool

*# Pegar URL do banco de variável de ambiente*

DATABASE\_URL = os.getenv("DATABASE\_URL", "sqlite:///database.db")

*# Configurações diferentes para SQLite vs PostgreSQL*

if DATABASE\_URL.startswith("sqlite"):

*# SQLite (desenvolvimento)*

engine = create\_engine(

DATABASE\_URL,

echo=True,

connect\_args={"check\_same\_thread": False}

)

else:

*# PostgreSQL (produção)*

engine = create\_engine(

DATABASE\_URL,

echo=False, *# Desabilitar logs em produção*

pool\_pre\_ping=True, *# Verificar conexão antes de usar*

poolclass=QueuePool,

pool\_size=5,

max\_overflow=10

)

def get\_session():

with Session(engine) as session:

yield session

def init\_db():

"""Inicializar banco de dados"""

SQLModel.metadata.create\_all(engine)

**🗄️ PostgreSQL Local (para testes)**

**Opção 1: Docker (Recomendado)**

Crie docker-compose.yml na raiz:

yaml

version: '3.8'

services:

postgres:

image: postgres:15-alpine

container\_name: salao\_postgres

environment:

POSTGRES\_USER: salao\_user

POSTGRES\_PASSWORD: salao\_pass

POSTGRES\_DB: salao\_db

ports:

- "5432:5432"

volumes:

- postgres\_data:/var/lib/postgresql/data

restart: unless-stopped

volumes:

postgres\_data:

**Comandos:**

bash

*# Subir PostgreSQL*

docker-compose up -d

*# Parar PostgreSQL*

docker-compose down

*# Ver logs*

docker-compose logs -f

**Opção 2: PostgreSQL Instalado Localmente**

* Windows: Baixar do [postgresql.org](https://www.postgresql.org/download/)
* Linux: sudo apt install postgresql postgresql-contrib
* Mac: brew install postgresql

**3️⃣ ADICIONAR DEPENDÊNCIAS DE PRODUÇÃO**

**📝 Atualizar requirements.txt**

txt

# Backend Core

fastapi==0.104.1

sqlmodel==0.0.14

uvicorn[standard]==0.24.0

python-jose[cryptography]==3.3.0

passlib[bcrypt]==1.7.4

python-multipart==0.0.6

python-dotenv==1.0.0

# Database

psycopg2-binary==2.9.9 # PostgreSQL

# Email & Jobs

sendgrid==6.11.0

apscheduler==3.10.4

# Reports

pandas==2.1.3

matplotlib==3.8.2

reportlab==4.0.6

# HTTP

requests==2.31.0

# Monitoring (Produção)

sentry-sdk[fastapi]==1.39.1

# Production Server

gunicorn==21.2.0

**📦 Instalar novas dependências**

bash

cd backend

pip install -r requirements.txt

**4️⃣ CONFIGURAR CI/CD (GITHUB ACTIONS)**

**📁 Criar .github/workflows/tests.yml**

yaml

name: Testes Backend

on:

push:

branches: [ main, develop ]

pull\_request:

branches: [ main, develop ]

jobs:

test:

runs-on: ubuntu-latest

services:

postgres:

image: postgres:15-alpine

env:

POSTGRES\_USER: test\_user

POSTGRES\_PASSWORD: test\_pass

POSTGRES\_DB: test\_db

ports:

- 5432:5432

options: >-

--health-cmd pg\_isready

--health-interval 10s

--health-timeout 5s

--health-retries 5

steps:

- uses: actions/checkout@v3

- name: Configurar Python

uses: actions/setup-python@v4

with:

python-version: '3.11'

- name: Instalar dependências

run: |

cd backend

pip install -r requirements.txt

pip install pytest pytest-cov

- name: Rodar testes

env:

DATABASE\_URL: postgresql://test\_user:test\_pass@localhost:5432/test\_db

SECRET\_KEY: test-secret-key-for-ci

run: |

cd backend

pytest tests/ -v --cov=app --cov-report=xml

- name: Upload coverage

uses: codecov/codecov-action@v3

with:

file: ./backend/coverage.xml

flags: backend

**📁 Criar .github/workflows/deploy.yml (Para quando hospedar)**

yaml

name: Deploy para Produção

on:

push:

branches: [ main ]

workflow\_dispatch:

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Deploy Backend

run: |

echo "🚀 Deploy será configurado quando escolher plataforma"

# Aqui vão os comandos específicos da plataforma escolhida

# Railway, Render, AWS, etc.

- name: Deploy Frontend

run: |

echo "🚀 Deploy frontend será configurado"

# Vercel, Netlify, etc.

**5️⃣ INTEGRAR SENTRY (MONITORAMENTO)**

**🔧 Atualizar main.py**

Adicione no início do arquivo:

python

import os

import sentry\_sdk

from sentry\_sdk.integrations.fastapi import FastApiIntegration

*# Configurar Sentry apenas em produção*

if os.getenv("ENVIRONMENT") == "production":

sentry\_sdk.init(

dsn=os.getenv("SENTRY\_DSN"),

integrations=[FastApiIntegration()],

traces\_sample\_rate=1.0,

environment=os.getenv("ENVIRONMENT", "development"),

)

**📝 Como obter Sentry DSN (GRÁTIS!)**

1. Acesse [sentry.io](https://sentry.io)
2. Crie uma conta gratuita
3. Crie um novo projeto Python/FastAPI
4. Copie o DSN fornecido
5. Adicione no .env.production

**✅ Benefícios do Sentry**

* ✅ Rastreamento automático de erros
* ✅ Alertas em tempo real
* ✅ Stack traces detalhados
* ✅ Monitoramento de performance
* ✅ Grátis até 5.000 eventos/mês

**6️⃣ SCRIPTS DE BACKUP DO BANCO**

**📁 Criar backend/scripts/backup\_database.py**

python

import os

import subprocess

from datetime import datetime

from pathlib import Path

def backup\_postgres():

"""Fazer backup do PostgreSQL"""

*# Configurações*

backup\_dir = Path("backups")

backup\_dir.mkdir(exist\_ok=True)

timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

backup\_file = backup\_dir / f"backup\_{timestamp}.sql"

*# Pegar credenciais do .env*

db\_url = os.getenv("DATABASE\_URL")

if not db\_url or db\_url.startswith("sqlite"):

print("❌ PostgreSQL não configurado")

return

print(f"📦 Criando backup: {backup\_file}")

try:

*# Comando pg\_dump*

subprocess.run([

"pg\_dump",

db\_url,

"-f", str(backup\_file)

], check=True)

print(f"✅ Backup criado com sucesso!")

*# Limpar backups antigos (manter últimos 30)*

cleanup\_old\_backups(backup\_dir, keep=30)

except Exception as e:

print(f"❌ Erro ao criar backup: {e}")

def cleanup\_old\_backups(backup\_dir: Path, keep: int = 30):

"""Remover backups antigos"""

backups = sorted(backup\_dir.glob("backup\_\*.sql"))

if len(backups) > keep:

to\_remove = backups[:-keep]

for backup in to\_remove:

backup.unlink()

print(f"🗑️ Removido: {backup.name}")

if \_\_name\_\_ == "\_\_main\_\_":

backup\_postgres()

**📁 Criar backend/scripts/restore\_database.py**

python

import os

import subprocess

from pathlib import Path

def restore\_postgres(backup\_file: str):

"""Restaurar backup do PostgreSQL"""

backup\_path = Path(backup\_file)

if not backup\_path.exists():

print(f"❌ Arquivo não encontrado: {backup\_file}")

return

db\_url = os.getenv("DATABASE\_URL")

if not db\_url or db\_url.startswith("sqlite"):

print("❌ PostgreSQL não configurado")

return

print(f"📥 Restaurando backup: {backup\_file}")

try:

subprocess.run([

"psql",

db\_url,

"-f", str(backup\_path)

], check=True)

print("✅ Backup restaurado com sucesso!")

except Exception as e:

print(f"❌ Erro ao restaurar: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) < 2:

print("Uso: python restore\_database.py <arquivo\_backup>")

else:

restore\_postgres(sys.argv[1])

**⏰ Automatizar Backups (Cron)**

**Linux/Mac - Adicionar no crontab:**

bash

*# Editar crontab*

crontab -e

*# Backup diário às 3h da manhã*

0 3 \* \* \* cd /caminho/do/projeto/backend && python scripts/backup\_database.py

**Windows - Task Scheduler:**

1. Abrir "Agendador de Tarefas"
2. Criar Tarefa Básica
3. Agendar: Diariamente às 3h
4. Ação: python scripts/backup\_database.py

**7️⃣ DOCUMENTAÇÃO FINAL**

**📁 Criar DEPLOYMENT.md**

markdown

# 🚀 Guia de Deploy - Sistema de Agendamento

## Pré-requisitos

- Python 3.11+

- Node.js 18+

- PostgreSQL 15+

## Backend

### Configuração Local

1. Clone o repositório

2. Crie ambiente virtual: `python -m venv venv`

3. Ative: `source venv/bin/activate` (Linux/Mac) ou `venv\Scripts\activate` (Windows)

4. Instale dependências: `pip install -r requirements.txt`

5. Configure `.env` baseado no `.env.example`

6. Rode migrações: `python -m app.database`

7. Inicie servidor: `uvicorn app.main:app --reload`

### Deploy Produção

1. Configure `.env.production` com credenciais reais

2. Configure PostgreSQL em produção

3. Configure Sentry DSN

4. Rode com Gunicorn: `gunicorn app.main:app -w 4 -k uvicorn.workers.UvicornWorker`

## Frontend

### Configuração Local

1. Entre na pasta: `cd frontend`

2. Instale dependências: `npm install`

3. Configure variáveis de ambiente

4. Rode dev server: `npm run dev`

### Deploy Produção

1. Build: `npm run build`

2. Deploy pasta `dist/` no servidor

## Backups

- Automático: Configurar cron job

- Manual: `python scripts/backup\_database.py`

- Restaurar: `python scripts/restore\_database.py backup\_YYYYMMDD\_HHMMSS.sql`

## Monitoramento

- Sentry: Acesse dashboard em sentry.io

- Logs: `tail -f logs/app.log`

**🎯 PRÓXIMOS PASSOS**

**Agora (Desenvolvimento):**

1. ✅ Criar arquivos .env.example e .env.production.example
2. ✅ Testar PostgreSQL localmente (com Docker)
3. ✅ Adicionar Sentry (conta grátis)
4. ✅ Criar scripts de backup
5. ✅ Configurar GitHub Actions

**Quando for Deploy:**

1. 🔜 Escolher plataformas de hospedagem
2. 🔜 Configurar domínio personalizado
3. 🔜 Configurar HTTPS/SSL
4. 🔜 Configurar banco PostgreSQL em nuvem
5. 🔜 Ativar monitoramento 24/7

**📚 Recursos Úteis**

* **Railway** (Backend): railway.app - Fácil deploy, plano grátis
* **Vercel** (Frontend): vercel.com - Deploy automático do GitHub
* **Sentry** (Monitoramento): sentry.io - 5k eventos grátis/mês
* **Neon** (PostgreSQL): neon.tech - PostgreSQL serverless grátis
* **Supabase** (PostgreSQL): supabase.com - PostgreSQL + Auth grátis

**⚠️ Segurança**

* Nunca commitar .env no Git
* Usar chaves fortes (32+ caracteres)
* Atualizar dependências regularmente
* Revisar logs do Sentry semanalmente
* Testar backups mensalmente
* Configurar rate limiting na API
* Usar HTTPS em produção

**🆘 Problemas Comuns**

**Erro de conexão PostgreSQL:**

* Verificar se o serviço está rodando
* Conferir credenciais no .env
* Testar conexão: psql -h host -U user -d database

**Build falha no CI:**

* Verificar versão do Python
* Conferir dependências no requirements.txt
* Ver logs detalhados no GitHub Actions

**Sentry não captura erros:**

* Verificar se DSN está correto
* Confirmar se ENVIRONMENT=production
* Testar forçando erro intencional

**✅ Checklist Final Antes do Deploy**

* Todos os testes passando
* .env.production configurado
* PostgreSQL em produção configurado
* Sentry DSN válido
* Backups automatizados testados
* CI/CD funcionando
* Documentação atualizada
* Secrets configurados no GitHub
* CORS configurado corretamente
* Rate limiting ativado

**🎉 Parabéns! Seu projeto está preparado para produção!**