Simulador de Producción de Impresoras 3D

1. Objetivo del reto

Desarrollar un software que permita simular, día a día, el ciclo completo de una planta que fabrica impresoras 3D. El enfoque está en la gestión de inventarios, las compras y la planificación de la producción. El alumno/usuario juega el papel de planificador: decide qué fabricar y qué comprar.

---

2. Alcance y requisitos

2.1 Requisitos funcionales mínimos

0. **Definición de condiciones iniciales:** Se determina el plan de producción (materiales necesarios para la fabricación de cada tipo de impresora y tiempo en la cadena de montaje), catalogo proveedores (Productos a la venta, precios x cantidades - ejemplo precio: por palet de 1000 unidades, precio por caja de 20 unidades - tiempo de entrega/lead time) , capacidad de almacen (para simplificar 1 unidad de cualquier material = 1 unidad almacenaje).
1. **Generación de demanda**: al iniciar cada día se crean aleatoriamente pedidos de fabricación (parámetros: media y varianza configurables).
2. **Tablero de control**: muestra los pedidos pendientes, la lista de materiales (BOM/Bill of Materials) de cada pedido y el nivel de inventario.
3. **Decisiones del usuario**:
    - Liberar pedidos a producción.
    - Emitir órdenes de compra (elegir producto, proveedor, cantidad y fecha).
4. **Simulación de eventos**:
    - Consumo de materias primas y fabricación limitada por capacidad diaria.
    - Llegada de compras según el *lead time* del proveedor.
5. **Avance de calendario**: un botón “Avanzar día” ejecuta las 24 h de simulación.
6. **Registro de eventos** para históricos y gráficas.
7. **Exportación / importación JSON** de inventario y eventos.
8. **API REST:** Toda funcionalidad / información presentada por la interfaz de usuario debe ser accesible desde una api REST documentada con SWAGGER / OpenAPI

2.2 Requisitos no funcionales

- Código claro, comentado y versionado con **Git**.
- Interfaz web sencilla; ninguna instalación compleja en el cliente.

- Compatible con Windows, macOS y Linux.

---

3. Pila tecnológica propuesta

Capa	Herramienta	Motivo
Lenguaje	**Python 3.11/ 3.12**	Ampliamente usado, sintaxis simple.
Simulación	**SimPy**	Motor de eventos discretos fácil de usar.
Persistencia	SQLite **y/o** archivo JSON	Ligeros y portables.
Back-end/API	Fastapi + Pydantic	Solo si Streamlit no cubre todas las vistas.
Interfaz	**Streamlit**	Construcción rápida de dashboards.
Gráficas	`matplotlib`	Integración directa en Streamlit.
Control de versiones	**Git + GitHub**	Flujo estándar.

---

4. Modelo de datos inicial aproximado

```
Producto(id, nombre, tipo)                  # materia prima o terminado
BOM(prod_terminado_id, material_id, cant)

Proveedor(id, nombre, producto_id,          # qué vende
          costo_unitario, lead_time_dias)

Inventario(producto_id, cantidad)

PedidoFab(id, fecha_creac, producto_id,
          cantidad, estado)

OrdenCompra(id, proveedor_id, producto_id,
            cantidad, fecha_emision, fecha_entrega_est, estado)

Evento(id, tipo, fecha_sim, detalle)        # todos los sucesos
```

4.1 Ejemplo plan de producción

```
{
  "capacity_per_day": 10,
  "models": {
    "P3D-Classic": {
      "bom": {
        "kit_piezas": 1,
        "pcb": 1,
        "pcb_ref": "CTRL-V2",
        "extrusor": 1,
        "cables_conexion": 2,
        "transformador_24v": 1,
        "enchufe_schuko": 1
      }
    },
    "P3D-Pro": {
      "bom": {
        "kit_piezas": 1,
        "pcb": 1,
        "pcb_ref": "CTRL-V3",
        "extrusor": 1,
        "sensor_autonivel": 1,
        "cables_conexion": 3,
        "transformador_24v": 1,
        "enchufe_schuko": 1
      }
    }
  },
  "plan": [
    {
      "day": 1,
      "orders": [
        { "model": "P3D-Classic", "quantity": 8 }
      ]
    },
    {
      "day": 2,
      "orders": [
        { "model": "P3D-Classic", "quantity": 5 },
        { "model": "P3D-Pro", "quantity": 4 }
      ]
    },
    {
      "day": 3,
      "orders": [
        { "model": "P3D-Pro", "quantity": 10 }
      ]
    }
```

```
  ]
}
```

5. Estructuras de datos sugeridas en Python

```python
from pydantic import BaseModel
from typing import Literal

class Product(BaseModel):
    id: int
    name: str
    type: Literal["raw", "finished"]

class InventoryItem(BaseModel):
    product_id: int
    qty: int

class Supplier(BaseModel):
    id: int
    product_id: int
    unit_cost: float
    lead_time: int  # días
```

(completa con las demás según el modelo)

---

6. Flujo diario resumido

1. **Avanzar día** → SimPy corre 24 h.
2. Se generan nuevos pedidos.
3. El tablero se actualiza y el alumno/usuario decide.
4. SimPy procesa compras y producción.
5. Fin del día: se registra todo en `Evento`.

---

7. Interfaz de usuario mínima

- **Encabezado**: Día simulado y botón *Avanzar día*.
- **Panel Pedidos**: tabla de pedidos pendientes con cálculo automático de BOM.
- **Panel Inventario**: niveles actuales y faltantes.
- **Panel Compras**: lista desplegable de proveedores, campo de cantidad, botón *Emitir orden*.

- **Panel Producción**: capacidad diaria, pedidos en cola y en curso.
- **Gráficas**: nivel de stock y número de pedidos terminados en el tiempo.

---

8. Escenario de ejemplo

- **Día 1**: stock inicial = 30 kits de piezas, capacidad = 10 impresoras/día. El generador crea dos pedidos de 8 y 6 unidades.
- El alumno libera solo el pedido de 8. El sistema verifica stock (8 kits). Quedan 22. No hace falta comprar todavía.
- **Día 2**: se crean otros dos pedidos de 5 y 7. El stock baja y el alumno decide comprar 20 kits al proveedor A (coste 90 €/kit, lead time 3 días).

(Sigue el ciclo…).

---

9. Entregables

1. **Repositorio Git** con:
   - Código fuente con README.
   - Instrucciones de instalación y uso.
2. **Informe PDF** (3 – 5 páginas):
   - Diseño de datos y decisiones tomadas.
   - Capturas de la interfaz.
   - Análisis de un escenario de prueba.
3. **Presentación corta** (máx. 10 diapositivas) para exponer el proyecto.

---

10. Recomendación de pasos

1. Crear el modelo de datos en Python.
2. Montar la simulación base con SimPy.
3. Conectar Streamlit mostrando un tablero mínimo.
4. Añadir cálculos de BOM y lógica de compras.
5. Mejorar la interfaz y las gráficas.
6. Testear con escenarios de ejemplo y documentar.

11. Consideraciones finales

Identificad hitos/checkpoints en el proyecto. Marcadlos en Github Issues
Haced un commit cada vez que llegueis a un hito con el código estable, poniendo el #12 numero de issue en el titulo del commit.

I**mportante :** Usad Vibe Coding sabiamente , generad documentos técnicos (tipo - cursor rules) pensando en que los agentes de programación van a leerlos en futuras interacciones (Este documento puede ser uno de ellos)

Inspiración mutua: podeis compartir ficheros de inicialización, importación y exportación de json y partes de código con otros equipos (max 20% codigo).

Como siempre invitad a granludo@gmail.com al repo. Pero desde el primer momento, no al final.