Auditoría de Refactor Backend – Alejandro R.

Diego

Ale

Nina

- Problema 1: Carpetas
  - Problema: dos apps (api y clientes) con nombres confusos y posible duplicación de lógica.
  - Categoría: Arquitectura.
  - Severidad: Media.
  - Impacto: Dificulta el mantenimiento y la escalabilidad.
  - Sugerencia: Unificar si clientes es parte del dominio de api o renombrar ambas para reflejar su función real. Centralizar código común (serializers, utils, validaciones) y eliminar duplicados.
- Problema 2: Archivos
  - Problema: el archivo de contenedor se llama dockerfile en minúsculas.
  - Categoría: DevOps/Construcción.
  - Severidad: Baja.
  - Impacto: Algunos flujos/CI o herramientas esperan Dockerfile con D mayúscula; puede romper builds automatizados o requerir configuración extra.
  - Sugerencia: Renombrar a Dockerfile
- Problema 3: requirements.txt
  - Problema: las versiones no fijadas (ej.: Django>=4.2, djangorestframework sin versión).
  - Categoría: Dependencias / Reproducibilidad.
  - Severidad: Media.
  - Impacto: instalaciones diferentes entre máquinas (y a futuro podrían romperse cosas al subir de versión mayor, p. ej. Django 5.x).
  - Sugerencia: fijar versiones estables. Para Django, usa un rango de LTS: Django~=4.2 (o Django==4.2.*). Fija también DRF, django-cors-headers, etc.
- Problema 4: requirements.txt
  - Problema: mezcla de dependencias de desarrollo y despliegue en un solo archivo.
  - Categoría: Organización / Dependencias.
  - Severidad: Baja.
  - Impacto: menos claridad; instalaciones innecesarias en ciertos entornos (por ejemplo, gunicorn no se usa al desarrollar localmente).
  - Sugerencia: separar en:
    - requirements.txt (base)
    - requirements-dev.txt (que empiece con -r requirements.txt y agregue libs de dev/pytest si usas)
    - requirements-prod.txt (que empiece con -r requirements.txt y agregue gunicorn si hace falta)
- Problema 5: proyecto_django/urls.py
  - Problema: dos prefijos de API mezclados: api/cliente/ (singular) y api/ (general).
  - Categoría: API / Consistencia de rutas.

- Severidad: Media.
- Impacto: confunde al consumidor de la API y al equipo ¿por qué uno cuelga de api/cliente y otro de api/?).
- Sugerencia: unificar bajo un único prefijo y convención plural:
- path("api/", include("api.urls"))
- dentro de api.urls, definir subrutas como clientes/, pedidos/, etc.
- Eliminar path("api/cliente/", include("clientes.urls")) o mover esas rutas dentro de api/ como api/clientes/.

- Problema 6: api/urls.py
  - Problema: Importaciones duplicadas (OrdenViewSet y HistorialPedidosAPIView aparecen dos veces en la lista de imports).
  - Categoría: Estilo / Limpieza.
  - Severidad: Baja.
  - Impacto: Ruido y riesgo de errores al mantener; dificulta leer qué se usa realmente.
  - Sugerencia: Eliminar las líneas duplicadas en los from .views import ...
- Problema 7: api/urls.py
  - Problema: Dependencia cruzada: from clientes.views import ClienteViewSet dentro de api/urls.py.
  - Categoría: Arquitectura / Modularidad.
  - Severidad: Media.
  - Impacto: Acopla las apps api y clientes; complica mover o renombrar cosas y rompe la idea de que cada app es autónoma.
  - Sugerencia: Mover ClienteViewSet a la app api (o unificar apps)
- Problema 8: api/views.py
  - Problema: clase duplicada TelaListAPIView aparece dos veces (una hereda de ListAPIView y otra de APIView).
  - Categoría: Bug / Organización.
  - Severidad: Alta.
  - Impacto: la segunda definición sobreescribe la primera en tiempo de carga; es confuso y puede romper expectativas de paginación/filtros.
  - Sugerencia: dejar una sola versión (idealmente la de ListAPIView con paginación/filters) y eliminar la otra.
- Problema 9: api/views.py
  - Problema: VentasPorFechaAPIView está definida dos veces (la misma clase reaparece más abajo).
  - Categoría: Bug / Duplicación.
  - Severidad: Alta.
  - Impacto: la segunda definición reemplaza a la primera; mantenimiento confuso y riesgo de inconsistencias.
  - Sugerencia: conservar una única definición y borrar la duplicada.
- Problema 10: api/views.py
  - Problema: dependencias cruzadas dentro de views.py:
  - from clientes.models import Compra as CompraCliente (no se usa).
  - from clientes.views import ClienteViewSet (no debería importarse desde otra app en este nivel).
  - Categoría: Arquitectura / Limpieza.
  - Severidad: Media.
  - Impacto: acopla apps y ensucia el archivo con imports innecesarios.

    - Sugerencia: Eliminar imports no usados y mover ClienteViewSet a la app api (o unificar apps), evitando importarlo desde clientes.
- Problema 11: clientes/views.py (igual esto debería dejar de existir en general así que lo voy a dejar de analizar)
    - Problema: Solapamiento de dominios entre apps: en api ya existen ProveedorViewSet y CompraViewSet, y en clientes aparecen otra vez ProveedorViewSet y CompraViewSet.
    - Categoría: Arquitectura / Duplicación.
    - Severidad: Alta.
    - Impacto: rutas y modelos duplicados o inconsistentes; riesgo de que el frontend golpee "la versión equivocada" y se partan datos/contratos.
    - Sugerencia: elegir una sola app como "fuente de verdad" para Proveedor y Compra (recomiendo api) y eliminar las versiones duplicadas en clientes.
- Problema 12: api/models.py
    - Problema: Venta.cliente_id es IntegerField en vez de una ForeignKey al cliente.
    - Categoría: Modelo de datos / Integridad.
    - Severidad: Alta.
    - Impacto: no hay integridad referencial; puedes guardar un cliente_id que no existe y luego es difícil hacer joins/consultas.
    - Sugerencia: convertir a cliente = models.ForeignKey(Cliente, on_delete=models.SET_NULL, null=True, blank=True, related_name="ventas").
- Problema 13: api/models.py
    - Problema: Orden / DetalleOrden duplican dominio comparado con Pedido en la otra app, O no esta especificado que es cada uno.
    - Categoría: Modelo de dominio / Duplicación.
    - Severidad: Alta.
    - Impacto: dos conceptos para lo mismo → deuda y confusión.
    - Sugerencia: Unificar: escoge Orden o Pedido y migra el otro; usa un solo conjunto de modelos/serializers/rutas.
- Problema 14: api/seriallizers
    - Problema: Hay dos clases ProductoSerializer definidas en el mismo archivo. La segunda sobreescribe la primera.
    - Categoría: Bug / Estilo.
    - Severidad: Alta.
    - Impacto: la primera definición (que anidaba CategoriaSerializer) queda inutilizada, lo que puede causar confusión en imports y serialización inconsistente.
    - Sugerencia: conservar solo una definición. La segunda se ve mas completa
- Problema 15: api/serializers
    - Problema: CompraSerializer usa el campo monto_total, pero en el modelo Compra (api/models.py) el campo se llama monto.
    - Categoría: Bug / Consistencia.
    - Severidad: Alta.
    - Impacto: genera error en tiempo de ejecución (FieldError: Unknown field(s) (monto_total)), o nunca se serializará el monto correcto.
    - Sugerencia: cambiar fields = [...] para usar monto en lugar de monto_total.
- Problema 16: api/serializers
    - Problema: Dependencias cruzadas con clientes:

from clientes.models import Compra as CompraCliente (no usado).
from clientes.models import Cliente y from clientes.serializers import ClienteSerializer (usados en VentaDetalleSerializer).

- Categoría: Arquitectura.
- Severidad: Media.
- Impacto: acopla api con clientes; si se elimina/renombra algo en clientes, falla la API de api.
- Sugerencia: unificar el modelo Cliente en api

Tablas: TOTAL: 30

Tablas de la app api (12)

- api_categoria - models.Categoria (api/models.py)
- api_compra - models.Compra (api/models.py)
- api_detalleorden - models.DetalleOrden (api/models.py) (una de estas dos se tiene que ir)
- api_detalleventa - models.DetalleVenta (api/models.py)
- api_hilo - models.Hilo (api/models.py)
- api_operacion - models.Operacion (api/models.py) (revisar)
- api_orden - models.Orden (api/models.py)
- api_producto - models.Producto (api/models.py) (cambiar para que ya no se llame a la tabla de uniformes si no a la de productos) (Ale)
- api_proveedor - models.Proveedor (api/models.py)
- api_tela - models.Tela (api/models.py)
- api_uniforme - models.Uniforme (api/models.py) (lo mismo que lo de api_producto)
- api_venta - models.Venta (api/models.py)

Tablas de la app clientes (8)

- clientes_cliente - models.Cliente (clientes/models.py)
- clientes_compra - models.Compra (clientes/models.py)
- clientes_compradetalle - models.CompraDetalle (clientes/models.py)
- clientes_cuentapagada - models.CuentaPagada (clientes/models.py)
- clientes_empresa - models.Empresa (clientes/models.py)
- clientes_pedido - models.Pedido (clientes/models.py)
- clientes_pedidodetalle - models.PedidoDetalle (clientes/models.py)
- clientes_proveedor - models.Proveedor (clientes/models.py)

Todas esas se van y pasan a ser parte de api

Tablas base de Django (10) – pos al parecer se crean por default por django

- auth_group

- auth_group_permissions
- auth_permission
- auth_user
- auth_user_groups
- auth_user_user_permissions
- django_admin_log
- django_content_type
- django_migrations
- django_session

Interpretación de cada fucking tabla:

(aclarar que si alguna es parte del reporte de ventas no lo se, ya que no hay maneras de ingresar datos para eso)

1. Api_categoria: las categorías en inventario
2. Api_compra: la compra necesita de un proveedor, no se que se supone que sea
3. Api_detalleorden: requiere talla, producto, color, tela, bordado … pero no es nada del inventario ni pedidos, “bordado” no es un campo que tenga ninguno de esos
4. Api_detalleventa: se usa en la parte de reporte de ventas
5. Api_hilo: los hilos en inventario
6. Api_operacion: es parte del sistema de contabilidad, las operaciones (movimientos) como ingresos o egresos
7. Api_orden: posiblemente es pedidos, ya que requiere cliente, fecha y total (el cual se calcula automáticamente)
8. Api_producto: se usa en el reporte de ventas
9. Api_proveedor: tabla de proveedores
10. Api_tela: tabla de telas en inventario
11. Api_uniforme: tabla de uniformes en inventario
12. Api_venta: porte del reporte de ventas
13. Auth_group: define que id 1 es administrador y el id 2 es empleado (por que esto esta separado?)
14. Auth_group_permissions: tabla vacia, se supone que define los permisos de los 2 id anteriores (util?)

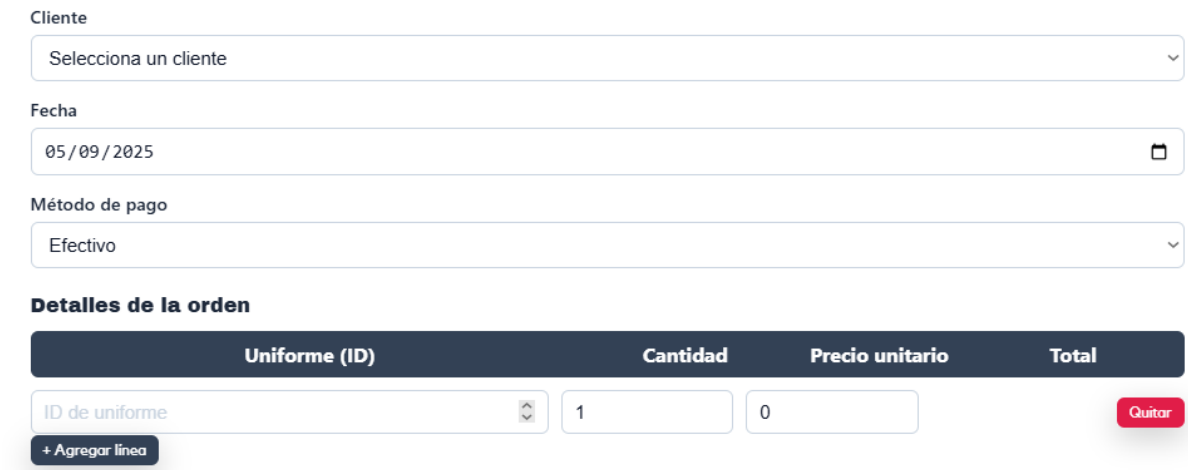
15. Auth_permmission: wtf es esto??

	id [PK] integer	name character varying (255)	content_type_id integer	codename character varying (100)
1	1	Can add log entry	1	add_logentry
2	2	Can change log entry	1	change_logentry
3	3	Can delete log entry	1	delete_logentry
4	4	Can view log entry	1	view_logentry
5	5	Can add permission	2	add_permission
6	6	Can change permission	2	change_permission
7	7	Can delete permission	2	delete_permission
8	8	Can view permission	2	view_permission
9	9	Can add group	3	add_group
10	10	Can change group	3	change_group
11	11	Can delete group	3	delete_group
12	12	Can view group	3	view_group
13	13	Can add user	4	add_user
14	14	Can change user	4	change_user
15	15	Can delete user	4	delete_user
16	16	Can view user	4	view_user
17	17	Can add content type	5	add_contenttype
18	18	Can change content type	5	change_contenttype
19	19	Can delete content type	5	delete_contenttype
20	20	Can view content type	5	view_contenttype
21	21	Can add session	6	add_session
22	22	Can change session	6	change_session

16. Auth_user: parte donde se guarda el usuario y su contraseña ya cifrada
17. Auth_user_groups: una table que guarda el id de los usuarios, les agrega un segundo id y dice si son del grupo 1 o 2 (que es esto?)
18. Auth_user_user_permissions: otra tabla vacia de permisos que no existen
19. Cliente_cliente: guarda los clientes
20. Clientes_compra: estoy casi seguro que no se usa en lo absoluto
21. Clientes_compradetalle: tampoco creo que sirva ni se use
22. Clientes_cuentapagada: lo mismo que la anterior
23. Clientes_empresa: empresa??? Una tabla para una variable??
24. Clientes_pedido: igual de inútil al parecer?
25. Clientes_pedidodetalle: no creo que se use
26. Clientes_proveedor: esta tabla si no se usa en lo absoluto, ya que tiene otra igual en api
27. Django_admin_log: no se usa, es default de django
28. Django_content_type: tabla de tablas? Xd no se no la entiendo
29. Django_migrations: tabla de migraciones (supongo, esta rara)
30. Django_sessions: esta toda cifrada

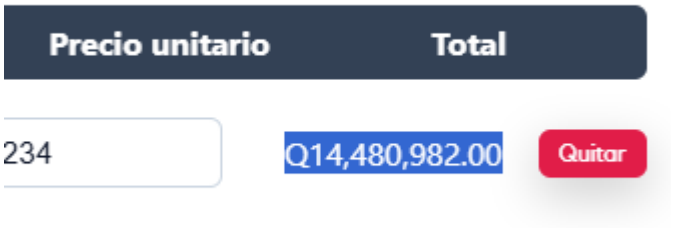
Dudas/Datos Extra:

1.



En pedidos, al agregar un pedido requiere el id de un uniforme, pero que es un uniforme? Es un producto??

2.



El Total, es invicible, pero si se calcula