

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
-

1 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!

2 PREPARAR POSTGRESQL PARA PRODUÇÃO

Adicionar dependência PostgreSQL

No `requirements.txt`, adicione:

```
txt  
  
psycopg2-binary==2.9.9
```

Atualizar `database.py`

```
python
```

```

import os
from sqlalchemy 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)
- Linux: `sudo apt install postgresql postgresql-contrib`
- Mac: `brew install postgresql`

3 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
```

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.
```

5 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
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

6 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`

7 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!

Criado para o Sistema de Agendamento - Salão de Beleza Última atualização: 2025