Instrucciones para la persona que hará el backend (Python Django)

1. Configuración inicial

- Crea un proyecto Django y una app llamada inventario.
- Instala dependencias:
    - django
    - djangorestframework
    - mssql-django (para SQL Server)
    - djangorestframework-simplejwt (para autenticación JWT)
    - django-cors-headers (para CORS)

2. Conexión a SQL Server

- En settings.py, configura la base de datos:

```
DATABASES = {
  'default': {
    'ENGINE': 'mssql',
    'NAME': 'restocontrol',
    'USER': 'usuario_sql',
    'PASSWORD': 'contraseña',
    'HOST': 'localhost',
    'PORT': '1433',
    'OPTIONS': {
      'driver': 'ODBC Driver 17 for SQL Server',
    },
  }
}
```

3. Modelos

- Crea modelos para:
    - **Usuario** (nombre, usuario, password, email, rol, activo, fecha_creacion)
    - **Producto** (nombre, descripcion, categoria, proveedor, stock, stock_minimo, precio, fecha_registro, activo)

    - **Categoría** (nombre, descripcion, activo)
    - **Proveedor** (nombre, contacto, telefono, email, direccion, activo)
    - **Configuración** (clave, valor)
- Usa claves foráneas para relaciones (producto-categoría, producto-proveedor).
- Ejemplo de modelo:

  class Categoria(models.Model):

    nombre = models.CharField(max_length=100)

    descripcion = models.TextField(blank=True)

    activo = models.BooleanField(default=True)

  class Producto(models.Model):

    nombre = models.CharField(max_length=100)

    descripcion = models.TextField(blank=True)

    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)

    proveedor = models.ForeignKey('Proveedor', on_delete=models.SET_NULL, null=True)

    stock = models.IntegerField(default=0)

    stock_minimo = models.IntegerField(default=0)

    precio = models.DecimalField(max_digits=10, decimal_places=2)

    fecha_registro = models.DateTimeField(auto_now_add=True)

    activo = models.BooleanField(default=True)

4. API REST

- Usa Django REST Framework para crear serializers y viewsets para cada modelo.
- Expón endpoints RESTful:
    - /api/productos/
    - /api/categorias/

    - /api/proveedores/
    - /api/usuarios/
    - /api/configuracion/
- Soporta métodos GET, POST, PUT/PATCH, DELETE.
- Implementa paginación, búsqueda y filtrado en los endpoints de productos y usuarios.

5. Autenticación y permisos

- Implementa autenticación JWT.
- Protege los endpoints sensibles (crear, editar, eliminar) para que solo usuarios autenticados y con permisos adecuados puedan usarlos.
- Define roles (admin, usuario, etc.) y verifica permisos en las vistas.

6. CORS

- Configura django-cors-headers para permitir peticiones desde el dominio del frontend (por ejemplo, http://localhost:5500).

7. Documentación

- Documenta los endpoints, los campos requeridos y ejemplos de request/response.
- Puedes usar Swagger (drf-yasg) para documentación automática.

8. Pruebas

- Proporciona ejemplos de cómo consumir la API desde JavaScript (fetch/axios).
- Ejemplo:

```
fetch('http://localhost:8000/api/productos/', {
  headers: { 'Authorization': 'Bearer <token>' }
})
.then(res => res.json())
.then(data => console.log(data));
```

Notas:

- El backend debe devolver y aceptar datos en formato JSON.
- El frontend se comunicará con la API usando HTTP (fetch/axios).