ChatGPT

# Simulador de Impresora ESC/POS

Este proyecto es un simulador de impresora ESC/POS con interfaz gráfica basada en PyQt5, que permite recibir comandos ESC/POS a través de un socket TCP (por defecto en el puerto 9100) y renderizar en pantalla los tickets impresos. Además, ofrece funcionalidades para guardar los tickets combinados como imágenes PNG o archivos PDF.

## Características principales

- **Servidor TCP**: Escucha en un puerto configurado (por defecto `0.0.0.0:9100`) para recibir datos de impresión.
- **Parsing de comandos ESC/POS**: Interpreta comandos básicos de impresión, cortes de papel, feeds, estilos (negrita, subrayado, alineación, tamaño de texto), códigos QR, códigos de barras e imágenes en modo `GS v 0`.
- **Renderizado en imágenes**: Genera dinámicamente imágenes en blanco y negro (modo "L") de los tickets, apilando todos los tickets recibidos hasta el momento.
- **Interfaz gráfica (PyQt5)**:
- Panel de log que muestra en tiempo real los bytes recibidos y los comandos ESC/POS interpretados.
- Vista de los tickets generados (imagen combinada).
- Controles para cambiar la IP/puerto de escucha y el ancho del ticket (en píxeles).
- Botones para guardar la imagen apilada de todos los tickets como PNG o PDF.
- Botón para "Reset" que limpia logs, buffer y tickets acumulados.

## Requisitos y dependencias

Antes de ejecutar o compilar el proyecto, asegúrate de tener instalado lo siguiente:

1. **Python 3.6+** (preferible 3.7 o superior).
2. **Pip** (gestor de paquetes de Python).
3. **Paquetes Python necesarios**:

```
pip install PyQt5 pillow qrcode python-barcode
```

- `PyQt5`: Para la interfaz gráfica.
- `Pillow`: Para crear y manipular imágenes (tickets, códigos de barras, QR).
- `qrcode`: Para generar imágenes QR.
- `python-barcode`: Para generar códigos de barras (`code128`).

Opcionalmente, si usas un entorno virtual (recomendado), puedes crear uno con:

```
python -m venv venv
source venv/bin/activate  # En macOS/Linux
venv\Scripts\activate     # En Windows
```

## Estructura del repositorio

```
simulador_impresora/
├── simulador_impresora.py    # Código fuente principal
├── README.md                  # Este archivo de documentación
├── requirements.txt           # Lista de dependencias (opcional)
└── assets/                    # Carpeta para guardar fuentes o assets
adicionales (si aplican)
```

Si deseas, puedes generar un `requirements.txt` con:

```
pip freeze > requirements.txt
```

## Ejecución en desarrollo

Para probar el simulador en tu máquina (sin compilar a ejecutable), sigue estos pasos:

1. Clona o copia este repositorio en tu sistema.
2. Instala las dependencias indicadas en la sección anterior.
3. Ejecuta el script principal:

```
python simulador_impresora.py
```

Si todo funciona correctamente, se abrirá la ventana del simulador. En la parte superior podrás ver campos para IP y puerto de escucha (por defecto `0.0.0.0` y `9100` respectivamente). Igualmente, podrás modificar el ancho del ticket (valor por defecto `400` píxeles) y hacer clic en "Aplicar" para reiniciar el servidor con los nuevos valores.

### Probar con un cliente TCP

Para enviar bytes de prueba al simulador, puedes usar herramientas como `netcat` o un script Python sencillo. Por ejemplo, en otro terminal:

```
# En Linux/macOS, enviar un comando ESC @ (reset) al puerto 9100
printf '\x1B\x40' | nc 127.0.0.1 9100
```

Cuando el simulador reciba datos, los mostrará en el panel de log y representará visualmente el ticket (aunque sea texto mínimo o solo comandos).

## Compilar a ejecutable en Windows

Para generar un único archivo `.exe` que funcione en Windows, recomendamos usar **PyInstaller**:

1. Instala PyInstaller si no lo tienes:

```
pip install pyinstaller
```

1. Sitúate en la carpeta donde se encuentra `simulador_impresora.py`.

2. Ejecuta PyInstaller con las siguientes opciones:

```
pyinstaller --onefile --windowed simulador_impresora.py
```

- `--onefile`: Agrupa todo en un único ejecutable.
- `--windowed`: Evita que se abra una consola de comandos adicional al iniciar la aplicación GUI.

- Tras el proceso, PyInstaller generará varias carpetas (`build/`, `dist/`, `__pycache__/`, etc.). El ejecutable final estará en:

```
dist\simulador_impresora.exe
```

1. Para distribuirlo, basta con copiar `simulador_impresora.exe` y, opcionalmente, incluir una carpeta `assets/` si tu aplicación requiere fuentes externas como `DejaVuSans.ttf`. Si todas las dependencias están embebidas correctamente, el `.exe` funcionará aislado.

### Personalización avanzada (icono y metadatos)

Si deseas agregar un ícono personalizado al `.exe`, prepara un archivo `icon.ico` y ejecuta:

```
pyinstaller --onefile --windowed --icon=icon.ico simulador_impresora.py
```

También puedes modificar el `spec` para incluir recursos adicionales.

## Generar ejecutable para macOS

En macOS, es posible usar PyInstaller o **py2app** para crear un paquete que funcione como aplicación `.app`. A continuación se describen ambas opciones:

### Opción 1: PyInstaller en macOS

> **Nota**: Debes ejecutar estos pasos en una Mac con macOS. No es posible generar un `.app` de macOS desde Windows.

1. Instala PyInstaller:

```
pip install pyinstaller
```

1. Desde la terminal, navega hasta la carpeta que contiene `simulador_impresora.py` y ejecuta:

```
pyinstaller --onefile --windowed simulador_impresora.py
```

1. Al finalizar, encontrarás un archivo ejecutable en `dist/simulador_impresora`. Para convertirlo en un paquete `.app`, podrías usar la opción:

```
pyinstaller --onefile --windowed --name "Simulador ESCPOS" --icon=icon.icns
simulador_impresora.py
```

- `--name`: Especifica el nombre de la aplicación.
- `--icon`: Archivo de ícono en formato `.icns` para macOS.

El resultado será un binario en `dist/` que puede ejecutarse directamente en macOS. Si deseas empaquetarlo como `.app`, PyInstaller internamente crea una estructura de aplicación, pero el ejecutable principal estará en `dist/Simulador ESCPOS.app/Contents/MacOS/Simulador ESCPOS`.

### Opción 2: py2app

1. Instala `py2app` en tu entorno virtual:

```
pip install py2app
```

1. Crea un archivo `setup.py` en la raíz del proyecto con este contenido:

```
# setup.py
from setuptools import setup

APP = ['simulador_impresora.py']
DATA_FILES = []  # Si necesitas incluir fuentes o assets, agrégalas aquí
OPTIONS = {
    'argv_emulation': True,
    'iconfile': 'icon.icns',  # Archivo de ícono opcional
    'packages': ['PyQt5', 'PIL', 'qrcode', 'barcode'],
    'plist': {'CFBundleName': 'Simulador ESCPOS',
'CFBundleShortVersionString': '1.0.0'},
}

setup(
    app=APP,
    name='Simulador ESCPOS',
    data_files=DATA_FILES,
    options={'py2app': OPTIONS},
    setup_requires=['py2app'],
)
```

1. Ejecuta:

```
python setup.py py2app
```

1. Después de unos minutos, tendrás una carpeta `dist/Simulador ESCPOS.app`. Esa carpeta es la aplicación macOS que puedes distribuir.

## Uso de la aplicación

1. Inicia el simulador:
2. En Windows: Ejecuta `simulador_impresora.exe`.
3. En macOS: Haz doble clic en `Simulador ESCPOS.app`.
4. En desarrollo (sin compilar): `python simulador_impresora.py`.
5. Guarda tu ticket desde cualquier cliente ESC/POS enviando datos al puerto configurado (por defecto 9100).
6. Observa los logs en el panel izquierdo y la imagen de los tickets en el panel derecho.
7. Para exportar:
8. Haz clic en "Guardar PNG" para guardar la imagen combinada en un archivo `.png`.
9. Haz clic en "Guardar PDF" para crear un archivo `.pdf` con los tickets.
10. Haz clic en "Reset" para limpiar todo y comenzar de nuevo.

## Ajustes y personalización

- **Ancho del ticket** (`paper_width`): Puedes cambiarlo directamente en la interfaz. Afecta el ancho (en píxeles) de las imágenes generadas.
- **Logs detallados**: El simulador muestra en hex y texto los bytes recibidos, así como los comandos ESC/POS interpretados.
- **Fuentes**: Por defecto se usa `DejaVuSans.ttf` (font de sistema). Si no lo encuentra, se usa la fuente por defecto de Pillow.
- **Timeouts y buffers**: Puedes ajustar en el código la forma en la que se procesa el buffer de datos si necesitas mayor rendimiento o compatibilidad con impresoras específicas.

## Solución de problemas comunes

- **Error de importación de PyQt5**: Asegúrate de tener instalada la versión de PyQt5 compatible con tu versión de Python. En Windows, a veces conviene usar:

```
pip install PyQt5==5.15.4
```

- **Fuentes faltantes**: Si no encuentras `DejaVuSans.ttf`, instala la fuente en tu sistema o coloca el archivo `.ttf` en la carpeta `assets/` y modifica la ruta en el código.

- **Problemas con PyInstaller**: Si al compilar faltan módulos (por ejemplo `qrcode` o `barcode`), asegúrate de que estén bien importados en el script. Puedes probar usando la opción `--hidden-import`:

```
pyinstaller --onefile --windowed --hidden-import=qrcode --hidden-
import=barcode.simplesuper --icon=icon.ico simulador_impresora.py
```

- **Permisos**: En macOS, si la aplicación no se abre por restricciones de seguridad, ve a "Preferencias del Sistema > Seguridad y Privacidad" y acepta la ejecución de la app.

## Conclusión

Este simulador permite validar y depurar flujos de impresión ESC/POS sin necesidad de una impresora física. Es útil para desarrolladores que integran sistemas de punto de venta y necesitan verificar cómo se interpretan los comandos de impresión, códigos de barras y QR. También facilita la generación rápida de tickets en formato digital para pruebas o demostraciones.

---

**Autor:** Miguel Britos (basado en requerimientos del proyecto)

**Fecha de creación:** Junio 2025