Agente-CEO Python

Complete Project & Code Guide

This guide provides the complete setup for the Agente-CEO Python project. The project's mission is to create a fully autonomous Python agent for dropshipping and infoproduct sales, leveraging a powerful backend stack with Supabase and FastAPI, and a complete frontend dashboard for monitoring and control.

Step 1: Configure Supabase Database

Run the following SQL script in your Supabase SQL editor. This will set up the necessary tables, functions, policies, and storage for the project.

Supabase Credentials
URL: `https://gpakoffbuypbmfiwewka.supabase.co`
Use the Service Role and ANON keys provided in your project documentation.

Supabase DDL — Copy

```
-- ============================
-- EXTENSIONS & UTILITIES
-- ============================
create extension if not exists pgcrypto; -- para gen_random_uuid()

-- Função genérica para updated_at automático
create or replace function public.set_updated_at()
returns trigger as $$
begin
  new.updated_at := now();
  return new;
```

```
end;
$$ language plpgsql;

-- ============================
-- TABELAS BÁSICAS / PERFIS (opcional)
-- ============================
-- Observação: Supabase já mantém auth.users. Se precisar perfis, use:
create table if not exists public.profiles (
  id uuid primary key default auth.uid(),
  full_name text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create trigger profiles_set_updated_at
before update on public.profiles
for each row execute function public.set_updated_at();

alter table public.profiles enable row level security;
create policy "profiles_select_own" on public.profiles
for select using (id = auth.uid());
create policy "profiles_ins_own" on public.profiles
for insert with check (id = auth.uid());
create policy "profiles_upd_own" on public.profiles
for update using (id = auth.uid());

-- ============================
-- TABELAS DE NEGÓCIO
-- ============================
-- Owner model simples: todas as tabelas têm owner_id = auth.uid()
-- (Service role bypassa RLS quando necessário)

-- PRODUCTS
create table if not exists public.products (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  title text not null,
  niche text,
  cost numeric(12,2) not null default 0,
  price numeric(12,2) not null default 0,
  margin_pct numeric(6,2) generated always as (
    case when price > 0 then round(((price - cost) / price) * 100, 2) else null end
  ) stored,
  source_url text,
  media_url text,
  status text not null default 'draft' check (status in ('draft','testing','active',
```

```
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);

create index if not exists idx_products_owner on public.products(owner_id);
create index if not exists idx_products_status on public.products(status);

create trigger products_set_updated_at
before update on public.products
for each row execute function public.set_updated_at();

alter table public.products enable row level security;
create policy "products_select_own" on public.products
for select using (owner_id = auth.uid());
create policy "products_ins_own" on public.products
for insert with check (owner_id = auth.uid());
create policy "products_upd_own" on public.products
for update using (owner_id = auth.uid());
create policy "products_del_own" on public.products
for delete using (owner_id = auth.uid());

-- PRODUCT SOURCES
create table if not exists public.product_sources (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  product_id uuid not null references public.products(id) on delete cascade,
  vendor text not null,
  vendor_product_id text,
  buy_url text,
  stock_hint int,
  shipping_sla text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_product_sources_product on public.product_sources(proc
create index if not exists idx_product_sources_owner on public.product_sources(owner_
create trigger product_sources_set_updated_at
before update on public.product_sources
for each row execute function public.set_updated_at();
alter table public.product_sources enable row level security;
create policy "product_sources_select_own" on public.product_sources for select using
create policy "product_sources_ins_own" on public.product_sources for insert with che
create policy "product_sources_upd_own" on public.product_sources for update using (
create policy "product_sources_del_own" on public.product_sources for delete using (

-- CREATIVES
```

```
create table if not exists public.creatives (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  product_id uuid not null references public.products(id) on delete cascade,
  status text not null default 'draft' check (status in ('draft','ready','queued','pu
  variant text,
  hook text,
  script text,
  cta text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_creatives_product on public.creatives(product_id);
create index if not exists idx_creatives_owner on public.creatives(owner_id);
create trigger creatives_set_updated_at
before update on public.creatives
for each row execute function public.set_updated_at();
alter table public.creatives enable row level security;
create policy "creatives_select_own" on public.creatives for select using (owner_id =
create policy "creatives_ins_own" on public.creatives for insert with check (owner_id
create policy "creatives_upd_own" on public.creatives for update using (owner_id = au
create policy "creatives_del_own" on public.creatives for delete using (owner_id = au

-- CREATIVE ASSETS
create table if not exists public.creative_assets (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  creative_id uuid not null references public.creatives(id) on delete cascade,
  asset_type text not null check (asset_type in ('thumbnail','video','caption')),
  storage_path text not null,
  duration_seconds int,
  checksum text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_creative_assets_creative on public.creative_assets(cre
create index if not exists idx_creative_assets_owner on public.creative_assets(owner_
create trigger creative_assets_set_updated_at
before update on public.creative_assets
for each row execute function public.set_updated_at();
alter table public.creative_assets enable row level security;
create policy "creative_assets_select_own" on public.creative_assets for select using
create policy "creative_assets_ins_own" on public.creative_assets for insert with che
create policy "creative_assets_upd_own" on public.creative_assets for update using (c
create policy "creative_assets_del_own" on public.creative_assets for delete using (c
```

```
-- PUBLICATIONS
create table if not exists public.publications (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  creative_id uuid not null references public.creatives(id) on delete cascade,
  platform text not null check (platform in ('tiktok','instagram','youtube')),
  scheduled_at timestamptz,
  published_at timestamptz,
  post_url text,
  status text not null default 'queued' check (status in ('queued','published','faile
  error text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_publications_creative on public.publications(creative_
create index if not exists idx_publications_owner on public.publications(owner_id);
create trigger publications_set_updated_at
before update on public.publications
for each row execute function public.set_updated_at();
alter table public.publications enable row level security;
create policy "publications_select_own" on public.publications for select using (owne
create policy "publications_ins_own" on public.publications for insert with check (ow
create policy "publications_upd_own" on public.publications for update using (owner_i
create policy "publications_del_own" on public.publications for delete using (owner_i

-- AD ACCOUNTS & PIXELS
create table if not exists public.ad_accounts (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  platform text not null check (platform in ('meta','tiktok')),
  external_id text not null,
  name text,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_ad_accounts_owner on public.ad_accounts(owner_id);
create trigger ad_accounts_set_updated_at
before update on public.ad_accounts
for each row execute function public.set_updated_at();
alter table public.ad_accounts enable row level security;
create policy "ad_accounts_select_own" on public.ad_accounts for select using (owner_
create policy "ad_accounts_ins_own" on public.ad_accounts for insert with check (owne
create policy "ad_accounts_upd_own" on public.ad_accounts for update using (owner_id
create policy "ad_accounts_del_own" on public.ad_accounts for delete using (owner_id

create table if not exists public.pixels (
```

```
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  platform text not null check (platform in ('meta','tiktok')),
  ad_account_id uuid not null references public.ad_accounts(id) on delete cascade,
  external_id text not null,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_pixels_owner on public.pixels(owner_id);
create index if not exists idx_pixels_account on public.pixels(ad_account_id);
create trigger pixels_set_updated_at
before update on public.pixels
for each row execute function public.set_updated_at();
alter table public.pixels enable row level security;
create policy "pixels_select_own" on public.pixels for select using (owner_id = auth
create policy "pixels_ins_own" on public.pixels for insert with check (owner_id = au
create policy "pixels_upd_own" on public.pixels for update using (owner_id = auth.ui
create policy "pixels_del_own" on public.pixels for delete using (owner_id = auth.ui

-- CAMPAIGNS / ADSETS / ADS / TRAFFIC METRICS
create table if not exists public.campaigns (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  platform text not null check (platform in ('meta','tiktok')),
  ad_account_id uuid not null references public.ad_accounts(id) on delete cascade,
  name text not null,
  objective text,
  status text not null default 'draft' check (status in ('draft','active','paused','
  daily_budget numeric(12,2) default 0,
  spend numeric(12,2) default 0,
  clicks int default 0,
  impressions int default 0,
  purchases int default 0,
  roas numeric(8,2),
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_campaigns_owner on public.campaigns(owner_id);
create index if not exists idx_campaigns_account on public.campaigns(ad_account_id);
create trigger campaigns_set_updated_at
before update on public.campaigns
for each row execute function public.set_updated_at();
alter table public.campaigns enable row level security;
create policy "campaigns_select_own" on public.campaigns for select using (owner_id =
create policy "campaigns_ins_own" on public.campaigns for insert with check (owner_i
create policy "campaigns_upd_own" on public.campaigns for update using (owner_id = a
```

```
create policy "campaigns_del_own" on public.campaigns for delete using (owner_id = a

create table if not exists public.adsets (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  campaign_id uuid not null references public.campaigns(id) on delete cascade,
  name text not null,
  targeting_json jsonb,
  bid_strategy text,
  status text not null default 'draft' check (status in ('draft','active','paused','
  budget numeric(12,2) default 0,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_adsets_campaign on public.adsets(campaign_id);
create index if not exists idx_adsets_owner on public.adsets(owner_id);
create trigger adsets_set_updated_at
before update on public.adsets
for each row execute function public.set_updated_at();
alter table public.adsets enable row level security;
create policy "adsets_select_own" on public.adsets for select using (owner_id = auth
create policy "adsets_ins_own" on public.adsets for insert with check (owner_id = au
create policy "adsets_upd_own" on public.adsets for update using (owner_id = auth.ui
create policy "adsets_del_own" on public.adsets for delete using (owner_id = auth.ui

create table if not exists public.ads (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  adset_id uuid not null references public.adsets(id) on delete cascade,
  name text not null,
  creative_ref text,
  status text not null default 'draft' check (status in ('draft','active','paused','
  cpc numeric(10,4),
  cpm numeric(10,4),
  ctr numeric(6,4),
  spend numeric(12,2) default 0,
  purchases int default 0,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_ads_adset on public.ads(adset_id);
create index if not exists idx_ads_owner on public.ads(owner_id);
create trigger ads_set_updated_at
before update on public.ads
for each row execute function public.set_updated_at();
alter table public.ads enable row level security;
```

```
create policy "ads_select_own" on public.ads for select using (owner_id = auth.uid()
create policy "ads_ins_own" on public.ads for insert with check (owner_id = auth.uid
create policy "ads_upd_own" on public.ads for update using (owner_id = auth.uid());
create policy "ads_del_own" on public.ads for delete using (owner_id = auth.uid());

create table if not exists public.traffic_metrics (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  date date not null,
  level text not null check (level in ('campaign','adset','ad')),
  ref_id uuid not null,
  spend numeric(12,2) default 0,
  clicks int default 0,
  impressions int default 0,
  purchases int default 0,
  revenue numeric(12,2),
  roas numeric(8,2),
  cac numeric(12,2),
  created_at timestamptz not null default now()
);
create index if not exists idx_traffic_metrics_owner_date on public.traffic_metrics(
create index if not exists idx_traffic_metrics_ref on public.traffic_metrics(ref_id)
alter table public.traffic_metrics enable row level security;
create policy "traffic_metrics_select_own" on public.traffic_metrics for select using
create policy "traffic_metrics_ins_own" on public.traffic_metrics for insert with ch

-- CLIENTES / LEADS / ORDERS
create table if not exists public.customers (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  name text,
  email text,
  phone text,
  first_seen_at timestamptz default now(),
  last_seen_at timestamptz,
  ltv numeric(12,2) default 0,
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_customers_owner on public.customers(owner_id);
create index if not exists idx_customers_email on public.customers(email);
create trigger customers_set_updated_at
before update on public.customers
for each row execute function public.set_updated_at();
alter table public.customers enable row level security;
create policy "customers_select_own" on public.customers for select using (owner_id =
```

```
create policy "customers_ins_own" on public.customers for insert with check (owner_i
create policy "customers_upd_own" on public.customers for update using (owner_id = a
create policy "customers_del_own" on public.customers for delete using (owner_id = a

create table if not exists public.leads (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  source text,
  product_id uuid references public.products(id) on delete set null,
  customer_id uuid references public.customers(id) on delete set null,
  utm_json jsonb,
  stage text default 'new' check (stage in ('new','contacted','qualified','won','los
  created_at timestamptz not null default now()
);
create index if not exists idx_leads_owner on public.leads(owner_id);
create index if not exists idx_leads_customer on public.leads(customer_id);
alter table public.leads enable row level security;
create policy "leads_select_own" on public.leads for select using (owner_id = auth.u
create policy "leads_ins_own" on public.leads for insert with check (owner_id = auth
create policy "leads_upd_own" on public.leads for update using (owner_id = auth.uid(
create policy "leads_del_own" on public.leads for delete using (owner_id = auth.uid(

create table if not exists public.orders (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  customer_id uuid references public.customers(id) on delete set null,
  product_id uuid references public.products(id) on delete set null,
  quantity int not null default 1,
  price numeric(12,2) not null default 0,
  discount numeric(12,2) default 0,
  revenue numeric(12,2) generated always as (round((price - coalesce(discount,0)) * 
  cost_product numeric(12,2) default 0,
  cost_shipping numeric(12,2) default 0,
  fees numeric(12,2) default 0,
  channel text,
  utm_json jsonb,
  campaign_ref uuid,
  purchased_at timestamptz not null default now(),
  status text default 'paid' check (status in ('paid','refunded','canceled','pending
  created_at timestamptz not null default now(),
  updated_at timestamptz not null default now()
);
create index if not exists idx_orders_owner on public.orders(owner_id);
create index if not exists idx_orders_customer on public.orders(customer_id);
create index if not exists idx_orders_product on public.orders(product_id);
create trigger orders_set_updated_at
```

```
before update on public.orders
for each row execute function public.set_updated_at();
alter table public.orders enable row level security;
create policy "orders_select_own" on public.orders for select using (owner_id = auth
create policy "orders_ins_own" on public.orders for insert with check (owner_id = au
create policy "orders_upd_own" on public.orders for update using (owner_id = auth.ui
create policy "orders_del_own" on public.orders for delete using (owner_id = auth.ui

create table if not exists public.order_items (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  order_id uuid not null references public.orders(id) on delete cascade,
  sku text,
  qty int not null default 1,
  unit_price numeric(12,2) not null default 0
);
create index if not exists idx_order_items_order on public.order_items(order_id);
create index if not exists idx_order_items_owner on public.order_items(owner_id);
alter table public.order_items enable row level security;
create policy "order_items_select_own" on public.order_items for select using (owner_
create policy "order_items_ins_own" on public.order_items for insert with check (own
create policy "order_items_upd_own" on public.order_items for update using (owner_id
create policy "order_items_del_own" on public.order_items for delete using (owner_id

-- ALERTS / TASKS / AUDIT
create table if not exists public.alerts (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  severity text not null check (severity in ('INFO','LOW','MEDIUM','HIGH','CRITICAL'
  title text not null,
  message text,
  ref_table text,
  ref_id uuid,
  ack boolean default false,
  created_at timestamptz not null default now()
);
create index if not exists idx_alerts_owner on public.alerts(owner_id);
alter table public.alerts enable row level security;
create policy "alerts_select_own" on public.alerts for select using (owner_id = auth
create policy "alerts_ins_own" on public.alerts for insert with check (owner_id = au

create table if not exists public.tasks (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  agent text not null,
  action text not null,
```

```
  payload_json jsonb,
  status text not null default 'pending' check (status in ('pending','running','done
  result_json jsonb,
  started_at timestamptz,
  finished_at timestamptz,
  created_at timestamptz not null default now()
);
create index if not exists idx_tasks_owner on public.tasks(owner_id);
create index if not exists idx_tasks_status on public.tasks(status);
alter table public.tasks enable row level security;
create policy "tasks_select_own" on public.tasks for select using (owner_id = auth.u
create policy "tasks_ins_own" on public.tasks for insert with check (owner_id = auth
create policy "tasks_upd_own" on public.tasks for update using (owner_id = auth.uid(

create table if not exists public.audit_log (
  id uuid primary key default gen_random_uuid(),
  owner_id uuid not null default auth.uid(),
  actor text not null,
  action text not null,
  details_json jsonb,
  created_at timestamptz not null default now()
);
create index if not exists idx_audit_owner on public.audit_log(owner_id);
alter table public.audit_log enable row level security;
create policy "audit_select_own" on public.audit_log for select using (owner_id = au
create policy "audit_ins_own" on public.audit_log for insert with check (owner_id = a

-- ============================
-- VIEWS ÚTEIS (SUMÁRIO DE MÉTRICAS)
-- ============================
create or replace view public.metrics_daily as
select
  o.owner_id,
  date_trunc('day', o.purchased_at) as day,
  count(*) as orders,
  sum(o.revenue) as revenue,
  sum(o.cost_product + o.cost_shipping + o.fees) as total_cost,
  coalesce(sum(tm.spend),0) as ad_spend,
  case when coalesce(sum(tm.spend),0) > 0 then round(sum(o.revenue)/sum(tm.spend),2)
from public.orders o
left join public.traffic_metrics tm
  on tm.owner_id = o.owner_id
 and tm.date = date_trunc('day', o.purchased_at)::date
where o.status = 'paid'
group by 1,2
order by 2 desc;
```

```
-- ============================
-- STORAGE BUCKET (mídias)
-- ============================
-- Executar no SQL Editor; requer permissão de storage_admin
select storage.create_bucket('media', true);
```

Step 2: Create Backend Project Files

Create the following files and directories in your local project folder. This provides the complete structure for the FastAPI backend application.

`requirements.txt`

Dependencies Copy

```
fastapi
uvicorn
httpx
pydantic
pydantic-settings
python-dotenv
supabase
apscheduler
loguru
playwright
playwright-stealth
beautifulsoup4
lxml
requests
facebook-business
pandas
```

`.env.example`

Environment Variables Copy

```
SUPABASE_URL=https://gpakoffbuypbmfiwewka.supabase.co
SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZi
SUPABASE_SERVICE_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJ
AUTO_MODE=true
APPROVAL_MODE=false
DRY_RUN=false
SAFE_BUDGET_CAP_DAILY=300.0
SAFE_BUDGET_CAP_WEEKLY=1500.0

META_APP_ID=
META_APP_SECRET=
META_ACCESS_TOKEN=
META_AD_ACCOUNT_ID=
META_PIXEL_ID=

TIKTOK_ACCESS_TOKEN=
TIKTOK_ADVERTISER_ID=
TIKTOK_OPEN_ID=
TIKTOK_PIXEL_ID=

INSTAGRAM_BUSINESS_ACCOUNT_ID=

PICTORY_API_KEY=

STORAGE_BUCKET=media

PLAYWRIGHT_PROXY_URL= # This variable is not directly used in the provided code, but
PROXY_SERVER= # e.g., "http://proxy.example.com:8080"
PROXY_USERNAME=
PROXY_PASSWORD=
```

app/main.py

Python Copy

```python
from fastapi import FastAPI
from .api.http import router as api_router
from .api.webhooks import router as hooks_router
from .core.scheduler import start as start_scheduler
from .services import publish
from .core.db import supabase
from apscheduler.triggers.cron import CronTrigger
from loguru import logger
```

```python
from datetime import time

app = FastAPI(title="Agente-CEO Python — Invisível")
app.include_router(api_router)
app.include_router(hooks_router)

@app.on_event("startup")
async def _startup():
    await start_scheduler()
    from .core.scheduler import scheduler

    scheduler.add_job(
        func=publish.process_publication_queue,
        trigger='interval',
        seconds=30,
        id='publication_queue_processor',
        name='Publication Queue Processor',
        replace_existing=True
    )

    logger.info("Scheduler jobs added.")
```

app/core/config.py

Python Copy

```python
from pydantic_settings import BaseSettings

class Settings(BaseSettings):
    SUPABASE_URL: str
    SUPABASE_ANON_KEY: str
    SUPABASE_SERVICE_KEY: str | None = None

    AUTO_MODE: bool = True
    APPROVAL_MODE: bool = False
    DRY_RUN: bool = False

    SAFE_BUDGET_CAP_DAILY: float = 300.0
    SAFE_BUDGET_CAP_WEEKLY: float = 1500.0

    META_ACCESS_TOKEN: str | None = None
    META_AD_ACCOUNT_ID: str | None = None
    META_PIXEL_ID: str | None = None
    META_APP_ID: str | None = None
    META_APP_SECRET: str | None = None
```

```
    TIKTOK_ACCESS_TOKEN: str | None = None
    TIKTOK_AD_ACCOUNT_ID: str | None = None
    TIKTOK_PIXEL_ID: str | None = None
    TIKTOK_OPEN_ID: str | None = None
    TIKTOK_ADVERTISER_ID: str | None = None

    INSTAGRAM_BUSINESS_ACCOUNT_ID: str | None = None

    PICTORY_API_KEY: str | None = None

    STORAGE_BUCKET: str = "media"

    PROXY_SERVER: str | None = None
    PROXY_USERNAME: str | None = None
    PROXY_PASSWORD: str | None = None

    class Config:
        env_file = ".env"

settings = Settings()
```

app/core/db.py

Python Copy

```
from supabase import create_client, Client
from .config import settings

supabase: Client = create_client(settings.SUPABASE_URL, settings.SUPABASE_ANON_KEY)

def service_client() -> Client:
    # quando precisar bypass de RLS para rotinas internas
    key = settings.SUPABASE_SERVICE_KEY or settings.SUPABASE_ANON_KEY
    return create_client(settings.SUPABASE_URL, key)
```

app/core/logs.py

Python Copy

```
from loguru import logger
import sys
```

```
logger.remove()
logger.add(sys.stdout, serialize=True, backtrace=True, diagnose=False)
```

app/core/scheduler.py

Python Copy

```python
from apscheduler.schedulers.asyncio import AsyncIOScheduler
from datetime import time
from loguru import logger

scheduler = AsyncIOScheduler()

async def start():
    scheduler.start()
    logger.info({"event": "scheduler_started"})
```

app/models/product.py

Python Copy

```python
from pydantic import BaseModel, HttpUrl
from typing import Optional

class ProductIn(BaseModel):
    title: str
    niche: Optional[str] = None
    cost: float = 0
    price: float = 0
    source_url: Optional[HttpUrl] = None
    media_url: Optional[HttpUrl] = None

class Product(ProductIn):
    id: str
    status: str
```

app/services/products.py

Python Copy

```
import asyncio
import random
import logging
import re
from typing import List, Dict, Any, Optional

from playwright.async_api import async_playwright, Browser, Page, Playwright
from bs4 import BeautifulSoup, Tag
from ..core.db import service_client
from loguru import logger

USER_AGENTS = [
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
    "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/11
]

async def _ethical_delay(min_delay: float = 2.0, max_delay: float = 5.0):
    delay = random.uniform(min_delay, max_delay)
    logger.info(f"Applying ethical delay of {delay:.2f} seconds.")
    await asyncio.sleep(delay)

def _get_random_user_agent() -> str:
    return random.choice(USER_AGENTS)

async def _init_browser(playwright: Playwright, proxy: Optional[str] = None) -> Brows
    browser_options = {
        'headless': True,
    }
    if proxy:
        browser_options['proxy'] = {'server': proxy}
        logger.info(f"Initializing browser with proxy: {proxy}")
    else:
        logger.info("Initializing browser without proxy.")

    browser = await playwright.chromium.launch(**browser_options)
    return browser

async def _scrape_aliexpress_search(page: Page, search_term: str) -> str:
    url = f"https://www.aliexpress.com/wholesale?SearchText={search_term.replace(' '
    logger.info(f"Navigating to AliExpress search URL: {url}")

    await page.goto(url, wait_until='domcontentloaded', timeout=60000)
```

```
    await page.wait_for_selector('div[id*="card-list"]', timeout=30000)
    await _ethical_delay(3, 6)

    for _ in range(3):
        await page.evaluate('window.scrollBy(0, document.body.scrollHeight)')
        await _ethical_delay(1, 2)

    return await page.content()

def _parse_aliexpress_html(html_content: str) -> List[Dict[str, Any]]:
    soup = BeautifulSoup(html_content, 'lxml')
    product_cards = soup.select('div[class*="product-card_root"]')

    logger.info(f"Found {len(product_cards)} potential product cards to parse.")

    parsed_products = []
    for card in product_cards:
        try:
            data = {}
            title_element = card.select_one('h3[class*="product-card_title"]')
            url_element = card.select_one('a[href]')
            if title_element and url_element:
                data['title'] = title_element.get_text(strip=True)
                data['url'] = url_element['href'] if url_element['href'].startswith(
            else:
                continue

            price_element = card.select_one('div[class*="product-card_price"] span')
            if price_element:
                price_text = price_element.get_text(strip=True).replace('$', '').rep
                match = re.search(r'[\\d.]+', price_text)
                if match:
                    data['supplier_price'] = float(match.group(0))

            sales_element = card.select_one('span[class*="product-card_sold"]')
            if sales_element:
                sales_text = sales_element.get_text(strip=True)
                match = re.search(r'(\\d+)', sales_text)
                if match:
                    data['orders'] = int(match.group(0))
            else:
                data['orders'] = 0

            shipping_element = card.select_one('span[class*="product-card_delivery"]
            if shipping_element and 'free' in shipping_element.get_text(lower=True):
                data['shipping_cost'] = 0.0
```

```
                data['shipping_days_max'] = 30
            else:
                data['shipping_cost'] = None
                data['shipping_days_max'] = 30

            if data.get('title') and data.get('supplier_price'):
                parsed_products.append(data)

        except (AttributeError, ValueError, TypeError) as e:
            logger.warning(f"Could not parse a product card. Error: {e}")
            continue

    return parsed_products

async def scout_once(
    search_term: str,
    target_margin: float = 3.0,
    max_shipping_days: int = 25,
    min_demand_orders: int = 100,
    proxy: Optional[str] = None
) -> List[Dict[str, Any]]:
    logger.info(f"Starting scout for '{search_term}'...")

    async with async_playwright() as p:
        browser = await _init_browser(p, proxy)
        context = await browser.new_context(user_agent=_get_random_user_agent())
        page = await context.new_page()

        try:
            html_content = await _scrape_aliexpress_search(page, search_term)
            raw_products = _parse_aliexpress_html(html_content)

            logger.info(f"Successfully parsed {len(raw_products)} products from the p

            promising_products_data = []
            for product in raw_products:
                if product.get('orders', 0) < min_demand_orders:
                    continue

                if product.get('shipping_days_max', 99) > max_shipping_days:
                    continue

                potential_retail_price = round(product['supplier_price'] * target_mar

                if product.get('title') and product.get('supplier_price'):
                    product_to_insert = {
```

```
                        "title": product['title'],
                        "niche": search_term,
                        "cost": product['supplier_price'],
                        "price": potential_retail_price,
                        "source_url": product['url'],
                        "status": "draft",
                    }
                    promising_products_data.append(product_to_insert)
                    logger.info(f"Identified promising product: '{product['title']}'

            if promising_products_data:
                cli = service_client()
                response = cli.table("products").insert(promising_products_data).exe
                inserted_count = len(response.data) if response.data else 0
                logger.info({"event": "products_scouted_and_inserted", "count": inse
                return response.data

        except Exception as e:
            logger.error(f"An error occurred during the scouting process: {e}")
            return []
        finally:
            await browser.close()
            logger.info("Scouting mission complete. Browser closed.")

    return []
```

app/services/creatives.py

Python Copy

```
import os
import asyncio
import random
from typing import List, Dict, Any, Optional

import httpx
from loguru import logger
from supabase import Client, create_client

from ..core.db import service_client
from ..core.config import settings

class PictoryAPIClient:
    def __init__(self):
```

```
        self.api_key = settings.PICTORY_API_KEY
        self.base_url = "https://api.pictory.ai/v1"
        if not self.api_key:
            logger.warning("PICTORY_API_KEY not found. Video generation will be skipp

    @property
    def is_configured(self) -> bool:
        return bool(self.api_key)

    async def create_video_from_script(self, script_text: str, product_name: str) ->
        if not self.is_configured:
            raise ConnectionError("Pictory API client is not configured.")

        payload = {
            "name": f"Ad Creative for {product_name}",
            "video_generation_settings": {
                "input_type": "script",
                "script": script_text,
                "output_format": "1080p",
                "aspect_ratio": "9:16",
            },
            "voiceover_settings": {
                "provider": "elevenlabs",
                "voice": "Brian",
                "speed": "1.0"
            },
            "branding": {
                "logo_url": "https://example.com/logo.png",
                "font": "Arial",
                "primary_color": "#FFFFFF"
            }
        }
        headers = {
            "Authorization": f"Bearer {self.api_key}",
            "Content-Type": "application/json"
        }

        async with httpx.AsyncClient(timeout=60.0) as client:
            response = await client.post(f"{self.base_url}/video.render", headers=hea
            response.raise_for_status()
            job_data = response.json()
            logger.info(f"Successfully submitted video generation job: {job_data.get

            await asyncio.sleep(5)
            return {
                "video_url": f"https://cdn.pictory.ai/videos/{job_data.get('job_id',
```

```
                "status": "generated",
                "vendor_job_id": job_data.get("job_id")
            }

def _generate_script_variants(product: Dict[str, Any], count: int) -> List[str]:
    hooks = [
        f"Stop scrolling! You need to see this.",
        f"This is the {product.get('title', 'product')} you've been waiting for.",
        f"Tired of {product.get('niche', 'the usual hassle')}? We have the solution.'
        f"The secret to {product.get('niche', 'success')} is finally out.",
        f"Transform your {product.get('niche', 'daily routine')} with this one simple
    ]
    benefits = [f"It solves your {product.get('niche')} problems!", "It's life-chang
    social_proofs = [
        f"Join thousands of happy customers who made the switch!",
        f"Rated 4.9 stars by people just like you.",
        f"As seen in Forbes and TechCrunch.",
        f"Don't just take our word for it, see the results for yourself."
    ]
    ctas = [
        f"Tap the link in bio to get your {product.get('title', 'product')} today!",
        f"Limited time offer! Shop now and get 20% off.",
        f"Discover the difference. Learn more at our website.",
        f"Don't wait! Your upgrade is just a click away."
    ]

    scripts = []
    for _ in range(count):
        script_parts = [
            random.choice(hooks),
            random.choice(benefits),
            random.choice(social_proofs),
            random.choice(ctas)
        ]
        random.shuffle(hooks)
        random.shuffle(benefits)
        random.shuffle(social_proofs)
        random.shuffle(ctas)
        scripts.append("\\n".join(script_parts))
    return scripts

async def _fetch_product_details(product_id: str) -> Optional[Dict[str, Any]]:
    cli = service_client()
    try:
        response = cli.table("products").select("*").eq("id", product_id).single().e
        return response.data
```

```
    except Exception as e:
        logger.error(f"Failed to fetch product {product_id} from Supabase: {e}")
        return None

async def generate_for_product(product_id: str, variants: int = 5) -> Dict[str, Any]
    logger.info(f"Starting creative generation for product_id='{product_id}' with {v

    product_details = await _fetch_product_details(product_id)
    if not product_details:
        raise ValueError(f"Product not found: {product_id}")

    scripts = _generate_script_variants(product_details, variants)
    pictory_client = PictoryAPIClient()
    creatives_to_insert = []
    generated_video_url = None

    for i, script in enumerate(scripts):
        video_url = None
        status = "ready_for_manual_production"
        vendor_job_id = None

        if pictory_client.is_configured:
            try:
                logger.info(f"Attempting video generation for variant {i+1}/{variant
                result = await pictory_client.create_video_from_script(script, produ
                video_url = result.get("video_url")
                status = result.get("status", "generated")
                vendor_job_id = result.get("vendor_job_id")
                generated_video_url = video_url
                logger.success(f"API call successful for variant {i+1}. Video URL: {
            except (httpx.HTTPStatusError, httpx.RequestError, ConnectionError, asyn
                logger.warning(f"API call failed for variant {i+1}: {e}. Triggering
        else:
             logger.warning(f"Pictory API not configured. Triggering fallback for va

        creatives_to_insert.append({
            "product_id": product_id,
            "script": script,
            "status": status if video_url else 'ready',
            "variant": f"v{i+1}",
            "hook": script.split('\\n')[0] if script else None,
            "cta": script.split('\\n')[-1] if script else None,
        })

    if creatives_to_insert:
        try:
```

```
            cli = service_client()
            response = cli.table("creatives").insert(creatives_to_insert).execute()
            inserted_creatives = response.data if response.data else []
            logger.info(f"Successfully inserted {len(inserted_creatives)} creatives i

            assets_to_insert = []
            for creative in inserted_creatives:
                if generated_video_url:
                    assets_to_insert.append({
                        "creative_id": creative["id"],
                        "asset_type": "video",
                        "storage_path": generated_video_url,
                        "duration_seconds": 30,
                        "checksum": "mock_checksum"
                    })
            if assets_to_insert:
                cli.table("creative_assets").insert(assets_to_insert).execute()
                logger.info(f"Successfully inserted {len(assets_to_insert)} creative
        except Exception as e:
            logger.error(f"Database insert failed: {e}")
            raise

    return {
        "message": f"Successfully processed {len(creatives_to_insert)} creatives for
        "creatives_generated": len(creatives_to_insert)
    }
```

app/services/publish.py

Python Copy

```
import os
import logging
import time
import asyncio
from supabase import create_client, Client
from playwright.async_api import async_playwright
from playwright_stealth import stealth_async
import requests
from datetime import datetime

from ..core.db import service_client
from ..core.config import settings
from loguru import logger
```

```
supabase_client: Client = service_client()
INSTAGRAM_BUSINESS_ACCOUNT_ID = settings.INSTAGRAM_BUSINESS_ACCOUNT_ID
META_ACCESS_TOKEN = settings.META_ACCESS_TOKEN
GRAPH_API_VERSION = "v19.0"
TIKTOK_ACCESS_TOKEN = settings.TIKTOK_ACCESS_TOKEN
TIKTOK_OPEN_ID = settings.TIKTOK_OPEN_ID

async def publish_reel_api(video_url: str, caption: str) -> str:
    logger.info(f"Attempting to publish Reel to Instagram via API for video: {video_u

    if not INSTAGRAM_BUSINESS_ACCOUNT_ID or not META_ACCESS_TOKEN:
        raise ValueError("Instagram business account ID or Meta access token not con

    create_container_url = f"https://graph.facebook.com/{GRAPH_API_VERSION}/{INSTAGRA
    container_params = {
        'media_type': 'REELS',
        'video_url': video_url,
        'caption': caption,
        'access_token': META_ACCESS_TOKEN,
    }
    response = requests.post(create_container_url, params=container_params)
    response.raise_for_status()
    container_id = response.json().get('id')
    if not container_id:
        raise Exception(f"Failed to create media container: {response.text}")
    logger.info(f"Successfully created Instagram media container: {container_id}")

    for _ in range(20):
        status_url = f"https://graph.facebook.com/{GRAPH_API_VERSION}/{container_id}
        status_params = {'fields': 'status_code', 'access_token': META_ACCESS_TOKEN}
        status_response = requests.get(status_url, params=status_params)
        status_response.raise_for_status()
        status = status_response.json().get('status_code')
        logger.info(f"Container {container_id} status: {status}")
        if status == 'FINISHED':
            break
        if status == 'ERROR':
            raise Exception(f"Container processing failed for {container_id}")
        await asyncio.sleep(15)
    else:
        raise Exception(f"Container {container_id} did not become ready in time.")

    publish_url = f"https://graph.facebook.com/{GRAPH_API_VERSION}/{INSTAGRAM_BUSINES
    publish_params = {
        'creation_id': container_id,
```

```
            'access_token': META_ACCESS_TOKEN
        }
        publish_response = requests.post(publish_url, params=publish_params)
        publish_response.raise_for_status()
        media_id = publish_response.json().get('id')
        if not media_id:
            raise Exception(f"Failed to publish container: {publish_response.text}")

        logger.info(f"Successfully published Instagram Reel with media ID: {media_id}")
        return media_id

    async def publish_tiktok_api(video_url: str, caption: str) -> str:
        logger.info(f"Attempting to publish to TikTok via API for video: {video_url}")

        if not TIKTOK_ACCESS_TOKEN:
            raise ValueError("TikTok access token not configured.")

        post_video_url = f"https://open.tiktokapis.com/v2/post/publish/video/init/"
        headers = {
            "Authorization": f"Bearer {TIKTOK_ACCESS_TOKEN}",
            "Content-Type": "application/json; charset=UTF-8",
        }
        payload = {
            "post_info": {
                "title": caption,
                "privacy_level": "PUBLIC_TO_EVERYONE",
            },
            "source_info": {
                "source": "PULL_FROM_URL",
                "video_url": video_url,
            }
        }
        response = requests.post(post_video_url, headers=headers, json=payload)
        response.raise_for_status()
        data = response.json()

        if data.get("error", {}).get("code") != "ok":
            raise Exception(f"TikTok API Error: {data.get('error')}")

        publish_id = data.get("data", {}).get("publish_id")
        if not publish_id:
            raise Exception(f"Failed to initiate TikTok publish: {response.text}")

        logger.info(f"Successfully initiated TikTok publish with ID: {publish_id}.")
        return publish_id
```

```
async def publish_playwright_fallback(platform: str, video_path: str, caption: str,
    logger.warning(f"Using Playwright FALLBACK to publish to {platform}.")
    async with async_playwright() as p:
        browser = await p.chromium.launch_persistent_context(
            user_data_dir=profile_path,
            headless=False,
            proxy=proxy,
            args=['--disable-blink-features=AutomationControlled']
        )
        await stealth_async(browser)
        page = await browser.new_page()

        try:
            if platform == 'instagram':
                await page.goto("https://www.instagram.com/")
                await page.wait_for_timeout(5000)
                await page.locator("svg[aria-label='New post']").first.click()
                async with page.expect_file_chooser() as fc_info:
                    await page.get_by_role("button", name="Select from computer").cli
                file_chooser = await fc_info.value
                await file_chooser.set_files(video_path)
                await page.get_by_role("button", name="Next").click()
                await page.get_by_role("button", name="Next").click()
                await page.locator("div[aria-label='Write a caption...']").fill(capt
                await page.get_by_role("button", name="Share").click()
                await page.wait_for_selector("text=Your reel has been shared.", time
                logger.info(f"Successfully published to Instagram via Playwright.")
            elif platform == 'tiktok':
                await page.goto("https://www.tiktok.com/upload")
                await page.wait_for_timeout(5000)
                upload_frame = page.frame_locator('iframe[data-tt="Upload_index_ifra
                async with upload_frame.locator('input[type="file"]').expect_file_ch
                    await upload_frame.locator('div.jsx-2178722955.upload-card').cli
                file_chooser = await fc_info.value
                await file_chooser.set_files(video_path)
                await upload_frame.locator('div.DraftEditor-editorContainer > div').
                await upload_frame.locator("button:has-text('Post')").wait_for(state
                await upload_frame.locator("button:has-text('Post')").click()
                await page.wait_for_selector("text=Your videos are being uploaded",
                logger.info(f"Successfully published to TikTok via Playwright.")
            else:
                raise ValueError(f"Playwright fallback not implemented for platform:
        except Exception as e:
            await page.screenshot(path=f"error_{platform}_{time.time()}.png")
            logger.error(f"Playwright fallback for {platform} failed: {e}")
            raise
```

```
        finally:
            await browser.close()

async def process_publication_queue():
    logger.info("Starting publication agent...")
    while True:
        try:
            response = supabase_client.table("publications").select("*, creatives(scr
            publications = response.data

            if not publications:
                logger.info("No queued publications found. Sleeping for 60 seconds."
                await asyncio.sleep(60)
                continue

            publication = publications[0]
            pub_id = publication['id']
            platform = publication['platform']

            creative_assets = publication.get('creatives', {}).get('creative_assets'
            video_asset = next((asset for asset in creative_assets if asset['asset_ty

            if not video_asset:
                error_msg = f"No video asset found for publication {pub_id}"
                logger.error(error_msg)
                supabase_client.table("publications").update({"status": "failed", "er
                continue

            video_url = video_asset['storage_path']
            caption = publication.get('creatives', {}).get('script', 'Check out this

            logger.info(f"Processing publication ID: {pub_id} for platform: {platfor

            try:
                if platform == 'instagram':
                    post_id = await publish_reel_api(video_url, caption)
                    update_payload = {"status": "published", "post_url": f"https://ww
                elif platform == 'tiktok':
                    post_id = await publish_tiktok_api(video_url, caption)
                    update_payload = {"status": "published", "post_url": f"https://ww
                else:
                    raise NotImplementedError(f"API publishing for {platform} not imp

                supabase_client.table("publications").update(update_payload).eq("id",
                logger.info(f"Successfully published and updated publication ID: {pub
```

```
            except Exception as api_error:
                logger.error(f"API publishing failed for pub ID {pub_id}: {api_error}
                video_path = "/tmp/downloaded_video.mp4"
                try:
                    profile_path = f"./profiles/publish_agent_{platform}"
                    proxy_config = {"server": settings.PROXY_SERVER, "username": set
                    await publish_playwright_fallback(platform, video_path, caption,
                    update_payload = {"status": "published", "error": None, "details
                    supabase_client.table("publications").update(update_payload).eq("
                    logger.info(f"Successfully published via fallback and updated pu
                except Exception as fallback_error:
                    error_message = f"API Error: {api_error}. Fallback Error: {fallb
                    logger.critical(f"ALL publishing methods failed for pub ID {pub_
                    update_payload = {"status": "failed", "error": error_message}
                    supabase_client.table("publications").update(update_payload).eq("
        except Exception as e:
            logger.error(f"An unexpected error occurred in the main processing loop:
            await asyncio.sleep(30)
```

app/services/ads.py

Python Copy

```
import os
import logging
from supabase import create_client, Client
from facebook_business.api import FacebookAdsApi
from facebook_business.adobjects.adaccount import AdAccount
from facebook_business.adobjects.campaign import Campaign
from facebook_business.adobjects.adset import AdSet
from facebook_business.adobjects.adcreative import AdCreative
from facebook_business.adobjects.ad import Ad
import requests

from ..core.db import service_client
from ..core.config import settings
from loguru import logger

supabase_client: Client = service_client()
META_APP_ID = settings.META_APP_ID
META_APP_SECRET = settings.META_APP_SECRET
META_ACCESS_TOKEN = settings.META_ACCESS_TOKEN
META_AD_ACCOUNT_ID = settings.META_AD_ACCOUNT_ID
TIKTOK_ACCESS_TOKEN = settings.TIKTOK_ACCESS_TOKEN
```

```
TIKTOK_ADVERTISER_ID = settings.TIKTOK_ADVERTISER_ID
TIKTOK_API_VERSION = "v1.3"
SAFE_BUDGET_CAP = settings.SAFE_BUDGET_CAP_DAILY
PAUSE_CPA_THRESHOLD = 50.0
SCALE_ROAS_THRESHOLD = 3.0
MIN_SPEND_FOR_DECISION = 100.0

def init_meta_api():
    if not all([META_APP_ID, META_APP_SECRET, META_ACCESS_TOKEN, META_AD_ACCOUNT_ID]
        logger.warning("Meta Ads API credentials not fully configured.")
        return None
    try:
        FacebookAdsApi.init(META_APP_ID, META_APP_SECRET, META_ACCESS_TOKEN)
        logger.info("Meta Ads API initialized successfully.")
        return AdAccount(f'act_{META_AD_ACCOUNT_ID}')
    except Exception as e:
        logger.critical(f"Failed to initialize Meta Ads API: {e}")
        return None

def create_meta_campaign(ad_account: AdAccount, name: str, objective: str, budget: f
    params = {
        'name': name,
        'objective': objective,
        'status': Campaign.Status.paused,
        'special_ad_categories': [],
    }
    if is_cbo:
        params['daily_budget'] = int(budget * 100)
        params['budget_rebalance_flag'] = True

    campaign = ad_account.create_campaign(params=params)
    logger.info(f"Created Meta Campaign '{name}' with ID: {campaign['id']}")
    return campaign['id']

def create_meta_adset(ad_account: AdAccount, campaign_id: str, name: str, budget: fl
    params = {
        'name': name,
        'campaign_id': campaign_id,
        'status': AdSet.Status.paused,
        'billing_event': AdSet.BillingEvent.impressions,
        'optimization_goal': AdSet.OptimizationGoal.conversions,
        'targeting': targeting,
        'pixel_id': settings.META_PIXEL_ID,
    }
    if is_abo:
        params['daily_budget'] = int(budget * 100)
```

```
    adset = ad_account.create_ad_set(params=params)
    logger.info(f"Created Meta AdSet '{name}' with ID: {adset['id']}")
    return adset['id']

def create_meta_ad(ad_account: AdAccount, adset_id: str, name: str, creative_id: str
    params = {
        'name': name,
        'adset_id': adset_id,
        'creative': {'creative_id': creative_id},
        'status': Ad.Status.paused,
    }
    ad = ad_account.create_ad(params=params)
    logger.info(f"Created Meta Ad '{name}' with ID: {ad['id']}")
    return ad['id']

def create_tiktok_campaign(name: str, objective: str, budget: float):
    if not all([TIKTOK_ACCESS_TOKEN, TIKTOK_ADVERTISER_ID]):
        logger.warning("TikTok Ads API credentials not fully configured.")
        return None
    url = f"https://business-api.tiktok.com/open_api/{TIKTOK_API_VERSION}/campaign/cr
    headers = {"Access-Token": TIKTOK_ACCESS_TOKEN}
    payload = {
        "advertiser_id": TIKTOK_ADVERTISER_ID,
        "campaign_name": name,
        "objective_type": objective,
        "budget_mode": "BUDGET_MODE_DAILY",
        "budget": budget,
    }
    response = requests.post(url, headers=headers, json=payload)
    response.raise_for_status()
    data = response.json()
    campaign_id = data.get('data', {}).get('campaign_id')
    logger.info(f"Created TikTok Campaign '{name}' with ID: {campaign_id}")
    return campaign_id

def optimize_campaigns():
    logger.info("Starting ad optimization routine...")
    ad_account = init_meta_api()
    if not ad_account:
        logger.warning("Meta Ads API not initialized, skipping.")
        return

    try:
        fields = ['name', 'daily_budget', 'insights.date_preset(last_7d){spend,cpa,rc
        adsets = ad_account.get_ad_sets(fields=fields, params={'effective_status': [
```

```
        for adset in adsets:
            insights = adset.get('insights', {}).get('data', [{}])[0]
            spend = float(insights.get('spend', 0))
            if spend < MIN_SPEND_FOR_DECISION:
                continue

            cpa = float(insights.get('cpa', [{}])[0].get('value', float('inf')))\n
                logger.warning(f"PAUSING AdSet '{adset['name']}'. CPA ${cpa} > thresh
                AdSet(adset['id']).api_update(params={'status': AdSet.Status.paused}
                supabase_client.table("adsets").update({"status": "paused"}).eq("id"
                continue

            roas = float(insights.get('roas', [{}])[0].get('value', 0))
            if roas > SCALE_ROAS_THRESHOLD:
                current_budget = adset['daily_budget'] / 100.0
                new_budget = min(current_budget * 1.2, SAFE_BUDGET_CAP)
                if new_budget > current_budget:
                    logger.info(f"SCALING AdSet '{adset['name']}'. ROAS {roas:.2f} >
                    AdSet(adset['id']).api_update(params={'daily_budget': int(new_bud
                    supabase_client.table("adsets").update({"budget": new_budget}).ec
    except Exception as e:
        logger.error(f"An error during campaign optimization: {e}")

async def sync_ads_from_db():
    logger.info("Checking for new campaigns to create...")
    response = supabase_client.table("campaigns").select("*, adsets(*, ads(*))").is_(
    campaign_requests = response.data

    if not campaign_requests:
        logger.info("No new campaigns to sync.")
        return

    ad_account = init_meta_api()

    for req in campaign_requests:
        try:
            logger.info(f"Syncing new campaign: {req['name']}")
            is_cbo = req.get('budget_type') == 'CBO'
            if req['platform'] == 'meta':
                if not ad_account: continue
                campaign_id = create_meta_campaign(ad_account, req['name'], req['obje
                await supabase_client.table("campaigns").update({"id": campaign_id, "
                for adset_req in req.get('adsets', []):
                    adset_id = create_meta_adset(ad_account, campaign_id, adset_req[
                    await supabase_client.table("adsets").update({"id": adset_id, "st
```

```
                    for ad_req in adset_req.get('ads', []):
                        if ad_req['creative_ref']:
                            ad_id = create_meta_ad(ad_account, adset_id, ad_req['name
                            await supabase_client.table("ads").update({"id": ad_id, "
                Campaign(campaign_id).api_update(params={'status': Campaign.Status.a
                await supabase_client.table("campaigns").update({"status": "active"}
            elif req['platform'] == 'tiktok':
                campaign_id = create_tiktok_campaign(req['name'], req['objective'], r
                if campaign_id:
                    await supabase_client.table("campaigns").update({"id": campaign_i
        except Exception as e:
            logger.error(f"Failed to sync campaign '{req['name']}': {e}")
            await supabase_client.table("campaigns").update({"status": "failed", "err
```

app/services/crm.py

Python Copy

```
import json
from datetime import datetime
from typing import Any, Dict, List, Optional
from loguru import logger
from supabase import Client
from ..core.db import service_client

async def _get_or_create_customer(payload: Dict[str, Any], db_client: Client) -> Opti
    customer_info = payload.get("customer", {})
    email = customer_info.get("email", "").lower().strip()
    phone = customer_info.get("phone")

    if not email and not phone:
        logger.warning("Webhook with no customer email or phone.")
        return None

    try:
        if email:
            response = db_client.table("customers").select("id").eq("email", email).m
            if response and response.data:
                logger.info(f"Found existing customer by email: {email}")
                db_client.table("customers").update({"last_seen_at": datetime.now().i
                return response.data["id"]

        if phone:
            response = db_client.table("customers").select("id").eq("phone", phone).m
```

```
            if response and response.data:
                logger.info(f"Found existing customer by phone: {phone}")
                db_client.table("customers").update({"last_seen_at": datetime.now().
                return response.data["id"]

        logger.info(f"Creating new customer for email: {email}")
        insert_data = {
            "name": customer_info.get("name"),
            "email": email,
            "phone": phone,
            "first_seen_at": datetime.now().isoformat(),
            "last_seen_at": datetime.now().isoformat(),
        }
        response = db_client.table("customers").insert(insert_data).execute()

        if response.data:
            customer_id = response.data[0]["id"]
            logger.success(f"Created new customer with ID: {customer_id}")
            return customer_id
        else:
            logger.error("Failed to create new customer.", response.error)
            return None

    except Exception as e:
        logger.opt(exception=True).error(f"Error during customer upsert for email '{
        return None

async def _find_campaign_reference(utm_params: Dict[str, Any], db_client: Client) ->
    utm_campaign_name = utm_params.get("utm_campaign")
    if not utm_campaign_name:
        return None

    try:
        response = db_client.table("campaigns").select("id").eq("name", utm_campaign_
        if response and response.data:
            logger.info(f"Associated order with campaign '{utm_campaign_name}'")
            return response.data['id']
        else:
            logger.info(f"No campaign found matching utm_campaign: '{utm_campaign_nam
            return None
    except Exception as e:
        logger.error(f"Error finding campaign reference for '{utm_campaign_name}': {
        return None

async def process_checkout_webhook(payload: Dict[str, Any]) -> Dict[str, Any]:
    db_client = service_client()
```

```
    logger.info("Processing checkout webhook...")

    customer_id = await _get_or_create_customer(payload, db_client)
    if not customer_id:
        return {"status": "error", "message": "Failed to get or create customer."}

    utm_params = payload.get("tracking", {}).get("utm", {})
    campaign_ref_id = await _find_campaign_reference(utm_params, db_client)

    try:
        order_payload = payload.get("order", {})
        order_data = {
            "customer_id": customer_id,
            "product_id": order_payload.get("product_id"),
            "quantity": order_payload.get("quantity", 1),
            "price": order_payload.get("price"),
            "discount": order_payload.get("discount", 0),
            "cost_product": order_payload.get("product_cost"),
            "fees": order_payload.get("gateway_fee"),
            "channel": payload.get("source", "Unknown"),
            "utm_json": json.dumps(utm_params) if utm_params else None,
            "campaign_ref": campaign_ref_id,
            "purchased_at": order_payload.get("purchased_at", datetime.now().isoforma
            "status": order_payload.get("status", "paid"),
        }

        order_response = db_client.table("orders").insert(order_data).execute()
        if not order_response.data:
            raise Exception(f"Failed to create order: {order_response.error}")

        new_order = order_response.data[0]
        order_id = new_order["id"]
        logger.success(f"Created order with ID: {order_id}")

        items_payload = order_payload.get("items", [])
        if items_payload:
            order_items_to_insert = [
                {
                    "order_id": order_id,
                    "sku": item.get("sku"),
                    "qty": item.get("quantity", 1),
                    "unit_price": item.get("price"),
                }
                for item in items_payload
            ]
            items_response = db_client.table("order_items").insert(order_items_to_ins
```

```
            if not items_response.data:
                logger.warning(f"Order {order_id} created, but failed to insert items

        return {"status": "success", "customer_id": customer_id, "order_id": order_i

    except Exception as e:
        logger.opt(exception=True).error(f"Critical error during order processing: {
        return {"status": "error", "message": str(e)}
```

app/services/analytics.py

Python Copy

```
from datetime import date, timedelta, datetime
from typing import Any, Dict, List
import pandas as pd
from loguru import logger
from supabase import Client
from ..core.db import service_client

DECISION_RULES = {
    "ESCALAR": {"roas_threshold": 2.0, "duration_days": 7, "reason": "ROAS > 2.0 for
    "PAUSAR": {"roas_threshold": 1.0, "duration_days": 3, "reason": "ROAS < 1.0 for
    "SUBSTITUIR_CRIATIVO": {"ctr_drop_pct": 0.40, "duration_days": 5, "reason": "CTR
}

ALERT_THRESHOLDS = {"CRITICAL_ROAS": 1.1, "ANOMALOUS_SPEND_STD_DEV": 3.0, "ANOMALOUS_

async def consolidate_daily_metrics(report_date: date) -> Dict[str, Any]:
    db_client = service_client()
    logger.info(f"Consolidating metrics for date: {report_date}")

    try:
        response = db_client.from_("metrics_daily").select("*").eq("day", report_dat

        if not response.data:
            logger.warning(f"No data in 'metrics_daily' for {report_date}.")
            return {"status": "warning", "message": "No data for date."}

        daily_summary = response.data[0]

        new_customers_resp = db_client.table("customers").select("id", count="exact"
        new_customers_count = new_customers_resp.count or 0
```

```
        ad_spend = daily_summary.get('ad_spend', 0)
        cac = (ad_spend / new_customers_count) if new_customers_count > 0 else 0
        profit = daily_summary.get('revenue', 0) - daily_summary.get('total_cost', 0

        record = {
            "date": report_date.isoformat(), "level": "account", "ref_id": None,
            "spend": ad_spend, "revenue": daily_summary.get('revenue'), "roas": daily
            "cac": cac, "purchases": daily_summary.get('orders'), "owner_id": daily_s
        }
        db_client.table("traffic_metrics").upsert(record, on_conflict="date,level").e

        summary = {
            "date": report_date.isoformat(), "revenue": record['revenue'], "ad_spend'
            "profit": profit, "roas": record['roas'], "cac": cac,
            "orders": record['purchases'], "new_customers": new_customers_count
        }
        logger.success(f"Successfully consolidated metrics: {summary}")
        return summary
    except Exception as e:
        logger.opt(exception=True).error(f"Failed to consolidate daily metrics: {e}"
        return {"status": "error", "message": str(e)}

async def generate_decision_signals() -> List[Dict[str, Any]]:
    db_client = service_client()
    logger.info("Analyst agent starting: Generating decision signals...")
    signals = []

    try:
        start_date = (date.today() - timedelta(days=14)).isoformat()
        response = db_client.table("traffic_metrics").select("*").gte("date", start_d

        if not response.data: return []

        df = pd.DataFrame(response.data)
        df['date'] = pd.to_datetime(df['date'])
        df = df.sort_values(by=['ref_id', 'date'])

        for ref_id, group in df.groupby('ref_id'):
            level = group['level'].iloc[0]

            rule_escalar = DECISION_RULES["ESCALAR"]
            last_n_days = group.tail(rule_escalar['duration_days'])
            if len(last_n_days) == rule_escalar['duration_days'] and (last_n_days['ro
                signals.append({"agent": "AdsManager", "action": "ESCALAR", "payload_

            rule_pausar = DECISION_RULES["PAUSAR"]
```

```
            last_n_days = group.tail(rule_pausar['duration_days'])
            if len(last_n_days) == rule_pausar['duration_days'] and (last_n_days['roa
                signals.append({"agent": "AdsManager", "action": "PAUSAR", "payload_

        if signals:
            tasks_to_create = [{"agent": s["agent"], "action": s["action"], "payload_
            db_client.table("tasks").insert(tasks_to_create).execute()
            logger.success(f"Created {len(signals)} tasks from decision signals.")

        return signals
    except Exception as e:
        logger.opt(exception=True).error(f"Failed to generate decision signals: {e}"
        return []

async def check_for_critical_conditions() -> None:
    db_client = service_client()
    logger.info("Checking for critical conditions...")
    try:
        today = date.today()
        response = db_client.from_("metrics_daily").select("roas").eq("day", today.is
        if response and response.data and response.data.get('roas', 99) < ALERT_THRES
            alert = {"severity": "CRITICAL", "title": "ROAS Below Critical Threshold"
            db_client.table("alerts").insert(alert).execute()

        min_days = ALERT_THRESHOLDS["ANOMALOUS_SPEND_MIN_DAYS"]
        start_date = (today - timedelta(days=min_days)).isoformat()
        response = db_client.from_("metrics_daily").select("ad_spend").gte("day", sta
        if response and response.data and len(response.data) >= min_days:
            spends = [d['ad_spend'] for d in response.data]
            spend_series = pd.Series(spends)
            mean_spend, std_spend = spend_series.mean(), spend_series.std()

            today_spend_resp = db_client.from_("metrics_daily").select("ad_spend").eq
            today_spend = today_spend_resp.data.get('ad_spend', 0) if today_spend_res

            if today_spend > (mean_spend + ALERT_THRESHOLDS["ANOMALOUS_SPEND_STD_DEV"
                alert = {"severity": "HIGH", "title": "Anomalous Ad Spend", "message"
                db_client.table("alerts").insert(alert).execute()
    except Exception as e:
        logger.opt(exception=True).error(f"Failed to check for critical conditions: {
```

app/services/supervisor.py

Python Copy

```python
from . import products, creatives, publish, ads, analytics
from loguru import logger
from datetime import date, timedelta
from ..core.db import service_client

async def daily_plan():
    logger.info({"event": "plan_start"})

    scouted_products = await products.scout_once(search_term="trending dropshipping p

    if scouted_products:
        for product in scouted_products:
            if product.get('status') == 'draft':
                await creatives.generate_for_product(product['id'], variants=3)

    cli = service_client()
    response = cli.table("creatives").select("id").eq("status", "ready").execute()
    ready_creative_ids = [c['id'] for c in response.data]

    if ready_creative_ids:
        await publish.queue_publications(ready_creative_ids, platform="tiktok")
        cli.table("creatives").update({"status": "queued"}).in_("id", ready_creative_

    await ads.sync_ads_from_db()
    ads.optimize_campaigns()

    today = date.today()
    await analytics.consolidate_daily_metrics(today)
    await analytics.generate_decision_signals()
    await analytics.check_for_critical_conditions()

    logger.info({"event": "plan_end"})
```

app/api/http.py

Python Copy

```python
from fastapi import APIRouter
from ..services import products, creatives, publish, ads, analytics, supervisor
from datetime import date

router = APIRouter(prefix="/api")
```

```python
@router.post("/run/scout")
async def run_scout(search_term: str = "trending dropshipping product"):
    return await products.scout_once(search_term=search_term)

@router.post("/run/creatives/{product_id}")
async def run_creatives(product_id: str, variants: int = 5):
    return await creatives.generate_for_product(product_id, variants=variants)

@router.post("/run/publish")
async def run_publish(creative_ids: list[str], platform: str = "tiktok"):
    return await publish.queue_publications(creative_ids, platform=platform)

@router.post("/ads/launch")
async def ads_launch(name: str, platform: str = "meta", daily_budget: float = 50.0):
    return await ads.launch_campaign(name=name, platform=platform, daily_budget=daily

@router.get("/metrics/summary")
async def metrics_summary():
    return await analytics.consolidate_daily_metrics(date.today())

@router.post("/run/plan/daily")
async def run_plan_daily():
    await supervisor.daily_plan()
    return {"ok": True}
```

app/api/webhooks.py

Python Copy

```python
from fastapi import APIRouter, Request
from ..core.db import service_client
from ..services import crm

router = APIRouter(prefix="/api/webhooks")

@router.post("/checkout")
async def checkout_hook(req: Request):
    body = await req.json()
    result = await crm.process_checkout_webhook(body)
    return result
```

Step 3: Create Frontend Dashboard Files

Create a `dashboard` folder in your project root and add the following files. This creates a fully functional frontend dashboard to interact with the backend API and monitor the agent's performance.

> **Dashboard Entry Point**
> The dashboard's `index.html` should be used as the main entry point for the user-facing application, separate from the FastAPI backend.

`dashboard/index.html`

HTML Copy

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Agente-CEO Dashboard</title>
    <script src="https://cdn.tailwindcss.com?plugins=typography"></script>
    <link href="style.css" rel="stylesheet">
    <script src="https://unpkg.com/lucide@latest"></script>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://unpkg.com/@supabase/supabase-js@2"></script>
</head>
<body class="bg-gray-800 text-gray-100">
    <!-- Login View -->
    <div id="login-view" class="flex items-center justify-center h-screen">
        <div class="w-full max-w-md p-8 space-y-8 bg-gray-900 rounded-lg shadow-lg">
            <div class="text-center">
                <h2 class="text-2xl font-bold text-white">Agente-CEO Login</h2>
                <p class="text-gray-400">Access your autonomous agent dashboard</p>
            </div>
            <form id="login-form" class="space-y-6">
                <div>
                    <label for="email" class="text-sm font-bold text-gray-400">Email
                    <input id="email" type="email" required class="w-full p-2 mt-1 t
                </div>
                <div>
                    <label for="password" class="text-sm font-bold text-gray-400">Pa
```

```
                    <input id="password" type="password" required class="w-full p-2 r
                </div>
                <button type="submit" class="w-full py-2 px-4 bg-sky-600 hover:bg-sky
                <p id="login-error" class="text-red-400 text-sm text-center"></p>
            </form>
        </div>
    </div>

    <!-- Dashboard View -->
    <div id="dashboard-view" class="hidden">
        <div class="flex h-screen bg-gray-900">
            <!-- Sidebar -->
            <div class="w-64 bg-gray-800 flex flex-col">
                <div class="flex items-center justify-center h-16 border-b border-gra
                    <h1 class="text-white text-xl font-bold">Agente-CEO</h1>
                </div>
                <nav class="flex-1 px-2 py-4 space-y-2">
                    <a href="#" data-view="dashboard" class="nav-link active"><i data
                    <a href="#" data-view="products" class="nav-link"><i data-lucide=
                    <a href="#" data-view="campaigns" class="nav-link"><i data-lucide
                </nav>
                <div class="px-2 py-4">
                    <button id="logout-button" class="w-full flex items-center justi
                        <i data-lucide="log-out" class="mr-3"></i>Logout
                    </button>
                </div>
            </div>
            <!-- Main Content -->
            <main class="flex-1 p-6 overflow-y-auto">
                <div id="dashboard-content" class="content-section">
                    <h2 class="text-2xl font-bold mb-4">Main Dashboard</h2>
                    <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6
                        <!-- KPI cards here -->
                    </div>
                    <div class="mt-8">
                        <canvas id="metrics-chart" height="100"></canvas>
                    </div>
                </div>
                <div id="products-content" class="content-section hidden">
                    <h2 class="text-2xl font-bold mb-4">Products</h2>
                    <div id="products-table-container"></div>
                </div>
                <div id="campaigns-content" class="content-section hidden">
                    <h2 class="text-2xl font-bold mb-4">Campaigns</h2>
                    <div id="campaigns-container"></div>
                </div>
```

```
            </main>
        </div>
    </div>

    <script src="auth.js" type="module"></script>
    <script src="api.js" type="module"></script>
    <script src="ui.js" type="module"></script>
    <script src="charts.js" type="module"></script>
    <script src="app.js" type="module"></script>
</body>
</html>
```

dashboard/style.css

CSS Copy

```
.nav-link {
    display: flex;
    align-items: center;
    padding: 0.75rem 1rem;
    border-radius: 0.375rem;
    color: #d1d5db;
    font-weight: 500;
    transition: background-color 0.2s, color 0.2s;
}

.nav-link:hover {
    background-color: #374151;
    color: #ffffff;
}

.nav-link.active {
    background-color: #0284c7;
    color: #ffffff;
}

.nav-link i {
    width: 1.25rem;
    height: 1.25rem;
}
```

dashboard/auth.js

JavaScript Copy

```
const SUPABASE_URL = 'https://gpakoffbuypbmfiwewka.supabase.co';
const SUPABASE_ANON_KEY = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFz

const supabase = self.supabase.createClient(SUPABASE_URL, SUPABASE_ANON_KEY);

export const login = async (email, password) => {
    const { data, error } = await supabase.auth.signInWithPassword({ email, password
    return { user: data?.user, error };
};

export const logout = async () => {
    await supabase.auth.signOut();
};

export const getUser = async () => {
    const { data: { session } } = await supabase.auth.getSession();
    return session?.user ?? null;
};

export const getSession = async () => {
    const { data: { session } } = await supabase.auth.getSession();
    return session;
}

export const onAuthStateChange = (callback) => {
    return supabase.auth.onAuthStateChange((event, session) => {
        callback(session);
    });
};
```

dashboard/api.js

JavaScript Copy

```
import { getSession } from './auth.js';

const API_BASE_URL = 'http://localhost:8080/api';

async function authedFetch(url, options = {}) {
    const session = await getSession();
    const token = session?.access_token;
```

```
    const headers = {
        'Content-Type': 'application/json',
        ...options.headers,
    };

    if (token) {
        headers['Authorization'] = `Bearer ${token}`;
    }

    const response = await fetch(url, { ...options, headers });

    if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.detail || 'API request failed');
    }
    return response.json();
}

export const getMetricsSummary = () => {
    return authedFetch(`${API_BASE_URL}/metrics/summary`);
};

export const getProducts = () => {
    // This should ideally fetch from the /api/products endpoint
    // Using a Supabase direct call as a placeholder if API is not ready
    return authedFetch(`${API_BASE_URL}/products`);
};

export const getCampaigns = () => {
    // This should ideally fetch from the /api/campaigns endpoint
    return authedFetch(`${API_BASE_URL}/campaigns`);
};
```

dashboard/ui.js

JavaScript Copy

```
export function showView(viewId) {
    document.getElementById('login-view').classList.add('hidden');
    document.getElementById('dashboard-view').classList.add('hidden');
    document.getElementById(viewId).classList.remove('hidden');
}

export function switchContent(view) {
```

```
    document.querySelectorAll('.content-section').forEach(section => {
        section.classList.add('hidden');
    });
    document.getElementById(`${view}-content`).classList.remove('hidden');

    document.querySelectorAll('.nav-link').forEach(link => {
        link.classList.remove('active');
    });
    document.querySelector(`.nav-link[data-view="${view}"]`).classList.add('active')
}

export function renderProducts(products) {
    const container = document.getElementById('products-table-container');
    if (!products || products.length === 0) {
        container.innerHTML = '<p>No products found.</p>';
        return;
    }
    const table = `
        <div class="overflow-x-auto bg-gray-800 rounded-lg">
            <table class="min-w-full text-sm text-left">
                <thead class="bg-gray-700">
                    <tr>
                        <th class="p-3">Title</th>
                        <th class="p-3">Niche</th>
                        <th class="p-3">Cost</th>
                        <th class="p-3">Price</th>
                        <th class="p-3">Status</th>
                    </tr>
                </thead>
                <tbody>
                    ${products.map(p => `
                        <tr class="border-b border-gray-700">
                            <td class="p-3">${p.title}</td>
                            <td class="p-3">${p.niche}</td>
                            <td class="p-3">$${p.cost.toFixed(2)}</td>
                            <td class="p-3">$${p.price.toFixed(2)}</td>
                            <td class="p-3"><span class="px-2 py-1 text-xs rounded-fu
                        </tr>
                    `).join('')}
                </tbody>
            </table>
        </div>
    `;
    container.innerHTML = table;
}
```

```
// Add renderCampaigns and other UI functions as needed
export function renderCampaigns(campaigns) {
    const container = document.getElementById('campaigns-container');
    container.innerHTML = '<p>Campaigns view is under construction.</p>';
}
```

dashboard/charts.js

JavaScript Copy

```
let metricsChart = null;

export function createMetricsChart(summary) {
    const ctx = document.getElementById('metrics-chart').getContext('2d');

    // Mock data for demonstration
    const labels = ['Day 1', 'Day 2', 'Day 3', 'Day 4', 'Day 5', 'Day 6', 'Today'];
    const revenueData = [120, 190, 150, 250, 220, 300, summary.revenue || 450];
    const spendData = [50, 60, 55, 70, 65, 80, summary.ad_spend || 90];

    if (metricsChart) {
        metricsChart.destroy();
    }

    metricsChart = new Chart(ctx, {
        type: 'line',
        data: {
            labels,
            datasets: [
                {
                    label: 'Revenue',
                    data: revenueData,
                    borderColor: 'rgb(56, 189, 248)',
                    backgroundColor: 'rgba(56, 189, 248, 0.2)',
                    fill: true,
                    tension: 0.4,
                },
                {
                    label: 'Ad Spend',
                    data: spendData,
                    borderColor: 'rgb(244, 63, 94)',
                    backgroundColor: 'rgba(244, 63, 94, 0.2)',
                    fill: true,
                    tension: 0.4,
```

```
                },
            ],
        },
        options: {
            responsive: true,
            maintainAspectRatio: true,
            scales: {
                y: {
                    beginAtZero: true,
                    ticks: { color: '#9ca3af' },
                    grid: { color: '#374151' }
                },
                x: {
                    ticks: { color: '#9ca3af' },
                    grid: { color: '#374151' }
                }
            },
            plugins: {
                legend: { labels: { color: '#d1d5db' } }
            }
        },
    });
}
```

dashboard/app.js

JavaScript Copy

```
import { onAuthStateChange, login, logout } from './auth.js';
import { getMetricsSummary, getProducts } from './api.js';
import { showView, switchContent, renderProducts, renderCampaigns } from './ui.js';
import { createMetricsChart } from './charts.js';

document.addEventListener('DOMContentLoaded', () => {
    lucide.createIcons();

    const loginForm = document.getElementById('login-form');
    const logoutButton = document.getElementById('logout-button');

    loginForm.addEventListener('submit', async (e) => {
        e.preventDefault();
        const email = document.getElementById('email').value;
        const password = document.getElementById('password').value;
        const { error } = await login(email, password);
        if (error) {
```

```
            document.getElementById('login-error').textContent = error.message;
        } else {
            document.getElementById('login-error').textContent = '';
        }
    });

    logoutButton.addEventListener('click', () => {
        logout();
    });

    onAuthStateChange(async (session) => {
        if (session) {
            showView('dashboard-view');
            initDashboard();
        } else {
            showView('login-view');
        }
    });

    document.querySelectorAll('.nav-link').forEach(link => {
        link.addEventListener('click', (e) => {
            e.preventDefault();
            const view = e.currentTarget.dataset.view;
            switchContent(view);
        });
    });
});

async function initDashboard() {
    try {
        const summary = await getMetricsSummary();
        createMetricsChart(summary);

        const products = await getProducts().catch(() => []); // Mock products
        renderProducts(products);

    } catch (error) {
        console.error("Failed to initialize dashboard:", error);
    }
}
```

Step 4: Credentials & Environment Variables

Create a `.env` file in your project root by copying `.env.example`. This file stores sensitive keys and configuration settings. Below is a detailed explanation of each variable you need to set.

Supabase

SUPABASE_URL

Purpose: The unique URL for your Supabase project's API.

How to get: In your Supabase project dashboard, navigate to `Project Settings > API` and copy the `Project URL`.

SUPABASE_ANON_KEY

Purpose: The public, anonymous key for your Supabase project. It is safe to expose in a browser client.

How to get: In your Supabase project dashboard, go to `Project Settings > API > Project API Keys` and copy the `anon` `public` key.

SUPABASE_SERVICE_KEY

Purpose: The secret service role key. This key can bypass Row Level Security (RLS) and should NEVER be exposed on the client-side. Used for backend administrative tasks.

How to get: In your Supabase project dashboard, go to `Project Settings > API > Project API Keys` and copy the `service_role` `secret` key.

Agent Behavior

AUTO_MODE

Purpose: A boolean flag (`true` or `false`) to determine if the agent runs its daily plan automatically.

How to get: Set to `true` for autonomous operation or `false` for manual control via API endpoints.

APPROVAL_MODE

Purpose: A boolean flag. If `true`, the agent will require human approval (via tasks in the DB) before executing critical actions like launching ad campaigns.

How to get: Set to `true` for a safety check or `false` for full automation.

DRY_RUN

Purpose: A boolean flag. If `true`, the agent will simulate its actions (e.g., creating campaigns) without actually spending money or publishing content.

How to get: Set to `true` for testing and `false` for live operation.

Budget & Guardrails

SAFE_BUDGET_CAP_DAILY

Purpose: The maximum amount of money the agent is allowed to spend on ads per day.

How to get: Define a daily budget cap you are comfortable with (e.g., `300.0`).

SAFE_BUDGET_CAP_WEEKLY

Purpose: The maximum amount of money the agent is allowed to spend on ads per week.

How to get: Define a weekly budget cap (e.g., `1500.0`).

ALLOWED_CATEGORIES

Purpose: (Optional) A comma-separated list of product categories the agent is allowed to search for.

How to get: Define your own list, e.g., `health,beauty,home improvement`.

BLOCKED_CATEGORIES

Purpose: (Optional) A comma-separated list of product categories the agent should avoid.

How to get: Define your own list to prevent work on sensitive items, e.g., `weapons,adult,gambling`.

Meta (Facebook/Instagram)

META_APP_ID

Purpose: The ID of your Facebook App used for Meta Ads and Instagram Graph API access.

How to get: Create an app on the Meta for Developers portal. Find the ID in the app's dashboard.

META_APP_SECRET

Purpose: The secret key for your Facebook App.

How to get: Found in your app's `Basic Settings` on the Meta for Developers portal.

META_ACCESS_TOKEN

Purpose: A long-lived User Access Token with permissions like `ads_management`, `instagram_basic`, and `instagram_content_publish`.

How to get: Generate this token through the Meta Graph API Explorer tool and extend its expiration.

META_AD_ACCOUNT_ID

Purpose: The ID of your Meta Ads account (without the `act_` prefix).

How to get: Find it in your Meta Ads Manager URL or account settings.

META_PIXEL_ID

Purpose: The ID for your Meta Pixel for conversion tracking.

How to get: Find it in the Events Manager within your Meta Business Suite.

INSTAGRAM_BUSINESS_ACCOUNT_ID

Purpose: The ID of your Instagram Business Account, needed to publish Reels via the API.

How to get: Connect your Instagram to a Facebook Page and find the ID in your Meta Business Suite settings or via a Graph API call.

TikTok

TIKTOK_ACCESS_TOKEN

Purpose: The access token required to use the TikTok for Business API.

How to get: Generate it through the developer portal for your app at TikTok for Business.

TIKTOK_AD_ACCOUNT_ID / TIKTOK_ADVERTISER_ID

Purpose: The ID of your TikTok Ads advertiser account. Note: The API often refers to this as `advertiser_id`.

How to get: Find this ID in your TikTok Ads Manager dashboard.

TIKTOK_OPEN_ID

Purpose: The Open ID for your TikTok account, used for API interactions like publishing content.

How to get: Obtain this through the TikTok for Business developer authentication flow.

TIKTOK_PIXEL_ID

Purpose: The ID for your TikTok Pixel for conversion tracking.

How to get: Find it in the Events Manager within your TikTok Ads Manager.

Other Services

PICTORY_API_KEY

Purpose: (Optional) The API key for Pictory.ai, used for automatic video generation from scripts.

How to get: Find this in your account settings after logging into Pictory.

STORAGE_BUCKET

Purpose: The name of the bucket in Supabase Storage where media files (videos, thumbnails) will be stored.

How to get: The provided DDL script creates a bucket named `media`. You can keep this value or change it if you use a different name.

PLAYWRIGHT_PROXY_URL / PROXY_*

Purpose: (Optional) Credentials for a proxy service. Used by Playwright for web scraping and publishing as a fallback to official APIs.

How to get: Provide the `PROXY_SERVER`, `PROXY_USERNAME`, and `PROXY_PASSWORD` from your proxy service provider.

Step 5: Installation & Execution

Follow these steps to get your backend application running locally.

1. Create a `.env` file in your project root by copying `.env.example` and fill in your Supabase and other credentials as explained in Step 4.

2. Install the required Python dependencies:

Shell Copy

```
pip install -r requirements.txt
```

3. Run the local development server for the backend:

Shell Copy

```
uvicorn app.main:app --reload --port 8080
```

4. Open the `dashboard/index.html` file in your browser to use the frontend application.

Useful Test Endpoints

POST /api/run/scout

POST /api/run/creatives/{product_id}

POST /api/run/publish

POST /api/ads/launch

GET /api/metrics/summary

POST /api/run/plan/daily

POST /api/webhooks/checkout