Laboratorio: Supabase BaaS

Backend de Ventas con Supabase + FE Cliente en Python

1) Objetivo

Construir un backend de ventas usando Supabase (PostgreSQL + PostgREST + Auth) que implemente:

- Modelo relacional con integridad referencial.
- RLS (Row-Level Security) por país del cliente y por categoría del producto.
- Usuarios de prueba para validar el RLS.
- Una app simple en Python (GUI) que se autentique y consuma endpoints (listar, filtrar e insertar una factura con su detalle). Puede generar la App GUI con algún GPT.

2) Alcance funcional

- CRUD básico de categorías y productos.
- CRUD básico de países y clientes.
- Registro de facturas y detalle de factura (líneas).
- Listados y filtros (por categoría, por país, por rango de fechas).
- RLS activo: un usuario solo ve/crea datos de sus países permitidos y sus categorías permitidas.

3) Requisitos técnicos

- Cuenta en Supabase y proyecto creado.
- Uso de Supabase Studio (SQL Editor, Table Editor, Auth).
- Postman o curl para probar endpoints.
- Python 3.10+ y paquete 'supabase' (pip install supabase).

4) Modelo de Datos (tablas y llaves)

Todos los nombres de tabla en public. Usar snake_case para columnas.

4.1 Dominios

countries (maestro de país)

- code text PK (ISO2 o ISO3)
- name text NOT NULL

categories (maestro de categoría de producto)

- id bigint PK identity
- name text UNIQUE NOT NULL

4.2 Comercial

products

- id bigint PK identity
- name text NOT NULL
- category_id bigint FK → categories.id
- unit_price numeric(12,2) NOT NULL CHECK unit_price >= 0
- created_at timestamptz default now()

customers

- id bigint PK identity
- name text NOT NULL
- email text
- country_code text FK → countries.code
- created_at timestamptz default now()

invoices (facturas)

- id bigint PK identity
- customer_id bigint FK → customers.id
- invoice_date date NOT NULL default current_date
- total_amount numeric(14,2) NOT NULL default 0
- created_at timestamptz default now()

invoice_lines (detalle de factura)

- id bigint PK identity
- invoice_id bigint FK → invoices.id ON DELETE CASCADE
- product_id bigint FK → products.id
- quantity numeric(12,2) NOT NULL CHECK quantity > 0
- unit_price numeric(12,2) NOT NULL CHECK unit_price >= 0
- line_total numeric(14,2) NOT NULL CHECK line_total >= 0

Reglas recomendadas: unit_price de la línea = precio del producto al momento de facturar (copiado). line_total = quantity * unit_price (validado por trigger o por aplicación).

4.3 Tablas de Autorización (para RLS)

user_allowed_country

- user_id uuid PK (→ auth.users.id)
- country_code text FK → countries.code NOT NULL

user_allowed_category

- user_id uuid PK (→ auth.users.id)
- category_id bigint FK → categories.id NOT NULL

Opcional: permitir múltiples países/categorías por usuario usando PK compuesta (user_id, country_code) y (user_id, category_id).

5) Script base (DDL)

```
-- Dominios

create table if not exists public.countries (

  code text primary key,

  name text not null

);


create table if not exists public.categories (

  id bigint generated always as identity primary key,

  name text not null unique

);


-- Comercial

create table if not exists public.products (

  id bigint generated always as identity primary key,

  name text not null,

  category_id bigint not null references public.categories(id),

  unit_price numeric(12,2) not null check (unit_price >= 0),

  created_at timestamptz default now()

);

…
```

6) Activar RLS y Políticas

```
alter table public.products enable row level security;
```

...

6.1 Políticas por categoría (products)

```
create policy "products_by_user_category_select"
on public.products for select
to authenticated
using (exists (
  select 1 from public.user_allowed_category u
  where u.user_id = auth.uid() and u.category_id = products.category_id
));

create policy "products_by_user_category_insert"
on public.products for insert
to authenticated
with check (exists (
  select 1 from public.user_allowed_category u
  where u.user_id = auth.uid() and u.category_id = products.category_id
));

create policy "products_by_user_category_update"
on public.products for update
to authenticated
using (exists (
  select 1 from public.user_allowed_category u
  where u.user_id = auth.uid() and u.category_id = products.category_id
```

```
))
with check (exists (
  select 1 from public.user_allowed_category u
  where u.user_id = auth.uid() and u.category_id = products.category_id
));

create policy "products_by_user_category_delete"
on public.products for delete
to authenticated
using (exists (
  select 1 from public.user_allowed_category u
  where u.user_id = auth.uid() and u.category_id = products.category_id
));
```

6.2 Políticas por país (customers)

```
create policy "customers_by_user_country_select"
on public.customers for select
to authenticated
using (exists (
  select 1 from public.user_allowed_country u
  where u.user_id = auth.uid() and u.country_code =
customers.country_code
));
```

…

6.3 Políticas en invoices (por país del cliente)

```
create policy "invoices_by_user_country_select"
on public.invoices for select
to authenticated
using (exists (
```

```
  select 1
  from public.customers c
  join public.user_allowed_country u
    on u.country_code = c.country_code and u.user_id = auth.uid()
  where c.id = invoices.customer_id
));
…
```

6.4 Políticas en invoice_lines (por país y categoría)

```
create policy "lines_by_country_and_category_select"
on public.invoice_lines for select
to authenticated
using (
  exists (
    select 1
    from public.invoices i
    join public.customers c on c.id = i.customer_id
    join public.user_allowed_country uc
      on uc.country_code = c.country_code and uc.user_id = auth.uid()
    where i.id = invoice_lines.invoice_id
  )
  and
  exists (
    select 1
    from public.products p
    join public.user_allowed_category ug
      on ug.category_id = p.category_id and ug.user_id = auth.uid()
    where p.id = invoice_lines.product_id
```

```
  )
);


create policy "lines_by_country_and_category_cud"
on public.invoice_lines for all
to authenticated
using (
  exists (
    select 1
    from public.invoices i
    join public.customers c on c.id = i.customer_id
    join public.user_allowed_country uc
      on uc.country_code = c.country_code and uc.user_id = auth.uid()
    where i.id = invoice_lines.invoice_id
  )
  and
  exists (
    select 1
    from public.products p
    join public.user_allowed_category ug
      on ug.category_id = p.category_id and ug.user_id = auth.uid()
    where p.id = invoice_lines.product_id
  )
)
with check (
  exists (
    select 1
    from public.invoices i
```

```
    join public.customers c on c.id = i.customer_id
    join public.user_allowed_country uc
      on uc.country_code = c.country_code and uc.user_id = auth.uid()
    where i.id = invoice_lines.invoice_id
  )
  and
  exists (
    select 1
    from public.products p
    join public.user_allowed_category ug
      on ug.category_id = p.category_id and ug.user_id = auth.uid()
    where p.id = invoice_lines.product_id
  )
);
```

7) Poblado mínimo y usuarios de prueba

1. Insertar countries (CR, US, ...), categories (Electronics, Furniture, ...).
2. Insertar products (con category_id válida).
3. Insertar customers (con country_code válido).
4. Crear usuarios en Auth → Users (confirmados).
5. Insertar filas en user_allowed_country y user_allowed_category con los UUID reales de los usuarios.

8) Endpoints REST de referencia (PostgREST)

```
-- Productos por categoría (RLS aplica)
GET /rest/v1/products?select=*&category_id=eq.<ID_CAT>

-- Clientes por país (RLS aplica)
GET /rest/v1/customers?select=*&country_code=eq.CR

-- Facturas con cliente embebido
```

```
GET /rest/v1/invoices?select=*,customers(*)


-- Detalle con producto embebido (RLS doble)

GET /rest/v1/invoice_lines?select=*,products(*)
```

Insertar factura y líneas (2 pasos): primero POST /rest/v1/invoices para obtener id; luego POST /rest/v1/invoice_lines con invoice_id, product_id, quantity, unit_price, line_total.

9) App cliente en Python (consola)

Requisitos: pip install supabase y variables de entorno SUPABASE_URL, SUPABASE_ANON_KEY, USER_EMAIL, USER_PASSWORD.

```
import os

from supabase import create_client, Client


URL = os.getenv("SUPABASE_URL")

KEY = os.getenv("SUPABASE_ANON_KEY")

EMAIL = os.getenv("USER_EMAIL")

PWD = os.getenv("USER_PASSWORD")


def login() -> Client:

    sb: Client = create_client(URL, KEY)

    auth = sb.auth.sign_in_with_password({"email": EMAIL, "password":
PWD})

    if not auth.session:

        raise SystemExit("Login failed.")

    print("Logged in:", auth.user.email)

    return sb


def list_my_products(sb: Client):

    res = sb.table("products").select("*").execute()
```

```
    print("Products (RLS applied):", res.data)


def list_my_customers(sb: Client):
    res = sb.table("customers").select("*").execute()
    print("Customers (RLS applied):", res.data)


def create_invoice(sb: Client, customer_id: int):
    inv = sb.table("invoices").insert({"customer_id":
customer_id}).select("*").execute()
    print("Invoice:", inv.data)
    return inv.data[0]["id"]


def add_line(sb: Client, invoice_id: int, product_id: int, qty: float,
unit_price: float):
    line_total = round(qty * unit_price, 2)
    line = {
        "invoice_id": invoice_id,
        "product_id": product_id,
        "quantity": qty,
        "unit_price": unit_price,
        "line_total": line_total
    }
    res = sb.table("invoice_lines").insert(line).select("*").execute()
    print("Line:", res.data)


def show_invoice_with_lines(sb: Client, invoice_id: int):
    inv = sb.table("invoices").select("*").eq("id",
invoice_id).execute()
    lines = sb.table("invoice_lines").select("*").eq("invoice_id",
invoice_id).execute()
```

```
    print("Invoice:", inv.data)

    print("Lines:", lines.data)


if __name__ == "__main__":

    sb = login()

    list_my_products(sb)

    list_my_customers(sb)

    # Completar: input() para IDs y crear factura + líneas
```

10) Entregables

1. Script SQL (/db/schema.sql) con DDL, índices, RLS y datos mínimos.
2. Colección Postman o guía de pruebas (/tests/postman.md).
3. App Python (/app/main.py) con README e instrucciones.
4. Video (≤ 5 min) o capturas mostrando RLS con dos usuarios diferentes.
5. Se revisará con el grupo, por tanto dejar todo listo para ejecutarse.

11) Extras (son obligatorios)

11.1. Crear una **función RPC** que **cree una factura y todas sus líneas en una única operación atómica** (transacción), expuesta por /rest/v1/rpc/....

- **RPC**: función SQL/PLpgSQL invocable por HTTP (PostgREST).
- **en transacción (PL/pgSQL)**: todo el trabajo (insertar factura + líneas) ocurre **o todo se guarda, o nada** (rollback si falla algo).
- **security invoker**: la función se ejecuta **con los permisos del usuario llamante**, por lo que **RLS se aplica** (no salta tus políticas).

```
{
  "customer_id": 123,
  "items": [
    { "product_id": 1, "quantity": 2 },                  // unit_price opcional
    { "product_id": 5, "quantity": 1, "unit_price": 39.9 }
  ]
}
```

```
create or replace function public.create_invoice(
  customer_id bigint,
  items jsonb
)
returns jsonb
language plpgsql
security invoker
as $$
$$
```

11.2. Vistas de reporte (vista ventas general para luego usar en las demás; ventas por categoría/país, top N productos).

create or replace view public.v_sales_fact as …
create or replace view public.v_sales_by_category as … group by …
create or replace view public.v_sales_by_country as … group by …
create or replace view public.v_top_products_30d as

Ejemplo de llamada:

```
GET {{SUPABASE_URL}}/rest/v1/v_sales_by_country?select=*&country_code=eq.CR
```

12) Generalidades

Entrega: 2 de octubre de 2022 hasta las 23:45 al TecDigital

Para realizar en Grupos de trabajo de proyectos.