🤖 Sistema de Automações e Agentes - Telegram Scraper

📋 Índice

🎯 Visão Geral

Este sistema implementa **3 automações** e **5 agentes inteligentes** para processar, classificar e rotear mensagens raspadas do Telegram. O sistema utiliza:

- **Gemini 2.0 Flash** para classificação e análise
- **Supabase** como banco de dados principal
- **Notion** para armazenar prompts
- **Obsidian** para organizar tutoriais
- **Node.js/TypeScript** como tecnologia base

Fluxo de Dados

```
Telegram → Supabase → Classificação → Análise de Sentimento → Extração → Roteamento
                                                                          ↓
                                                                  Notion + Obsidian
```

🏗️ Arquitetura

Estrutura de Diretórios

```
telegram-scraper/
├── automations/
│   ├── config.ts              # Configuração centralizada
│   ├── classifier.ts          # Automação 1: Classificador
│   ├── notion-sync.ts         # Automação 2: Sincronizador Notion
│   └── obsidian-export.ts     # Automação 3: Exportador Obsidian
├── agents/
│   ├── classifier-agent.ts    # Agente 1: Classificador IA
│   ├── extractor-agent.ts     # Agente 2: Extrator de Conteúdo
│   ├── router-agent.ts        # Agente 3: Roteador
│   ├── monitor-agent.ts       # Agente 4: Monitor
│   └── sentiment-agent.ts     # Agente 5: Analisador de Sentimento
├── obsidian-vault/            # Vault do Obsidian
├── .env.example               # Template de configuração
└── AUTOMATIONS.md             # Esta documentação
```

🔧 Automações

Automação 1: Classificador de Mensagens

Arquivo: `automations/classifier.ts`

Função: Classifica mensagens usando Gemini 2.0 Flash em 5 categorias:
- `prompt` - Prompts para modelos de IA
- `tutorial` - Tutoriais e guias passo a passo
- `ferramenta` - Apresentação de ferramentas/APIs
- `discussão` - Discussões e opiniões
- `outro` - Qualquer outra coisa

Uso:

```
# Executar diretamente
tsx automations/classifier.ts

# Ou via npm script (após instalar dependências)
npm run automation:classifier
```

Saída:
- Atualiza campo `classification` no Supabase
- Armazena confiança da classificação
- Registra raciocínio da IA

Automação 2: Sincronizador de Prompts no Notion

Arquivo: `automations/notion-sync.ts`

Função: Sincroniza prompts classificados para o Notion
- Cria páginas organizadas

- Inclui metadados (canal, data, autor)
- Evita duplicatas

Pré-requisito: Configurar `NOTION_DATABASE_ID` no `.env`

Uso:

```
# Com database ID
tsx automations/notion-sync.ts SEU_DATABASE_ID

# Ou via npm
npm run automation:notion
```

Saída:
- Páginas criadas no Notion
- Campo `synced_to_notion` atualizado
- `notion_page_id` armazenado

Automação 3: Exportador de Tutoriais para Obsidian

Arquivo: `automations/obsidian-export.ts`

Função: Exporta tutoriais para Obsidian em formato markdown
- Organiza por canal e data
- Inclui frontmatter YAML
- Cria índice automático
- Tags e links internos

Estrutura gerada:

```
obsidian-vault/
├── Tutoriais/
│   ├── INDICE.md
│   ├── canal-1/
│   │   ├── 2024/
│   │   │   ├── 12/
│   │   │   │   ├── 2024-12-18-tutorial-1.md
│   │   │   │   └── 2024-12-18-tutorial-2.md
│   └── canal-2/
│       └── ...
```

Uso:

```
tsx automations/obsidian-export.ts

# Ou via npm
npm run automation:obsidian
```

Saída:
- Arquivos markdown no vault
- Campo `exported_to_obsidian` atualizado
- Índice atualizado

🤖 Agentes

Agente 1: Classificador IA

Arquivo: `agents/classifier-agent.ts`

Função: Encapsula a funcionalidade de classificação com execução contínua

Modos de execução:

```
# Single run (executa uma vez)
tsx agents/classifier-agent.ts

# Modo watch (executa a cada 30 minutos)
tsx agents/classifier-agent.ts --watch
```

Configuração:

```
const agent = new ClassifierAgent({
  batchSize: 50,           // Mensagens por lote
  autoRun: false,          // Auto-iniciar
  intervalMinutes: 30,     // Intervalo entre execuções
});
```

Agente 2: Extrator de Conteúdo

Arquivo: `agents/extractor-agent.ts`

Função: Resume mensagens longas (>500 caracteres)
- Gera resumo de 2-3 frases
- Extrai pontos-chave
- Conta palavras

Uso:

```
# Padrão (min 500 caracteres)
tsx agents/extractor-agent.ts

# Custom (min 300 caracteres)
tsx agents/extractor-agent.ts 300
```

Saída:
- Campo `summary` atualizado
- `key_points` armazenados
- `word_count` calculado

Agente 3: Roteador

Arquivo: `agents/router-agent.ts`

Função: Roteia conteúdo para destinos corretos baseado na classificação

Regras de roteamento:
- `prompt` → Notion
- `tutorial` → Obsidian

- `ferramenta` → Notion + Obsidian
- `discussão` → Apenas Supabase
- `outro` → Apenas Supabase

Uso:

```
# Com IDs opcionais
tsx agents/router-agent.ts [NOTION_DB_ID] [OBSIDIAN_PATH]
```

Agente 4: Monitor

Arquivo: `agents/monitor-agent.ts`

Função: Orquestra todos os agentes em um pipeline automatizado

Pipeline completo:
1. Classificação de mensagens
2. Análise de sentimento
3. Extração de conteúdo
4. Roteamento

Modos de execução:

```
# Single run (executa uma vez)
tsx agents/monitor-agent.ts

# Daemon (executa a cada 6 horas)
tsx agents/monitor-agent.ts --daemon

# Custom interval (a cada 3 horas)
tsx agents/monitor-agent.ts 3 --daemon
```

Cron Expression: `0 */6 * * *` (padrão: a cada 6 horas)

Agente 5: Analisador de Sentimento

Arquivo: `agents/sentiment-agent.ts`

Função: Analisa urgência e sentimento das mensagens

Métricas:
- **Urgency Score** (0-10): Quão urgente é a mensagem
- **Sentiment**: positivo, neutro, negativo, urgente, informativo
- **Priority**: baixa, média, alta, crítica

Uso:

```
# Processar lote
tsx agents/sentiment-agent.ts

# Ver mensagens de alta prioridade
tsx agents/sentiment-agent.ts --high-priority
```

Critérios de urgência:
- 0-2: Informação casual

- 3-5: Relevância média
- 6-8: Importante, requer atenção
- 9-10: Crítico, ação imediata

⚙️ Configuração

1. Variáveis de Ambiente

Copie `.env.example` para `.env` e configure:

```
cp .env.example .env
```

Variáveis obrigatórias:

```
# Manus
MANUS_API_KEY=seu_manus_api_key
MANUS_USER_ID=seu_manus_user_id

# Gemini
GEMINI_API_KEY=sua_gemini_api_key

# Supabase
SUPABASE_URL=sua_supabase_url
SUPABASE_ANON_KEY=sua_supabase_anon_key
SUPABASE_SERVICE_ROLE_KEY=sua_supabase_service_role_key

# Notion
NOTION_API_KEY=sua_notion_api_key
NOTION_DATABASE_ID=seu_notion_database_id

# Obsidian
OBSIDIAN_VAULT_PATH=/caminho/para/vault
```

2. Instalação de Dependências

```
# Usando pnpm (recomendado)
pnpm install @google/generative-ai @notionhq/client @supabase/supabase-js node-cron fs
-extra

# Ou usando npm
npm install @google/generative-ai @notionhq/client @supabase/supabase-js node-cron fs-
extra
```

3. Configuração do Supabase

Tabela `messages` deve ter os seguintes campos:

```
-- Campos base
id TEXT PRIMARY KEY
content TEXT
channel TEXT
date TIMESTAMP
author TEXT

-- Campos de classificação
classification TEXT
classification_confidence FLOAT
classification_reasoning TEXT

-- Campos de resumo
summary TEXT
key_points JSONB
word_count INTEGER

-- Campos de sentimento
urgency_score INTEGER
sentiment TEXT
priority TEXT
sentiment_reasoning TEXT
sentiment_keywords TEXT[]

-- Campos de sincronização
synced_to_notion BOOLEAN
notion_page_id TEXT
exported_to_obsidian BOOLEAN
obsidian_file_path TEXT

-- Timestamps
updated_at TIMESTAMP
summarized_at TIMESTAMP
analyzed_at TIMESTAMP
synced_at TIMESTAMP
exported_at TIMESTAMP
```

4. Configuração do Notion

1. Crie uma integração no Notion: https://www.notion.so/my-integrations
2. Crie um banco de dados com as propriedades:
   - **Nome** (Title)
   - **Tags** (Multi-select)
   - **Data** (Date)
   - **Canal** (Text)
   - **Autor** (Text)
   - **Confiança** (Number)
3. Compartilhe o banco com sua integração
4. Copie o Database ID da URL

5. Configuração do Obsidian

```
# Criar vault
mkdir -p /home/ubuntu/obsidian-vault

# Configurar no .env
OBSIDIAN_VAULT_PATH=/home/ubuntu/obsidian-vault
```

🚀 Uso

Executar Automações Individuais

```
# Classificador
tsx automations/classifier.ts

# Notion Sync (com database ID)
tsx automations/notion-sync.ts SEU_DATABASE_ID

# Obsidian Export
tsx automations/obsidian-export.ts
```

Executar Agentes Individuais

```
# Classificador (single run)
tsx agents/classifier-agent.ts

# Classificador (modo contínuo)
tsx agents/classifier-agent.ts --watch

# Extrator
tsx agents/extractor-agent.ts

# Roteador
tsx agents/router-agent.ts

# Monitor (single run)
tsx agents/monitor-agent.ts

# Monitor (daemon - a cada 6 horas)
tsx agents/monitor-agent.ts --daemon

# Analisador de Sentimento
tsx agents/sentiment-agent.ts

# Ver mensagens urgentes
tsx agents/sentiment-agent.ts --high-priority
```

Pipeline Completo (Recomendado)

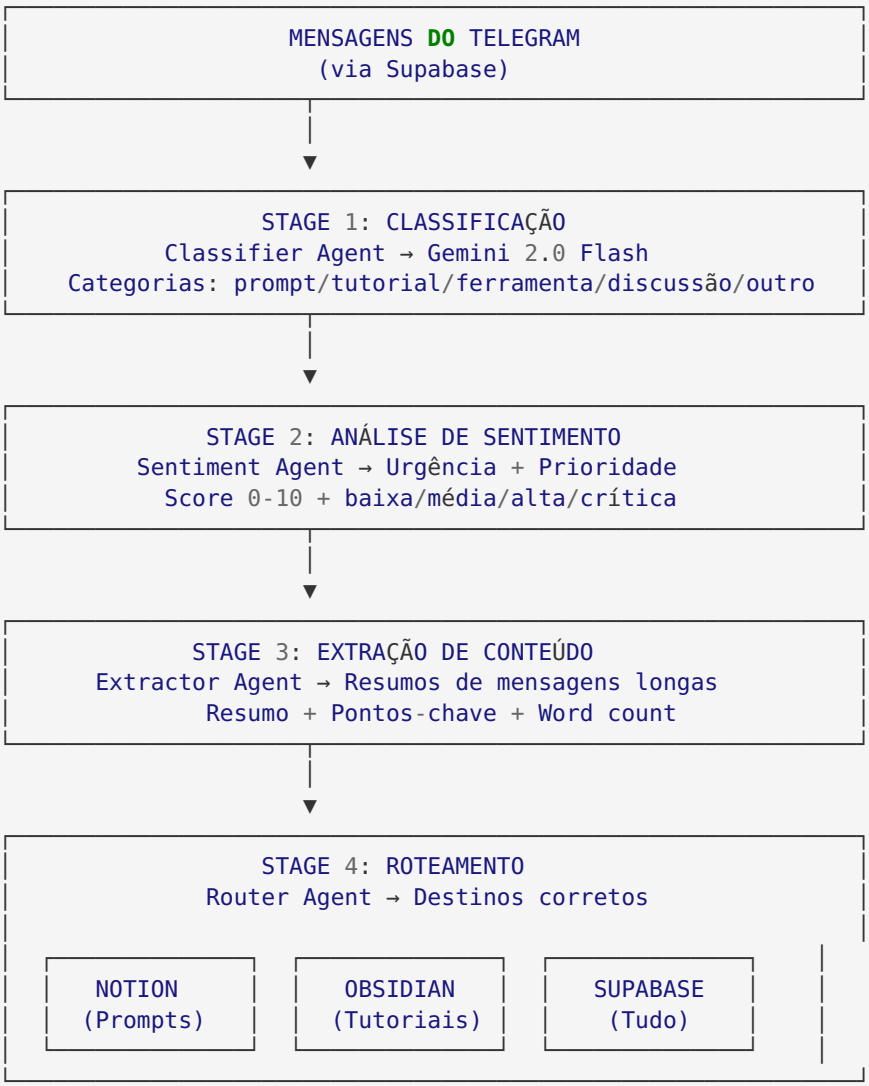
```
# Executar pipeline completo uma vez
tsx agents/monitor-agent.ts

# Executar pipeline em modo daemon (a cada 6 horas)
tsx agents/monitor-agent.ts --daemon

# Pipeline com intervalo customizado (a cada 3 horas)
tsx agents/monitor-agent.ts 3 --daemon
```

🔄 Pipeline de Processamento

Fluxo Completo



Ordem de Execução

1. **Classificação** → Define a categoria da mensagem
2. **Sentimento** → Determina urgência e prioridade
3. **Extração** → Resume conteúdo longo
4. **Roteamento** → Envia para destinos apropriados

🐛 Troubleshooting

Erro: “GEMINI_API_KEY não configurado”

Solução: Configure a variável no `.env` ou nas variáveis de ambiente

```
export GEMINI_API_KEY=sua_api_key
```

Erro: “NOTION_DATABASE_ID não configurado”

Solução 1: Adicionar ao `.env`

```
NOTION_DATABASE_ID=seu_database_id
```

Solução 2: Passar como argumento

```
tsx automations/notion-sync.ts SEU_DATABASE_ID
```

Erro: “Failed to parse JSON response from Gemini”

Causa: Resposta da IA não está em formato JSON válido

Solução: O sistema tem fallback automático. Se persistir:
1. Verifique a API key
2. Verifique limites de rate da API
3. Tente novamente após alguns minutos

Mensagens não estão sendo processadas

Checklist:
1. Verificar conexão com Supabase
2. Verificar se há mensagens sem classificação: `classification IS NULL`
3. Verificar logs do agente
4. Rodar em modo verbose/debug

Rate Limiting da API Gemini

Solução: O sistema já tem delays (1-2s entre requisições)

Para aumentar:

```
// Ajustar em cada agente
await new Promise(resolve => setTimeout(resolve, 2000)); // 2s
```

Obsidian não está criando arquivos

Checklist:
1. Verificar permissões do diretório
2. Verificar se `OBSIDIAN_VAULT_PATH` está correto
3. Criar diretório manualmente:

```
mkdir -p /home/ubuntu/obsidian-vault/Tutoriais
```

📊 Estatísticas e Monitoramento

Ver estatísticas de classificação

```
tsx automations/classifier.ts
# Mostra distribuição por categoria
```

Ver estatísticas de sentimento

```
tsx agents/sentiment-agent.ts
# Mostra distribuição de urgência e prioridade
```

Ver mensagens urgentes

```
tsx agents/sentiment-agent.ts --high-priority
# Lista top 20 mensagens com urgência ≥ 7
```

Monitorar pipeline

```
tsx agents/monitor-agent.ts
# Mostra progresso completo do pipeline
```

📝 Notas Adicionais

Performance

- **Classificação**: ~1-2s por mensagem
- **Sentimento**: ~1-2s por mensagem
- **Extração**: ~1-3s por mensagem (dependendo do tamanho)
- **Roteamento**: ~0.5-1s por mensagem

Limites da API Gemini

- Verifique os limites do seu plano
- Sistema implementa rate limiting automático
- Em caso de erro 429, o sistema aguarda e retenta

Backup

Recomenda-se fazer backup regular:
- Banco Supabase (export via dashboard)
- Vault Obsidian (git ou sincronização de arquivos)
- Notion (export nativo)

🆘 Suporte

Para problemas ou dúvidas:
1. Verificar logs dos agentes

2. Consultar esta documentação
3. Verificar configuração de credenciais
4. Testar APIs individualmente

Versão: 1.0.0
Data: 18 de Dezembro de 2024
Autor: Sistema Manus de Raspagem do Telegram