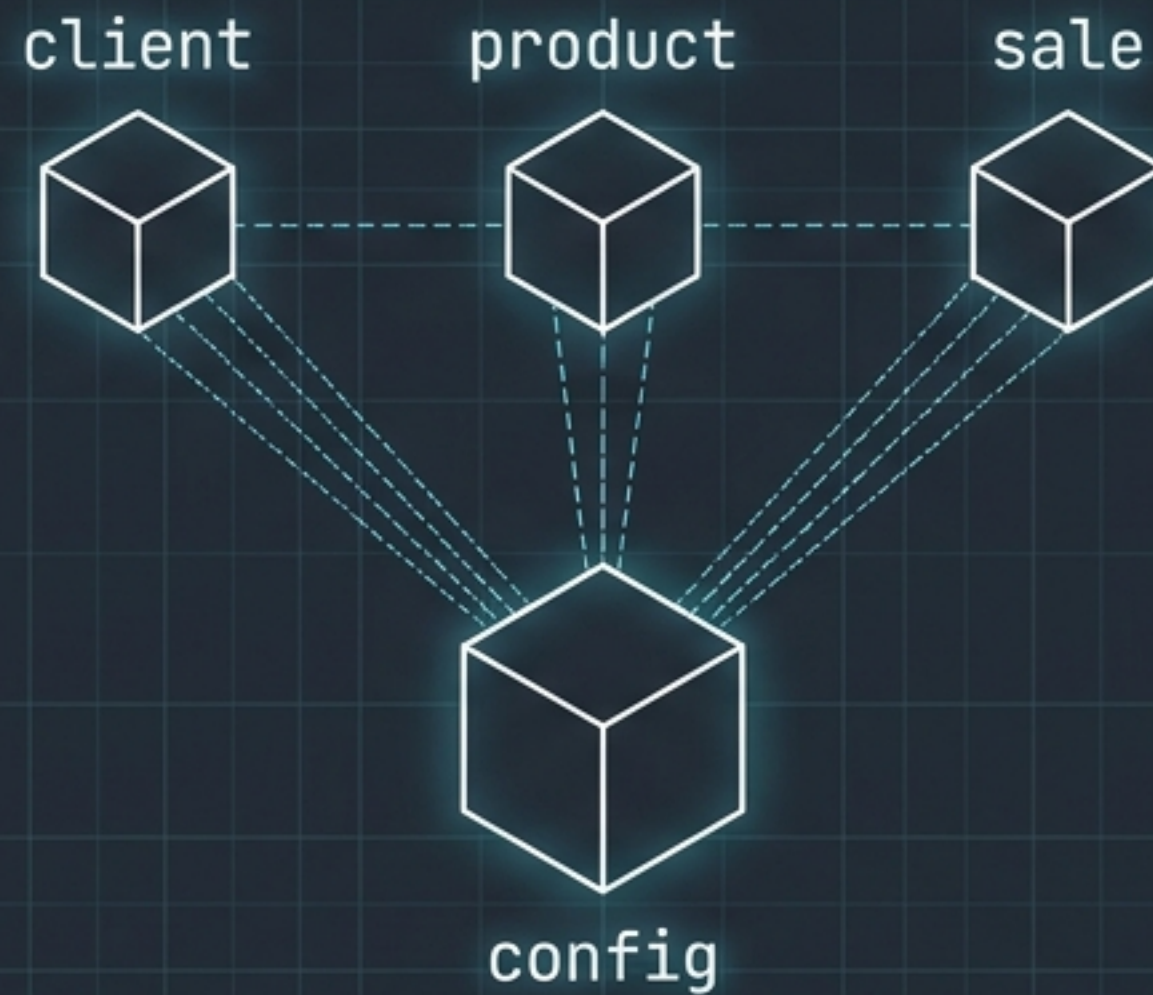




ISS-02 — Apps client, product y sale

Fundamentos de Arquitectura y Registro en Django



Sistemas Operativos



ONLINE

[x] ISS-01 Superado de manera exitosa.

100%

CHECK_ID: ISS-01

STATUS: PASSED



PASS

[x] Entorno virtual .venv activado en la sesión.

ENV_STATUS: ACTIVE

PYTHON_VERSION: 3.11.4

SESSION_ID: VENV-SESSION-45X9



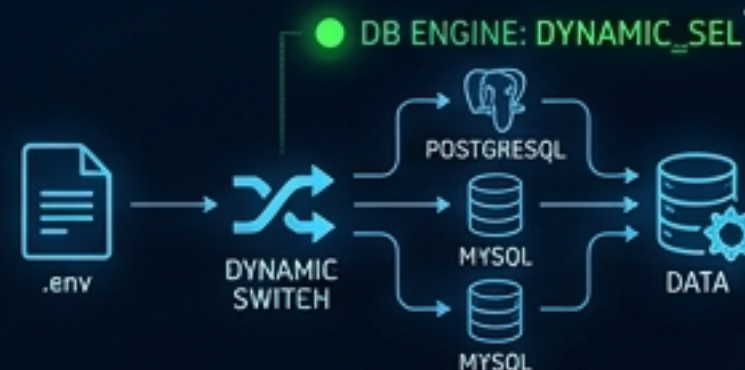
PASS

[x] Configuración .env y selección dinámica de motor de base de datos operando.

CONFIG_FILE: LOADED

DB_ENGINE_SELECT: POSTGRESQL

CONNECTION_POOL: ESTABLISHED



SYSTEM DIAGNOSTIC: 100% COMPLETE - ALL SYSTEMS GO

CPU LOAD: 12%
MEMORY: 22GB
NETWORK: ONLINE
LAST DIAGNOSTIC: [CURRENT DATE/TIME]

REGLA CRÍTICA DE ESTADO

AÚN NO SE CREAN MODELOS.

Ni vistas, ni URLs.



Los modelos de negocio se posponen estrictamente hasta el ISS-03 (Custom User).

El Motivo: Evitar conflictos irreversibles en las migraciones iniciales de Django. La arquitectura modular debe existir antes que las tablas.

SYSTEM DIAGNOSTIC: 100% COMPLETE - ALL SYSTEMS GO

CPU LDAB: 12S
MEMGRV: 226S
NETWRX: ONLINE
LAST DIAGNOSTIC: [CURRENT DATE/TIME]



Definición de Dominios de Negocio

Creación y registro de las tres Django apps estructurales. Organización modular estricta dentro de una carpeta contenedora principal.

apps/



client

Gestión de compradores.



product

Catálogo e inventario.



sale

Transacciones operativas.



De Plano Secundario a Estructura Modular

Antes: Proyecto raíz limpio

```
storelab/  
├── config/  
└── manage.py
```

El directorio raíz no se
contamina.

Después: Contenedor central

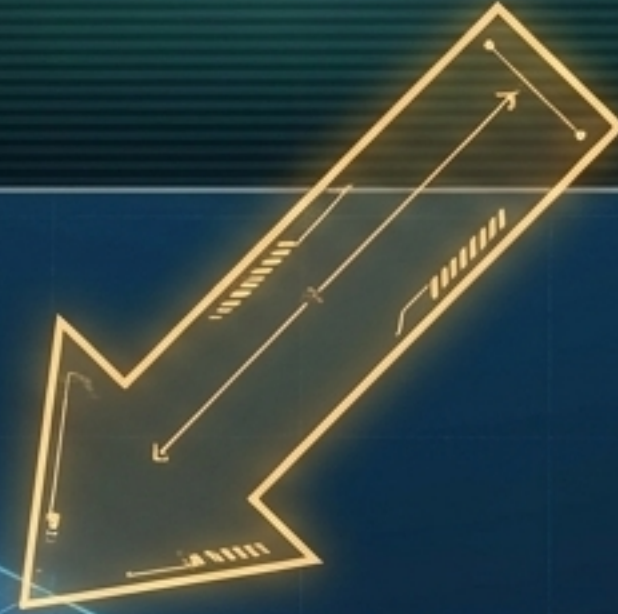
```
storelab/  
├── apps/  
│   ├── client/  
│   ├── product/  
│   └── sale/  
└── config/  
    └── manage.py
```

Aislando y albergando los tres
dominios de negocio.



Cimentación del Espacio Físico

```
$ mkdir -p apps/client apps/product apps/sale  
> █
```



La Herramienta: La CLI de Django (`startapp`) exige que el directorio destino ya exista físicamente cuando utilizamos rutas de ubicación anidadas.

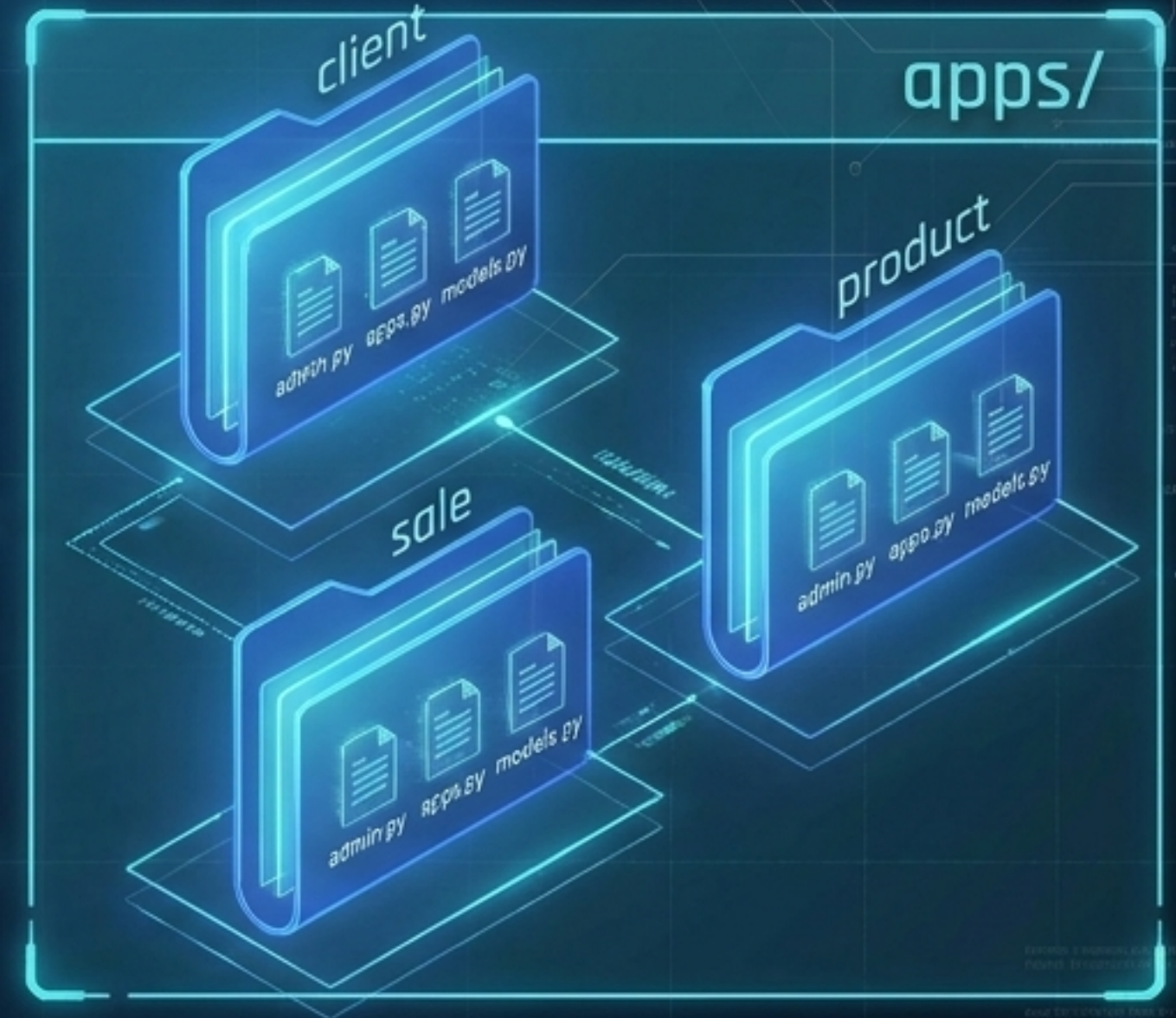


Generación de Boilerplate Estándar

```
$ python manage.py startapp  
client apps/client
```

```
$ python manage.py startapp  
product apps/product
```

```
$ python manage.py startapp  
sale apps/sale
```



Regla de oro: No copiamos archivos a mano.
La CLI garantiza la integridad de la estructura estándar.



El Adhesivo del Ecosistema Python

```
$ cat > apps/__init__.py <<'EOF'  
> █
```

El Rol Estructural

Transforma un directorio regular del sistema operativo en un Paquete Python real.

Dependencia Crítica

Sin este paso, Django y el intérprete de Python jamás encontrarán nuestras aplicaciones anidadas.



Identidad vs. Ubicación (AppConfig)



Ubicación

```
name = 'apps.client'
```

La ruta absoluta del módulo en Python. Django lo utiliza exclusivamente para ubicar e importar el código.

apps.py



Identidad

```
label = 'client'
```

El **identificador único** del dominio. Evita que las migraciones de base de datos arrastren el prefijo 'apps_' en los nombres físicos de las tablas.



Integración a la Red Central

client

apps.py

product

apps.py

sale

apps.py

config/settings/_init_.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
  
    # Local Apps  
    'apps.core',  
    'apps.client.apps.ClientConfig',  
    'apps.product.apps.ProductConfig',  
    'apps.sale.apps.SaleConfig',  
]
```

El motor central de configuraciones (settings) ahora reconoce oficialmente los tres dominios.



El Patrón Arquitectónico de Registro

Este ciclo define la incorporación correcta de cualquier aplicación dentro del framework StoreLab.

1. Generación CLI

mkdir + startapp
crea la base física.

2. Identidad AppConfig

Parcheo de name y
label para ajustar la
identidad de base de
datos.

3. Registro Central

Inyección en
INSTALLED_APPS
para conectar el
cableado al sistema.



Validación Estructural del Sistema

```
$ python manage.py check
```

```
System check identified  
no issues (0 silenced).
```

¿Qué garantiza esta validación?

- ✓ - Ausencia de errores de sintaxis en los parches aplicados.
- ✓ - Resolución correcta de los paquetes anidados.
- ✓ - Coherencia absoluta entre los AppConfig declarados y lo registrado en INSTALLED_APPS.



Métricas de Aceptación y Evidencia

Criterios de Aceptación (AC)

- ✓ AC-02-01: Apps creadas utilizando la herramienta startapp.
- ✓ AC-02-02: Ejecución de check sin reportes de errores.
- ✓ AC-02-03: Validación visual de cero modelos de negocio.

Artefacto de Evidencia

EVI-02-01

```
$ python manage.py check
```

```
System check identified  
no issues (0 silenced).
```

Captura del System check confirmando que no existen problemas de registro estructural.



Autorización de Fase (GATE)

Criterio	Método de Verificación	Evidencia	Resultado
AC-02-01 (Directorios de apps)	Inspección del Árbol	Repositorio	PASS
AC-02-02 (Verificación de integridad)	Comando CLI check	EVI-02-01	PASS
AC-02-03 (Restricción de modelos)	Revisión de código	Archivos vacíos	PASS





Demuestra tu Dominio

1.

¿Por qué la herramienta `startapp` de Django fallaría si omitimos ejecutar `mkdir -p` al trabajar con directorios anidados?

2.

¿Qué diferencia conceptual exacta existe entre asignar un `name` y un `label` dentro de `apps.py`?

3.

Si una app existe físicamente en el árbol de archivos pero olvidamos registrarla en `INSTALLED_APPS`, ¿qué comportamiento asume Django frente a ella?