30-11 Olla App streamlit

Qué incluye (resumen rápido)

- **Admin Dashboard**
  - Métricas: ventas (count), ingresos total, comisión, bypass detectados.
  - Tabla de usuarios con botones para bloquear/activar (hace call a Supabase si está configurado; si no, usa datos mock/cached).
  - Exportar Excel con reporte financiero (download button).
  - Exportar PDF diario (ReportLab cuando está disponible, fallback a text file).
  - Monitor básico de salud (Supabase/Notion/config).
- **Modo Abuela**
  - Interfaz de alto contraste y **letras gigantes**.
  - Ventas del día mostradas grande y clara.
  - **Menú de la mañana** (campo de texto + botón para "Enviar a Notion" — placeholder con mensaje si no configuraste Notion).
  - Ganancias proyectadas y gráfico simple.
  - Botones grandes: **Hoy / Semana / Mes**.
  - Botón para guardar menú localmente (modo offline).
- **Características técnicas**
  - Conexión con **Supabase** vía `supabase-py` (usa `SUPABASE_URL` y `SUPABASE_KEY` desde las variables de entorno).
  - **Auto-refresh cada 2 minutos** implementado con un pequeño script JS (recarga la página).
  - Diseño pensado **mobile-first** (columnas adaptables, botones grandes).
  - **Modo offline básico**: cache en `st.session_state` y toggle en la barra lateral.
  - Export PDF y Excel.
  - Código incluye *fallbacks* y mocks para que puedas probar sin Supabase/Notion.

Cómo probarlo localmente

1. Descarga el archivo desde el enlace de arriba.
2. Instala dependencias (recomendado crear un virtualenv):

```
pip install streamlit supabase xlsxwriter reportlab
```

   - `reportlab` es opcional: si no está instalado, se usa un fallback simple al exportar PDF.
3. (Opcional) configura variables de entorno para conectar a Supabase y Notion:
   - `SUPABASE_URL`, `SUPABASE_KEY`
   - `NOTION_TOKEN`, `NOTION_DB_ID` (si querés la función real para Notion)
4. Ejecuta:

```
streamlit run /mnt/data/olla_dashboard_app.py
```

Notas y siguientes mejoras que puedo hacer si querés

- Integración real y segura con Notion (crear páginas en la DB con `notion-client`).
- Webhooks y logs de bypass más sofisticados + alertas por email/SMS.
- Gráficos más amigables (Recharts o Plotly) y filtros por fecha/productor.
- Tests unitarios y despliegue (Streamlit Cloud / Docker).
- Hacer que el auto-refresh sea más elegante sin recargar usando sockets/Streamlit session updates.

Si querés, puedo:

- Ajustar texto/colores/contraste exacto para "modo abuela".
- Añadir autenticación (Auth Supabase) y roles (admin / productor / abuela).
- Implementar la integración real con Notion y mostrarte exactamente qué campos necesitas en la DB.

1) Arquitectura propuesta (rápida)

- `streamlit-app/` → `olla_dashboard_app.py` (tu app actual).
- `webhook-service/` → FastAPI que recibe webhooks de bypass, guarda en Supabase y emite alertas.
- `notifier/` → módulo reutilizable para enviar email (SMTP) y SMS (Twilio) + registro a Supabase.
- `docker-compose.yml` → orquesta Streamlit + Webhook API + (opcional) local SMTP dev.
- `tests/` → pytest unit + integration tests.
- CICD: GitHub Actions para tests y build.

---

2) Webhook service (FastAPI) — recibir bypass / logs y almacenar en Supabase + emitir alerta

Crea `webhook-service/main.py`:

```
# webhook-service/main.py
from fastapi import FastAPI, Request, HTTPException
from pydantic import BaseModel
from datetime import datetime
import os
import asyncio
from supabase import create_client
from notifier.notifier import notify_bypass_if_needed

SUPABASE_URL = os.getenv("SUPABASE_URL")
SUPABASE_KEY = os.getenv("SUPABASE_KEY")

app = FastAPI(title="Olla Webhooks")

if not SUPABASE_URL or not SUPABASE_KEY:
    raise RuntimeError("SUPABASE_URL and SUPABASE_KEY must be set")
```

```
sb = create_client(SUPABASE_URL, SUPABASE_KEY)

class BypassEvent(BaseModel):
    order_id: int
    producer_id: str
    reason: str
    score: float = 0.0
    meta: dict = {}

@app.post("/webhook/bypass")
async def bypass(event: BypassEvent, request: Request):
    payload = event.dict()
    payload["created_at"] = datetime.utcnow().isoformat()
    # store in supabase table 'bypass_alerts'
    try:
        res = sb.table("bypass_alerts").insert(payload).execute()
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
    # async notify (don't block the webhook)
    asyncio.create_task(notify_bypass_if_needed(payload, sb))
    return {"status": "received", "id": res.data if hasattr(res, "data") else res}
```

---

3) Notifier: email + SMS + escalation logic

Crea `notifier/notifier.py`:

```
# notifier/notifier.py
import os
import smtplib
from email.message import EmailMessage
from twilio.rest import Client as TwilioClient
from datetime import datetime

SMTP_HOST = os.getenv("SMTP_HOST")
SMTP_PORT = int(os.getenv("SMTP_PORT", "587"))
SMTP_USER = os.getenv("SMTP_USER")
```

```
SMTP_PASS = os.getenv("SMTP_PASS")
TWILIO_SID = os.getenv("TWILIO_SID")
TWILIO_TOKEN = os.getenv("TWILIO_TOKEN")
TWILIO_FROM = os.getenv("TWILIO_FROM")
ALERT_EMAIL_TO = os.getenv("ALERT_EMAIL_TO")  # comma-separated
ALERT_SMS_TO = os.getenv("ALERT_SMS_TO")      # comma-separated E.1
64

def send_email(subject, body, to_list):
    msg = EmailMessage()
    msg["Subject"] = subject
    msg["From"] = SMTP_USER
    msg["To"] = to_list
    msg.set_content(body)
    with smtplib.SMTP(SMTP_HOST, SMTP_PORT) as s:
        s.starttls()
        s.login(SMTP_USER, SMTP_PASS)
        s.send_message(msg)

def send_sms(body, to_number):
    client = TwilioClient(TWILIO_SID, TWILIO_TOKEN)
    client.messages.create(to=to_number, from_=TWILIO_FROM, body=bod
y)

async def notify_bypass_if_needed(payload: dict, supabase_client=None):
    # Decide severity: score > 0.8 high
    score = payload.get("score", 0.0)
    reason = payload.get("reason", "unknown")
    order = payload.get("order_id")
    prod = payload.get("producer_id")
    created = payload.get("created_at", datetime.utcnow().isoformat())

    subject = f"[ALERTA BYPASS] {reason} - order #{order}"
    body = f"Alerta bypass detectada:\n\norder: {order}\nproducer: {prod}\n
reason: {reason}\nscore: {score}\ncreated: {created}\n\nPayload: {payloa
d}"

    # Always log to supabase table `bypass_alerts_log` for audit (optional)
```

```
    try:
        if supabase_client:
            supabase_client.table("bypass_alerts_log").insert({
                "order_id": order,
                "producer_id": prod,
                "reason": reason,
                "score": score,
                "payload": payload,
                "created_at": created
            }).execute()
    except Exception as e:
        # log to stdout; don't raise
        print("Error logging bypass to supabase:", e)

    # Escalation: high score ⇒ email + SMS; medium score ⇒ email
    if score >= 0.8:
        if SMTP_HOST and SMTP_USER:
            send_email(subject, body, os.getenv("ALERT_EMAIL_TO"))
        if TWILIO_SID and TWILIO_TOKEN and os.getenv("ALERT_SMS_TO"):
            for n in os.getenv("ALERT_SMS_TO").split(","):
                send_sms(subject + "\n" + reason, n.strip())
    elif score >= 0.5:
        if SMTP_HOST and SMTP_USER:
            send_email("[WARNING] " + subject, body, os.getenv("ALERT_EMAI
L_TO"))
    else:
        print("Bypass low score; logged only.")
```

> Variables de entorno: SMTP_HOST, SMTP_PORT, SMTP_USER, SMTP_PASS, TWILIO_SID, TWILIO_TOKEN, TWILIO_FROM, ALERT_EMAIL_TO, ALERT_SMS_TO.

---

4) Integración con Supabase: tablas sugeridas (migrations conceptuales)

- `bypass_alerts` (guardado inmediato de webhook)

- id (uuid), order_id (int), producer_id (text), reason (text), score (float), payload (json), created_at (timestamp)

- `bypass_alerts_log` (audit)
  - id, order_id, producer_id, reason, score, payload, created_at, handled_by, status
- `alerts_history` (si querés un resumen de notificaciones)

---

5) Plotly (gráficos más amigables) — snippets para añadir en Streamlit

Instalar: `pip install plotly`

Dentro del Streamlit (`olla_dashboard_app.py`) reemplaza el `st.line_chart` por Plotly para más control:

```
import plotly.express as px

# suponer df_daily: DataFrame with columns date, amount
df_daily = df.set_index("created_at").resample("D").sum().reset_index()
fig = px.line(df_daily, x="created_at", y="amount", title="Ingresos diarios (úl
timos 30 días)", markers=True)
fig.update_layout(margin=dict(l=10,r=10,t=40,b=10), height=300)
st.plotly_chart(fig, use_container_width=True)
```

Para filtros por productor + rango de fecha (añadilo en sidebar):

```
prod_selected = st.sidebar.selectbox("Productor", options=["Todos"] + list
(per_producer["producer_id"].unique()))
date_from = st.sidebar.date_input("Desde", value=datetime.today()-timedel
ta(days=7))
date_to = st.sidebar.date_input("Hasta", value=datetime.today())
# filtrar df con created_at entre date_from y date_to y por productor
```

---

6) Auto-refresh elegante sin recargar toda la página

Opciones que implementé/te recomiendo (ordenadas por simpleza → sofisticado):

1. `st.experimental_data_editor` + `st_autorefresh`:
   - Usa `st.experimental_singleton` o `st.cache_data` y `st_autorefresh` para recargar solo las partes que dependen de datos.
   - Ejemplo:

```
from streamlit_autorefresh import st_autorefresh
# pip install streamlit-autorefresh
count = st_autorefresh(interval=120000, limit=0, key="auto_refresh")
# on every tick, re-query caché (fetch_data_from_supabase) que está cach
eada por 60s
```

1. Long-polling interno (background thread que actualiza `st.session_state`):
   - Desencadena `st.experimental_rerun()` solo si hay cambios críticos (p. ej. nuevo bypass).
   - Ejemplo (simplificado):

```
import threading, time
def poller():
    while True:
        # check supabase latest bypass count
        # if changed: st.session_state['new_bypass']=True ; break
        time.sleep(60)

if 'poller_started' not in st.session_state:
    t = threading.Thread(target=poller, daemon=True)
    t.start()
    st.session_state['poller_started'] = True
if st.session_state.get('new_bypass'):
```

```
    st.experimental_rerun()
```

1. Socket / SSE: montar un microservicio (FastAPI) que emite SSE y en la app Streamlit usar JS para escuchar y actualizar DOM parciales. Es más trabajo, pero más instantáneo.

Te recomiendo la opción 1 o 2 para empezar y si querés implementamos SSE luego.

---

7) Tests unitarios (pytest) — ejemplos

`tests/test_notifier.py`:

```
from notifier.notifier import send_email, notify_bypass_if_needed

def test_dummy_notify_logs(monkeypatch):
    # monkeypatch SMTP/Twilio to avoid external calls and check they are in
voked
    calls = {}
    def fake_send_email(subject, body, to_list):
        calls['email'] = True
    monkeypatch.setattr("notifier.notifier.send_email", fake_send_email)
    # run
    import asyncio
    asyncio.run(notify_bypass_if_needed({"order_id":123,"producer_id":"p
1","reason":"test","score":0.9}))
    assert calls.get('email') is True
```

`tests/test_webhook.py` (fastapi test client):

```
from fastapi.testclient import TestClient
from webhook_service.main import app

client = TestClient(app)
def test_receive_bypass():
    payload = {"order_id": 999, "producer_id": "pX", "reason": "test", "scor
```

```
e": 0.9}
    res = client.post("/webhook/bypass", json=payload)
    assert res.status_code == 200
    assert res.json()["status"] == "received"
```

---

8) Docker + docker-compose (local)

`Dockerfile` para webhook:

```
# webhook-service/Dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY ./requirements.txt /app/requirements.txt
RUN pip install -r requirements.txt
COPY . /app
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--proxy-
headers"]
```

`docker-compose.yml` (simplificado):

```
version: "3.8"
services:
  streamlit:
    build: .
    image: olla_streamlit
    volumes:
      - ./streamlit-app:/app
    command: streamlit run olla_dashboard_app.py --server.port 8501 --serv
er.address 0.0.0.0
    ports:
      - "8501:8501"
    environment:
      - SUPABASE_URL=${SUPABASE_URL}
      - SUPABASE_KEY=${SUPABASE_KEY}
```

```
  webhook:
    build: ./webhook-service
    ports:
      - "8000:8000"
    environment:
      - SUPABASE_URL=${SUPABASE_URL}
      - SUPABASE_KEY=${SUPABASE_KEY}
      - SMTP_HOST=${SMTP_HOST}
      - SMTP_USER=${SMTP_USER}
      - SMTP_PASS=${SMTP_PASS}
```

---

9) Streamlit Cloud / despliegue

- Para Streamlit Cloud: push the repo, set environment variables in the app settings (SUPABASE_URL, SUPABASE_KEY, etc). Streamlit Cloud sirve solo al front — poné webhook service en un host (Railway/Heroku/Render) o en la misma máquina con Docker en VPS.
- Para despliegue con Docker: build images y subir a un registry (Docker Hub) y desplegar con `docker-compose` en VPS o usar GitHub Actions para build+push.

GitHub Actions minimal para tests + lint:

`.github/workflows/ci.yml`:

```
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
      - name: Install deps
```

```
      run: |
        pip install -r requirements.txt
        pip install -r requirements-dev.txt
    - name: Run tests
      run: pytest -q
```

---

10) Auth Supabase + roles (Admin / Productor / Abuela)

En Streamlit:

```
from supabase import create_client
from streamlit import session_state as ss

supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
# Login flow (email link or magic link)
email = st.text_input("Email")
if st.button("Entrar"):
    res = supabase.auth.sign_in_with_otp({"email": email})
    st.info("Chequeá tu email para el link mágico")

# after login, fetch user role (assuming 'role' in users table)
user = supabase.auth.get_user()
if user:
    row = supabase.table("users").select("role").eq("id", user['id']).single().e
xecute()
    role = row.data.get('role')
    ss['role'] = role
```

- Admins: pueden ver Admin Dashboard y bloquear usuarios.
- Productores: vista con métricas por productor.
- Abuela: modo abuela forzado si role == 'abuela'.

---

11) Ajustes UI "Modo Abuela" (colores exactos)

Propuesta CSS (añadir a `olla_dashboard_app.py`):

```
<style>
:root{
  --bg:#000000;
  --fg:#ffffff;
  --accent:#FFD700; /* amarillo fuerte */
  --btn-bg:#004AAD; /* azul saturado */
}
body { background: var(--bg); color: var(--fg); }
.hc { background: var(--bg); color: var(--fg); }
.hc .big { font-size: 48px; font-weight:800; letter-spacing:1px; }
.button-large { font-size: 28px; padding:18px 24px; border-radius:14px; }
</style>
```

Textos: usa frases cortas y verbo imperativo: “HOY: Ventas”, “VER MENÚ”.

---

12) Mapeo Notion para "Menú de la mañana"

Si implementás la integración con Notion (API v1), te recomiendo usar una DB con estas columnas:

- `title` (Título) → e.g. "Menú - 2025-12-01"
- `Fecha` (Date)
- `Contenido` (Rich text / plain text)
- `Autor` (Person/text)
- `Publicado` (checkbox)
- `Visibilidad` (select: Cocina/Docentes/Publico)

Con eso el push al DB es directo: crear página con propiedades.

---

13) Prioridad / Roadmap sugerido (rápido)

1. Implementar webhook FastAPI + notifier básico (email). (48–72h dev)
2. Añadir logging supabase + visualización en Admin (lista + filtros). (24h)
3. Reemplazar charts por Plotly y filtros por fecha/productor. (8–12h)
4. Tests + Docker + GitHub Actions. (8–16h)
5. Auto-refresh elegante (long-polling) / SSE si querés tiempo real. (16–24h)
6. Optional: Twilio SMS, Notion final, Auth Supabase. (variable)

---

14) Qué te entrego ya (lista concreta)

- Snippets completos arriba que podés pegar en tu repo (webhook, notifier, Docker, tests).
- Mapeo de tablas de Supabase y campos de Notion.
- Recomendaciones de despliegue y CI.
- CSS para Modo Abuela y Plotly snippets.

Si querés lo hago por vos (yo actualizo el repo / genero los archivos) — decime si preferís que:

A) **Implemente y te devuelva los archivos** (webhook-service, notifier, docker-compose, tests) listos para commitear;

B) **Solo snippets y guía** (lo que ya te pasé) y te ayudo a integrarlos paso a paso;

C) **Prioricemos** y yo haga primero Webhook + Notifier + Admin UI para ver logs en el dashboard.

Indicame A / B / C y genero los archivos listos (si elegís A o C, los crearé y te daré enlaces de descarga).