🚀 Resumo Rápido de Integração - Vision Trading Agent

📍 Onde Adicionar os Novos Componentes

1️⃣ Backend (Supabase)

```
supabase/
├── functions/
│   └── vision-agent-signal/     ← ✨ NOVO: Recebe sinais do Vision Agent
│       └── index.ts
├── migrations/
│   └── [timestamp]_vision_agent.sql  ← ✨ NOVO: Tabelas do Vision Agent
```

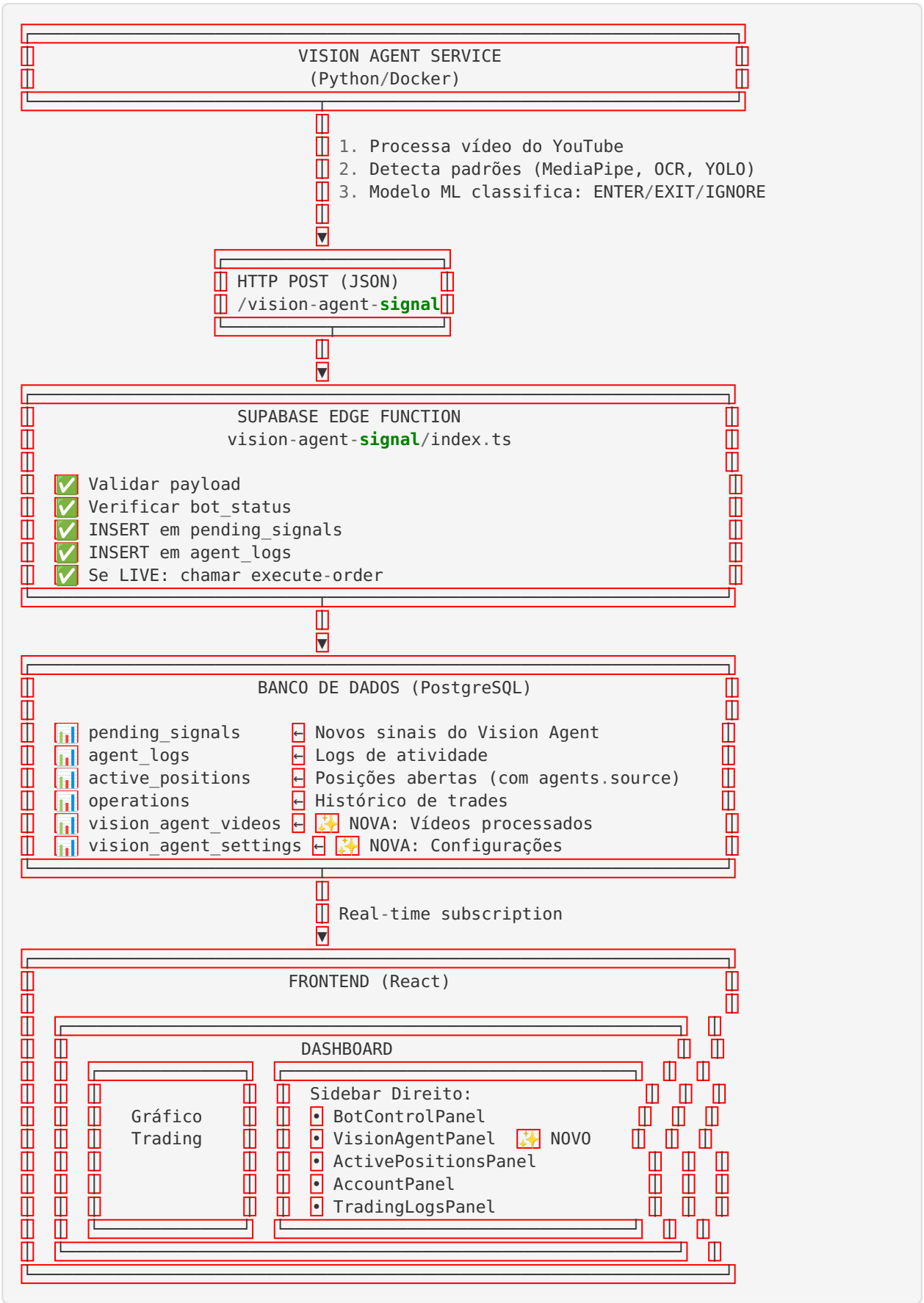
2️⃣ Frontend (React)

```
src/
├── components/
│   └── trading/
│       ├── BotControlPanel.tsx          (existente)
│       ├── VisionAgentPanel.tsx         ← ✨ NOVO: Painel do Vision Agent
│       ├── ActivePositionsPanel.tsx     ← 🔧 MODIFICAR: Badge "Vision Agent"
│       └── TradingLogsPanel.tsx         (existente - logs automáticos)
├── pages/
│   ├── Dashboard.tsx                    ← 🔧 MODIFICAR: Adicionar VisionAgentPanel
│   └── VisionAgentSettings.tsx          ← ✨ NOVO: Página de configurações
└── integrations/
    └── supabase/
        └── types.ts                     ← 🔄 ATUALIZAR: Novos tipos após migração
```

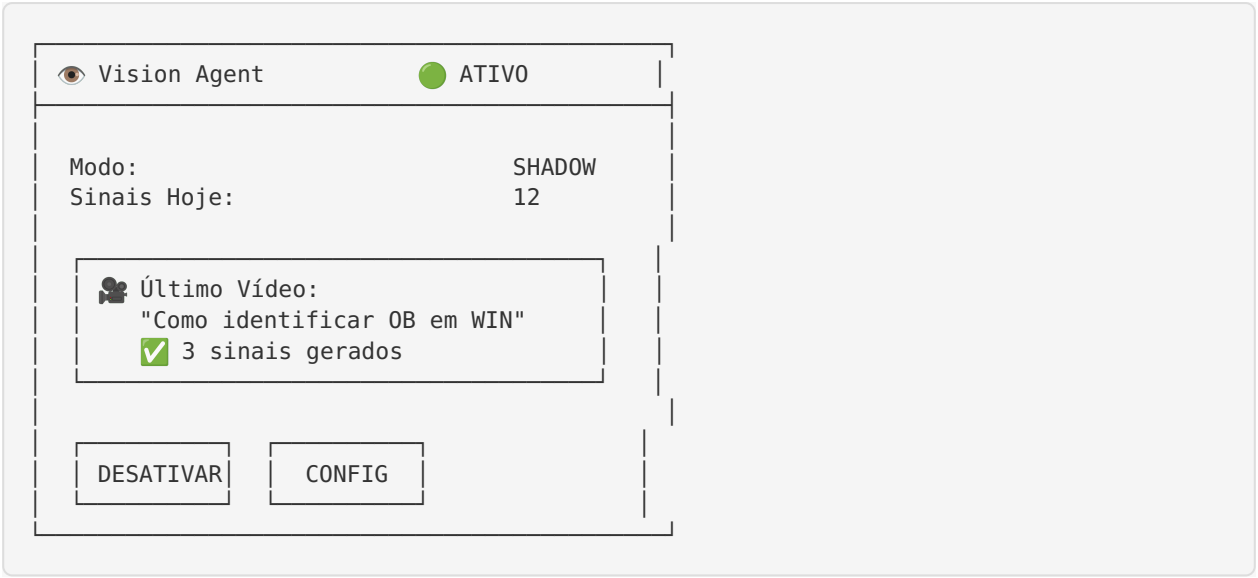
3️⃣ Serviço Externo (Python)

```
vision-agent-service/                    ← ✨ NOVO: Serviço separado (Python)
├── src/
│   ├── agent/
│   │   ├── video_processor.py
│   │   ├── feature_extractor.py
│   │   ├── model_inference.py
│   │   └── signal_sender.py             ← Envia para Edge Function
│   └── main.py
├── models/
│   └── model_seq_v20251125.h5
├── requirements.txt
└── Dockerfile
```

🔗 Fluxo de Dados Simplificado



🎨 Visualização do VisionAgentPanel



✅ Checklist Rápido de Implementação

Backend

- [ ] Criar arquivo `supabase/migrations/[timestamp]_vision_agent.sql`
- [ ] Adicionar tabelas: `vision_agent_videos`, `vision_agent_settings`
- [ ] Criar pasta `supabase/functions/vision-agent-signal/`
- [ ] Implementar `index.ts` da Edge Function
- [ ] Executar migração: `npx supabase db push`
- [ ] Atualizar tipos: `npx supabase gen types typescript --local > src/integrations/supabase/types.ts`

Frontend

- [ ] Criar `src/components/trading/VisionAgentPanel.tsx`
- [ ] Editar `src/pages/Dashboard.tsx` (adicionar `<VisionAgentPanel />`)
- [ ] Editar `src/components/trading/ActivePositionsPanel.tsx` (adicionar badge)
- [ ] Criar `src/pages/VisionAgentSettings.tsx`
- [ ] Adicionar rota em `src/App.tsx`

Vision Agent Service

- [ ] Criar diretório `vision-agent-service/` (fora do dashboard)
- [ ] Implementar processamento de vídeo (Python)
- [ ] Integrar MediaPipe, OCR, YOLO
- [ ] Implementar modelo ML (LSTM/Transformer)
- [ ] Criar `signal_sender.py` para comunicação HTTP
- [ ] Configurar variáveis de ambiente (SUPABASE_URL, SERVICE_KEY)
- [ ] Dockerizar o serviço

🔐 Variáveis de Ambiente Necessárias

Vision Agent Service (.env)

```
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_SERVICE_ROLE_KEY=your_service_role_key
VISION_AGENT_USER_ID=uuid_do_usuario
CONFIDENCE_THRESHOLD=0.70
MODE=SHADOW  # SHADOW | PAPER | LIVE
YOUTUBE_PLAYLIST_URL=https://youtube.com/playlist?list=...
```

Dashboard (.env - já existente)

```
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_PUBLISHABLE_KEY=your_anon_key
```

🚦 Ordem de Implementação (Passo a Passo)

Dia 1: Preparação Backend

1. Criar migração SQL com novas tabelas
2. Executar migração no Supabase
3. Atualizar tipos TypeScript

Dia 2-3: Edge Function

1. Criar pasta `vision-agent-signal/`
2. Implementar lógica de validação e inserção
3. Testar com Postman/curl

Dia 4: Frontend - Componente Básico

1. Criar `VisionAgentPanel.tsx` (versão simples)
2. Integrar no Dashboard
3. Testar exibição de status

Dia 5-10: Vision Agent Service (MVP)

1. Estrutura básica do projeto Python
2. Implementar processamento de 1 vídeo
3. Integrar MediaPipe + OCR básico
4. Criar modelo ML simples (ou mock)
5. Implementar envio de sinal via HTTP

Dia 11-12: Testes End-to-End

1. Processar vídeo de teste
2. Verificar sinal no dashboard
3. Validar logs e auditoria

Dia 13-14: Refinamento

1. Melhorar UI/UX do painel

2. Adicionar página de configurações
3. Implementar scheduler para processar playlist

📊 Exemplo de Payload do Sinal

Requisição do Vision Agent → Edge Function

```
POST /functions/v1/vision-agent-signal
Authorization: Bearer SERVICE_ROLE_KEY

{
  "user_id": "uuid-do-usuario",
  "action": "ENTER",
  "symbol": "WIN$",
  "confidence": 0.82,
  "video_id": "dQw4w9WgXcQ",
  "frame_index": 2400,
  "entry_price": 134.50,
  "stop_loss": 133.80,
  "take_profit": 136.60,
  "risk_reward": 3.0,
  "model_version": "model_seq_v20251125.h5",
  "features_summary": {
    "hands_detected": 1,
    "draw_count": 2,
    "ocr_text": "Entry 134.50"
  }
}
```

Resposta da Edge Function

```
{
  "status": "signal_created",
  "signal_id": "uuid-do-sinal",
  "message": "Sinal recebido e armazenado"
}
```

⚠️ Pontos de Atenção

✅ O que PODE fazer:

- Adicionar novos componentes React
- Criar novas tabelas no banco
- Adicionar novas Edge Functions
- Criar serviços externos (Python)

❌ O que NÃO PODE fazer:

- Modificar estrutura de tabelas existentes
- Alterar design/layout atual
- Mudar lógica de trading existente

- Remover ou renomear componentes atuais

🛡️ Segurança Obrigatória:

- Sempre começar no modo **SHADOW**
- Validar `confidence >= threshold`
- Verificar `bot_status === "running"`
- Limitar quantidade de sinais por período
- Logs completos para auditoria

🎯 Resultado Final

Após a integração completa, o usuário terá:

1. **Um painel visual** no dashboard mostrando status do Vision Agent
2. **Processamento automático** de vídeos do YouTube
3. **Sinais de trading** gerados por ML em tempo real
4. **Execução automática** (se configurado em LIVE)
5. **Logs completos** de toda atividade
6. **Configuração flexível** via interface web
7. **Sistema modular** que pode ser desativado a qualquer momento

Sem quebrar nada do sistema existente! 🎉

Documento: Resumo de Integração
Versão: 1.0
Data: 25 de Novembro de 2025