Documentación Técnica: Integración con Supabase

---

1. Arquitectura de Conexión

La conexión con Supabase se maneja a través de un patrón Singleton para asegurar que solo exista una instancia del cliente en toda la aplicación.

1.1 Configuración (`config.py`)

Carga las variables de entorno y valida que las credenciales existan.

```
class Config: SUPABASE_URL = os.getenv('SUPABASE_URL') # Clave de servicio para operaciones de backend (bypasses RLS) SUPABASE_BACKEND_KEY = os.getenv('SUPABASE_SERVICE_KEY')
```

1.2 Módulo de Base de Datos (`database.py`)

Este módulo es el puente principal. Utiliza la librería oficial `supabase-py`.

```
from supabase import create_client _supabase_client = None def get_supabase_client(): global _supabase_client if _supabase_client is None: # Inicialización única _supabase_client = create_client( Config.SUPABASE_URL, Config.SUPABASE_BACKEND_KEY ) return _supabase_client
```

---

2. Operaciones CRUD con Supabase

Supabase utiliza una sintaxis fluida (method chaining) similar a SQL. A continuación se detallan las operaciones implementadas en los modelos.

2.1 Consultas (SELECT)

Ejemplo: Listar Productos (`producto_model.py`)

```
response = supabase.table('producto')\ .select('*')\ .order('nombre')\ .execute() return response.data # Retorna una lista de diccionarios
```

Ejemplo: Filtrar por ID (`usuario_model.py`)

```
response = supabase.table('usuario')\ .select('*, empleado(*)')\ # Join con
tabla empleado .eq('id_usuario', usuario_id)\ # WHERE id_usuario = ...
.execute()
```

Nota: `empleado(*)` realiza un JOIN automático gracias a las Foreign Keys definidas en Supabase.

2.2 Inserciones (INSERT)

Ejemplo: Crear Venta (`venta_model.py`)

```
data = { 'fecha': date.today().isoformat(), 'monto_total': total, 'id_cliente':
id_cliente } # .insert(data) inserta el diccionario response =
supabase.table('venta_completa').insert(data).execute() id_venta =
response.data[0]['id_venta_completa']
```

2.3 Actualizaciones (UPDATE)

Ejemplo: Actualizar Stock (`producto_model.py`)

```
# Actualiza el campo 'stock' donde 'id_producto' coincide
supabase.table('producto')\ .update({'stock': nuevo_stock})\ .eq('id_producto',
id_producto)\ .execute()
```

2.4 Eliminaciones (DELETE)

Ejemplo: Eliminar Cliente (`cliente_model.py`)

```
supabase.table('cliente')\ .delete()\ .eq('id_cliente', id_cliente)\ .execute()
```

---

3. Lógica de Negocio Implementada

3.1 Autenticación (`usuario_model.py`)

El sistema valida credenciales consultando directamente la tabla `usuario`. 1. Busca el usuario por ID. 2. Verifica que `activo` sea `True`. 3. Compara la contraseña (actualmente texto plano, preparado para bcrypt). 4. Retorna el objeto usuario con los datos de su empleado asociado.

3.2 Procesamiento de Ventas (`venta_model.py`)

Esta es la operación más compleja del sistema. Realiza los siguientes pasos de forma secuencial:

1. **Calcula Totales**: Suma los subtotales del carrito.
2. **Crea Venta Completa**: Inserta el registro "padre" en `venta_completa`.
3. **Procesa Items**:
4. Itera sobre cada producto del carrito.
5. Inserta el registro en `venta`.
6. **Actualiza el Stock**: Resta la cantidad vendida del inventario en `producto`.
7. **Maneja Donaciones**: Si hay redondeo, crea un registro en la tabla `donacion`.

3.3 Gestión de Inventario (`punto_venta_controller.py`)

El controlador ahora carga los productos dinámicamente al iniciar:

```
def cargar_productos(self): # Obtiene datos frescos de Supabase productos_db = self.ProductoModel.listar_todos() # Mapea al formato requerido por la UI self.productos = [...]
```

---

4. Estructura de Datos (Schema)

El sistema se basa en 7 tablas relacionales.

- **Tablas Catálogo**: `cliente`, `empleado`, `producto`.
- **Tablas Transaccionales**: `venta_completa`, `venta`, `donacion`.
- **Tabla de Seguridad**: `usuario`.

Tipos de Datos Clave

- **SERIAL**: Para IDs autoincrementales.
- **DECIMAL(n,2)**: Para importes monetarios y stock preciso.
- **VARCHAR**: Para textos con longitud definida.
- **DATE/TIME**: Para registro temporal exacto.

---

5. Seguridad y Buenas Prácticas

1. **Variables de Entorno**: Las credenciales no están en el código (`.env`).
2. **Service Role**: Se utiliza la clave de servicio para operaciones de backend, permitiendo gestionar datos sin restricciones de RLS (Row Level Security) desde la aplicación administrativa.
3. **Validación de Datos**: Los modelos incluyen validaciones básicas (ej. longitud de contraseña) antes de enviar a la base de datos.