

PROMPT DE EXECUÇÃO

Guia Completo de Implementação

Dashboard Analítico + Agentes de IA

Clínica Médica Dr. Igor

DOCUMENTO PARA DESENVOLVEDOR

Siga cada passo na ordem apresentada

INSTRUÇÕES GERAIS

Este documento contém instruções detalhadas para implementação do sistema. Siga cada etapa na ordem apresentada.

REGRAS IMPORTANTES:

1. NÃO pule etapas. Cada uma depende da anterior.
2. TESTE cada integração antes de avançar.
3. DOCUMENTE problemas encontrados e soluções.
4. USE Git para versionamento desde o início.
5. NUNCA commite credenciais no repositório.

SEGUIE URL DO REPOSITÓRIO DO GITHUB:

https://github.com/mateusolintof/dashboard_igor.git

ETAPA 1: SETUP DO PROJETO

1.1 Criar Repositório Git e Fazer Push para Repo Remoto do Github

```
git init
git branch -M main
```

1.2 Estrutura de Pastas

Crie a seguinte estrutura:

```
dashboard-clinica/
├── frontend/      # Next.js
├── backend/       # FastAPI
├── docker/        # Dockerfiles
├── docs/          # Documentação
└── docker-compose.yml
```

1.3 Setup Frontend (Next.js)

```
npx create-next-app@latest frontend --typescript --tailwind --eslint --app
--src-dir
```

Quando perguntado:

- Would you like to use Turbopack? → No
- Would you like to customize the default import alias? → No

Instale as dependências:

```
cd frontend
npm install @tanstack/react-query zustand recharts framer-motion
npm install lucide-react date-fns axios
npx shadcn@latest init
```

Configure o shadcn/ui:

- Style: Default
- Base color: Slate
- CSS variables: Yes

Instale componentes shadcn:

```
npx shadcn@latest add button card input select tabs table badge
npx shadcn@latest add dropdown-menu dialog sheet tooltip separator
```

1.4 Setup Backend (FastAPI)

```
cd ../backend
python -m venv venv
source venv/bin/activate # Linux/Mac
# ou: venv\Scripts\activate # Windows
```

Crie requirements.txt:

```
fastapi==0.109.0
uvicorn[standard]==0.27.0
```

```
sqlalchemy[asyncio]==2.0.25
asynccpg==0.29.0
alembic==1.13.1
pydantic==2.6.0
pydantic-settings==2.1.0
redis==5.0.1
celery==5.3.6
httpx==0.26.0
python-multipart==0.0.6
python-jose[cryptography]==3.3.0
anthropic==0.18.0
google-generativeai==0.4.0
aiofiles==23.2.1
pip install -r requirements.txt
```

1.5 Estrutura do Backend

Crie a seguinte estrutura dentro de /backend:

```
backend/
├── app/
│   ├── __init__.py
│   ├── main.py           # Entry point FastAPI
│   ├── config.py         # Settings/Environment
│   ├── database.py       # SQLAlchemy setup
│   └── api/
│       ├── __init__.py
│       └── routes/
│           ├── __init__.py
│           ├── dashboard.py
│           ├── leads.py
│           ├── campaigns.py
│           ├── instagram.py
│           └── agents.py
│       ├── deps.py       # Dependencies
│       └── models/       # SQLAlchemy models
│           ├── __init__.py
│           ├── lead.py
│           ├── campaign.py
│           └── ...
│       ├── schemas/      # Pydantic schemas
│           ├── __init__.py
│           ├── lead.py
│           └── ...
│       ├── services/     # Business logic
│           ├── __init__.py
│           ├── kommo.py
│           ├── meta_ads.py
│           ├── google_ads.py
│           ├── instagram.py
│           └── agents/
│               ├── __init__.py
│               ├── campaign_analyst.py
│               ├── lead_analyst.py
│               └── copywriter.py
│       └── utils/
│           ├── __init__.py
│           └── helpers.py
├── alembic/              # Migrations
├── tests/
├── requirements.txt
└── alembic.ini
```


1.6 Arquivo de Configuração (config.py)

Crie o arquivo app/config.py:

```
from pydantic_settings import BaseSettings
from functools import lru_cache

class Settings(BaseSettings):
    # App
    APP_NAME: str = 'Dashboard Clinica Dr Igor'
    DEBUG: bool = False

    # Database
    DATABASE_URL: str
    REDIS_URL: str = 'redis://localhost:6379/0'

    # Kommo CRM
    KOMMO_DOMAIN: str
    KOMMO_CLIENT_ID: str
    KOMMO_CLIENT_SECRET: str
    KOMMO_ACCESS_TOKEN: str
    KOMMO_REFRESH_TOKEN: str

    # Meta Ads
    META_APP_ID: str
    META_APP_SECRET: str
    META_ACCESS_TOKEN: str
    META_AD_ACCOUNT_ID: str

    # Google Ads
    GOOGLE_ADS_DEVELOPER_TOKEN: str
    GOOGLE_ADS_CLIENT_ID: str
    GOOGLE_ADS_CLIENT_SECRET: str
    GOOGLE_ADS_REFRESH_TOKEN: str
    GOOGLE_ADS_CUSTOMER_ID: str

    # Instagram
    INSTAGRAM_BUSINESS_ID: str
    INSTAGRAM_ACCESS_TOKEN: str

    # LLMs
    ANTHROPIC_API_KEY: str
    GOOGLE_AI_API_KEY: str
    PERPLEXITY_API_KEY: str = ''

    # Agenda
    AGENDA_API_URL: str = ''
```

```
AGENDA_API_KEY: str = ''

class Config:
    env_file = '.env'

@lru_cache()
def get_settings():
    return Settings()
```

1.7 Database Setup (database.py)

Crie o arquivo app/database.py:

```
from sqlalchemy.ext.asyncio import create_async_engine, AsyncSession
from sqlalchemy.ext.asyncio import async_sessionmaker
from sqlalchemy.orm import DeclarativeBase
from app.config import get_settings

settings = get_settings()

engine = create_async_engine(
    settings.DATABASE_URL,
    echo=settings.DEBUG,
    pool_size=20,
    max_overflow=10
)

AsyncSessionLocal = async_sessionmaker(
    engine,
    class_=AsyncSession,
    expire_on_commit=False
)

class Base(DeclarativeBase):
    pass

async def get_db():
    async with AsyncSessionLocal() as session:
        try:
            yield session
        finally:
            await session.close()
```

1.8 Main Application (main.py)

Crie o arquivo app/main.py:

```
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.config import get_settings
from app.api.routes import dashboard, leads, campaigns, instagram, agents

settings = get_settings()

app = FastAPI(
    title=settings.APP_NAME,
```

```
        version='1.0.0',
        docs_url='/docs',
        redoc_url='/redoc'
    )

app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'], # Em produção, especifique domínios
    allow_credentials=True,
    allow_methods=['*'],
    allow_headers=['*'],
)

# Rotas
app.include_router(dashboard.router, prefix='/api/dashboard',
tags=['Dashboard'])
app.include_router(leads.router, prefix='/api/leads', tags=['Leads'])
app.include_router(campaigns.router, prefix='/api/campaigns',
tags=['Campaigns'])
app.include_router(instagram.router, prefix='/api/instagram',
tags=['Instagram'])
app.include_router(agents.router, prefix='/api/agents', tags=['AI Agents'])

@app.get('/health')
async def health_check():
    return {'status': 'healthy'}
```

ETAPA 2: MODELOS DO BANCO DE DADOS

2.1 Modelo de Lead (models/lead.py)

```
from sqlalchemy import Column, Integer, String, DateTime, ForeignKey, Enum, Text
from sqlalchemy.orm import relationship
from datetime import datetime
import enum
from app.database import Base

class PipelineType(enum.Enum):
    ATENDIMENTO_IA = 'atendimento_ia'
    ATENDIMENTO_HUMANO = 'atendimento_humano'
    DR_IGOR = 'dr_igor'

class Pipeline(Base):
    __tablename__ = 'pipelines'
    __table_args__ = {'schema': 'crm'}

    id = Column(Integer, primary_key=True)
    kommo_id = Column(Integer, unique=True, index=True)
    nome = Column(String(255), nullable=False)
    tipo = Column(Enum(PipelineType), nullable=False)

    etapas = relationship('Etapa', back_populates='pipeline')
    leads = relationship('Lead', back_populates='pipeline')

class Etapa(Base):
    __tablename__ = 'etapas'
    __table_args__ = {'schema': 'crm'}

    id = Column(Integer, primary_key=True)
    kommo_id = Column(Integer, unique=True, index=True)
    pipeline_id = Column(Integer, ForeignKey('crm.pipelines.id'))
    nome = Column(String(255), nullable=False)
    ordem = Column(Integer, default=0)

    pipeline = relationship('Pipeline', back_populates='etapas')
    leads = relationship('Lead', back_populates='etapa')

class Lead(Base):
    __tablename__ = 'leads'
    __table_args__ = {'schema': 'crm'}

    id = Column(Integer, primary_key=True)
```

```
kommo_id = Column(Integer, unique=True, index=True)
nome = Column(String(255))
telefone = Column(String(50))
email = Column(String(255))
pipeline_id = Column(Integer, ForeignKey('crm.pipelines.id'))
etapa_id = Column(Integer, ForeignKey('crm.etapas.id'))
origem = Column(String(100))
campanha_id = Column(Integer, ForeignKey('marketing.campanhas.id'),
nullable=True)
utm_source = Column(String(100))
utm_medium = Column(String(100))
utm_campaign = Column(String(255))
created_at = Column(DateTime, default=datetime.utcnow)
updated_at = Column(DateTime, default=datetime.utcnow,
onupdate=datetime.utcnow)

pipeline = relationship('Pipeline', back_populates='leads')
etapa = relationship('Etapa', back_populates='leads')
historico = relationship('LeadHistorico', back_populates='lead')
```

2.2 Modelo de Campanha (models/campaign.py)

```

from sqlalchemy import Column, Integer, String, DateTime, Float, Enum, Text
from sqlalchemy.orm import relationship
from datetime import datetime
import enum
from app.database import Base

class Plataforma(enum.Enum):
    META = 'meta'
    GOOGLE = 'google'

class Campanha(Base):
    __tablename__ = 'campanhas'
    __table_args__ = {'schema': 'marketing'}

    id = Column(Integer, primary_key=True)
    plataforma = Column(Enum(Plataforma), nullable=False)
    external_id = Column(String(100), unique=True, index=True)
    nome = Column(String(255), nullable=False)
    objetivo = Column(String(100))
    status = Column(String(50))
    budget_diario = Column(Float)
    created_at = Column(DateTime, default=datetime.utcnow)

    anuncios = relationship('Anuncio', back_populates='campanha')
    metricas = relationship('MetricaDiaria', back_populates='campanha')

class Anuncio(Base):
    __tablename__ = 'anuncios'
    __table_args__ = {'schema': 'marketing'}

    id = Column(Integer, primary_key=True)
    campanha_id = Column(Integer, ForeignKey('marketing.campanhas.id'))
    external_id = Column(String(100), unique=True, index=True)
    nome = Column(String(255))
    tipo_criativo = Column(String(50)) # image, video, carousel
    url_criativo = Column(Text)
    texto_primario = Column(Text)
    headline = Column(String(255))
    status = Column(String(50))

    campanha = relationship('Campanha', back_populates='anuncios')

class MetricaDiaria(Base):

```

```
__tablename__ = 'metricas_diarias'
__table_args__ = {'schema': 'marketing'}

id = Column(Integer, primary_key=True)
campanha_id = Column(Integer, ForeignKey('marketing.campanhas.id'))
anuncio_id = Column(Integer, ForeignKey('marketing.anuncios.id'),
nullable=True)
data = Column(DateTime, nullable=False)
impressoes = Column(Integer, default=0)
cliques = Column(Integer, default=0)
ctr = Column(Float)
cpc = Column(Float)
cpm = Column(Float)
conversoes = Column(Integer, default=0)
custo = Column(Float, default=0)
leads_gerados = Column(Integer, default=0)

campanha = relationship('Campanha', back_populates='metricas')
```

ETAPA 3: SERVIÇOS DE INTEGRAÇÃO

3.1 Serviço Kommo CRM (services/kommo.py)

```
import httpx

from app.config import get_settings
from typing import Optional, List, Dict, Any

class KommoService:
    def __init__(self):
        self.settings = get_settings()
        self.base_url = f'https://{self.settings.KOMMO_DOMAIN}/api/v4'
        self.headers = {
            'Authorization': f'Bearer {self.settings.KOMMO_ACCESS_TOKEN}',
            'Content-Type': 'application/json'
        }

    async def _request(self, method: str, endpoint: str, **kwargs) -> Dict:
        async with httpx.AsyncClient() as client:
            response = await client.request(
                method,
                f'{self.base_url}{endpoint}',
                headers=self.headers,
                **kwargs
            )
            response.raise_for_status()
            return response.json()

    async def get_pipelines(self) -> List[Dict]:
        '''Obtém todos os pipelines'''
        data = await self._request('GET', '/leads/pipelines')
        return data.get('_embedded', {}).get('pipelines', [])

    async def get_leads(self, pipeline_id: Optional[int] = None,
                        status_id: Optional[int] = None,
                        limit: int = 250,
                        page: int = 1) -> List[Dict]:
        '''Obtém leads com filtros'''
        params = {'limit': limit, 'page': page}
        if pipeline_id:
            params['filter[pipeline_id]'] = pipeline_id
        if status_id:
            params['filter[status_id]'] = status_id

        data = await self._request('GET', '/leads', params=params)
        return data.get('_embedded', {}).get('leads', [])
```

```
async def get_lead_by_id(self, lead_id: int) -> Dict:
    '''Obtém detalhes de um lead específico'''
    data = await self._request('GET', f'/leads/{lead_id}')
    return data

async def get_lead_events(self, lead_id: int) -> List[Dict]:
    '''Obtém histórico de eventos do lead'''
    params = {'filter[entity_id]': lead_id, 'filter[entity_type]': 'lead'}
    data = await self._request('GET', '/events', params=params)
    return data.get('_embedded', {}).get('events', [])
```

3.2 Serviço Meta Ads (services/meta_ads.py)

```
import httpx

from app.config import get_settings
from typing import List, Dict, Optional
from datetime import date, timedelta

class MetaAdsService:
    def __init__(self):
        self.settings = get_settings()
        self.base_url = 'https://graph.facebook.com/v18.0'
        self.access_token = self.settings.META_ACCESS_TOKEN
        self.ad_account_id = self.settings.META_AD_ACCOUNT_ID

    async def _request(self, endpoint: str, params: Dict = None) -> Dict:
        params = params or {}
        params['access_token'] = self.access_token

        async with httpx.AsyncClient() as client:
            response = await client.get(
                f'{self.base_url}/{endpoint}',
                params=params
            )
            response.raise_for_status()
            return response.json()

    async def get_campaigns(self, status: str = None) -> List[Dict]:
        '''Obtém campanhas da conta'''
        params = {
            'fields':
'id,name,objective,status,daily_budget,lifetime_budget,created_time'
        }
        if status:
            params['effective_status'] = [status]

        data = await self._request(f'/{self.ad_account_id}/campaigns', params)
        return data.get('data', [])

    async def get_campaign_insights(self, campaign_id: str,
                                    date_from: date = None,
                                    date_to: date = None) -> List[Dict]:
        '''Obtém métricas de uma campanha'''
        date_from = date_from or (date.today() - timedelta(days=30))
        date_to = date_to or date.today()

        params = {
```

```

        'fields':
'impressions,clicks,ctr,cpc,cpm,spend,actions,cost_per_action_type',
        'time_range': {'since': str(date_from), 'until': str(date_to)},
        'time_increment': 1 # Daily breakdown
    }

    data = await self._request(f'/{campaign_id}/insights', params)
    return data.get('data', [])

    async def get_ads(self, campaign_id: str = None) -> List[Dict]:
        '''Obtém anúncios'''
        params = {
            'fields':
'id,name,status,creative{id,thumbnail_url,body,title,image_url,video_id}'
        }

        endpoint = f'/{campaign_id}/ads' if campaign_id else
f'/{self.ad_account_id}/ads'
        data = await self._request(endpoint, params)
        return data.get('data', [])

    async def get_ad_creative(self, ad_id: str) -> Dict:
        '''Obtém criativo de um anúncio'''
        params = {'fields':
'thumbnail_url,body,title,image_url,video_id,object_story_spec'}
        return await self._request(f'/{ad_id}/adcreatives', params)

```

3.3 Serviço Instagram (services/instagram.py)

```
import httpx

from app.config import get_settings
from typing import List, Dict
from datetime import date, timedelta

class InstagramService:
    def __init__(self):
        self.settings = get_settings()
        self.base_url = 'https://graph.facebook.com/v18.0'
        self.access_token = self.settings.INSTAGRAM_ACCESS_TOKEN
        self.ig_user_id = self.settings.INSTAGRAM_BUSINESS_ID

    async def _request(self, endpoint: str, params: Dict = None) -> Dict:
        params = params or {}
        params['access_token'] = self.access_token

        async with httpx.AsyncClient() as client:
            response = await client.get(f'{self.base_url}{endpoint}',
params=params)
            response.raise_for_status()
            return response.json()

    async def get_account_info(self) -> Dict:
        '''Obtém informações da conta'''
        params = {'fields':
'id,username,followers_count,follows_count,media_count,biography'}
        return await self._request(f'/{self.ig_user_id}', params)

    async def get_media(self, limit: int = 50) -> List[Dict]:
        '''Obtém posts recentes'''
        params = {
            'fields':
'id,caption,media_type,media_url,thumbnail_url,permalink,timestamp,like_count,co
mments_count',
            'limit': limit
        }
        data = await self._request(f'/{self.ig_user_id}/media', params)
        return data.get('data', [])

    async def get_media_insights(self, media_id: str) -> Dict:
        '''Obtém métricas de um post específico'''
        params = {'metric': 'engagement,impressions,reach,saved,shares'}
        data = await self._request(f'/{media_id}/insights', params)
        return {item['name']: item['values'][0]['value'] for item in
data.get('data', [])}
```

```
async def get_account_insights(self, period: str = 'day',
                                days: int = 30) -> List[Dict]:
    '''Obtém métricas da conta'''
    params = {
        'metric': 'impressions,reach,profile_views,follower_count',
        'period': period
    }
    data = await self._request(f'/{self.ig_user_id}/insights', params)
    return data.get('data', [])
```

ETAPA 4: AGENTES DE IA

4.1 Base do Agente (services/agents/base.py)

```
from abc import ABC, abstractmethod
from typing import Dict, Any, Optional
import anthropic
import google.generativeai as genai
from app.config import get_settings

class BaseAgent(ABC):
    def __init__(self):
        self.settings = get_settings()
        self.claude_client = anthropic.Anthropic(api_key=self.settings.ANTHROPIC_API_KEY)
        genai.configure(api_key=self.settings.GOOGLE_AI_API_KEY)
        self.gemini_model = genai.GenerativeModel('gemini-2.0-flash')

    async def call_claude(self, prompt: str, model: str = 'claude-sonnet-4-20250514',
                        max_tokens: int = 4096) -> str:
        '''Chama Claude API'''
        response = self.claude_client.messages.create(
            model=model,
            max_tokens=max_tokens,
            messages=[{'role': 'user', 'content': prompt}]
        )
        return response.content[0].text

    async def call_gemini(self, prompt: str, image_data: bytes = None) -> str:
        '''Chama Gemini API (com suporte a imagem)'''
        if image_data:
            response = self.gemini_model.generate_content([prompt, image_data])
        else:
            response = self.gemini_model.generate_content(prompt)
        return response.text

    @abstractmethod
    async def analyze(self, data: Dict[str, Any]) -> Dict[str, Any]:
        '''Método principal de análise - implementar nas subclasses'''
        pass
```

4.2 Agente de Análise de Campanhas (services/agents/campaign_analyst.py)

```
from typing import Dict, Any, List
from .base import BaseAgent
```

```

import json

class CampaignAnalystAgent(BaseAgent):

    async def analyze(self, data: Dict[str, Any]) -> Dict[str, Any]:
        '''Analisa performance das campanhas'''
        campaigns = data.get('campaigns', [])
        metrics = data.get('metrics', [])

        # Prepara contexto para análise
        context = self._prepare_context(campaigns, metrics)

        # Análise de métricas com Claude Sonnet
        metrics_analysis = await self._analyze_metrics(context)

        # Recomendações com Claude Opus
        recommendations = await self._generate_recommendations(context,
metrics_analysis)

        return {
            'summary': metrics_analysis,
            'recommendations': recommendations,
            'top_campaigns': self._get_top_performers(campaigns, metrics),
            'alerts': self._get_alerts(metrics)
        }

    def _prepare_context(self, campaigns: List, metrics: List) -> str:
        '''Prepara contexto textual para LLM'''
        return f'''
DADOS DE CAMPANHAS:
Total de campanhas: {len(campaigns)}

MÉTRICAS AGREGADAS:
{json.dumps(metrics, indent=2, ensure_ascii=False)}
'''

    async def _analyze_metrics(self, context: str) -> str:
        prompt = f'''
        Você é um analista de marketing digital especializado em tráfego pago
para clínicas médicas.

        Analise os seguintes dados e forneça:
        1. Resumo executivo da performance
        2. Principais KPIs e sua evolução
        3. Campanhas com melhor e pior performance
        4. Padrões identificados

```

```

    {context}

    Responda em português, de forma objetiva e acionável.
    '''
    return await self.call_claude(prompt, model='claude-sonnet-4-20250514')

    async def _generate_recommendations(self, context: str, analysis: str) ->
    str:
        prompt = f'''
        Você é um consultor sênior de marketing digital.

        Baseado na análise abaixo, gere recomendações estratégicas:

        ANÁLISE:
        {analysis}

        DADOS ORIGINAIS:
        {context}

        Forneça:
        1. 3-5 ações prioritárias
        2. Sugestões de otimização de budget
        3. Testes A/B recomendados
        4. Próximos passos

        Seja específico e prático.
        '''
        return await self.call_claude(prompt, model='claude-opus-4-20250514')

```

4.3 Agente de Copywriting (services/agents/copywriter.py)

```

from typing import Dict, Any, List
from .base import BaseAgent
import httpx
import base64

class CopywriterAgent(BaseAgent):

    async def analyze(self, data: Dict[str, Any]) -> Dict[str, Any]:
        '''Gera conteúdo criativo'''
        request_type = data.get('type') # 'copy', 'roteiro', 'card'
        context = data.get('context', {})

        if request_type == 'copy':
            return await self._generate_copy(context)
        elif request_type == 'roteiro':
            return await self._generate_roteiro(context)
        elif request_type == 'card':
            return await self._generate_card(context)
        else:
            raise ValueError(f'Tipo não suportado: {request_type}')

    async def _generate_copy(self, context: Dict) -> Dict:
        '''Gera copies para anúncios'''
        top_performers = context.get('top_performers', [])
        tema = context.get('tema', 'consulta médica')
        tom = context.get('tom', 'profissional e acolhedor')

        prompt = f'''
        Você é um copywriter especializado em marketing para clínicas médicas.

        CONTEXTO:
        - Clínica do Dr. Igor
        - Tom de voz: {tom}
        - Tema: {tema}

        COPIES QUE PERFORMARAM BEM ANTERIORMENTE:
        {top_performers}

        Gere 5 variações de copy para anúncios, incluindo:
        - Headline principal (máx 40 caracteres)
        - Texto primário (máx 125 caracteres)
        - Descrição (máx 30 caracteres)
        - CTA sugerido
    
```

```

        Use gatilhos mentais adequados para área médica (autoridade, prova
social, escassez moderada).

        NÃO use termos proibidos pela publicidade médica.
'''

        result = await self.call_claude(prompt,
model='claude-sonnet-4-20250514')
        return {'copies': result}

    async def _generate_card(self, context: Dict) -> Dict:
        '''Gera imagem de card usando Nano Banana Pro'''
        tema = context.get('tema', '')
        texto = context.get('texto', '')
        estilo = context.get('estilo', 'clean, profissional, tons azuis')

        # Usar Gemini com Nano Banana Pro para gerar imagem
        prompt = f'''
Crie um card para Instagram com as seguintes características:
- Tema: {tema}
- Texto a incluir: {texto}
- Estilo visual: {estilo}
- Formato: quadrado 1080x1080
- Texto legível e bem posicionado
- Paleta de cores profissional para área médica
'''

        # Chamada para Gemini 3 com Nano Banana Pro
        genai_model = genai.GenerativeModel('gemini-2.0-flash-image')
        response = genai_model.generate_content(prompt)

        # Extraí imagem da resposta
        for part in response.candidates[0].content.parts:
            if part.inline_data:
                image_data = base64.b64encode(part.inline_data.data).decode()
                return {
                    'image_base64': image_data,
                    'mime_type': part.inline_data.mime_type
                }

        return {'error': 'Falha ao gerar imagem'}

```

ETAPA 5: FRONTEND - ESTRUTURA

5.1 Estrutura de Pastas do Frontend

```
frontend/src/
├── app/
│   ├── layout.tsx           # Layout principal
│   ├── page.tsx            # Dashboard principal
│   ├── atendimento-ia/
│   │   └── page.tsx
│   ├── atendimento-humano/
│   │   └── page.tsx
│   ├── leads-organicos/
│   │   └── page.tsx
│   ├── trafego-pago/
│   │   └── page.tsx
│   ├── instagram/
│   │   └── page.tsx
│   ├── agentes/
│   │   └── page.tsx
│   └── configuracoes/
│       └── page.tsx
├── components/
│   ├── layout/
│   │   ├── Sidebar.tsx
│   │   ├── Header.tsx
│   │   └── PageContainer.tsx
│   ├── dashboard/
│   │   ├── KPICard.tsx
│   │   ├── AreaChart.tsx
│   │   ├── FunnelChart.tsx
│   │   └── DataTable.tsx
│   ├── agents/
│   │   ├── AgentCard.tsx
│   │   ├── AnalysisResult.tsx
│   │   └── GeneratorForm.tsx
│   └── ui/                  # shadcn components
├── lib/
│   ├── api.ts              # API client
│   ├── utils.ts
│   └── constants.ts
├── hooks/
│   ├── useLeads.ts
│   ├── useCampaigns.ts
│   └── useAgents.ts
└── stores/
```

```
|   └─ useAppStore.ts      # Zustand store
└─ types/
    └─ index.ts
```

5.2 API Client (lib/api.ts)

```
import axios from 'axios';

const api = axios.create({
  baseURL: process.env.NEXT_PUBLIC_API_URL || 'http://localhost:8000/api',
  headers: { 'Content-Type': 'application/json' }
});

// Dashboard
export const getDashboardKPIs = (period: string) =>
  api.get(`/dashboard/kpis?period=${period}`);

export const getDashboardCharts = (period: string) =>
  api.get(`/dashboard/charts?period=${period}`);

// Leads
export const getLeads = (params: any) =>
  api.get('/leads', { params });

export const getLeadById = (id: number) =>
  api.get(`/leads/${id}`);

// Campaigns
export const getCampaigns = (params: any) =>
  api.get('/campaigns', { params });

export const getCampaignInsights = (id: string, period: string) =>
  api.get(`/campaigns/${id}/insights?period=${period}`);

// Instagram
export const getInstagramMetrics = () =>
  api.get('/instagram/metrics');

export const getInstagramPosts = (limit: number) =>
  api.get(`/instagram/posts?limit=${limit}`);

// Agents
export const runCampaignAnalysis = (params: any) =>
  api.post('/agents/campaign-analysis', params);
```

```
export const runLeadAnalysis = (params: any) =>
  api.post('/agents/lead-analysis', params);

export const generateCopy = (params: any) =>
  api.post('/agents/generate-copy', params);

export const generateCard = (params: any) =>
  api.post('/agents/generate-card', params);

export default api;
```

5.3 Sidebar Component (components/layout/Sidebar.tsx)

```
'use client';

import Link from 'next/link';
import { usePathname } from 'next/navigation';
import { cn } from '@lib/utils';
import {
  LayoutDashboard, Bot, Users, Leaf, Megaphone,
  Instagram, Sparkles, Settings
} from 'lucide-react';

const navigation = [
  { name: 'Dashboard', href: '/', icon: LayoutDashboard },
  { name: 'Atendimento IA', href: '/atendimento-ia', icon: Bot },
  { name: 'Atendimento Humano', href: '/atendimento-humano', icon: Users },
  { name: 'Leads Orgânicos', href: '/leads-organicos', icon: Leaf },
  { name: 'Tráfego Pago', href: '/trafego-pago', icon: Megaphone },
  { name: 'Instagram', href: '/instagram', icon: Instagram },
  { name: 'Agentes IA', href: '/agentes', icon: Sparkles },
  { name: 'Configurações', href: '/configuracoes', icon: Settings },
];

export function Sidebar() {
  const pathname = usePathname();

  return (
    <aside className='w-64 min-h-screen bg-gradient-to-b from-slate-900
via-slate-800 to-slate-900'>
      <div className='p-6'>
        <h1 className='text-xl font-bold text-white'>
          Clínica Dr. Igor
        </h1>
        <p className='text-sm text-slate-400'>Dashboard Analytics</p>
      </div>

      <nav className='px-3 space-y-1'>
        {navigation.map((item) => {
          const isActive = pathname === item.href;
          return (
            <Link
              key={item.name}
              href={item.href}
              className={cn(
                'flex items-center gap-3 px-3 py-2.5 rounded-lg transition-all',

```

```
        isActive
        ? 'bg-blue-600 text-white'
        : 'text-slate-300 hover:bg-slate-700/50 hover:text-white'
      )}
    >
    <item.icon className='w-5 h-5' />
    <span className='font-medium'>{item.name}</span>
  </Link>
  );
  })}
</nav>
</aside>
);
}
```

5.4 KPI Card Component (components/dashboard/KPICard.tsx)

```
'use client';

import { Card, CardContent } from '@components/ui/card';
import { cn } from '@lib/Utils';
import { TrendingUp, TrendingDown, Minus } from 'lucide-react';

interface KPICardProps {
  title: string;
  value: string | number;
  change?: number;
  changeLabel?: string;
  icon?: React.ReactNode;
  format?: 'number' | 'currency' | 'percent';
}

export function KPICard({
  title,
  value,
  change,
  changeLabel = 'vs período anterior',
  icon,
  format = 'number'
}: KPICardProps) {
  const formatValue = (val: string | number) => {
    if (typeof val === 'string') return val;
    switch (format) {
      case 'currency':
        return new Intl.NumberFormat('pt-BR', {
          style: 'currency',
          currency: 'BRL'
        }).format(val);
      case 'percent':
        return `${val.toFixed(1)}%`;
      default:
        return new Intl.NumberFormat('pt-BR').format(val);
    }
  };

  const getTrendIcon = () => {
    if (!change) return <Minus className='w-4 h-4 text-slate-400' />;
    if (change > 0) return <TrendingUp className='w-4 h-4 text-emerald-500' />;
    return <TrendingDown className='w-4 h-4 text-red-500' />;
  };
}
```

```

return (
  <Card className='bg-white border-slate-200 hover:shadow-lg
transition-shadow'>
    <CardContent className='p-6'>
      <div className='flex items-start justify-between'>
        <div>
          <p className='text-sm font-medium text-slate-500'>{title}</p>
          <p className='text-3xl font-bold text-slate-900 mt-2'>
            {formatValue(value)}
          </p>
          {change !== undefined && (
            <div className='flex items-center gap-1 mt-2'>
              {getTrendIcon()}
              <span className={cn(
                'text-sm font-medium',
                change > 0 ? 'text-emerald-600' : change < 0 ? 'text-red-600'
: 'text-slate-500'
              )}>
                {change > 0 ? '+' : ''}{change?.toFixed(1)}%
              </span>
              <span className='text-xs text-slate-400'>{changeLabel}</span>
            </div>
          )}
        </div>
        {icon && (
          <div className='p-3 bg-blue-50 rounded-lg'>
            {icon}
          </div>
        )}
      </div>
    </CardContent>
  </Card>
);
}

```

ETAPA 6: DOCKER E DEPLOY

6.1 Dockerfile Backend (docker/backend.Dockerfile)

```
FROM python:3.12-slim

WORKDIR /app

# Install dependencies
COPY backend/requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application
COPY backend/app ./app
COPY backend/alembic ./alembic
COPY backend/alembic.ini .

# Create non-root user
RUN useradd -m appuser && chown -R appuser:appuser /app
USER appuser

EXPOSE 8000

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

6.2 Docker Compose (docker-compose.yml)

```
version: '3.8'

services:
  backend:
    build:
      context: .
      dockerfile: docker/backend.Dockerfile
    ports:
      - '8000:8000'
    env_file:
      - .env
    depends_on:
      - redis
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    ports:
```

```

    - '6379:6379'

volumes:
  - redis_data:/data

restart: unless-stopped

celery_worker:
  build:
    context: .
    dockerfile: docker/backend.Dockerfile
  command: celery -A app.celery_app worker --loglevel=info
  env_file:
    - .env
  depends_on:
    - redis
    - backend
  restart: unless-stopped

volumes:
  redis_data:

```

6.3 Deploy Backend na Hostinger

1. Acesse o EasyPanel da sua VPS
2. Crie um novo projeto 'dashboard-clinica'
3. Adicione serviço 'App' com Docker
4. Configure o repositório Git
5. Adicione variáveis de ambiente do .env
6. Configure domínio e SSL
7. Deploy!

6.4 Deploy Frontend na Vercel

1. Acesse vercel.com e faça login
2. Clique em 'Add New Project'
3. Importe o repositório Git
4. Configure Root Directory: frontend
5. Adicione variável: NEXT_PUBLIC_API_URL = https://api.seudominio.com
6. Deploy!

CHECKLIST DE VALIDAÇÃO

Antes de considerar cada etapa concluída, valide:

Setup

- ☐ npm run dev funciona no frontend
- ☐ uvicorn app.main:app --reload funciona no backend
- ☐ /docs abre documentação Swagger
- ☐ /health retorna {status: 'healthy'}

Integrações

- ☐ Kommo: GET /api/leads retorna dados
- ☐ Meta Ads: GET /api/campaigns retorna campanhas
- ☐ Instagram: GET /api/instagram/metrics funciona
- ☐ Chat History: Conexão com n8n_chat_history OK

Dashboard

- ☐ Sidebar navega entre páginas
- ☐ KPIs carregam dados reais
- ☐ Gráficos renderizam corretamente
- ☐ Filtros de período funcionam

Agentes IA

- ☐ POST /api/agents/campaign-analysis retorna análise
- ☐ POST /api/agents/generate-copy retorna copies
- ☐ POST /api/agents/generate-card retorna imagem base64

Deploy

- ☐ Backend acessível via HTTPS
- ☐ Frontend acessível via HTTPS
- ☐ CORS configurado corretamente
- ☐ Variáveis de ambiente em produção

FIM DO DOCUMENTO DE EXECUÇÃO

Siga cada etapa na ordem. Teste antes de avançar.

Em caso de dúvidas, consulte a documentação oficial de cada tecnologia.