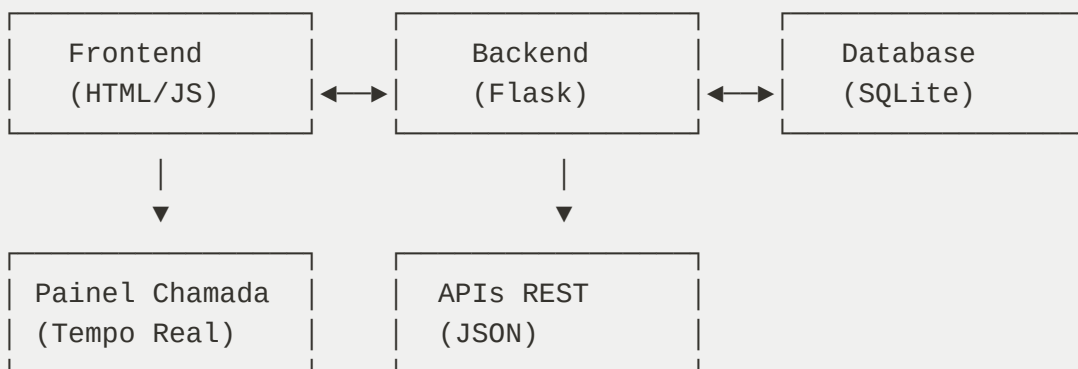


# Especificações Técnicas - Sistema de Agendamentos

## Arquitetura do Sistema

### Visão Geral

Plain Text



## Stack Tecnológico

### Backend

- **Framework:** Flask 2.3.3
- **Linguagem:** Python 3.11+
- **ORM:** SQLAlchemy 2.0
- **CORS:** Flask-CORS 4.0
- **Banco:** SQLite 3
- **Autenticação:** Flask Sessions
- **Deploy:** Manus Cloud Platform

### Frontend

- **Tecnologias:** HTML5, CSS3, JavaScript ES6+
- **Arquitetura:** SPA (Single Page Application)
- **Comunicação:** Fetch API
- **Áudio:** Web Audio API + Speech Synthesis API
- **Responsividade:** CSS Grid + Flexbox
- **Compatibilidade:** Chrome 90+, Firefox 88+, Safari 14+, Edge 90+

## Infraestrutura

- **Hospedagem:** Manus Cloud
- **SSL:** Certificado automático
- **CDN:** Distribuição global
- **Backup:** Automático
- **Monitoramento:** 24/7

---

## Estrutura de Arquivos

Plain Text

```
sistema_agendamentos_novo/  
├── src/  
│   ├── main.py           # Servidor Flask principal  
│   ├── static/           # Arquivos estáticos  
│   │   ├── index.html    # Interface principal SPA  
│   │   └── painel.html    # Painel de chamada  
│   ├── models/           # Modelos de dados  
│   │   └── user.py        # Modelo de usuário  
│   ├── database/         # Banco de dados  
│   │   └── app.db         # SQLite database  
│   └── templates/        # Templates (se necessário)  
└── requirements.txt       # Dependências Python
```

└─ README.md  
└─ .gitignore

# Documentação básica  
# Arquivos ignorados

## Modelo de Dados

### Usuários (Backend - SQLAlchemy)

Python

```
class User(db.Model):
    __tablename__ = 'users'

    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(255), nullable=False)
    perfil = db.Column(db.Enum(PerfilEnum), nullable=False)
    ativo = db.Column(db.Boolean, default=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)

class PerfilEnum(enum.Enum):
    ASSISTENTE = "ASSISTENTE"
    PROFISSIONAL = "PROFISSIONAL"
```

### Clientes (Frontend - JavaScript)

JavaScript

```
const Cliente = {
  id: Number,           // ID único
  nome: String,         // Nome completo (obrigatório)
  cpf: String,          // CPF formatado (obrigatório)
  telefone: String,     // Telefone formatado (obrigatório)
  email: String,        // Email (opcional)
  dataNascimento: String, // Data no formato YYYY-MM-DD (opcional)
  endereco: String,     // Endereço completo (opcional)
  observacoes: String,  // Observações gerais (opcional)
  createdAt: String,    // Data de criação ISO
```

```
    updatedAt: String          // Data de atualização ISO
};
```

## 17 Agendamentos (Frontend - JavaScript)

### JavaScript

```
const Agendamento = {
  id: Number,          // ID único
  clienteId: Number,   // ID do cliente
  clienteNome: String, // Nome do cliente (cache)
  data: String,        // Data no formato YYYY-MM-DD
  horario: String,     // Horário no formato HH:MM
  status: StatusEnum,  // Status atual
  observacoes: String, // Observações do agendamento
  prioridade: Boolean, // Se tem prioridade na fila
  chamadoEm: String,   // Timestamp da chamada
  createdAt: String,   // Data de criação
  updatedAt: String    // Data de atualização
};

const StatusEnum = {
  AGENDADO: "AGENDADO", // Cliente agendou
  PRESENTE: "PRESENTE", // Fez check-in
  EM_ATENDIMENTO: "EM_ATENDIMENTO", // Sendo atendido
  ATENDIDO: "ATENDIDO"  // Atendimento finalizado
};
```

## Estado Global (Backend - Python)

### Python

```
# Estado global para gerenciar chamadas
estado_chamada = {
  'cliente_atual': {
    'nome': str,          # Nome do cliente chamado
    'cpf': str,           # CPF formatado
    'horario': str,       # Horário do agendamento
    'prioridade': bool    # Se tem prioridade
  },
  'timestamp_chamada': str # ISO timestamp da chamada
}
```

# APIs e Endpoints

## Autenticação

### POST /api/login

Autentica usuário no sistema.

#### Headers:

Plain Text

Content-Type: application/json

#### Request Body:

JSON

```
{
  "email": "string (required)",
  "password": "string (required)"
}
```

#### Response (200):

JSON

```
{
  "success": true,
  "message": "Login realizado com sucesso",
  "user": {
    "email": "string",
    "perfil": "ASSISTENTE|PROFISSIONAL"
  }
}
```

#### Response (401):

JSON

```
{
  "success": false,
  "message": "Credenciais inválidas"
}
```

## POST /api/logout

Encerra sessão do usuário.

### Response (200):

JSON

```
{
  "success": true,
  "message": "Logout realizado com sucesso"
}
```

## Gestão de Clientes

### GET /api/clientes

Lista todos os clientes cadastrados.

### Query Parameters:

- `search` (optional): Busca por nome ou CPF
- `limit` (optional): Limite de resultados (default: 100)
- `offset` (optional): Offset para paginação (default: 0)

### Response (200):

JSON

```
{
  "success": true,
  "clientes": [
    {
      "id": 1,
      "nome": "João Silva",

```

```
        "cpf": "123.456.789-00",
        "telefone": "(11) 99999-9999",
        "email": "joao@email.com",
        "dataNascimento": "1990-01-01",
        "endereco": "Rua A, 123",
        "observacoes": "Cliente VIP"
    },
    "total": 1
}
```

## POST /api/clientes

Cria novo cliente.

### Request Body:

JSON

```
{
  "nome": "string (required, max 100)",
  "cpf": "string (required, format: 000.000.000-00)",
  "telefone": "string (required, format: (00) 00000-0000)",
  "email": "string (optional, valid email)",
  "dataNascimento": "string (optional, format: YYYY-MM-DD)",
  "endereco": "string (optional, max 200)",
  "observacoes": "string (optional, max 500)"
}
```

### Response (201):

JSON

```
{
  "success": true,
  "message": "Cliente criado com sucesso",
  "cliente": {
    "id": 1,
    "nome": "João Silva",
    // ... outros campos
  }
}
```

## PUT /api/clientes/{id}

Atualiza cliente existente.

#### Path Parameters:

- `id` : ID do cliente (integer)

**Request Body:** Mesmo formato do POST

#### Response (200):

JSON

```
{
  "success": true,
  "message": "Cliente atualizado com sucesso",
  "cliente": { /* dados atualizados */ }
}
```

#### DELETE /api/clientes/{id}

Remove cliente.

#### Response (200):

JSON

```
{
  "success": true,
  "message": "Cliente removido com sucesso"
}
```

## Gestão de Agendamentos

#### GET /api/agendamentos

Lista agendamentos com filtros.

#### Query Parameters:

- `data` (optional): Filtrar por data (YYYY-MM-DD)
- `status` (optional): Filtrar por status



- `cliente_id` (optional): Filtrar por cliente

### Response (200):

JSON

```
{
  "success": true,
  "agendamentos": [
    {
      "id": 1,
      "cliente": {
        "id": 1,
        "nome": "João Silva",
        "cpf": "123.456.789-00",
        "telefone": "(11) 99999-9999"
      },
      "data": "2025-07-13",
      "horario": "09:00",
      "status": "AGENDADO",
      "observacoes": "Primeira consulta",
      "prioridade": false
    }
  ],
  "estatisticas": {
    "total_agendados": 5,
    "presentes": 2,
    "em_atendimento": 1,
    "atendidos": 2
  }
}
```

### POST /api/agendamentos

Cria novo agendamento.

### Request Body:

JSON

```
{
  "cliente_id": "number (required)",
  "data": "string (required, format: YYYY-MM-DD, not past)",
  "horario": "string (required, format: HH:MM)",
}
```

```
"observacoes": "string (optional, max 500)"
}
```

## Fila de Atendimento

### GET /api/fila

Obtém estado atual da fila.

#### Response (200):

JSON

```
{
  "success": true,
  "fila": [
    {
      "id": 1,
      "cliente": {
        "id": 1,
        "nome": "João Silva",
        "cpf": "123.456.789-00"
      },
      "horario": "09:00",
      "status": "PRESENTE",
      "posicao": 1,
      "prioridade": false,
      "tempo_espera": "00:15:30"
    }
  ],
  "em_atendimento": {
    "id": 2,
    "cliente": {
      "nome": "Maria Santos",
      "cpf": "987.654.321-00"
    },
    "horario": "10:00",
    "chamado_em": "15:30:45"
  },
  "estatisticas": {
    "total_agendados": 5,
    "presentes": 3,
    "em_atendimento": 1,
    "atendidos": 1,
    "tempo_medio_espera": "00:12:30"
  }
}
```

```
}  
}
```

## POST /api/fila/chamar-proximo

Chama próximo cliente da fila.

### Request Body:

JSON

```
{  
  "cliente": {  
    "nome": "string (required)",  
    "cpf": "string (required)",  
    "horario": "string (required)",  
    "prioridade": "boolean (optional, default: false)"  
  }  
}
```

### Response (200):

JSON

```
{  
  "success": true,  
  "message": "Próximo cliente chamado",  
  "cliente_chamado": {  
    "nome": "João Silva",  
    "cpf": "123.456.789-00",  
    "horario": "09:00",  
    "prioridade": false  
  },  
  "timestamp": "2025-07-13T15:02:57.211508"  
}
```



## Painel de Chamada

## GET /api/painel/status

Obtém status atual do painel para exibição.

## Response (200):

JSON

```
{
  "success": true,
  "cliente_atual": {
    "nome": "João Silva",
    "cpf": "123.456.789-00",
    "horario": "09:00",
    "prioridade": false
  },
  "timestamp_chamada": "2025-07-13T15:02:57.211508",
  "proximos": [
    {
      "nome": "Maria Santos",
      "horario": "10:00",
      "posicao": 1,
      "prioridade": false
    },
    {
      "nome": "Pedro Costa",
      "horario": "11:00",
      "posicao": 2,
      "prioridade": true
    }
  ],
  "estatisticas": {
    "total_agendados": 5,
    "presentes": 2,
    "em_atendimento": 1,
    "atendidos": 2
  }
}
```

## Frontend - Especificações

### Responsividade

#### Breakpoints

CSS

```

/* Mobile First Approach */
/* Mobile: 320px - 767px */
@media (max-width: 767px) {
    .sidebar { transform: translateX(-100%); }
    .content { margin-left: 0; }
    .table-responsive { overflow-x: auto; }
}

/* Tablet: 768px - 1199px */
@media (min-width: 768px) and (max-width: 1199px) {
    .sidebar { width: 200px; }
    .content { margin-left: 200px; }
}

/* Desktop: 1200px+ */
@media (min-width: 1200px) {
    .sidebar { width: 250px; }
    .content { margin-left: 250px; }
}

```

## Adaptações Mobile

- Menu lateral colapsável com overlay
- Tabelas com scroll horizontal
- Botões maiores (min 44px) para touch
- Fonte mínima 16px para evitar zoom
- Modais em fullscreen em telas pequenas

## Design System

### Cores Principais

#### CSS

```

:root {
    /* Cores Primárias */
    --primary-color: #4A90E2;      /* Azul principal */
    --secondary-color: #7B68EE;     /* Roxo secundário */
    --success-color: #28A745;       /* Verde sucesso */
    --warning-color: #FFC107;        /* Amarelo aviso */
}

```

```

--danger-color: #DC3545;          /* Vermelho perigo */

/* Cores de Fundo */
--bg-primary: #2C3E50;            /* Fundo sidebar */
--bg-secondary: #34495E;          /* Fundo hover */
--bg-light: #F8F9FA;              /* Fundo claro */
--bg-white: #FFFFFF;              /* Fundo branco */

/* Cores de Texto */
--text-primary: #2C3E50;          /* Texto principal */
--text-secondary: #6C757D;        /* Texto secundário */
--text-light: #FFFFFF;             /* Texto claro */
--text-muted: #ADB5BD;            /* Texto esmaecido */

/* Gradientes */
--gradient-primary: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
--gradient-painel: linear-gradient(135deg, #4A90E2 0%, #7B68EE 100%);
}

```

## Tipografia

### CSS

```

/* Fontes */
--font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
--font-size-xs: 0.75rem;          /* 12px */
--font-size-sm: 0.875rem;         /* 14px */
--font-size-base: 1rem;           /* 16px */
--font-size-lg: 1.125rem;         /* 18px */
--font-size-xl: 1.25rem;          /* 20px */
--font-size-2xl: 1.5rem;          /* 24px */
--font-size-3xl: 1.875rem;        /* 30px */
--font-size-4xl: 2.25rem;         /* 36px */

/* Pesos */
--font-weight-normal: 400;
--font-weight-medium: 500;
--font-weight-semibold: 600;
--font-weight-bold: 700;

```

## Espaçamentos

### CSS

```
/* Espaçamentos (baseado em 8px) */
--spacing-1: 0.25rem; /* 4px */
--spacing-2: 0.5rem; /* 8px */
--spacing-3: 0.75rem; /* 12px */
--spacing-4: 1rem; /* 16px */
--spacing-5: 1.25rem; /* 20px */
--spacing-6: 1.5rem; /* 24px */
--spacing-8: 2rem; /* 32px */
--spacing-10: 2.5rem; /* 40px */
--spacing-12: 3rem; /* 48px */
```

## Especificações de Áudio

### Web Audio API

JavaScript

```
// Configuração do sinal sonoro
const audioContext = new (window.AudioContext || window.webkitAudioContext)
();

function criarSinalSonoro() {
  const oscillator = audioContext.createOscillator();
  const gainNode = audioContext.createGain();

  oscillator.connect(gainNode);
  gainNode.connect(audioContext.destination);

  // Configurações
  oscillator.frequency.setValueAtTime(800, audioContext.currentTime); //
800Hz
  gainNode.gain.setValueAtTime(0.3, audioContext.currentTime); //
Volume 30%

  // Duração: 200ms
  oscillator.start(audioContext.currentTime);
  oscillator.stop(audioContext.currentTime + 0.2);
}
```

### Speech Synthesis API

JavaScript

```
// Configuração da síntese de voz
function configurarVoz() {
  const utterance = new SpeechSynthesisUtterance();

  // Configurações
  utterance.lang = 'pt-BR';           // Português brasileiro
  utterance.rate = 0.8;               // Velocidade 80%
  utterance.pitch = 1.0;              // Tom normal
  utterance.volume = 0.8;             // Volume 80%

  // Selecionar voz portuguesa se disponível
  const voices = speechSynthesis.getVoices();
  const portugueseVoice = voices.find(voice =>
    voice.lang.includes('pt') || voice.lang.includes('BR')
  );

  if (portugueseVoice) {
    utterance.voice = portugueseVoice;
  }

  return utterance;
}
```



## Performance

### Métricas Alvo

- **First Contentful Paint:** < 1.5s
- **Largest Contentful Paint:** < 2.5s
- **Cumulative Layout Shift:** < 0.1
- **First Input Delay:** < 100ms
- **Time to Interactive:** < 3s

### Otimizações Implementadas

- CSS e JS inline para reduzir requests
- Lazy loading de imagens
- Debounce em buscas (300ms)



- Cache de dados do cliente
- Minificação automática no deploy

---

## Segurança

### Autenticação e Autorização

#### Sessões

Python

```
# Configuração de sessão segura
app.config['SECRET_KEY'] = 'sistema-agendamentos-2024'
app.config['SESSION_COOKIE_SECURE'] = True          # HTTPS only
app.config['SESSION_COOKIE_HTTPONLY'] = True       # Não acessível via JS
app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'      # CSRF protection
app.config['PERMANENT_SESSION_LIFETIME'] = timedelta(hours=8)
```

#### Validação de Dados

Python

```
# Validações no backend
def validar_cpf(cpf):
    # Remove formatação
    cpf = re.sub(r'^\0-9]', '', cpf)

    # Verifica se tem 11 dígitos
    if len(cpf) != 11:
        return False

    # Verifica se não são todos iguais
    if cpf == cpf[0] * 11:
        return False

    # Algoritmo de validação do CPF
    # ... implementação completa

    return True
```

```
def validar_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
```

## Proteções Implementadas

### CORS

Python

```
# Configuração CORS
CORS(app,
      origins="*",          # Em produção: domínio específico
      supports_credentials=True, # Permite cookies
      allow_headers=["Content-Type", "Authorization"],
      methods=["GET", "POST", "PUT", "DELETE", "OPTIONS"]
)
```

### Sanitização

JavaScript

```
// Sanitização no frontend
function sanitizeInput(input) {
    return input
        .replace(/[<>]/g, '')          // Remove < e >
        .replace(/javascript:/gi, '')  // Remove javascript:
        .replace(/on\w+=/gi, '')        // Remove eventos on*
        .trim();                       // Remove espaços
}
```

### Rate Limiting

Python

```
# Implementação básica de rate limiting
from functools import wraps
from time import time

request_counts = {}

def rate_limit(max_requests=60, window=60):
    def decorator(f):
```

```

@wraps(f)
def decorated_function(*args, **kwargs):
    client_ip = request.remote_addr
    current_time = time()

    # Limpa requests antigos
    if client_ip in request_counts:
        request_counts[client_ip] = [
            req_time for req_time in request_counts[client_ip]
            if current_time - req_time < window
        ]
    else:
        request_counts[client_ip] = []

    # Verifica limite
    if len(request_counts[client_ip]) >= max_requests:
        return jsonify({
            'success': False,
            'message': 'Rate limit exceeded'
        }), 429

    # Adiciona request atual
    request_counts[client_ip].append(current_time)

    return f(*args, **kwargs)
return decorated_function
return decorator

```

## Deploy e DevOps

### Manus Cloud Platform

#### Configuração de Deploy

YAML

```

# manus.yml (configuração de deploy)
name: sistema-agendamentos
framework: flask
python_version: "3.11"
entry_point: src/main.py

build:

```

```
commands:
  - pip install -r requirements.txt
  - python src/models/user.py # Inicializar DB
```

```
runtime:
  environment:
    - FLASK_ENV=production
    - PYTHONPATH=/app/src
```

```
health_check:
  path: /api/health
  interval: 30s
  timeout: 10s
  retries: 3
```

```
scaling:
  min_instances: 1
  max_instances: 3
  cpu_threshold: 70
  memory_threshold: 80
```

## Variáveis de Ambiente

### Bash

```
# Produção
FLASK_ENV=production
SECRET_KEY=sistema-agendamentos-2024-prod
DATABASE_URL=sqlite:///database/app.db
CORS_ORIGINS=https://w5hni7c7l7n0.manus.space

# Desenvolvimento
FLASK_ENV=development
SECRET_KEY=sistema-agendamentos-2024-dev
DATABASE_URL=sqlite:///database/app_dev.db
CORS_ORIGINS=http://localhost:5000
```

## Monitoramento

## Health Check

### Python

```

@app.route('/api/health')
def health_check():
    try:
        # Verifica conexão com banco
        db.session.execute('SELECT 1')

        # Verifica arquivos estáticos
        static_files = ['index.html', 'painel.html']
        for file in static_files:
            if not os.path.exists(f'static/{file}'):
                raise Exception(f'Missing static file: {file}')

        return jsonify({
            'status': 'healthy',
            'timestamp': datetime.utcnow().isoformat(),
            'version': '2.0',
            'database': 'connected',
            'static_files': 'ok'
        })
    except Exception as e:
        return jsonify({
            'status': 'unhealthy',
            'error': str(e),
            'timestamp': datetime.utcnow().isoformat()
        }), 500

```

## Logs

Python

```

import logging

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

logger = logging.getLogger(__name__)

# Uso nos endpoints
@app.route('/api/fila/chamar-proximo', methods=['POST'])

```

```
def chamar_proximo():
    try:
        data = request.get_json()
        logger.info(f"Chamando próximo cliente: {data['cliente']['nome']}")

        # ... lógica do endpoint

        logger.info(f"Cliente {data['cliente']['nome']} chamado com sucesso")
        return jsonify(response)

    except Exception as e:
        logger.error(f"Erro ao chamar próximo cliente: {str(e)}")
        return jsonify({'success': False, 'message': str(e)}), 500
```

---

## Testes

### Testes Unitários

#### Backend (Python)

Python

```
import unittest
from src.main import app, db
from src.models.user import User

class TestAuth(unittest.TestCase):
    def setUp(self):
        self.app = app.test_client()
        self.app_context = app.app_context()
        self.app_context.push()
        db.create_all()

    def tearDown(self):
        db.session.remove()
        db.drop_all()
        self.app_context.pop()

    def test_login_success(self):
        # Criar usuário de teste
        user = User(
            email='test@test.com',
            password='test123',
```

```

        perfil=PerfilEnum.ASSISTENTE
    )
    db.session.add(user)
    db.session.commit()

    # Testar login
    response = self.app.post('/api/login',
        json={'email': 'test@test.com', 'password': 'test123'}
    )

    self.assertEqual(response.status_code, 200)
    data = response.get_json()
    self.assertTrue(data['success'])

def test_login_invalid_credentials(self):
    response = self.app.post('/api/login',
        json={'email': 'invalid@test.com', 'password': 'wrong'}
    )

    self.assertEqual(response.status_code, 401)
    data = response.get_json()
    self.assertFalse(data['success'])

```

## Frontend (JavaScript)

### JavaScript

```

// tests/frontend.test.js
describe('Sistema de Agendamentos', () => {
    describe('Validações', () => {
        test('deve validar CPF corretamente', () => {
            expect(validarCPF('123.456.789-09')).toBe(true);
            expect(validarCPF('111.111.111-11')).toBe(false);
            expect(validarCPF('123.456.789-00')).toBe(false);
        });

        test('deve validar telefone corretamente', () => {
            expect(validarTelefone('(11) 99999-9999')).toBe(true);
            expect(validarTelefone('11999999999')).toBe(false);
            expect(validarTelefone('(11) 9999-9999')).toBe(false);
        });
    });
});

describe('API Calls', () => {
    test('deve fazer login com sucesso', async () => {
        // Mock fetch
        global.fetch = jest.fn(() =>

```

```

        Promise.resolve({
            ok: true,
            json: () => Promise.resolve({
                success: true,
                user: { email: 'test@test.com' }
            })
        })
    );

    const result = await fazerLogin('test@test.com', 'test123');
    expect(result.success).toBe(true);
    expect(fetch).toHaveBeenCalledWith('/api/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            email: 'test@test.com',
            password: 'test123'
        })
    });
});
});
});

```

## Testes de Integração

Python

```

class TestIntegration(unittest.TestCase):
    def test_fluxo_completo_agendamento(self):
        # 1. Login
        login_response = self.app.post('/api/login',
            json={'email': 'assistente@escritorio.com', 'password':
'assistente123'})
        self.assertEqual(login_response.status_code, 200)

        # 2. Criar cliente
        cliente_response = self.app.post('/api/clientes',
            json={
                'nome': 'João Teste',
                'cpf': '123.456.789-09',
                'telefone': '(11) 99999-9999'
            })
        self.assertEqual(cliente_response.status_code, 201)
        cliente_id = cliente_response.get_json()['cliente']['id']

```



```
# 3. Criar agendamento
agendamento_response = self.app.post('/api/agendamentos',
    json={
        'cliente_id': cliente_id,
        'data': '2025-07-14',
        'horario': '09:00'
    }
)
self.assertEqual(agendamento_response.status_code, 201)

# 4. Check-in
agendamento_id = agendamento_response.get_json()['agendamento']['id']
checkin_response = self.app.post(f'/api/checkin/{agendamento_id}')
self.assertEqual(checkin_response.status_code, 200)

# 5. Chamar próximo
chamar_response = self.app.post('/api/fila/chamar-proximo',
    json={
        'cliente': {
            'nome': 'João Teste',
            'cpf': '123.456.789-09',
            'horario': '09:00'
        }
    }
)
self.assertEqual(chamar_response.status_code, 200)

# 6. Verificar painel
painel_response = self.app.get('/api/painel/status')
self.assertEqual(painel_response.status_code, 200)
painel_data = painel_response.get_json()
self.assertEqual(painel_data['cliente_atual']['nome'], 'João Teste')
```



## Performance e Otimização



### Otimizações Implementadas

#### Frontend

JavaScript

```

// Debounce para buscas
function debounce(func, wait) {
  let timeout;
  return function executedFunction(...args) {
    const later = () => {
      clearTimeout(timeout);
      func(...args);
    };
    clearTimeout(timeout);
    timeout = setTimeout(later, wait);
  };
}

// Uso em busca de clientes
const buscarClientesDebounce = debounce(buscarClientes, 300);

// Cache de dados
const cache = {
  clientes: null,
  agendamentos: null,
  timestamp: null,

  get(key) {
    if (this.timestamp && Date.now() - this.timestamp < 30000) { // 30s
      return this[key];
    }
    return null;
  },

  set(key, value) {
    this[key] = value;
    this.timestamp = Date.now();
  }
};

```

## Backend

### Python

```

# Cache simples em memória
from functools import lru_cache
from datetime import datetime, timedelta

# Cache para dados que mudam pouco
@lru_cache(maxsize=100)
def get_clientes_cache():

```

```

    return db.session.query(Cliente).all()

# Cache com TTL
cache_agendamentos = {}
cache_ttl = timedelta(minutes=5)

def get_agendamentos_cached(data=None):
    cache_key = f"agendamentos_{data or 'all'}"

    if (cache_key in cache_agendamentos and
        datetime.now() - cache_agendamentos[cache_key]['timestamp'] <
        cache_ttl):
        return cache_agendamentos[cache_key]['data']

    # Buscar dados frescos
    agendamentos = buscar_agendamentos(data)
    cache_agendamentos[cache_key] = {
        'data': agendamentos,
        'timestamp': datetime.now()
    }

    return agendamentos

```



## Métricas de Performance

### Tempos de Resposta Alvo

- **Login:** < 500ms
- **Listar clientes:** < 300ms
- **Criar agendamento:** < 400ms
- **Chamar próximo:** < 200ms
- **Status do painel:** < 100ms

### Otimizações de Banco

#### SQL

```

-- Índices para melhor performance
CREATE INDEX idx_agendamentos_data ON agendamentos(data);
CREATE INDEX idx_agendamentos_status ON agendamentos(status);
CREATE INDEX idx_agendamentos_cliente_id ON agendamentos(cliente_id);

```

```
CREATE INDEX idx_clientes_cpf ON clientes(cpf);
CREATE INDEX idx_clientes_nome ON clientes(nome);

-- Query otimizada para fila
SELECT a.*, c.nome, c.cpf
FROM agendamentos a
JOIN clientes c ON a.cliente_id = c.id
WHERE a.data = CURRENT_DATE
      AND a.status = 'PRESENTE'
ORDER BY a.prioridade DESC, a.horario ASC;
```

## Manutenção

### Backup e Recuperação

#### Backup Automático

Python

```
import shutil
import os
from datetime import datetime

def backup_database():
    """Cria backup do banco de dados"""
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    backup_name = f'backup_sistema_agendamentos_{timestamp}.db'
    backup_path = f'backups/{backup_name}'

    # Criar diretório se não existir
    os.makedirs('backups', exist_ok=True)

    # Copiar banco
    shutil.copy2('database/app.db', backup_path)

    # Manter apenas últimos 30 backups
    cleanup_old_backups()

    return backup_path

def cleanup_old_backups():
    """Remove backups antigos (mantém últimos 30)"""
    backup_dir = 'backups'
```

```
if not os.path.exists(backup_dir):
    return

backups = sorted([
    f for f in os.listdir(backup_dir)
    if f.startswith('backup_sistema_agendamentos_')
])

# Remove backups antigos
for backup in backups[:-30]:
    os.remove(os.path.join(backup_dir, backup))
```

## Restauração

Python

```
def restore_database(backup_path):
    """Restaura banco de dados a partir de backup"""
    if not os.path.exists(backup_path):
        raise FileNotFoundError(f'Backup não encontrado: {backup_path}')

    # Backup do banco atual antes de restaurar
    current_backup = backup_database()

    try:
        # Restaurar backup
        shutil.copy2(backup_path, 'database/app.db')
        return True
    except Exception as e:
        # Se falhar, restaurar backup atual
        shutil.copy2(current_backup, 'database/app.db')
        raise e
```



## Logs e Auditoria

### Sistema de Logs

Python

```
import logging
from logging.handlers import RotatingFileHandler

# Configuração avançada de logs
def setup_logging():
```

```

# Formatter
formatter = logging.Formatter(
    '%(asctime)s - %(name)s - %(levelname)s - %(funcName)s:%(lineno)d - %
(message)s'
)

# Handler para arquivo (rotativo)
file_handler = RotatingFileHandler(
    'logs/sistema_agendamentos.log',
    maxBytes=10*1024*1024, # 10MB
    backupCount=5
)
file_handler.setFormatter(formatter)
file_handler.setLevel(logging.INFO)

# Handler para console
console_handler = logging.StreamHandler()
console_handler.setFormatter(formatter)
console_handler.setLevel(logging.DEBUG)

# Logger principal
logger = logging.getLogger('sistema_agendamentos')
logger.setLevel(logging.DEBUG)
logger.addHandler(file_handler)
logger.addHandler(console_handler)

return logger

# Auditoria de ações
def log_user_action(user_email, action, details=None):
    """Log de ações do usuário para auditoria"""
    logger = logging.getLogger('sistema_agendamentos.audit')

    log_entry = {
        'timestamp': datetime.utcnow().isoformat(),
        'user': user_email,
        'action': action,
        'details': details,
        'ip': request.remote_addr if request else None
    }

    logger.info(f"AUDIT: {json.dumps(log_entry)}")

```

## Atualizações

### Versionamento

---

## Python

```
# version.py
VERSION = {
    'major': 2,
    'minor': 0,
    'patch': 0,
    'build': '20250713'
}

def get_version_string():
    return f"{VERSION['major']}.{VERSION['minor']}.{VERSION['patch']}.
{VERSION['build']}"

# Endpoint de versão
@app.route('/api/version')
def get_version():
    return jsonify({
        'version': get_version_string(),
        'release_date': '2025-07-13',
        'features': [
            'Painel de chamada automático',
            'Síntese de voz',
            'Sistema de priorização',
            'Interface responsiva'
        ]
    })
```

## Migração de Dados

## Python


```
# migrations.py
def migrate_v1_to_v2():
    """Migração da versão 1.0 para 2.0"""
    try:
        # Adicionar colunas se não existirem
        db.session.execute('''
            ALTER TABLE agendamentos
            ADD COLUMN prioridade BOOLEAN DEFAULT FALSE
        ''')

        db.session.execute('''
            ALTER TABLE agendamentos
            ADD COLUMN chamado_em TIMESTAMP NULL
        ''')
```

```
        db.session.commit()
        logger.info("Migração v1 -> v2 concluída com sucesso")

except Exception as e:
    db.session.rollback()
    logger.error(f"Erro na migração v1 -> v2: {str(e)}")
    raise e
```

---

 **Esta documentação técnica fornece todas as especificações necessárias para manutenção, desenvolvimento e deploy do Sistema de Agendamentos.**