Teste da Lógica de Prioridade - Atende+

📋 Status do Servidor

✅ **Servidor rodando em:** http://localhost:3001

🎯 Lógica de Prioridade Implementada

A lógica de prioridade está implementada no arquivo server.js e funciona da seguinte forma:

Regras de Ordenação da Fila

1. **Prioridade Principal**: Senhas prioritárias (`prioritaria` ou `prioritaria+`) são chamadas ANTES de senhas normais
2. **Ordem de Chegada**: Dentro do mesmo nível de prioridade, a ordem é por `horaGeracao` (FIFO - First In, First Out)

```
const filaOrdenada = candidatos.sort((a, b) => {
    if (a.prioridade === b.prioridade) {
        return new Date(a.horaGeracao).getTime() - new Date(b.horaGe
    }
    return a.prioridade === 'prioritaria' ? -1 : 1;
});
```

Tipos de Senha

- **Normal (N)**: Prefixo `N` + contador (ex: N001, N002, N003...)
- **Prioritária (P)**: Prefixo `P` + contador (ex: P001, P002, P003...)

🧪 Plano de Testes

Teste 1: Prioridade Básica

Objetivo: Verificar se senhas prioritárias são chamadas antes das normais

Passos:

1. Acesse http://localhost:3001
2. Faça login como **Gerador**
3. Gere as seguintes senhas nesta ordem:
    - Senha N001 (Normal)
    - Senha N002 (Normal)
    - Senha P001 (Prioritária)
    - Senha N003 (Normal)
4. Faça login como **Atendente** em outra aba/janela
5. Chame a próxima senha

Resultado Esperado: A senha **P001** deve ser chamada primeiro, mesmo tendo sido gerada depois das senhas N001 e N002.

Teste 2: Ordem de Chegada (FIFO)

Objetivo: Verificar se senhas do mesmo tipo respeitam a ordem de chegada

Passos:

1. Gere 3 senhas normais:
    - Senha N001
    - Senha N002
    - Senha N003
2. Chame as senhas uma por uma

Resultado Esperado: As senhas devem ser chamadas na ordem: N001 → N002 → N003

Teste 3: Múltiplas Prioridades Intercaladas

Objetivo: Testar cenário complexo com múltiplas prioridades

Passos:

1. Gere as senhas nesta ordem:
    - N001 (Normal)
    - P001 (Prioritária)
    - N002 (Normal)
    - P002 (Prioritária)
    - N003 (Normal)
    - P003 (Prioritária)
2. Chame todas as senhas

Resultado Esperado:

- Ordem de chamada: **P001 → P002 → P003 → N001 → N002 → N003**

Teste 4: Concorrência (Múltiplos Atendentes)

Objetivo: Verificar se a lógica atômica previne conflitos

Passos:

1. Gere 5 senhas prioritárias (P001 a P005)
2. Abra 2 ou 3 abas como atendentes diferentes
3. Clique em "Chamar Senha" simultaneamente em todas as abas

Resultado Esperado:

- Cada atendente deve receber uma senha diferente
- Nenhuma senha deve ser chamada duas vezes
- A ordem deve respeitar P001 → P002 → P003 → P004 → P005

Teste 5: Filtro por Tipo de Atendimento

Objetivo: Verificar se o filtro de tipos de atendimento funciona corretamente

Passos:

1. Configure um atendente para atender apenas "Cadastro Novo"
2. Gere senhas:
    - P001 - Cadastro Novo (Prioritária)
    - P002 - Atualização (Prioritária)
    - N001 - Cadastro Novo (Normal)
3. Chame a próxima senha com o atendente configurado

Resultado Esperado:

- A senha **P001** deve ser chamada (prioritária + tipo correto)
- A senha P002 deve ser ignorada (tipo diferente)
- A senha N001 só será chamada depois que P001 for finalizada

---

🔍 Como Verificar os Resultados

No Painel de Atendimento

- A senha chamada aparece em destaque
- Verifique o número da senha e a ordem

No Console do Servidor

- Abra o terminal onde o servidor está rodando
- Procure por mensagens como: `Senha P001 chamada no Guichê 1 (Atômico)`

No Banco de Dados

Execute o seguinte comando para ver a ordem das senhas:

```
npx prisma studio
```

Ou use o script de inspeção:

```
node inspect_db.js
```

⚠️ Problemas Conhecidos

Prioridade+ não implementada

O código atual trata `prioritaria` e `prioritaria+` da mesma forma. Para diferenciar, seria necessário ajustar a lógica de ordenação:

```
const prioridadeValor = (p) => {
    if (p === 'prioritaria+') return 3;
    if (p === 'prioritaria') return 2;
    return 1; // normal
};

const filaOrdenada = candidatos.sort((a, b) => {
    const prioA = prioridadeValor(a.prioridade);
    const prioB = prioridadeValor(b.prioridade);

    if (prioA !== prioB) {
        return prioB - prioA; // Maior prioridade primeiro
    }
    return new Date(a.horaGeracao).getTime() - new Date(b.horaGeraca
});
```

📊 Checklist de Testes

- [ ] Teste 1: Prioridade Básica
- [ ] Teste 2: Ordem de Chegada (FIFO)
- [ ] Teste 3: Múltiplas Prioridades Intercaladas
- [ ] Teste 4: Concorrência (Múltiplos Atendentes)

- ☐ Teste 5: Filtro por Tipo de Atendimento

🚀 Credenciais de Teste

Administrador:

- Email: `admin`
- Senha: `admin`

Criar outros usuários: Use o painel de administração para criar atendentes e geradores conforme necessário.
