CenterMind — Documento de Proyecto

Referencia técnica, hoja de ruta y lecciones aprendidas

Arquitectura general

CenterMind Server	Google Drive	Telegram API
centermind_core.py (orquestador)	fotos por distribuidora, grupo y fecha	un bot por distribuidora, evita saturar cuota

Flujo completo de una exhibición:

```
Vendedor

| (1) Envía foto al grupo de Telegram

▼

BotWorker

| (2) Pide NRO CLIENTE (texto numérico)

| (3) Muestra botones de tipo PDV

| (4) Descarga foto y la sube a Drive

| (5) Registra en SQLite — estado: 'Pendiente'

| (6) Envía mensaje de confirmación en el grupo

▼

SQLite.exhibiciones (estado = Pendiente, synced_telegram = 0)

| (7) Supervisor abre Streamlit y evalúa

▼

SQLite.exhibiciones (estado = Aprobado/Rechazado/Destacado)

| (8) sync job del bot cada 30s detecta synced_telegram = 0

| (9) Edita el mensaje original en Telegram con el resultado

| (10) Marca synced_telegram = 1

▼

Vendedor ve el resultado en el grupo
```

Estructura de archivos

```
CenterMind/ <- Raíz del servidor
|-- bot_worker.py [OK] Bot por distribuidora
|-- centermind_core.py [OK] Orquestador multi-bot
|-- setup_drive_oauth.py [OK] Setup OAuth2 Drive (con fix SSL)
|
|-- base_datos/
| +-- centermind.db <- Base de datos SQLite (hacer backup!)
|
|-- credencial_oauth.json <- OAuth2 Desktop (nunca a git!)
|-- token_drive.json <- Token generado por setup_drive_oauth.py
|
|-- streamlit_app/ [TODO] Fase 3
| |-- app.py
| |-- pages/
| | |-- 01_visor.py <- Evaluación de exhibiciones
| | |-- 02_dashboard.py <- Modo TV / ranking
| | +-- 03_admin.py <- Gestión de distribuidoras y usuarios
| +-- utils/
| +-- db.py <- Acceso a SQLite compartido
|
+-- logs/ <- Auto-generado

Portal_Exhibiciones/ [TODO] Fase 5 (.exe cliente)
```

Base de datos — Nomenclatura real de columnas

> ■■ IMPORTANTE: El esquema fue creado antes del código. Los nombres de columnas en la DB difieren de los usados en Python. La clase Database en bot_worker.py hace el mapeo con aliases SQL (AS). No renombrar columnas en la DB sin actualizar los queries en el código.

distribuidores

Columna en DB	Alias en código	Tipo	Notas

id_distribuidor	id	INTEGER PK	—
nombre_empresa	nombre	TEXT	—
token_bot	token_bot	TEXT	Token de BotFather
ruta_credencial_drive	—	TEXT	No se usa (credencial global)
id_carpeta_drive	drive_folder_id	TEXT	ID de carpeta raíz en Drive
estado	estado	TEXT	'activo' / 'inactivo'
admin_telegram_id	admin_telegram_id	TEXT	Recibe notificaciones del bot

exhibiciones

Columna en DB	Alias en código	Tipo	Notas
uuid	uuid	TEXT (UUID)	—
id_distribuidor	distribuidor_id	INTEGER FK	—
id_grupo	chat_id	INTEGER	Grupo donde se cargó
id_integrante	vendedor_id	INTEGER	telegram_user_id del vendedor
numero_cliente_local	nro_cliente	TEXT	Número de cliente
comentarios_telegram	tipo_pdv	TEXT	■■ Guarda tipo de PDV (nombre heredado)
url_foto_drive	drive_link	TEXT	Link de Drive a la foto
estado	estado	TEXT	Pendiente -> Aprobado / Rechazado / Destacado
supervisor_nombre	supervisor_nombre	TEXT	Quién evaluó (desde Streamlit)
comentarios	comentarios	TEXT	Comentario del supervisor
telegram_msg_id	telegram_msg_id	INTEGER	Mensaje a editar en Telegram
telegram_chat_id	telegram_chat_id	INTEGER	Chat donde vive ese mensaje
synced_telegram	synced_telegram	INTEGER	0 = notificar / 1 = ya notificado
timestamp_subida	created_at	DATETIME	Momento de carga
timestamp_evaluacion	evaluated_at	DATETIME	Momento de evaluación

> ■■ comentarios_telegram guarda el tipo de PDV. El nombre es confuso y heredado. A corregir en futura migración.

Hoja de ruta

■ Fase 1 — Infraestructura base (COMPLETADO)

- Estructura de carpetas del proyecto
- Base de datos SQLite con tablas relacionales
- Datos de prueba (distribuidora test, vendedores, grupo)
- Google Cloud: proyecto 'Bot-SQL', OAuth2 Desktop configurado
- Carpeta Drive CenterMind_Fotos autorizada con OAuth2
- credencial_oauth.json + token_drive.json en el servidor

■ Fase 2 — Bot de Telegram (COMPLETADO — testing OK)

- bot_worker.py: flujo completo foto → cliente → tipo PDV → Drive → SQLite
- centermind_core.py: orquestador multi-bot con reinicio automático (hasta 10 veces, delay 15s)
- Ráfaga de fotos: hasta 5 fotos en 8 segundos en una misma carga
- Hibernación configurable con flag HIBERNACION_ACTIVA
- Sync job cada 30s: edita mensajes en Telegram cuando Streamlit evalúa
- Sin evaluación en Telegram — solo carga
- Estado actual: Bot levanta, responde comandos y sube fotos a Drive. Testing completado.

■ Fase 3 — Streamlit: Visor de evaluación (PRÓXIMO)

• Login con usuarios_portal
• Vista de exhibiciones Pendientes: foto, vendedor, cliente, tipo PDV
• Botones Aprobar / Rechazar / Destacado con comentario opcional
• Al evaluar: actualiza estado y pone synced_telegram = 0
• Sync job del bot detecta el cambio y edita el mensaje en Telegram

■ Fase 4 — Streamlit: Dashboard y Admin

• Dashboard modo TV: ranking del mes en tiempo real
• Panel admin: CRUD de distribuidoras, usuarios del portal
• Reportes exportables CSV/Excel por período

■ Fase 5 — Portal Exhibiciones (.exe cliente)

• App Streamlit empaquetada con PyInstaller para Windows
• Módulos: Login, Visor, Dashboard (TV), Reportes

■ Fase 6 — Hardening y producción

• Logs centralizados con rotación
• Backup automático de centermind.db
• Monitoreo de uptime de los bots
• Script de onboarding para nuevas distribuidoras

Lecciones aprendidas (trampas ya encontradas)

Problema	Causa	Solución
database is locked	DB Browser abierto al correr el bot	Cerrar DB Browser antes de levantar el bot
no such column: id	Columna real es id_distribuidor	Aliases SQL en queries (id_distribuidor AS id)
no such column: rol	Columna real es rol_telegram	Aliases SQL en queries
no such column: distribuidor_id	En exhibiciones es id_distribuidor	Corregido en queries
JobQueue.run_daily() unexpected keyword 'timezone'	Este wrapper de PTB no acepta timezone como kwarg separado	Pasar tzinfo dentro del objeto time: dt_time(22, 0, tzinfo=AR_TZ)
ADD COLUMN IF NOT EXISTS syntax error	DB Browser usa SQLite < 3.37	Ejecutar ALTER TABLE sin IF NOT EXISTS, ignorar 'duplicate column'
Bot ignora fotos de noche	Hibernación activa 22:00-06:00	HIBERNACION_ACTIVA = False para testing
Error 400: redirect_uri_mismatch	Se creó credencial tipo 'Aplicación web' en lugar de 'Escritorio'	Eliminar credencial y crear nueva de tipo 'Aplicación de escritorio' (Desk
Error 403: access_denied	App en modo Testing y el email no está en la lista de usuarios de prueba	Ir a Pantalla de consentimiento OAuth → Usuarios de pr
Could not find TLS CA certificate bundle	PostgreSQL sobreescribe SSL_CERT_FILE del sistema apuntando a un path inexistente	En el script de arranque: os.environ.pop('SSL_CERT_FILE', None) antes de importar psycopg2 (usa certifi). En equipo nuevo:

Cómo correr el sistema

```
# Instalar dependencias (una sola vez)
```

```
pip install python-telegram-bot google-auth google-auth-oauthlib
google-api-python-client pillow certifi


# Setup OAuth2 Drive (una sola vez por servidor)

cd CenterMind

python setup_drive_oauth.py


# Levantar UN bot (debug / testing)

python bot_worker.py --distribuidor-id 1


# Levantar TODOS los bots (producción)

python centermind_core.py


# Solo una distribuidora desde el orquestador

python centermind_core.py --distribuidor-id 1
```

Archivos críticos

Archivo	Criticidad	Qué hacer si se pierde
base_datos/centermind.db	CRÍTICO	Se pierden todos los datos. Backup diario obligatorio
credencial_oauth.json	CRÍTICO	Sin esto no se suben fotos. Regenerar en Google Cloud Console
token_drive.json	ALTO	Correr setup_drive_oauth.py para regenerarlo
token_bot en distribuidores	ALTO	Crear bots nuevos en @BotFather y actualizar la DB
bot_worker.py / centermind_core.py	MEDIO	Están en el repositorio, se recuperan

---

Glosario

Término	Significado
Distribuidora	Cliente del sistema. Tiene su propio bot de Telegram y carpeta en Drive
PDV	Punto de Venta (kiosco, supermercado, almacén, etc.)
Exhibición	Una carga de foto por un vendedor, con sus metadatos
Integrante	Usuario de Telegram registrado en un grupo (vendedor o supervisor)