🔧 Especificações Técnicas - Sistema SaaS de Agendamento Online

📋 Visão Geral do Sistema

Arquitetura

- **Padrão**: Microserviços containerizados
- **Proxy Reverso**: NGINX
- **Containerização**: Docker + Docker Compose
- **Orquestração**: Docker Compose (produção simples)

Stack Tecnológica

Backend

- **Framework**: Flask 2.3+
- **Linguagem**: Python 3.11+
- **ORM**: SQLAlchemy 2.0+
- **Autenticação**: JWT (JSON Web Tokens)
- **Validação**: Marshmallow
- **Cache**: Redis 7.0+
- **Task Queue**: Celery (opcional)

Frontend

- **Framework**: React 18+
- **Build Tool**: Vite
- **Linguagem**: JavaScript/JSX

- **Styling**: TailwindCSS 3.0+
- **Componentes**: shadcn/ui
- **Roteamento**: React Router 6+
- **Estado**: Context API + useState/useEffect
- **HTTP Client**: Axios

Banco de Dados

- **Principal**: PostgreSQL 15+
- **Cache**: Redis 7.0+
- **Backup**: pg_dump + compressão gzip

Infraestrutura

- **Servidor Web**: NGINX 1.20+
- **SSL**: Let's Encrypt (Certbot)
- **Firewall**: UFW (Ubuntu Firewall)
- **Monitoramento**: Docker Health Checks
- **Logs**: Rotação automática com logrotate

🗄️ Estrutura do Banco de Dados

Tabelas Principais

users

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    phone VARCHAR(20),
    role VARCHAR(20) DEFAULT 'client',
    avatar_url VARCHAR(500),
    is_active BOOLEAN DEFAULT true,
    email_verified BOOLEAN DEFAULT false,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

providers

```sql
CREATE TABLE providers (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    business_name VARCHAR(255) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    address TEXT,
    city VARCHAR(100),
    state VARCHAR(50),
    postal_code VARCHAR(20),
    latitude DECIMAL(10, 8),
    longitude DECIMAL(11, 8),
    phone VARCHAR(20),
    website VARCHAR(255),
    instagram VARCHAR(100),
    facebook VARCHAR(100),
    is_verified BOOLEAN DEFAULT false,
    plan VARCHAR(20) DEFAULT 'free',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

categories

```sql
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    slug VARCHAR(100) UNIQUE NOT NULL,
    description TEXT,
    icon VARCHAR(50),
    color VARCHAR(7),
    is_active BOOLEAN DEFAULT true,
    sort_order INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

services

```sql
CREATE TABLE services (
    id SERIAL PRIMARY KEY,
    provider_id INTEGER REFERENCES providers(id) ON DELETE CASCADE,
    category_id INTEGER REFERENCES categories(id),
    name VARCHAR(255) NOT NULL,
    description TEXT,
    duration INTEGER NOT NULL, -- em minutos
    price DECIMAL(10, 2),
    is_active BOOLEAN DEFAULT true,
    sort_order INTEGER DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

appointments

```sql
CREATE TABLE appointments (
    id SERIAL PRIMARY KEY,
    client_id INTEGER REFERENCES users(id),
    provider_id INTEGER REFERENCES providers(id),
    service_id INTEGER REFERENCES services(id),
    appointment_date DATE NOT NULL,
    appointment_time TIME NOT NULL,
    duration INTEGER NOT NULL,
    status VARCHAR(20) DEFAULT 'pending',
    client_name VARCHAR(255),
    client_email VARCHAR(255),
    client_phone VARCHAR(20),
    client_notes TEXT,
    provider_notes TEXT,
    total_price DECIMAL(10, 2),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Índices Importantes

```sql
-- Índices para performance
CREATE INDEX idx_appointments_provider_date ON appointments(provider_id,
appointment_date);
CREATE INDEX idx_appointments_client ON appointments(client_id);
CREATE INDEX idx_appointments_status ON appointments(status);
CREATE INDEX idx_services_provider ON services(provider_id);
CREATE INDEX idx_providers_slug ON providers(slug);
CREATE INDEX idx_users_email ON users(email);
```

🔌 API Endpoints

Autenticação

```
POST   /api/auth/register     - Registrar usuário
POST   /api/auth/login        - Login
POST   /api/auth/logout       - Logout
POST   /api/auth/refresh      - Renovar token
GET    /api/auth/me           - Dados do usuário logado
POST   /api/auth/forgot       - Esqueci senha
POST   /api/auth/reset        - Resetar senha
```

Usuários

```
GET    /api/users/profile     - Perfil do usuário
PUT    /api/users/profile     - Atualizar perfil
GET    /api/users/appointments - Agendamentos do usuário
POST   /api/users/deactivate  - Desativar conta
```

Prestadores

```
GET    /api/providers         - Listar prestadores
GET    /api/providers/:slug   - Dados do prestador
GET    /api/providers/profile - Perfil do prestador logado
PUT    /api/providers/profile - Atualizar perfil
GET    /api/providers/dashboard - Dashboard do prestador
GET    /api/providers/available-times - Horários disponíveis
```

Serviços

```
GET    /api/services/provider/:id - Serviços do prestador
POST   /api/services          - Criar serviço
PUT    /api/services/:id      - Atualizar serviço
DELETE /api/services/:id      - Deletar serviço
GET    /api/services/my-services - Meus serviços
POST   /api/services/reorder  - Reordenar serviços
```

Agendamentos

```
POST   /api/appointments      - Criar agendamento
GET    /api/appointments/:id  - Dados do agendamento
PUT    /api/appointments/:id/status - Atualizar status
GET    /api/appointments/provider/:id - Agendamentos do prestador
DELETE /api/appointments/:id  - Cancelar agendamento
GET    /api/appointments/calendar/:id - Calendário
```

Categorias

```
GET    /api/categories        - Listar categorias
POST   /api/categories        - Criar categoria (admin)
PUT    /api/categories/:id    - Atualizar categoria (admin)
DELETE /api/categories/:id    - Deletar categoria (admin)
```

🔐 Segurança

Autenticação JWT

```
// Estrutura do Token
{
  "user_id": 123,
  "email": "user@example.com",
  "role": "provider",
  "exp": 1640995200,
  "iat": 1640908800
}
```

Rate Limiting

- **API Geral**: 100 requests/minuto por IP
- **Login**: 5 tentativas/minuto por IP
- **Registro**: 3 registros/hora por IP

Validação de Dados

```
# Exemplo de schema Marshmallow
class UserSchema(Schema):
    name = fields.Str(required=True, validate=Length(min=2, max=255))
    email = fields.Email(required=True)
    password = fields.Str(required=True, validate=Length(min=8))
    phone = fields.Str(validate=Length(max=20))
```

Headers de Segurança

```
 add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header X-Content-Type-Options "nosniff" always;
add_header Referrer-Policy "no-referrer-when-downgrade" always;
add_header Strict-Transport-Security "max-age=63072000" always;
```

🐳 Configuração Docker

docker-compose.yml

```yaml
version: '3.8'
services:
  postgres:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: saas_agendamentos
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 30s
      timeout: 10s
      retries: 3

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass ${REDIS_PASSWORD}
    healthcheck:
      test: ["CMD", "redis-cli", "ping"]
      interval: 30s
      timeout: 10s
      retries: 3

  backend:
    build: ./backend
    environment:
      DATABASE_URL:
postgresql://postgres:${DB_PASSWORD}@postgres:5432/saas_agendamentos
      REDIS_URL: redis://:${REDIS_PASSWORD}@redis:6379
    depends_on:
      postgres:
        condition: service_healthy
      redis:
        condition: service_healthy

  frontend:
    build: ./frontend
    depends_on:
      - backend

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
      - /etc/letsencrypt:/etc/letsencrypt
    depends_on:
      - frontend
      - backend
```

📊 Monitoramento e Logs

Health Checks

```
# Verificar saúde dos containers
docker-compose ps

# Verificar logs
docker-compose logs -f backend
docker-compose logs -f frontend

# Verificar recursos
docker stats
```

Estrutura de Logs

```
backend/logs/
├── app.log          # Logs da aplicação
├── error.log        # Logs de erro
├── access.log       # Logs de acesso
└── debug.log        # Logs de debug (dev only)
```

Métricas Importantes

- **Uptime**: Tempo de atividade do sistema
- **Response Time**: Tempo de resposta das APIs
- **Error Rate**: Taxa de erros por endpoint
- **Database Connections**: Conexões ativas no banco
- **Memory Usage**: Uso de memória por container
- **Disk Usage**: Espaço em disco utilizado

🔧 Configurações de Performance

NGINX

```
 # Configurações de performance
worker_processes auto;
worker_connections 1024;

# Gzip compression
gzip on;
gzip_vary on;
gzip_min_length 1024;
gzip_comp_level 6;

# Cache de arquivos estáticos
location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
    expires 1y;
    add_header Cache-Control "public, immutable";
}
```

PostgreSQL

```
 -- Configurações recomendadas para PostgreSQL
shared_buffers = 256MB
effective_cache_size = 1GB
maintenance_work_mem = 64MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
```

Redis

```
 # Configurações de memória
maxmemory 256mb
maxmemory-policy allkeys-lru

# Persistência
save 900 1
save 300 10
save 60 10000
```

📱 Responsividade

Breakpoints TailwindCSS

```
/* Mobile first approach */
sm: 640px   /* Tablet pequeno */
md: 768px   /* Tablet */
lg: 1024px  /* Desktop pequeno */
xl: 1280px  /* Desktop */
2xl: 1536px /* Desktop grande */
```

Componentes Responsivos

```
// Exemplo de componente responsivo
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">
  {/* Cards responsivos */}
</div>
```

🔄 Backup e Restore

Estratégia de Backup

1. **Banco de Dados**: Dump diário às 2h
2. **Uploads**: Backup semanal
3. **Configurações**: Backup mensal
4. **Retenção**: 30 dias local, 90 dias nuvem

Scripts Automatizados

```
# Backup automático
./scripts/backup.sh

# Restore específico
./scripts/restore.sh db_backup_20240101_120000.sql.gz

# Listar backups
./scripts/restore.sh --list
```

🚀 Deploy e CI/CD

Deploy Manual

```
# Deploy completo
sudo ./scripts/deploy.sh --domain seusite.com --ssl

# Atualização
sudo ./scripts/deploy.sh --update
```

Deploy Automatizado (GitHub Actions)

```
name: Deploy to Production
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Deploy to server
        run: |
          ssh user@server 'cd /path/to/project && git pull &&
./scripts/deploy.sh --update'
```

📈 Escalabilidade

Horizontal Scaling

- **Load Balancer**: NGINX upstream
- **Database**: Read replicas
- **Cache**: Redis Cluster
- **Storage**: Shared filesystem (NFS/S3)

Vertical Scaling

- **CPU**: Aumentar cores do servidor
- **RAM**: Aumentar memória disponível
- **Storage**: SSD NVMe para melhor I/O

Otimizações

```
# Conexão pool do banco
SQLALCHEMY_ENGINE_OPTIONS = {
    'pool_size': 20,
    'pool_recycle': 3600,
    'pool_pre_ping': True
}

# Cache de queries
@cache.memoize(timeout=300)
def get_providers_by_category(category_id):
    return Provider.query.filter_by(category_id=category_id).all()
```

🔍 Debugging

Logs de Debug

```
# Configuração de logging
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('logs/app.log'),
        logging.StreamHandler()
    ]
)
```

Ferramentas Úteis

- **Flask Debug Toolbar**: Debug em desenvolvimento
- **PostgreSQL Logs**: Análise de queries lentas
- **Redis CLI**: Monitoramento de cache
- **Docker Logs**: Logs dos containers

📋 Checklist de Produção

Antes do Deploy

- [ ] Configurar variáveis de ambiente

- [ ] Gerar senhas seguras
- [ ] Configurar DNS
- [ ] Testar backup/restore
- [ ] Configurar monitoramento

Após o Deploy

- [ ] Verificar SSL
- [ ] Testar todas as funcionalidades
- [ ] Configurar alertas
- [ ] Documentar credenciais
- [ ] Treinar usuários

Manutenção Regular

- [ ] Backup semanal manual
- [ ] Atualização de segurança mensal
- [ ] Limpeza de logs
- [ ] Monitoramento de recursos
- [ ] Análise de performance

📊 Sistema otimizado para alta performance e escalabilidade

Para dúvidas técnicas específicas, consulte a documentação do código ou entre em contato com a equipe de desenvolvimento.