Evaluación N° 4 – Backend.

Evaluación - UNIDAD 3

Instrucciones
La siguiente evaluación tiene como objetivo medir tu capacidad de desarrollar un proyecto con las herramientas entregadas.
Proyecto de Backend.

Se deberá implementar un desarrollo de una API, usando Django REST Framework para ser usada

Criterio	Puntos
Diseño (MER, normalización y modelos) (al menos las tablas indicadas en la descripción)	**12**
Implementación Auth & Roles (JWT, permission classes)	**14**
Funcionalidad básica (Products, Inventory, Branches (sucursales), Suppliers)	**14**
Ventas & Orders (POS + e-commerce + checkout)	**14**
Validaciones (RUT, fechas, numéricos, textos (al menos 1 de cada 1))	**8**
Templates y UX (Bootstrap, control de secciones por rol)	**14**
Configuración de Nginx y Gunicorn	**8**
Despliegue EC2	**10**
Documentación y comentarios	**6**
Total	**100**

Desarrollo & Diseño “TemucoSoft S.A.” —con base en Temuco que ofrece soluciones de software a pymes (tiendas, farmacias y pequeños supermercados). Desean desarrollar una solución modular de **POS + e-commerce** con gestión de inventario, proveedores y sucursales, ofrecida bajo **modelo de suscripción** (planes: Básico / Estándar / Premium) y con control de acceso por roles.

Objetivo del proyecto (encargo): Desarrollar una aplicación web basada en APIs (Django REST Framework) y templates (UI minimalista con Bootstrap) que permita a comercios clientes gestionar ventas presenciales (POS), ventas en línea (e-commerce), inventario, proveedores y sucursales. El sistema se desplegará en AWS EC2 y usará PostgreSQL como BD de producción. Se usará JWT para autenticar clientes (propietarios/empleados) y controlar acceso por roles (admin, gerente, vendedor).

Usuarios / roles (mínimos):

* `super_admin` (solo para la empresa TemucoSoft, configura cuentas de clientes, planes y facturación).
* `admin_cliente` (cliente: dueño de tienda — puede administrar todo en su "tenant")(tenant = arrendatario de la aplicación o suscriptor).
* `gerente` (gestión de inventario, reportes, proveedores).

- `vendedor` (realiza ventas POS, gestiona caja, no puede cambiar precios).
- `cliente_final` (usuario público para e-commerce; opcional)

Nota: para el ejercicio se pide implementar al menos `admin_cliente`, `gerente`, `vendedor` y `super_admin` (este último generado por el admin del sistema).

1. Diseñar y modelar una aplicación RESTful compleja (entidades: Product, Inventory, Supplier, Branch, Sale, Order, Subscription, User).
2. Implementar autenticación con JWT y autorización basada en roles.
3. Validar datos sensibles locales (RUT chileno) y campos (fechas, cantidades, precios).
4. Implementar endpoints REST con Django REST Framework y vistas templates con Bootstrap.
5. Integrar Postgres y desplegar la aplicación en EC2 (básico: servidor, gunicorn + nginx).
6. Entregar documentación, scripts de creación de datos y pruebas de endpoints (curl/Postman).
7. Trabajar en equipo y presentar la solución.

Requisitos funcionales (mínimos obligatorios)

Gestión de cuentas y roles

- Usuario custom (`AUTH_USER_MODEL`) con campos: `username`, `email`, `password`, `role` (choices), `is_active`, `rut` (RUT chileno), `company` (cliente/tenant), `created_at`.
- `super_admin` puede crear cuentas `admin_cliente` (para cada cliente) desde una interfaz (template) administrativa.
- Los usuarios se crean también vía `createsuperuser` para evaluación si es necesario.

Suscripciones y planes

- Modelo `Subscription` con `plan_name` (Básico/Estándar/Premium), `start_date`, `end_date`, `active`.
- El acceso a ciertas funcionalidades estará condicionado por el plan (ej.: límite de sucursales, reports, API para integración).

Productos e inventario

- `Product`: sku, name, description, price, cost, category.
- `Branch` (sucursal): nombre, dirección, teléfono.
- `Inventory`: relación `Branch x Product` con `stock`, `reorder_point`.
- `Supplier`: nombre, rut (validar), contacto.
- Movimientos de inventario (ingreso por compra de proveedor, salida por venta, ajuste).

Ventas (POS) y e-commerce (Orders)

- `Sale` (POS): branch, user (vendedor), items (product, quantity, price), total, payment_method, created_at.
- `Order` (ecommerce): cliente_final (nombre y email), items, estado (pendiente/enviado/entregado), total, created_at.

Carro de compras (Shop templates)

* Páginas templates: catálogo, detalle, carrito, checkout (confirmación).
* Para ecommerce se podrá comprar sin autenticación (opcional) o con usuario.

Proveedores

* CRUD de `Supplier`, vincular compras y registros a inventario (ingreso de stock).

Reportes (mínimos)

* Reporte de stock por sucursal.
* Reporte de ventas por periodo (día/mes) por sucursal.
* Reporte simple de proveedores (productos asociados y últimos pedidos).

API y templates

* Todas las funciones deben tener **endpoints DRF** para consumo futuro por frontend/APP.
* **NO** usar la interfaz automática de browsable API como UI principal, sí usar templates Bootstrap para la presentación.
* Templates deben usar la página de login; el menú muestra/oculta secciones según roles.

Seguridad y validaciones

* Validador RUT Chile (algoritmo DV).
* Validación de fechas (no aceptar fechas de venta futuras en POS, etc).
* Validación de campos numéricos (precios >= 0, stock >= 0).
* JWT para APIs (SimpleJWT) y control de roles en permission classes.
* `is_active` debe ser verificado antes de permitir el acceso.

Requisitos no funcionales

* BD: PostgreSQL (en producción) — en desarrollo puede usar sqlite3 pero final debe ser compatible con Postgres.
* Despliegue: instancias EC2 (una para la app + nginx + gunicorn; otra opcional para DB o RDS).
* Autenticación API: `Authorization: Bearer <access_token>`.
* UI: Bootstrap o similar.
* Documentación: README con pasos para levantar en local y deploy en EC2.

Modelo conceptual (alto nivel)

Entidades principales: `User`, `Company` (cliente/tenant), `Subscription`, `Branch`, `Supplier`, `Product`, `Inventory`, `Purchase` (orden de entrada desde proveedor), `Sale`, `Order`, `CartItem`, `Payment` (simplificado).

No doy MER completo aquí, pero en la actividad se pide que envíen un MER + tablas normalizadas (3NF).

Endpoints API mínimos (DRF)

Nota: prefijo `/api/` en todos.

Autenticación

* `POST /api/token/` — obtener JWT (username/password)
* `POST /api/token/refresh/` — refresh token

Usuarios

* `POST /api/users/` — crear usuario (solo super_admin o admin_cliente según contexto)
* `GET /api/users/me/` — info del usuario autenticado

Company / Subscription

* `GET /api/companies/`
* `POST /api/companies/` (super_admin)
* `POST /api/companies/{id}/subscribe/` (super_admin) — activar plan

Productos

* `GET /api/products/` — listado (publico si ecommerce)
* `POST /api/products/` — admin_cliente/gerente
* `PUT/PATCH /api/products/{id}/` — admin_cliente/gerente

Sucursales

* `GET /api/branches/`
* `POST /api/branches/` — admin_cliente
* `GET /api/branches/{id}/inventory/` — inventario por sucursal

Inventario

* `GET /api/inventory/?branch=...`
* `POST /api/inventory/adjust/` — ingreso/salida manual

Proveedores

* `GET /api/suppliers/`
* `POST /api/suppliers/` — admin_cliente/gerente

Compras / Ingreso de stock

* `POST /api/purchases/` — registrar compra a proveedor e incrementar stock

Ventas (POS)

* `POST /api/sales/` — registrar venta POS (requiere branch y vendedor)
* `GET /api/sales/?branch=&date_from=&date_to=` — reportes (gerente/admin)

Shop / Carro

* `GET /shop/products/` — template catálogo

- `GET /shop/products/{id}/` — template detalle
- `POST /api/cart/add/` — API para agregar (JWT)
- `GET /shop/cart/` — ver carro (template; si user logged show server cart)
- `POST /api/cart/checkout/` — convertir carro en `Order` y (si corresponde) vaciar stock

Reports

- `GET /api/reports/stock/` — stock por branch
- `GET /api/reports/sales/` — ventas por range

Validaciones específicas

Validación RUT (Chile)

- Implementar función validadora que reciba `rut` (con o sin puntos, con o sin guion) y calcule dígito verificador (DV). Rechazar si inválido.

Validación fechas

- `Sale.created_at` no puede estar en el futuro (comprobar con timezone.now()).
- `Purchase.date` no mayor a hoy.
- `Subscription.end_date` > `start_date`.

Validación numérica

- `Product.price` >= 0
- `Inventory.stock` >= 0 (no permitir stock negativo excepto por ajustes con justificación)
- `CartItem.quantity` >= 1

Templates y UX (solo templates; no usar el Admin DRF)

- Templates con Bootstrap 5.
- `login.html` con opción de obtener JWT (botón que solicita token y almacena en localStorage) *y* login por sesión para demostrar ambos enfoques.
- Menú principal muestra u oculta links según `user.role`.
    - `admin_cliente` ve: Productos, Proveedores, Sucursales, Inventario, Suscripción, Reportes, Usuarios (crear).
    - `gerente` ve: Productos, Proveedores, Inventario, Reportes.
    - `vendedor` ve: POS (vender), Productos (ver), Carro.
- Formularios con validación en frontend (HTML5 + pattern para RUT) y mensajes de error del backend.
- Páginas mínimas requeridas: login, dashboard (según rol), catálogo, producto detalle, cart/checkout, branch inventory, supplier list, purchase create, sales list, stock report.

Requisitos de despliegue (AWS EC2 + Postgres (EC2))

Topología recomendada mínima de evaluación

* **Opción A (mínima):** 1 EC2 (Linux (cualquiera)) con PostgreSQL instalado localmente, Nginx + Gunicorn + app. (más simple para alumnos).
* **Opción B (recomendada):** 2 EC2: app (Gunicorn + Nginx) y DB (Postgres) *o* RDS para la DB (más realista). Usar Security Groups para permitir sólo tráfico HTTP(S) en puerto 80/443 y solo conexiones Postgres desde la app (5432) desde la IP privada.

Configuración básica

* Crear instancia EC2 (capa gratuita), asociar Elastic IP.
* Security Group:
    - 22 (SSH)
    - 80 (HTTP)
    - 5432 (desde app EC2 private IP) DB en otra instancia.
* Gunicorn service + nginx config + collectstatic.
* JWT: mantener SECRET_KEY en variable de entorno.

Notas de evaluación: los alumnos deben entregar un documento con pasos de despliegue en EC2 y demostrar con captura de pantalla la app en funcionamiento desde la Elastic IP.

Requisitos técnicos mínimos que se evaluarán

* Uso de DRF para APIs y templates Bootstrap para UI.
* Implementación de JWT y control de acceso por roles (permision classes).
* Validadores (RUT, fechas, números) funcionando.
* Base de datos Postgres (o compatibilidad) y despliegue básico en EC2.
* Documentación clara y scripts reproducibles.

¡¡¡ ATENCION !!!

COPIA DE CUALQUIER TIPO A ALGUNO DE SUS COMPAÑEROS O TRAER ARCHIVOS DESDE CASA SERAN EVALUADOS CON NOTA MINIMA (1.0).