📊 Análise de Estrutura e Integração do Vision Trading Agent

📋 Sumário Executivo

Este documento apresenta uma análise completa da estrutura atual do **SMC Alpha Dashboard** e identifica os melhores pontos de integração para o **Vision Trading Agent** sem quebrar funcionalidades existentes ou alterar o design.

🏗️ 1. Estrutura Atual do Projeto

1.1 Visão Geral Técnica

- **Framework Frontend**: React 18.3.1 + TypeScript
- **Build Tool**: Vite 5.4.19
- **Roteamento**: React Router DOM 6.30.1
- **UI Framework**: shadcn/ui + Radix UI
- **Estilização**: Tailwind CSS 3.4.17
- **Backend**: Supabase (PostgreSQL + Edge Functions)
- **State Management**: React Query (@tanstack/react-query 5.83.0)
- **Autenticação**: Supabase Auth
- **Gráficos**: Recharts 2.15.4

1.2 Estrutura de Diretórios

```
smc-alpha-dashboard-main/
├── public/                          # Arquivos estáticos
│   ├── favicon.ico
│   ├── placeholder.svg
│   └── robots.txt
├── src/
│   ├── components/                  # Componentes React
│   │   ├── settings/                # Configurações
│   │   │   └── SettingsDialog.tsx
│   │   ├── trading/                 # Componentes de trading
│   │   │   ├── AccountPanel.tsx     # Painel de conta/saldo
│   │   │   ├── ActivePositionsPanel.tsx  # Posições abertas
│   │   │   ├── BotControlPanel.tsx  # ⭐ Controle do bot
│   │   │   ├── SMCPanel.tsx         # Análise SMC
│   │   │   ├── TopBar.tsx           # Barra superior
│   │   │   ├── TradingChart.tsx     # Gráfico principal
│   │   │   ├── TradingChartOverlay.tsx
│   │   │   └── TradingLogsPanel.tsx # Logs de trading
│   │   ├── ui/                      # Componentes shadcn
│   │   ├── NavLink.tsx
│   │   └── ProtectedRoute.tsx
│   ├── hooks/                       # Custom React Hooks
│   │   ├── use-mobile.tsx
│   │   ├── use-toast.ts
│   │   ├── useAuth.tsx              # Hook de autenticação
│   │   └── useMultiTimeframeAnalysis.ts
│   ├── integrations/                # Integrações externas
│   │   └── supabase/
│   │       ├── client.ts            # Cliente Supabase
│   │       └── types.ts             # ⭐ TypeScript types (1071 linhas)
│   ├── lib/
│   │   └── utils.ts                 # Utilitários
│   ├── pages/                       # Páginas da aplicação
│   │   ├── Auth.tsx                 # Página de autenticação
│   │   ├── Dashboard.tsx            # ⭐ Dashboard principal
│   │   ├── Index.tsx                # Página inicial
│   │   └── NotFound.tsx
│   ├── App.tsx                      # ⭐ App principal
│   └── main.tsx                     # Entry point
├── supabase/                        # Backend Supabase
│   ├── functions/                   # ⭐ Edge Functions (Deno)
│   │   ├── analyze-multi-timeframe/
│   │   ├── close-position/
│   │   ├── encrypt-api-credentials/
│   │   ├── execute-order/           # ⭐ Execução de ordens
│   │   ├── monitor-positions/
│   │   ├── sync-real-balance/
│   │   └── test-broker-connection/
│   ├── migrations/                  # Migrações SQL
│   └── config.toml                  # Configuração
└── [arquivos de configuração]

⭐ = Pontos críticos para integração
```

🗄️ 2. Arquitetura de Backend (Supabase)

2.1 Tabelas Principais do Banco de Dados

active_positions (Posições Abertas)

```
{
  id: string (UUID)
  user_id: string (FK → auth.users)
  asset: string (ex: "BTCUSDT", "WIN$")
  direction: string ("LONG" | "SHORT")
  entry_price: number
  stop_loss: number
  take_profit: number
  risk_reward: number
  current_price: number | null
  current_pnl: number | null
  projected_profit: number
  session: string | null
  agents: Json | null        // ⭐ CAMPO RELEVANTE PARA VISION AGENT
  opened_at: timestamp
  updated_at: timestamp
}
```

operations (Histórico de Operações)

```
{
  id: string
  user_id: string
  asset: string
  direction: string
  entry_price: number
  exit_price: number
  stop_loss: number
  take_profit: number
  risk_reward: number
  pnl: number
  profit_percent: number
  result: string ("WIN" | "LOSS")
  entry_time: timestamp
  exit_time: timestamp
  session: string | null
  agents: Json | null        // ⭐ CAMPO RELEVANTE
}
```

user_settings (Configurações do Usuário)

```
{
  id: string
  user_id: string
  bot_status: string ("stopped" | "running" | "paused")
  paper_mode: boolean
  balance: number
  risk_per_trade: number
  leverage: number | null
  max_positions: number | null
  profit_target_percent: number | null
  active_strategies: string[] | null  // ⭐ Lista de estratégias ativas
  trading_strategy: string | null
  single_position_mode: boolean | null
}
```

pending_signals (Sinais Pendentes)

```
{
  id: string
  user_id: string
  asset: string
  signal_type: string ("ENTER" | "EXIT")
  direction: string | null
  entry_price: number | null
  stop_loss: number | null
  take_profit: number | null
  risk_reward: number | null
  confidence: number | null   // ⭐ Confidence do modelo
  signal_data: Json | null    // ⭐ Dados adicionais do sinal
  status: string ("pending" | "executed" | "cancelled")
  created_at: timestamp
  executed_at: timestamp | null
}
```

agent_logs (Logs de Agentes)

```
{
  id: string
  user_id: string
  agent_name: string          // ⭐ "vision_trading_agent"
  action: string
  status: string
  details: Json | null
  created_at: timestamp
}
```

user_api_credentials (Credenciais de API)

```
{
  id: string
  user_id: string
  broker_type: string ("binance" | "forex")
  encrypted_api_key: string | null
  encrypted_api_secret: string | null
  broker_name: string | null
  is_active: boolean
  test_status: string ("success" | "failed" | "pending")
}
```

2.2 Edge Functions (Serverless Deno)

execute-order (`supabase/functions/execute-order/`)

- **Responsabilidade**: Executar ordens de trading
- **Validações**:
- `bot_status === "running"`
- Verificar posições existentes
- Validar saldo e risk management
- Modo paper vs real
- ⭐ **Ponto de Integração**: Aceitar sinais do Vision Agent

monitor-positions (`supabase/functions/monitor-positions/`)

- Monitora posições abertas
- Atualiza PnL em tempo real
- Fecha posições quando TP/SL atingido

analyze-multi-timeframe (`supabase/functions/analyze-multi-timeframe/`)

- Análise SMC em múltiplos timeframes
- Detecta FVG, OB, Liquidity Sweeps

🎨 3. Frontend - Componentes Existentes

3.1 Dashboard Principal (`src/pages/Dashboard.tsx`)

Estrutura atual:

```
<Dashboard>
  <TopBar />              // Seleção de symbol/interval
  <TradingChart />        // Gráfico principal (Recharts)
  <RightSidebar>
    <BotControlPanel />   // ⭐ Controle START/PAUSE/STOP
    <ActivePositionsPanel />
    <AccountPanel />
    <SMCPanel />
    <TradingLogsPanel />  // ⭐ Logs de trading
  </RightSidebar>
</Dashboard>
```

3.2 BotControlPanel (`src/components/trading/BotControlPanel.tsx`)

Funcionalidades atuais:
- Iniciar/Pausar/Parar o bot
- Exibir status: 🟢 ATIVO | 🟡 PAUSADO | 🔴 PARADO
- Mostrar modo: 📄 PAPER | 💰 REAL
- Contador de posições abertas e trades do dia

⭐ **Ponto de Integração**: Adicionar indicador do Vision Agent

3.3 ActivePositionsPanel (`src/components/trading/ActivePositionsPanel.tsx`)

- Lista posições abertas em tempo real
- Mostra PnL atual, preço de entrada, TP/SL
- Botão de fechar posição manualmente

⭐ **Ponto de Integração**: Exibir se a posição foi originada pelo Vision Agent

3.4 TradingLogsPanel (`src/components/trading/TradingLogsPanel.tsx`)

- Exibe logs de trading em tempo real
- Conectado à tabela `operations`

⭐ **Ponto de Integração**: Exibir logs do Vision Agent (processamento de vídeos, sinais gerados)

🤖 4. Vision Trading Agent - Descrição Completa

4.1 Visão Geral

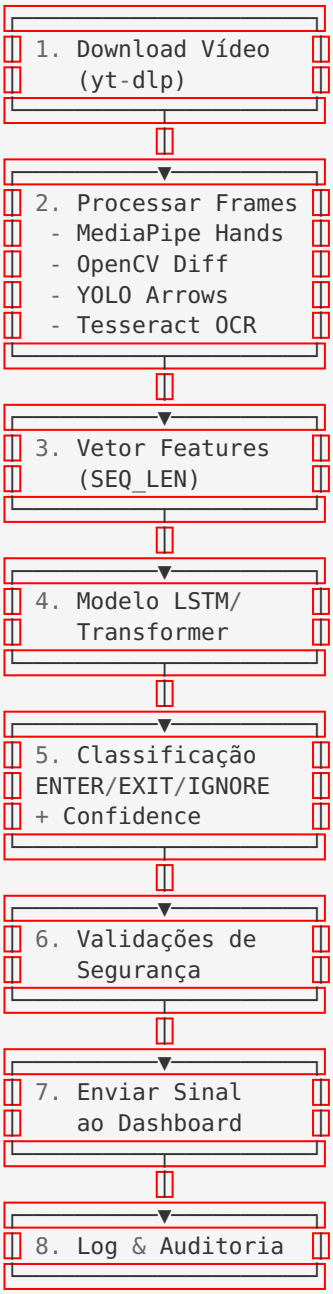
O **Vision Trading Agent** é um sistema de visão computacional e machine learning que:

1. **Assiste automaticamente vídeos** de um canal/playlist do YouTube
2. **Detecta padrões visuais** usando:
   - MediaPipe (gestos e mãos)
   - OpenCV (traços, riscos, linhas)
   - YOLO (setas, formas)
   - OCR/Tesseract (níveis de preço, RR, texto)
3. **Processa frames sequencialmente** criando vetores de features temporais
4. **Classifica ações** usando modelo LSTM/Transformer:
   - **ENTER** (entrar em posição)
   - **EXIT** (sair da posição)
   - **IGNORE** (nenhuma ação)
5. **Envia sinais em tempo real** para o dashboard via API
6. **Evolui continuamente** através de re-treinamento periódico

4.2 Modos de Operação

Modo	Comportamento
SHADOW (padrão)	Apenas observa e registra logs. Não executa nada.
PAPER	Gera sinais e envia para o painel em modo simulado.
LIVE	Executa ordens reais com validações de segurança obrigatórias.

4.3 Pipeline do Agente



4.4 Payload do Sinal (JSON)

Quando o Vision Agent detecta um padrão, ele envia:

```
{
  "action": "ENTER",
  "timestamp": "2025-11-25T12:45:32Z",
  "confidence": 0.82,
  "symbol": "WIN$",
  "video_id": "abc123",
  "frame_index": 2400,
  "features_summary": {
    "hands": 1,
    "draw_count": 2,
    "ocr": "1.3450"
  },
  "model_version": "model_seq_v20251125.h5",
  "entry_price": 134.50,
  "stop_loss": 133.80,
  "take_profit": 136.60,
  "risk_reward": 3.0
}
```

4.5 Tecnologias do Agente

- **OpenCV** - Processamento de vídeo e detecção de diferenças
- **MediaPipe Holistic** - Detecção de landmarks de mãos
- **Tesseract OCR** - Leitura de texto nos vídeos
- **YOLO (Ultralytics)** - Detecção de setas/linhas
- **TensorFlow/Keras** - Modelo LSTM/Transformer
- **yt-dlp** - Download de vídeos do YouTube
- **Python 3.10+** - Linguagem principal

🔗 5. Pontos de Integração (Sem Quebrar Nada)

5.1 Backend - Nova Edge Function

Criar: `supabase/functions/vision-agent-signal/index.ts`

Responsabilidade:
- Receber sinais do Vision Agent (via HTTP POST)
- Validar estrutura do payload
- Autenticar o agente (token seguro)
- Inserir sinal na tabela `pending_signals`
- Triggerar a Edge Function `execute-order` se modo LIVE
- Registrar log na tabela `agent_logs`

Endpoint: `POST /functions/v1/vision-agent-signal`

Exemplo de implementação:

```
// supabase/functions/vision-agent-signal/index.ts
import { serve } from "https://deno.land/std@0.168.0/http/server.ts";
import { createClient } from "https://esm.sh/@supabase/supabase-js@2.39.3";

serve(async (req) => {
  if (req.method !== 'POST') {
    return new Response('Method not allowed', { status: 405 });
  }

  const supabase = createClient(
    Deno.env.get('SUPABASE_URL')!,
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!
  );

  // Validar token do Vision Agent
  const authHeader = req.headers.get('Authorization');
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return new Response('Unauthorized', { status: 401 });
  }

  const payload = await req.json();
  const { action, confidence, symbol, video_id, entry_price, stop_loss, take_profit, r
isk_reward } = payload;

  // Validações
  if (!['ENTER', 'EXIT', 'IGNORE'].includes(action)) {
    return new Response('Invalid action', { status: 400 });
  }

  if (action === 'IGNORE') {
    return new Response(JSON.stringify({ status: 'ignored' }), { status: 200 });
  }

  // Buscar user_id (assumindo que o token contém user_id ou é configurado por
usuário)
  const user_id = payload.user_id; // ou extrair do token

  // Inserir sinal pendente
  const { data: signal, error: signalError } = await supabase
    .from('pending_signals')
    .insert({
      user_id,
      asset: symbol,
      signal_type: action,
      direction: action === 'ENTER' ? 'LONG' : null,
      entry_price,
      stop_loss,
      take_profit,
      risk_reward,
      confidence,
      signal_data: {
        source: 'vision_trading_agent',
        video_id,
        model_version: payload.model_version,
        features: payload.features_summary
      },
      status: 'pending'
    })
    .select()
    .single();

  if (signalError) {
```

```
    return new Response(JSON.stringify({ error: signalError.message }), { status:
500 });
  }

  // Log do agente
  await supabase.from('agent_logs').insert({
    user_id,
    agent_name: 'vision_trading_agent',
    action: `signal_${action.toLowerCase()}`,
    status: 'success',
    details: { signal_id: signal.id, video_id, confidence }
  });

  // Se modo LIVE e bot está running, executar ordem imediatamente
  const { data: settings } = await supabase
    .from('user_settings')
    .select('bot_status, paper_mode')
    .eq('user_id', user_id)
    .single();

  if (settings?.bot_status === 'running' && action === 'ENTER') {
    // Chamar execute-order
    const executeResponse = await fetch(`${Deno.env.get('SUPABASE_URL')}/functions/v1/
execute-order`, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        asset: symbol,
        direction: 'LONG',
        entry_price,
        stop_loss,
        take_profit,
        risk_reward,
        signal_data: {
          source: 'vision_trading_agent',
          video_id,
          confidence
        }
      })
    });

    const executeResult = await executeResponse.json();

    return new Response(JSON.stringify({
      status: 'executed',
      signal_id: signal.id,
      execution: executeResult
    }), { status: 200 });
  }

  return new Response(JSON.stringify({
    status: 'signal_created',
    signal_id: signal.id
  }), { status: 200 });
});
```

5.2 Backend - Nova Tabela `vision_agent_videos`

Criar migração: `supabase/migrations/[timestamp]_create_vision_agent_tables.sql`

```sql
-- Tabela para rastrear vídeos processados
CREATE TABLE public.vision_agent_videos (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL,
  video_id TEXT NOT NULL,
  youtube_url TEXT NOT NULL,
  title TEXT,
  channel TEXT,
  status TEXT CHECK (status IN ('pending', 'processing', 'completed', 'failed')) DE-
FAULT 'pending',
  total_frames INT,
  processed_frames INT DEFAULT 0,
  signals_generated INT DEFAULT 0,
  model_version TEXT,
  processing_started_at TIMESTAMP WITH TIME ZONE,
  processing_completed_at TIMESTAMP WITH TIME ZONE,
  error_message TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

-- Índices
CREATE INDEX idx_vision_agent_videos_user_id ON public.vision_agent_videos(user_id);
CREATE INDEX idx_vision_agent_videos_status ON public.vision_agent_videos(status);

-- RLS
ALTER TABLE public.vision_agent_videos ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view their own videos"
ON public.vision_agent_videos FOR SELECT
TO authenticated
USING (auth.uid() = user_id);

CREATE POLICY "Users can insert their own videos"
ON public.vision_agent_videos FOR INSERT
TO authenticated
WITH CHECK (auth.uid() = user_id);

CREATE POLICY "Users can update their own videos"
ON public.vision_agent_videos FOR UPDATE
TO authenticated
USING (auth.uid() = user_id);

-- Tabela para configurações do Vision Agent
CREATE TABLE public.vision_agent_settings (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id UUID REFERENCES auth.users(id) ON DELETE CASCADE NOT NULL UNIQUE,
  enabled BOOLEAN DEFAULT false,
  mode TEXT CHECK (mode IN ('SHADOW', 'PAPER', 'LIVE')) DEFAULT 'SHADOW',
  confidence_threshold NUMERIC(3,2) DEFAULT 0.70,
  youtube_playlist_url TEXT,
  model_version TEXT DEFAULT 'model_seq_v20251125.h5',
  auto_process_new_videos BOOLEAN DEFAULT false,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT now(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

-- RLS
ALTER TABLE public.vision_agent_settings ENABLE ROW LEVEL SECURITY;

CREATE POLICY "Users can view their own settings"
ON public.vision_agent_settings FOR SELECT
```

```
TO authenticated
USING (auth.uid() = user_id);

CREATE POLICY "Users can insert their own settings"
ON public.vision_agent_settings FOR INSERT
TO authenticated
WITH CHECK (auth.uid() = user_id);

CREATE POLICY "Users can update their own settings"
ON public.vision_agent_settings FOR UPDATE
TO authenticated
USING (auth.uid() = user_id);
```

5.3 Backend - Atualizar Tipos TypeScript

Executar após criar as tabelas:

```
cd smc-alpha-dashboard-main
npx supabase gen types typescript --local > src/integrations/supabase/types.ts
```

5.4 Frontend - Novo Componente `VisionAgentPanel`

Criar: `src/components/trading/VisionAgentPanel.tsx`

Localização: Adicionar no sidebar direito do Dashboard, entre `BotControlPanel` e `ActivePosition-sPanel`

Funcionalidades:
- Exibir status do Vision Agent (🟢 Ativo | 🔴 Desativado)
- Mostrar modo atual (SHADOW/PAPER/LIVE)
- Listar últimos vídeos processados
- Exibir contador de sinais gerados hoje
- Botão para ativar/desativar o agente
- Link para configurações avançadas

Exemplo de implementação:

```tsx
// src/components/trading/VisionAgentPanel.tsx
import { useState, useEffect } from "react";
import { Card } from "@/components/ui/card";
import { Button } from "@/components/ui/button";
import { Badge } from "@/components/ui/badge";
import { Eye, EyeOff, Video, Settings } from "lucide-react";
import { supabase } from "@/integrations/supabase/client";
import { useToast } from "@/hooks/use-toast";
import { useAuth } from "@/hooks/useAuth";

export const VisionAgentPanel = () => {
  const { user } = useAuth();
  const { toast } = useToast();
  const [agentEnabled, setAgentEnabled] = useState(false);
  const [mode, setMode] = useState<"SHADOW" | "PAPER" | "LIVE">("SHADOW");
  const [signalsToday, setSignalsToday] = useState(0);
  const [lastVideo, setLastVideo] = useState<any>(null);
  const [loading, setLoading] = useState(false);

  const fetchAgentStatus = async () => {
    if (!user) return;

    try {
      // Buscar configurações do Vision Agent
      const { data: settings } = await supabase
        .from("vision_agent_settings")
        .select("enabled, mode")
        .eq("user_id", user.id)
        .maybeSingle();

      if (settings) {
        setAgentEnabled(settings.enabled);
        setMode(settings.mode);
      }

      // Buscar último vídeo processado
      const { data: videos } = await supabase
        .from("vision_agent_videos")
        .select("*")
        .eq("user_id", user.id)
        .order("created_at", { ascending: false })
        .limit(1);

      if (videos && videos.length > 0) {
        setLastVideo(videos[0]);
      }

      // Buscar sinais gerados hoje
      const today = new Date().toISOString().split("T")[0];
      const { count } = await supabase
        .from("agent_logs")
        .select("id", { count: "exact" })
        .eq("user_id", user.id)
        .eq("agent_name", "vision_trading_agent")
        .like("action", "signal_%")
        .gte("created_at", today);

      setSignalsToday(count || 0);
    } catch (error) {
      console.error("Erro ao buscar status do Vision Agent:", error);
    }
  };
```

```
  useEffect(() => {
    fetchAgentStatus();
    const interval = setInterval(fetchAgentStatus, 10000);
    return () => clearInterval(interval);
  }, [user]);

  const toggleAgent = async () => {
    if (!user) return;
    setLoading(true);

    try {
      // Verificar se já existe configuração
      const { data: existing } = await supabase
        .from("vision_agent_settings")
        .select("id")
        .eq("user_id", user.id)
        .maybeSingle();

      if (existing) {
        // Atualizar
        const { error } = await supabase
          .from("vision_agent_settings")
          .update({ enabled: !agentEnabled })
          .eq("user_id", user.id);

        if (error) throw error;
      } else {
        // Criar
        const { error } = await supabase
          .from("vision_agent_settings")
          .insert({ user_id: user.id, enabled: true });

        if (error) throw error;
      }

      toast({
        title: !agentEnabled ? "🤖 Vision Agent Ativado" : "Vision Agent Desativado",
        description: !agentEnabled
          ? "O agente começará a processar vídeos automaticamente"
          : "Processamento de vídeos pausado",
      });

      fetchAgentStatus();
    } catch (error: any) {
      toast({
        title: "Erro ao alterar status",
        description: error.message,
        variant: "destructive",
      });
    } finally {
      setLoading(false);
    }
  };

  return (
    <Card className="p-4 m-4">
      <div className="flex items-center justify-between mb-4">
        <div className="flex items-center gap-2">
          <Eye className="w-4 h-4 text-muted-foreground" />
          <h3 className="font-bold text-foreground">Vision Agent</h3>
        </div>
        <Badge variant={agentEnabled ? "default" : "outline"}>
```

```
            {agentEnabled ? "🟢 ATIVO" : "🔴 DESATIVADO"}
          </Badge>
        </div>

        <div className="space-y-3">
          {/* Modo */}
          <div className="flex items-center justify-between text-xs">
            <span className="text-muted-foreground">Modo:</span>
            <Badge variant="secondary">{mode}</Badge>
          </div>

          {/* Sinais gerados hoje */}
          <div className="flex items-center justify-between text-xs">
            <span className="text-muted-foreground">Sinais Hoje:</span>
            <span className="font-bold text-foreground">{signalsToday}</span>
          </div>

          {/* Último vídeo */}
          {lastVideo && (
            <div className="flex items-start gap-2 p-2 bg-muted/50 rounded-md">
              <Video className="w-4 h-4 text-muted-foreground mt-0.5 flex-shrink-0" />
              <div className="flex-1 min-w-0">
                <p className="text-xs font-medium text-foreground truncate">
                  {lastVideo.title || "Vídeo sem título"}
                </p>
                <p className="text-[10px] text-muted-foreground">
                  {lastVideo.status === "completed"
                    ? `✅ ${lastVideo.signals_generated} sinais`
                    : lastVideo.status === "processing"
                    ? "⏳ Processando..."
                    : "❌ Erro"}
                </p>
              </div>
            </div>
          )}

          {/* Botões */}
          <div className="grid grid-cols-2 gap-2 pt-2">
            <Button
              onClick={toggleAgent}
              disabled={loading}
              size="sm"
              variant={agentEnabled ? "destructive" : "default"}
            >
              {agentEnabled ? (
                <>
                  <EyeOff className="w-4 h-4 mr-1" />
                  DESATIVAR
                </>
              ) : (
                <>
                  <Eye className="w-4 h-4 mr-1" />
                  ATIVAR
                </>
              )}
            </Button>

            <Button size="sm" variant="outline">
              <Settings className="w-4 h-4 mr-1" />
              CONFIG
            </Button>
          </div>
        </div>
```

```
    </Card>
  );
};
```

5.5 Frontend - Integrar no Dashboard

Editar: `src/pages/Dashboard.tsx`

```
// Adicionar import
import { VisionAgentPanel } from "@/components/trading/VisionAgentPanel";

// Adicionar no sidebar (após BotControlPanel)
<div className="w-96 flex flex-col border-l border-border">
  <div className="h-full overflow-y-auto pb-4">
    <BotControlPanel />
    <VisionAgentPanel />  {/* ⭐ NOVO */}
    <ActivePositionsPanel />
    <AccountPanel />
    <SMCPanel symbol={symbol} interval={interval} />
    <TradingLogsPanel />
  </div>
</div>
```

5.6 Frontend - Atualizar ActivePositionsPanel

Editar: `src/components/trading/ActivePositionsPanel.tsx`

Adicionar badge indicando se a posição veio do Vision Agent:

```
// Dentro do map de posições
{position.agents?.source === 'vision_trading_agent' && (
  <Badge variant="outline" className="text-[10px]">
    👁️ Vision Agent
  </Badge>
)}
```

5.7 Frontend - Criar Página de Configurações

Criar: `src/pages/VisionAgentSettings.tsx`

Funcionalidades:
- Configurar URL da playlist do YouTube
- Selecionar modo (SHADOW/PAPER/LIVE)
- Ajustar confidence threshold
- Ativar/desativar processamento automático
- Visualizar histórico de vídeos processados
- Fazer upload manual de vídeo para teste

5.8 Backend - Serviço Python do Vision Agent

Criar: `vision-agent-service/` (diretório separado do dashboard)

Estrutura:

```
vision-agent-service/
├── src/
│   ├── agent/
│   │   ├── video_processor.py      # Processar frames
│   │   ├── feature_extractor.py    # MediaPipe, OCR, YOLO
│   │   ├── model_inference.py      # LSTM/Transformer
│   │   └── signal_sender.py        # Enviar para Supabase
│   ├── models/
│   │   └── model_seq_v20251125.h5  # Modelo treinado
│   ├── config/
│   │   └── config.yaml             # Configurações
│   ├── main.py                     # Entry point
│   └── scheduler.py                # Processar playlist periodicamente
├── requirements.txt
├── Dockerfile
└── README.md
```

Comunicação com o Dashboard:
- Fazer requisições HTTP POST para a Edge Function `vision-agent-signal`
- Usar token de autenticação seguro
- Atualizar progresso na tabela `vision_agent_videos`

🛡️ 6. Segurança e Validações

6.1 Autenticação do Vision Agent

Opções:

1. **Service Role Key** (recomendado para MVP)
   - O Vision Agent usa o `SUPABASE_SERVICE_ROLE_KEY`
   - Passar `user_id` no payload

2. **Token JWT por Usuário**
   - Cada usuário gera um token exclusivo para seu Vision Agent
   - Armazenar na tabela `vision_agent_settings.api_token`

3. **API Key dedicada**
   - Criar sistema de API keys na tabela `vision_agent_api_keys`

6.2 Validações Obrigatórias

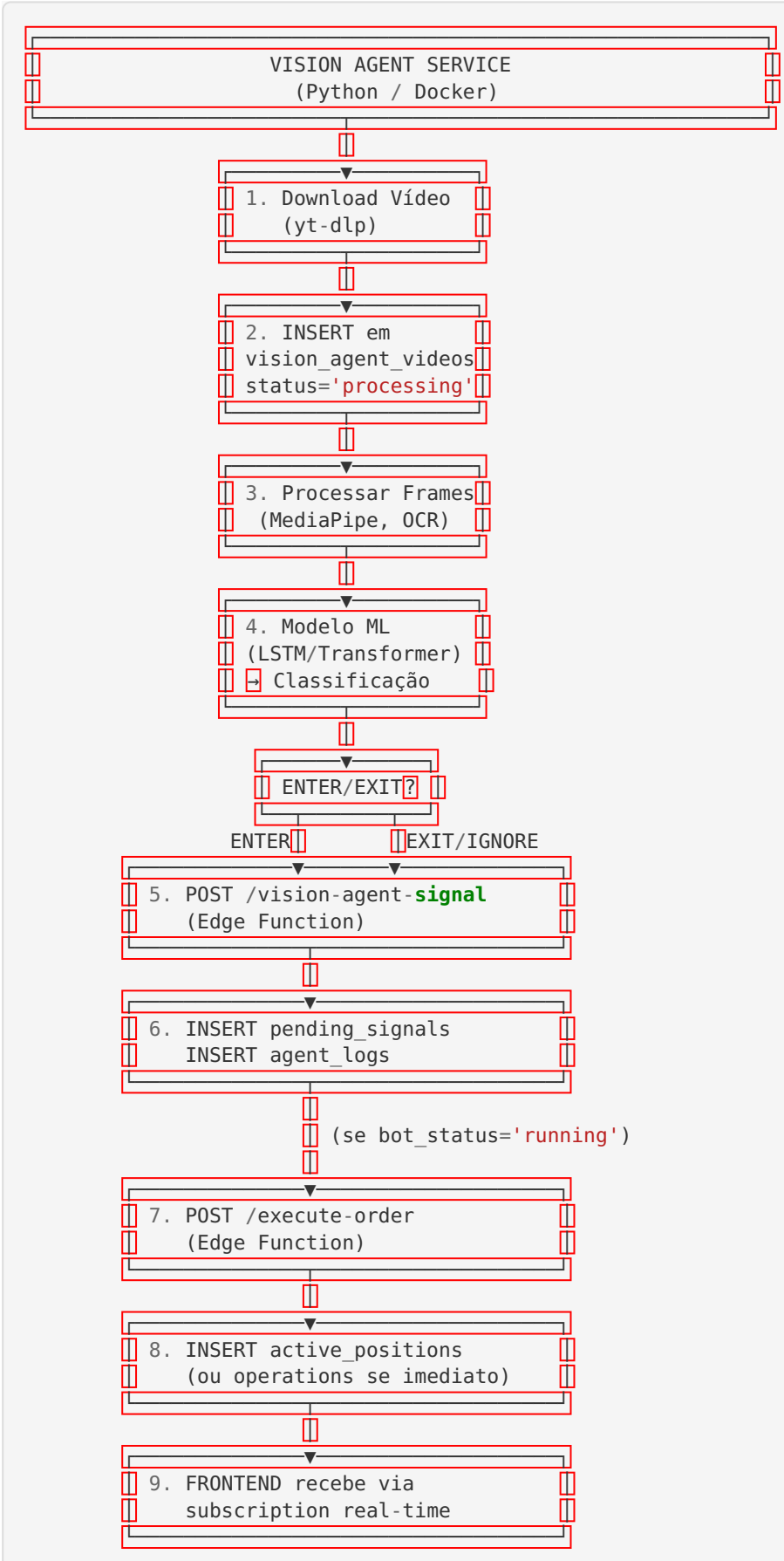
Na Edge Function `vision-agent-signal`:
- ✅ Validar estrutura do payload
- ✅ Verificar se `bot_status === "running"`
- ✅ Verificar `confidence >= threshold` configurado
- ✅ Validar símbolo/asset é suportado
- ✅ Verificar limites de trades diários
- ✅ Validar risk management (não ultrapassar max_positions)
- ✅ Modo PAPER vs LIVE (não executar em LIVE se papel)

6.3 Rate Limiting

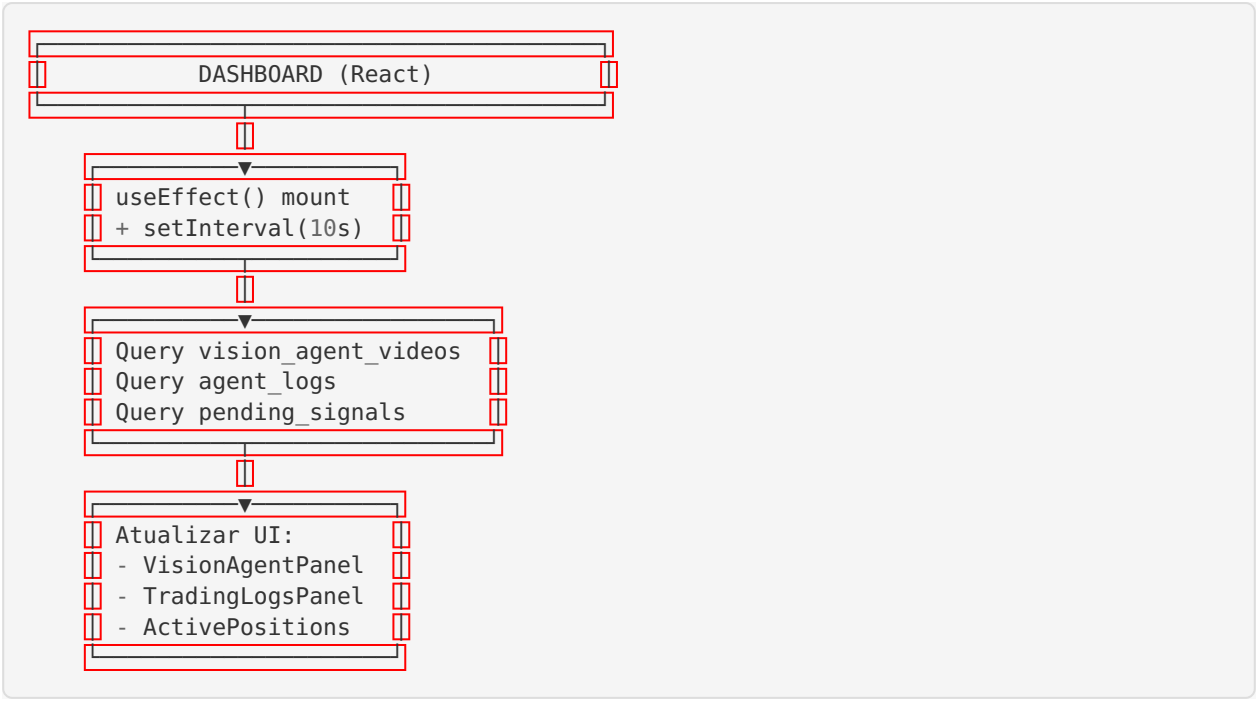
- Limitar quantidade de sinais por minuto (ex: máx 10/min)
- Implementar cooldown entre sinais do mesmo asset

📊 7. Fluxo Completo de Integração

7.1 Fluxo de Processamento de Vídeo



7.2 Fluxo de Exibição no Frontend



🚀 8. Roadmap de Implementação

Fase 1: Backend Foundation (Semana 1)

- ✅ Criar tabelas `vision_agent_videos` e `vision_agent_settings`
- ✅ Criar Edge Function `vision-agent-signal`
- ✅ Atualizar tipos TypeScript
- ✅ Testar autenticação e validações

Fase 2: Frontend UI (Semana 1-2)

- ✅ Criar componente `VisionAgentPanel`
- ✅ Integrar no Dashboard
- ✅ Atualizar `ActivePositionsPanel` para mostrar origem
- ✅ Criar página de configurações básica

Fase 3: Vision Agent Service (Semana 2-3)

- ✅ Implementar `video_processor.py`
- ✅ Integrar MediaPipe + OpenCV + OCR
- ✅ Implementar modelo LSTM/Transformer
- ✅ Criar `signal_sender.py` para comunicação com Edge Function
- ✅ Dockerizar o serviço

Fase 4: Testes & Validação (Semana 3-4)

- ✅ Testar em modo SHADOW com vídeos reais
- ✅ Validar qualidade dos sinais
- ✅ Ajustar thresholds e parâmetros
- ✅ Testar modo PAPER

Fase 5: Produção (Semana 4+)

- ✅ Deploy do Vision Agent Service
- ✅ Configurar monitoramento e logs
- ✅ Habilitar modo LIVE (após validação completa)
- ✅ Documentação final

🎯 9. Checklist de Integração

Backend

- [ ] Criar migração para novas tabelas
- [ ] Implementar Edge Function `vision-agent-signal`
- [ ] Atualizar tipos TypeScript
- [ ] Configurar RLS nas novas tabelas
- [ ] Criar índices necessários
- [ ] Documentar API da Edge Function

Frontend

- [ ] Criar `VisionAgentPanel.tsx`
- [ ] Atualizar `Dashboard.tsx`
- [ ] Modificar `ActivePositionsPanel.tsx` (badge)
- [ ] Criar página de configurações
- [ ] Adicionar rota no `App.tsx`
- [ ] Criar hook `useVisionAgent.ts`

Vision Agent Service

- [ ] Estrutura de diretórios
- [ ] Implementar processamento de vídeo
- [ ] Integrar MediaPipe/OCR/YOLO
- [ ] Implementar modelo ML
- [ ] Criar comunicação com Supabase
- [ ] Dockerizar
- [ ] Configurar variáveis de ambiente
- [ ] Implementar scheduler para processamento contínuo

Testes

- [ ] Testar Edge Function com Postman
- [ ] Testar criação de sinais
- [ ] Testar execução de ordens
- [ ] Testar UI no navegador
- [ ] Testar Vision Agent com vídeo de exemplo
- [ ] Testar modo SHADOW end-to-end
- [ ] Validar logs e auditoria

Documentação

- [ ] README do Vision Agent Service
- [ ] Documentação da API
- [ ] Guia de configuração para usuários
- [ ] Diagrama de arquitetura atualizado

💡 10. Considerações Importantes

10.1 Não Quebrar Funcionalidades Existentes

✅ **Garantias**:
- Nenhuma tabela existente será modificada (apenas novas tabelas)
- Nenhum componente existente será alterado estruturalmente
- Design permanece idêntico (apenas adiciona novos componentes)
- Edge Functions existentes não serão modificadas
- Lógica de trading atual permanece intacta

10.2 Modularidade

✅ **Arquitetura**:
- Vision Agent é completamente **opcional** e pode ser desativado
- Funciona de forma **independente** do sistema principal
- Usa **tabelas dedicadas** para evitar conflitos
- Comunicação via **API bem definida** (Edge Function)

10.3 Performance

✅ **Otimizações**:
- Vision Agent roda em **serviço separado** (Python/Docker)
- Não impacta performance do frontend React
- Edge Functions são serverless e escaláveis
- Processamento de vídeo é assíncrono

10.4 Segurança

✅ **Medidas**:
- Autenticação obrigatória via token
- Validações rigorosas no backend
- Modo SHADOW como padrão (seguro)
- Validações de risk management antes de executar ordens
- Logs completos para auditoria

📝 11. Resumo Executivo

O **Vision Trading Agent** pode ser integrado ao **SMC Alpha Dashboard** de forma **completamente modular e não-invasiva**, seguindo estes princípios:

✅ O que NÃO será alterado:

- Design e layout existente

- Tabelas do banco de dados atuais
- Edge Functions existentes
- Lógica de trading atual
- Componentes React atuais

✅ O que será ADICIONADO:

- **2 novas tabelas**: `vision_agent_videos`, `vision_agent_settings`
- **1 nova Edge Function**: `vision-agent-signal`
- **1 novo componente**: `VisionAgentPanel`
- **1 serviço externo**: Vision Agent Service (Python)
- **1 nova página**: Configurações do Vision Agent

✅ Pontos de Integração Identificados:

1. **Backend**: Edge Function para receber sinais
2. **Banco de Dados**: Novas tabelas dedicadas
3. **Frontend**: Novo painel no sidebar do Dashboard
4. **API**: Comunicação via HTTP POST segura
5. **Logs**: Integração com `agent_logs` e `TradingLogsPanel`

✅ Segurança:

- Modo SHADOW como padrão
- Validações obrigatórias
- Autenticação robusta
- Auditoria completa

🎬 12. Próximos Passos

1. **Revisar e aprovar** este documento de integração
2. **Criar branch** no Git: `feature/vision-agent-integration`
3. **Implementar Fase 1** (Backend Foundation)
4. **Testar isoladamente** cada componente
5. **Implementar Fase 2** (Frontend UI)
6. **Implementar Fase 3** (Vision Agent Service)
7. **Testes completos** em ambiente de desenvolvimento
8. **Deploy gradual** (SHADOW → PAPER → LIVE)

Documento criado em: 25 de Novembro de 2025
Versão: 1.0
Status: ✅ Análise Completa e Pronta para Implementação