Checklist de Despliegue Físico - LabPiPanel

Table of Contents

Proyecto: Sistema de Control de Laboratorio LabPiPanel
Fecha: _____
Responsable: _____
Versión: 1.0

Instrucciones de Uso

Este checklist debe completarse **secuencialmente**, verificando cada ítem antes de continuar. **NO** omitir pasos, ya que existen dependencias críticas entre fases. Marcar cada casilla solo cuando la verificación sea exitosa.

Convenciones:

- ☐ Tarea pendiente
- ✓ Tarea completada y verificada
- ⚠ Punto crítico de seguridad
- Requiere documentación

FASE 1: PREPARACIÓN DEL RASPBERRY PI

1.1 Sistema Operativo Base

☐ Raspbian OS instalado (Bullseye o superior)
☐ Sistema actualizado: `sudo apt update &amp;&amp; sudo apt upgrade`
☐ SSH habilitado y funcionando
☐ Contraseña predeterminada cambiada
☐ Hostname configurado: `sudo raspi-config` → Network → Hostname
☐ IP estática configurada (opcional pero recomendado)

Verificación:

```
uname -a  # Verificar versión del kernel
hostname -I  # Verificar IP asignada
```

 Documentar: Hostname: _____ | IP: _____

1.2 Dependencias del Sistema

☐ Git instalado: `sudo apt-get install git`
☐ Build tools instalados: `sudo apt-get install build-essential`
☐ Python 3.9+ verificado: `python3 --version`
☐ pip actualizado: `python3 -m pip install --upgrade pip`

Verificación:

```
python3 --version  # Debe mostrar 3.9 o superior
```

FASE 2: INSTALACIÓN DE DRIVERS DAQ USB-5203

2.1 Dependencias USB y HID

☐ libusb instalado: `sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev`
☐ libudev instalado: `sudo apt-get install libudev-dev`
☐ Herramientas autotools: `sudo apt-get install libfox-1.6-dev autotools-dev autoconf automake libtool`

2.2 Instalación de HIDAPI

☐ Repositorio clonado: `git clone git://github.com/signal11/hidapi.git`
☐ Bootstrap ejecutado: `cd hidapi &amp;&amp; ./bootstrap`
☐ Configurado: `./configure`
☐ Compilado: `make`
☐ Instalado: `sudo make install`

Verificación:

```
ldconfig -p | grep hidapi  # Debe mostrar libhidapi
```

2.3 Instalación de Drivers MCC Linux

☐ Repositorio clonado: `git clone https://github.com/wjasper/Linux_Drivers.git`
☐ Directorio correcto: `cd ~/Linux_Drivers/USB/mcc-libusb`
☐ Compilado: `make`
☐ Instalado: `sudo make install`
☐ Librerías cargadas: `sudo ldconfig`

Verificación:

```
ls ~/Linux_Drivers/USB/mcc-libusb/test-usb5203  # Debe existir
file ~/Linux_Drivers/USB/mcc-libusb/test-usb5203  # Debe ser ejecutable
```

2.4 Configuración de Reglas udev

☐ Reglas copiadas: `sudo cp ~/Linux_Drivers/USB/61-mcc.rules /etc/udev/rules.d/99-mcc.rules`
☐ Permisos correctos: `sudo chmod 644 /etc/udev/rules.d/99-mcc.rules`
☐ Reglas recargadas: `sudo udevadm control --reload-rules`
☐ Sistema reiniciado: `sudo reboot`

2.5 Conexión y Verificación del DAQ

☐ DAQ USB-5203 conectado al puerto USB
☐ LED de alimentación encendido en el DAQ
☐ Dispositivo detectado: `lsusb | grep "09db"`

Verificación esperada:

```
Bus 001 Device 004: ID 09db:0086 Measurement Computing Corp.
```

☐ Programa de prueba ejecuta: `~/Linux_Drivers/USB/mcc-libusb/test-usb5203`
☐ Menú interactivo se muestra correctamente

⚠ **CRÍTICO:** Si `lsusb` no muestra el dispositivo 09db, detener y revisar conexiones USB y drivers.

 Documentar ruta del driver: _____

FASE 3: CONFIGURACIÓN DE LA FUENTE XLN30052

3.1 Conexión de Red

☐ Fuente conectada a la red local (cable Ethernet)
☐ Fuente encendida y lista
☐ Menú frontal accesible

3.2 Configuración desde Panel Frontal

☐ Presionar botón **[Menu]**
☐ Seleccionar **"1. SYSTEM SETTING"**
☐ **REMOTE CONTROL** → **ETHERNET**
☐ **IP CONFIG** → **STATIC**
☐ **IP ADDRESS** → Introducir IP estática
☐ **GATEWAY** → Configurar si es necesario
☐ **NETMASK** → Configurar (típicamente 255.255.255.0)

 Documentar IP de la fuente: _____

3.3 Verificación de Conectividad

☐ Ping exitoso: `ping &lt;IP_FUENTE&gt;`
☐ Conexión Telnet puerto 5024: `telnet &lt;IP_FUENTE&gt; 5024`
☐ Comando `*IDN?` enviado por Telnet
☐ Respuesta recibida: `BK PRECISION,XLN30052,&lt;SN&gt;,&lt;FW&gt;`

Verificación de comandos básicos:

```
telnet 192.168.1.100 5024
*IDN?           # Debe responder con identificación
VSET?           # Debe responder con voltaje configurado
ISET?           # Debe responder con corriente configurada
```

3.4 Configuración de Protecciones (CRÍTICO)

⚠ **ANTES DEL PRIMER USO, configurar desde panel frontal:**

☐ Presionar **[Menu]** → **"3. PROTECTION"**
☐ **OVP** (Sobrevoltaje) → **ON**
☐ **OVP SET** → **310V** (ligeramente por encima del máximo esperado)
☐ **OCP** (Sobrecorriente) → **ON**
☐ **OCP SET** → **5.5A**
☐ **OPP** (Sobrepotencia) → **ON**
☐ **OPP SET** → **1600W**

⚠ **Estas protecciones de hardware son la última línea de defensa ante fallos del software.**

FASE 4: CONFIGURACIÓN DEL MÓDULO DE RELÉS

4.1 Identificación de Hardware

☐ Modelo del módulo de relés identificado: _____
☐ Datasheet descargado y revisado
☐ Especificaciones de carga confirmadas: _____A @ _____V
☐ Lógica de activación confirmada: ☐ Activo BAJO ☐ Activo ALTO

4.2 Conexiones Físicas

☐ Alimentación 5V conectada (pin 2 o 4 del GPIO)
☐ GND conectado (pin 6, 9, 14, 20, 25, 30, 34, o 39)
☐ GPIO 26 → IN1 (Relay 1)
☐ GPIO 20 → IN2 (Relay 2)
☐ GPIO 21 → IN3 (Relay 3)
☐ GPIO 16 → IN4 (Relay 4)
☐ Conexiones verificadas con multímetro

⚠ **Verificar voltaje entre VCC y GND = 5V ±5%**

4.3 Prueba Individual de Relés

☐ Script de prueba creado (`test_relay.py`)
☐ Relé 1 activa/desactiva correctamente
☐ Relé 2 activa/desactiva correctamente
☐ Relé 3 activa/desactiva correctamente
☐ Relé 4 activa/desactiva correctamente
☐ LEDs indicadores funcionan
☐ Sonido mecánico del relé audible (click)

Script de prueba básico:

```
import RPi.GPIO as GPIO
import time

RELAY_PINS = [26, 20, 21, 16]
GPIO.setmode(GPIO.BCM)

for pin in RELAY_PINS:
    GPIO.setup(pin, GPIO.OUT)
    GPIO.output(pin, GPIO.HIGH)

for i, pin in enumerate(RELAY_PINS, 1):
    print(f"Activando Relé {i}")
    GPIO.output(pin, GPIO.LOW)
    time.sleep(2)
```

```
    GPIO.output(pin, GPIO.HIGH)
    time.sleep(1)

GPIO.cleanup()
```

⚠ **PROBAR PRIMERO CON LEDs antes de conectar cargas reales.**

FASE 5: CONFIGURACIÓN DEL SOFTWARE

5.1 Entorno Virtual Python

☐ Directorio del proyecto creado: `mkdir -p ~/LabPiPanel`
☐ Navegado al directorio: `cd ~/LabPiPanel`
☐ Entorno virtual creado: `python3 -m venv venv`
☐ Entorno activado: `source venv/bin/activate`
☐ Prompt muestra `(venv)`

5.2 Instalación de Dependencias

☐ Archivo `requirements.txt` creado con:

```
Flask==2.3.0
RPi.GPIO==0.7.1
pexpect==4.8.0
requests==2.31.0
python-dotenv==1.0.0
gunicorn==21.2.0
```

☐ Dependencias instaladas: `pip install -r requirements.txt`

Verificación:

```
pip list | grep Flask
pip list | grep RPi.GPIO
pip list | grep pexpect
```

5.3 Estructura de Directorios

☐ Directorio `templates/` creado
☐ `index.html` movido a `templates/`
☐ Directorio `static/` creado (CSS, JS, imágenes)
☐ Directorio `logs/` creado: `mkdir -p logs`
☐ Directorio `config/` creado
☐ Permisos correctos: `chmod 755 logs/`

Estructura esperada:

```
~/LabPiPanel/
├── venv/
├── templates/
│   └── index.html
├── static/
├── logs/
├── config/
├── labpipanel.py
├── fuente_xln.py
├── daq_usb5203.py
├── relay_controller.py
├── config.py
```

```
├── .env
└── requirements.txt
```

5.4 Archivos de Configuración

☐ Archivo `config.py` creado con clases de configuración
☐ Archivo `.env` creado con variables:

```
FLASK_ENV=production
SECRET_KEY=&lt;generar clave segura&gt;
XLN_HOST=&lt;IP de la fuente&gt;
XLN_TELNET_PORT=5024
DAQ_DRIVER_PATH=/home/pi/Linux_Drivers/USB/mcc-libusb/test-usb5203
LOG_LEVEL=INFO
LOG_FILE=/var/log/labpipanel/app.log
```

☐ `.env` agregado a `.gitignore`
☐ SECRET_KEY generada: `python3 -c "import secrets; print(secrets.token_hex(32))"`

 Documentar SECRET_KEY en lugar seguro (NO en Git)

5.5 Actualización del Código

☐ `labpipanel.py` actualizado con:

- ☐ Importación de `config.py`
- ☐ Carga de variables desde `.env`
- ☐ Integración de `fuente_xln.py`
- ☐ Integración de `daq_usb5203.py`
- ☐ Integración de control de relés
- ☐ Manejo de excepciones en todos los endpoints
- ☐ Logging estructurado

☐ `fuente_xln.py` actualizado:

- ☐ Puerto Telnet cambiado a 5024
- ☐ Terminación de comandos con `\r\n`
- ☐ Método `verificar_estado()` implementado
- ☐ Método `set_voltage_safe()` con validación

☐ `daq_usb5203.py` actualizado:

- ☐ Timeout aumentado a 10 segundos
- ☐ Validación de canal (0-7)
- ☐ Validación de tipo de termopar
- ☐ Manejo de `pexpect.TIMEOUT`

FASE 6: PRUEBAS UNITARIAS

6.1 Prueba de DAQ USB-5203

☐ Script de prueba creado: `test_daq.py`
☐ Lectura de canal 0 exitosa
☐ Lectura de todos los canales (0-7) exitosa
☐ Temperatura dentro de rango esperado
☐ Sin errores de timeout

Comando de prueba:

```
python3 -c "from daq_usb5203 import leer_temperatura; print(leer_temperatura(0, 'K'))"
```

 Documentar temperaturas leídas:

- CH0: _____ °C
- CH1: _____ °C
- CH2: _____ °C
- CH3: _____ °C

6.2 Prueba de Fuente XLN30052

☐ Script de prueba creado: `test_xln.py`
☐ Identificación correcta con `*IDN?`
☐ Configuración de voltaje seguro (5V) exitosa
☐ Lectura de voltaje configurado coincide
☐ Configuración de corriente (0.5A) exitosa
☐ Activación de salida funciona
☐ Desactivación de salida funciona

⚠ IMPORTANTE: Probar sin carga conectada primero

Comandos de prueba:

```
from fuente_xln import FuenteXLN
fuente = FuenteXLN(host="&amp;lt;IP&amp;gt;")
print(fuente.enviar_comando("*IDN?"))
fuente.enviar_comando("VSET 5")
print(fuente.enviar_comando("VSET?"))
fuente.enviar_comando("OUT 0")  # Asegurar salida apagada
```

6.3 Prueba de Endpoints Flask

☐ Flask iniciado en modo desarrollo: `python3 labpipanel.py`
☐ Servidor escuchando en puerto 5000
☐ Navegador accede a `http://&lt;IP_RASPBERRY&gt;:5000/`
☐ Página principal carga correctamente

Pruebas con curl:

☐ GET `/`: `curl http://localhost:5000/`
☐ POST `/set_voltage`: `curl -X POST -H "Content-Type: application/json" -d '{"voltage":5}' http://localhost:5000/set_voltage`
☐ GET `/get_voltage_current`: `curl http://localhost:5000/get_voltage_current`
☐ GET `/read_daq`: `curl http://localhost:5000/read_daq`
☐ POST `/Relay`: `curl -X POST -d "Relay1=0" http://localhost:5000/Relay`

 Documentar respuestas exitosas: ✓

6.4 Prueba de Integración Completa

☐ Experimento simulado:

1. ☐ Configurar voltaje a 5V
2. ☐ Leer voltaje real
3. ☐ Activar relé 1
4. ☐ Leer temperatura canal 0
5. ☐ Desactivar relé 1
6. ☐ Configurar voltaje a 0V

☐ Secuencia exitosa sin errores
☐ Logs generados correctamente en `logs/`

FASE 7: HARDENING Y SEGURIDAD

7.1 Validación de Entradas

☐ Validación de rangos implementada:

- ☐ Voltaje: 0-300V
- ☐ Corriente: 0-5.2A
- ☐ Canal DAQ: 0-7

☐ Mensajes de error informativos implementados
☐ Códigos HTTP correctos (400, 500) retornados

7.2 Límites de Seguridad

☐ Soft limit implementado para voltaje >50V
☐ Confirmación de usuario requerida (modal en frontend)
☐ Hard limit en código: voltaje máx = 300V
☐ Hard limit en código: corriente máx = 5.2A

⚠ **Probar que voltaje >300V es rechazado**

7.3 Sistema de Logging

☐ Logging configurado en `labpipanel.py`
☐ Nivel de log: INFO en producción
☐ Rotación de archivos implementada (RotatingFileHandler)
☐ Tamaño máximo: 10 MB
☐ Backup count: 5 archivos
☐ Formato incluye timestamp, nivel, módulo, mensaje

Verificación:

```
tail -f logs/app.log  # Debe mostrar logs en tiempo real
```

7.4 Modo Debug Deshabilitado

☐ `debug=False` en `labpipanel.py`
☐ Variable `FLASK_ENV=production` en `.env`
☐ Sin stack traces expuestos al usuario

FASE 8: DEPLOYMENT EN PRODUCCIÓN

8.1 Creación de Servicio systemd

☐ Archivo `labpipanel.service` creado
☐ Usuario y grupo configurados: `User=pi`, `Group=pi`
☐ WorkingDirectory correcto: `/home/pi/LabPiPanel`
☐ ExecStart con Gunicorn configurado

☐ Servicio copiado: `sudo cp labpipanel.service /etc/systemd/system/`
☐ Permisos correctos: `sudo chmod 644 /etc/systemd/system/labpipanel.service`

8.2 Configuración de Logs del Sistema

☐ Directorio de logs creado: `sudo mkdir -p /var/log/labpipanel`
☐ Permisos asignados: `sudo chown pi:pi /var/log/labpipanel`
☐ Verificación: `ls -ld /var/log/labpipanel`

8.3 Instalación de Gunicorn

☐ Gunicorn instalado en venv: `pip install gunicorn`
☐ Configuración verificada en `labpipanel.service`:

- ☐ Workers: 2-4 (según CPU)
- ☐ Bind: 127.0.0.1:5000
- ☐ Timeout: 120 segundos

8.4 Habilitación del Servicio

☐ Daemon recargado: `sudo systemctl daemon-reload`
☐ Servicio habilitado: `sudo systemctl enable labpipanel.service`
☐ Servicio iniciado: `sudo systemctl start labpipanel.service`
☐ Estado verificado: `sudo systemctl status labpipanel.service`

Verificación esperada:

```
● labpipanel.service - LabPiPanel - Sistema de Control de Laboratorio
   Loaded: loaded (/etc/systemd/system/labpipanel.service; enabled)
   Active: active (running)
```

☐ Logs del servicio: `sudo journalctl -u labpipanel.service -f`

8.5 Prueba de Reinicio Automático

☐ Sistema reiniciado: `sudo reboot`
☐ Servicio inicia automáticamente después del boot
☐ Aplicación accesible en `http://<IP>:5000/`

FASE 9: PRUEBAS DE ESTRÉS Y CONFIABILIDAD

9.1 Pruebas de Estrés

☐ Script de estrés creado con 100+ comandos
☐ Ejecución exitosa sin fallos
☐ Uso de memoria estable (monitoreado con `htop`)
☐ Temperatura del CPU <70°C
☐ Sin degradación de rendimiento

Comando de monitoreo:

```
watch -n 1 'vcgencmd measure_temp'  # Monitorear temperatura CPU
```

9.2 Pruebas de Fallos

☐ Desconexión de DAQ durante operación:

- ☐ Error manejado correctamente
- ☐ Mensaje de error informativo
- ☐ Aplicación no crashea

☐ Desconexión de fuente XLN durante operación:

- ☐ Timeout manejado
- ☐ Error logged
- ☐ Aplicación no crashea

☐ Simulación de crash del servicio:

- ☐ `sudo systemctl kill -s SIGKILL labpipanel.service`
- ☐ Servicio se reinicia automáticamente
- ☐ Tiempo de recuperación <30 segundos

9.3 Pruebas de Recuperación

☐ Reconexión de DAQ después de desconexión funciona
☐ Reconexión de fuente XLN funciona
☐ No se requiere reinicio manual del servicio

FASE 10: SEGURIDAD ELÉCTRICA

10.1 Inspección Física

⚠ **ANTES DE CONECTAR CARGAS REALES:**

☐ Todas las conexiones eléctricas verificadas
☐ No hay cables pelados o expuestos
☐ Terminales de la fuente apretados correctamente
☐ Relés montados en superficie aislante
☐ Área de trabajo seca y limpia
☐ Extinguidor de incendios accesible

10.2 Botón de Emergencia (RECOMENDADO)

☐ Botón de emergencia físico instalado
☐ Botón corta alimentación principal
☐ Ubicación claramente marcada
☐ Accesible desde posición de operador

10.3 Procedimientos de Seguridad

☐ Manual de operación escrito
☐ Procedimientos de emergencia documentados
☐ Personal capacitado en:

- ☐ Riesgos de alto voltaje (>50V)
- ☐ Primeros auxilios ante shock eléctrico
- ☐ Uso del botón de emergencia

☐ Equipo de protección personal disponible:

- ☐ Guantes dieléctricos (si voltaje >50V)
- ☐ Gafas de seguridad
- ☐ Tapete aislante (si voltaje >50V)

FASE 11: DOCUMENTACIÓN FINAL

11.1 Documentación Técnica

☐ README.md completo con:

- ☐ Descripción del proyecto
- ☐ Requisitos de hardware
- ☐ Instrucciones de instalación
- ☐ Comandos para iniciar/detener servicio

☐ Diagramas de conexión creados
☐ Especificaciones de hardware documentadas
☐ API endpoints documentados

11.2 Documentación de Usuario

☐ Manual de operación escrito
☐ Guía de troubleshooting creada
☐ FAQ documentada con errores comunes
☐ Screenshots de la interfaz incluidos

11.3 Información de Configuración

 Completar esta sección para referencia:

Hardware:

- Modelo Raspberry Pi: _____
- IP del Raspberry Pi: _____
- IP de la fuente XLN: _____
- Modelo del módulo de relés: _____

Software:

- Ruta del driver DAQ: _____
- Puerto de la aplicación: _____
- Ubicación de logs: _____

Configuración de Termopares:

- Canal 0: Tipo _____, ubicación: _____
- Canal 1: Tipo _____, ubicación: _____
- Canal 2: Tipo _____, ubicación: _____
- Canal 3: Tipo _____, ubicación: _____

Configuración de Relés:

- Relé 1: Controla: _____
- Relé 2: Controla: _____
- Relé 3: Controla: _____
- Relé 4: Controla: _____

FASE 12: BACKUP Y RECOVERY

12.1 Script de Backup

☐ Script de backup creado: `backup_labpipanel.sh`
☐ Incluye respaldo de:

- ☐ Archivos de configuración (`.env`, `config.py`)
- ☐ Código fuente
- ☐ Logs importantes

☐ Cron job configurado (opcional):

```
0 2 * * * /home/pi/LabPiPanel/backup_labpipanel.sh
```

12.2 Procedimiento de Recovery

☐ Procedimiento documentado para:

- ☐ Restauración desde backup
- ☐ Reinstalación de drivers
- ☐ Reconfiguración de servicio

VERIFICACIÓN FINAL

Checklist de Operación

Antes de declarar el sistema listo para producción, verificar:

☐ Sistema completo funciona durante 24 horas continuas sin fallos
☐ Todos los componentes responden correctamente
☐ Logs se escriben sin errores
☐ Servicio se reinicia automáticamente tras reboot
☐ Personal operativo capacitado y certificado
☐ Documentación completa y accesible
☐ Procedimientos de emergencia practicados

SIGN-OFF FINAL

Responsable Técnico:

Nombre: _____
Firma: _____
Fecha: _____

Supervisor de Laboratorio:

Nombre: _____
Firma: _____
Fecha: _____

Oficial de Seguridad (si aplica):

Nombre: _____
Firma: _____
Fecha: _____

Notas y Observaciones

Usar este espacio para documentar cualquier desviación del procedimiento estándar, problemas encontrados y sus soluciones, o configuraciones personalizadas:

FIN DEL CHECKLIST

Versión: 1.0
Última actualización: Octubre 30, 2025
Proyecto: LabPiPanel - Sistema de Control de Laboratorio