HOJA DE RUTA COMPLETA - LabPiPanel con Expansión Arduino e OLED

Table of Contents

Proyecto: Sistema Integrado de Control de Laboratorio para Investigación Térmica
Fase Actual: Definición de Despliegue Base + Expansión
Fecha de Creación: 30 de Octubre de 2025
Responsable: _____

DOCUMENTO EJECUTIVO

Este roadmap define **DOS FASES CLARAS**:

1. **FASE BASE (Semanas 1-4)**: Despliegue del sistema LabPiPanel core con hardware existente (Raspberry Pi, Fuente XLN, DAQ USB-5203, Relés)
2. **FASE EXPANDIDA (Semanas 5-8)**: Integración de Arduino, Display OLED y automatización completa

Tiempo Total Estimado: 8 semanas para sistema completamente funcional

PARTE 1: CHECKLIST DE DESPLIEGUE BASE (SEMANAS 1-4)

SEMANA 1: Preparación de Hardware y Software Base

1.1 Preparación Inicial (Días 1-2)

☐ **Hardware Adquirido y Verificado**

- ☐ Raspberry Pi 3/4 confirmado disponible
- ☐ Fuente XLN30052 funcionando
- ☐ DAQ USB-5203 sin defectos físicos
- ☐ Módulo de 4 relés en buen estado
- ☐ Todos los cables necesarios verificados
- ☐ Termopares tipo K verificados (mínimo 8)

☐ **Documentación Impresa**

- ☐ Manual BK Precision XLN30052 en mano
- ☐ Manual USB-5203 User Guide disponible
- ☐ Diagramas de conexión listos

- ☐ Especificaciones de componentes compiladas

Verificación: Fotografiar y documentar estado inicial de hardware

1.2 Sistema Operativo y Dependencias Base (Días 2-3)

☐ **Instalación Raspbian OS**

- ☐ Raspbian Bullseye o superior instalado en tarjeta SD
- ☐ Sistema actualizado: `sudo apt update &amp;&amp; sudo apt upgrade`
- ☐ SSH habilitado y funcionando
- ☐ Contraseña predeterminada cambiada
- ☐ Hostname configurado: `labpipanel`
- ☐ IP estática asignada (documentar: _____)

☐ **Python y Dependencias Base**

- ☐ Python 3.9+ verificado: `python3 --version`
- ☐ pip3 actualizado
- ☐ Git instalado: `sudo apt install git`
- ☐ Build tools instalados: `sudo apt install build-essential`

Comando de Verificación:

```
python3 --version &amp;amp;&amp;amp; pip3 --version &amp;amp;&amp;amp; git --version
```

1.3 Instalación de Drivers DAQ (Días 3-4)

☐ **Dependencias USB y HID**

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

☐ **Instalación de HIDAPI**

- ☐ Repositorio clonado: `git clone git://github.com/signal11/hidapi.git`
- ☐ Bootstrap: `cd hidapi &amp;&amp; ./bootstrap`
- ☐ Configurado: `./configure`
- ☐ Compilado: `make`
- ☐ Instalado: `sudo make install`
- ☐ Verificación: `ldconfig -p | grep hidapi`

☐ **Instalación de Drivers MCC Linux**

- ☐ Repositorio clonado: `git clone https://github.com/wjasper/Linux_Drivers.git`
- ☐ Compilado: `cd Linux_Drivers/USB/mcc-libusb &amp;&amp; make`
- ☐ Instalado: `sudo make install`
- ☐ Librerías cargadas: `sudo ldconfig`
- ☐ Verificación: `ls ~/Linux_Drivers/USB/mcc-libusb/test-usb5203`

☐ **Configuración de Reglas udev**

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

Verificación Post-Reboot:

```
lsusb | grep "09db"  # Debe mostrar DAQ USB-5203
~/Linux_Drivers/USB/mcc-libusb/test-usb5203  # Debe mostrar menú
```

 Documentar: Ruta driver DAQ: _____

SEMANA 2: Configuración de Hardware de Prueba

2.1 Configuración de Fuente XLN30052 (Días 5-6)

☐ **Configuración de Red**

- ☐ Fuente conectada a red local (cable Ethernet)
- ☐ Desde panel frontal: [Menu] → "1. SYSTEM SETTING"
- ☐ REMOTE CONTROL → ETHERNET
- ☐ IP CONFIG → STATIC
- ☐ IP configurada (anotar: _____)
- ☐ Indicador "RMT" visible en pantalla frontal

☐ **Verificación de Conectividad**

- ☐ Ping exitoso: `ping &lt;IP_FUENTE&gt;`
- ☐ Telnet puerto 5024: `telnet &lt;IP&gt; 5024`
- ☐ Comando *IDN? retorna: BK PRECISION,XLN30052,...
- ☐ Comando VSET? retorna valor

⚠ **CRÍTICO**: Configurar protecciones desde panel frontal ANTES de cualquier prueba:

- ☐ [Menu] → "3. PROTECTION"
- ☐ OVP → ON, SET → 310V
- ☐ OCP → ON, SET → 5.5A
- ☐ OPP → ON, SET → 1600W

 Documentar:

- IP de fuente: _____
- Puerto Telnet: _____
- Protecciones verificadas: ✓

2.2 Instalación y Prueba de Termopares (Día 6)

☐ **Conexión Física de Termopares**

- ☐ Termopares tipo K conectados a DAQ USB-5203
- ☐ Canal 0-3: Evaporador (posiciones 1-4)
- ☐ Canal 4-7: Condensador (posiciones 1-4)
- ☐ Voltajes verificados entre CxH y CxL: ±1-10mV típico

☐ **Prueba de Lectura**

- ☐ Comando `./test-usb5203` ejecuta sin error
- ☐ Lectura de canal 0 : Temp = _____ °C
- ☐ Lectura de canal 7 : Temp = _____ °C
- ☐ Sin errores de timeout ni "OTD" (Open Thermocouple Detected)

2.3 Configuración de Relés y Pruebas (Día 7)

☐ **Verificación de Pines GPIO**

- ☐ Relé 1 (GPIO 26): Funcionando
- ☐ Relé 2 (GPIO 20): Funcionando
- ☐ Relé 3 (GPIO 21): Funcionando
- ☐ Relé 4 (GPIO 16): Funcionando
- ☐ LEDs indicadores encendiendo/apagando correctamente

☐ **Asignación Preliminar de Relés**

- ☐ Relé 1: Reservado para Bomba de Fluido (fase expandida)
- ☐ Relé 2: Reservado para Control Fuente (fase expandida)
- ☐ Relé 3: Disponible
- ☐ Relé 4: Disponible

☐ **Test de Seguridad**

- ☐ Voltaje 5V/GND medido entre pins: 5.0V ±5%
- ☐ Capacidad de carga verificada sin calentamiento
- ☐ Ruido mecánico audible (click) en activación

 Documentar:

- GPIO 26 funcional: ✓
- GPIO 20 funcional: ✓
- GPIO 21 funcional: ✓
- GPIO 16 funcional: ✓

SEMANA 3: Desarrollo y Configuración de Software

3.1 Estructura Base del Proyecto (Día 8)

☐ **Creación de Estructura de Directorios**

- ☐ Directorio base: `mkdir ~/LabPiPanel`
- ☐ Subdir templates: `mkdir templates`
- ☐ Subdir static: `mkdir static`
- ☐ Subdir logs: `mkdir logs`
- ☐ Subdir config: `mkdir config`
- ☐ Permisos correctos: `chmod 755 logs/`

☐ **Entorno Virtual Python**

- ☐ Venv creado: `python3 -m venv venv`
- ☐ Activado: `source venv/bin/activate`
- ☐ pip actualizado en venv
- ☐ Prompt muestra `(venv)`

☐ **Instalación de Dependencias Core**

- ☐ Flask instalado: `pip install Flask==2.3.0`
- ☐ RPi.GPIO instalado: `pip install RPi.GPIO==0.7.1`
- ☐ pexpect instalado: `pip install pexpect==4.8.0`
- ☐ requests instalado: `pip install requests==2.31.0`
- ☐ python-dotenv instalado: `pip install python-dotenv==1.0.0`
- ☐ Gunicorn instalado: `pip install gunicorn==21.2.0`

Verificación:

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

3.2 Configuración de Archivos Base (Día 9)

☐ **Archivo** config.py

- ☐ Clases de configuración: DevelopmentConfig, ProductionConfig, TestingConfig
- ☐ Parámetros de fuente XLN: IP, puertos, timeout
- ☐ Ruta de driver DAQ: `/home/pi/Linux_Drivers/USB/mcc-libusb/test-usb5203`
- ☐ Pines GPIO: [26, 20, 21, 16]
- ☐ Límites de seguridad: Voltaje (0-300V), Corriente (0-5.2A)

☐ **Archivo .env**

- ☐ FLASK_ENV=development (cambiar a production después)
- ☐ SECRET_KEY generada: `python3 -c "import secrets; print(secrets.token_hex(32))"`
- ☐ XLN_HOST configurado (documentar: _____)
- ☐ XLN_TELNET_PORT=5024
- ☐ DAQ_DRIVER_PATH verificado
- ☐ LOG_LEVEL=INFO
- ☐ .env agregado a .gitignore

☐ **Archivo requirements.txt**

- ☐ Contenido correcto con todas las dependencias
- ☐ Verificable: `pip freeze > requirements.txt`

3.3 Integración de Módulos de Hardware (Día 10)

☐ labpipanel.py **Actualizado**

- ☐ Importa config.py
- ☐ Carga variables desde .env
- ☐ Integra fuente_xln.py
- ☐ Integra daq_usb5203.py
- ☐ Integra relay_controller.py
- ☐ Endpoints para: relés, fuente, DAQ
- ☐ Manejo de excepciones en todos los endpoints
- ☐ Logging estructurado

☐ **Módulo fuente_xln.py**

- ☐ Puerto Telnet cambiado a 5024 (NO 23)
- ☐ Terminación de comandos: `\r\n`
- ☐ Método verificar_estado() implementado
- ☐ Método set_voltage_safe() con validación
- ☐ Manejo de timeouts robusto

☐ **Módulo daq_usb5203.py**

- ☐ Timeout: 10 segundos (NO 5)
- ☐ Validación de canal: 0-7
- ☐ Validación de tipo de termopar: J, K, R, S, T, N, E, B
- ☐ Manejo de pexpect.TIMEOUT
- ☐ Sanity check de temperatura: -270°C < T < 2000°C

☐ **Módulo relay_controller.py**

- ☐ GPIO.setmode(GPIO.BCM)
- ☐ Pines correctos: 26, 20, 21, 16
- ☐ Estado inicial: HIGH (relés inactivos)
- ☐ Funciones activar/desactivar implementadas

3.4 Archivo index.html Mejorado (Día 10)

☐ **Frontend Funcional Mínimo**

- ☐ Botones para activar/desactivar relés
- ☐ Input para configurar voltaje (0-300V)
- ☐ Input para configurar corriente (0-5.2A)
- ☐ Botón para leer temperatura (todos los canales)
- ☐ Display de valores reales (voltaje, corriente salida)
- ☐ Log de eventos en página

SEMANA 4: Pruebas Unitarias y Deployment Base

4.1 Pruebas Unitarias de Componentes (Días 11-12)

☐ **Test de DAQ USB-5203**

- ☐ Lectura de canal 0 exitosa
- ☐ Lectura de todos los canales (0-7) exitosa
- ☐ Temperatura dentro de rango esperado (20-30°C ambiente)
- ☐ Sin errores de timeout
- ☐ Reproducibilidad: múltiples lecturas consistentes

Script de prueba:

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

☐ **Test de Fuente XLN30052**

- ☐ Identificación: `*IDN?` retorna correctamente
- ☐ Set voltage 5V exitoso, verificación: `VSET?` = 5.0V
- ☐ Set current 0.5A exitoso, verificación: `ISET?` = 0.5A
- ☐ Salida activada: `OUT 1` exitoso
- ☐ Salida desactivada: `OUT 0` exitoso
- ☐ Estado desactivado verificado

⚠ **SIN CARGA CONECTADA EN ESTA ETAPA**

☐ **Test de Relés**

- ☐ Relé 1 activa/desactiva (GPIO 26)
- ☐ Relé 2 activa/desactiva (GPIO 20)
- ☐ Relé 3 activa/desactiva (GPIO 21)
- ☐ Relé 4 activa/desactiva (GPIO 16)
- ☐ LEDs parpadean correctamente
- ☐ Click mecánico audible

4.2 Pruebas de Endpoints Flask (Día 13)

☐ **GET /?**

```
curl http://localhost:5000/
# Retorna: página HTML index.html<a></a>
```

☐ **POST /set_voltage**

```
curl -X POST -H "Content-Type: application/json" \
  -d '{"voltage":5}' http://localhost:5000/set_voltage
# Retorna: {"status": "ok"}<a></a>
```

☐ **GET /get_voltage_current**

```
curl http://localhost:5000/get_voltage_current
# Retorna: {"voltage": 5.0, "current": 0.5}<a></a>
```

☐ **GET /read_daq**

```
curl http://localhost:5000/read_daq
# Retorna: {"temperatures": [23.4, 23.2, 23.1, 23.3, ...]}<a></a>
```

☐ **POST /Relay**

```
curl -X POST -d "Relay1=0" http://localhost:5000/Relay
# Retorna: 204 No Content (éxito)<a></a>
```

4.3 Configuración de Servicio systemd (Día 14)

☐ **Archivo labpipanel.service Creado**

- ☐ Unit, Service, Install sections correctos
- ☐ ExecStart con Gunicorn: 2-4 workers
- ☐ Timeout: 120 segundos
- ☐ Reinicio automático: always
- ☐ Usuario: pi, Grupo: pi

☐ **Instalación de Servicio**

- ☐ Copiado: `sudo cp labpipanel.service /etc/systemd/system/`
- ☐ Permisos: `sudo chmod 644 /etc/systemd/system/labpipanel.service`
- ☐ Recargado: `sudo systemctl daemon-reload`
- ☐ Habilitado: `sudo systemctl enable labpipanel.service`
- ☐ Iniciado: `sudo systemctl start labpipanel.service`

☐ **Verificación de Servicio**

- ☐ Estado: `sudo systemctl status labpipanel.service` → active (running)
- ☐ Accesible: `curl http://<PI_IP>:5000/` retorna HTML
- ☐ Logs: `sudo journalctl -u labpipanel.service -f` muestra actividad

☐ **Prueba de Auto-Inicio**

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

4.4 Documentación de Despliegue Base (Día 14)

☐ **Documentación Completada**

- ☐ README.md actualizado con instrucciones de instalación
- ☐ Diagramas de conexión fotografiados y documentados
- ☐ Especificaciones de hardware compiladas
- ☐ IPs y puertos documentados
- ☐ Procedimiento de emergencia escrito

 DOCUMENTAR CONFIGURACIÓN FINAL BASE:

Parámetro	Valor
IP Raspberry Pi	_____

Parámetro	Valor
IP Fuente XLN	_____
Puerto Fuente	5024
Ruta Driver DAQ	/home/pi/Linux_Drivers/USB/mcc-libusb/test-usb5203
Pines GPIO Relés	26, 20, 21, 16
Puerto Flask	5000
Estado del Sistema	✓ OPERACIONAL

PARTE 2: ROADMAP DE EXPANSIÓN (SEMANAS 5-8)

SEMANA 5: Preparación de Expansión

5.1 Adquisición de Componentes (Día 15)

☐ **Arduino para I2C**

- ☐ Modelo seleccionado: _____
- ☐ ✓ Recomendado: Arduino Nano 33 IoT (3.3V nativo)
- ☐ ✓ Alternativa: Arduino Pro Mini 3.3V
- ☐ Especificación verificada: 3.3V I2C slave compatible
- ☐ Adquirido y probado con blink básico

☐ **Display OLED SSD1306**

- ☐ Modelo: 0.96" 128x64 píxeles
- ☐ Interfaz: I2C (confirmado)
- ☐ Dirección I2C verificada: 0x3C (típica)
- ☐ Alimentación: 3.3V-5V compatible
- ☐ Adquirido y funcionando en test básico

☐ **Componentes de Soporte (Si Aplica)**

- ☐ Si Arduino es 5V: Módulo convertidor de niveles lógicos BSS138 o TXS0108E
- ☐ Si DAQ requiere alimentación externa: Adaptador AC verificado
- ☐ Hub USB alimentado (si se planea agregar más dispositivos)
- ☐ Bomba de fluido: Especificaciones verificadas (12-24V DC, <2A típico)
- ☐ Diodos flyback (1N4007) para relés inductivos

5.2 Validación de Componentes Nuevos (Día 16)

☐ **Test Arduino Independiente**

- ☐ Arduino IDE instalado en laptop/escritorio
- ☐ Blink test funciona en Arduino
- ☐ I2C funcionando: scan I2C muestra dirección 0x08
- ☐ Comunicación serial USB funciona

☐ **Test OLED Independiente**

- ☐ Conectado a laptop por FTDI o Arduino
- ☐ i2cdetect detecta dirección 0x3C
- ☐ Pantalla enciende y muestra contenido
- ☐ Brillo y contraste configurables

☐ **Presupuesto de Corriente USB Verificado**

- ☐ DAQ + Arduino + OLED corriente estimada: ~250mA
- ☐ Margen disponible: ~950mA
- ☐ ✓ DENTRO DE LÍMITES (1.2A total)
- ☐ Decisión: ☐ Hub USB alimentado ☐ Alimentación externa Arduino ☐ Sin cambios

SEMANA 6: Integración de Arduino e OLED

6.1 Integración I2C - Arduino (Día 17)

☐ **Resolución del Problema de Niveles Lógicos**

- ☐ Opción elegida:
  - ☐ Arduino 3.3V nativo (SIN convertidor)
  - ☐ Convertidor de niveles lógicos instalado
  - ☐ Otra: _____

☐ **Cableado I2C Raspberry Pi ↔ Arduino**

- ☐ Pi GPIO2 (SDA) → Arduino A4 (SDA) (con convertidor si aplica)
- ☐ Pi GPIO3 (SCL) → Arduino A5 (SCL) (con convertidor si aplica)
- ☐ Pi GND → Arduino GND (conexión común)
- ☐ Sin 5V del Pi al Arduino (alimentación externa)

☐ **Firmware Arduino I2C Slave Cargado**

- ☐ Código de ejemplo compilado sin errores
- ☐ Dirección I2C: 0x08
- ☐ Funciones implementadas: lectura de ADC, control de pines
- ☐ Cargado en Arduino vía USB

☐ **Verificación de I2C en Raspberry Pi**

- ☐ I2C habilitado: `sudo raspi-config` → Interface Options → I2C
- ☐ Herramientas instaladas: `i2c-tools python3-smbus`
- ☐ Scan I2C: `i2cdetect -y 1` muestra 0x08

6.2 Integración OLED SSD1306 (Día 18)

☐ **Cableado OLED**

- ☐ VCC → Pin 1 (3.3V del Pi) ⚠ 3.3V, NO 5V
- ☐ GND → Pin 6 (GND)
- ☐ SCL → Pin 5 (GPIO3)
- ☐ SDA → Pin 3 (GPIO2)

☐ **Verificación I2C**

- ☐ i2cdetect detecta: 0x3C (OLED) y 0x08 (Arduino)
- ☐ Ambos en mismo bus I2C funcionan correctamente
- ☐ Sin conflictos de dirección

☐ **Librerías OLED Instaladas**

- ☐ PIL instalada: `pip install Pillow`
- ☐ adafruit-circuitpython-ssd1306: `pip install adafruit-circuitpython-ssd1306`
- ☐ board instalada: incluida en adafruit

☐ **Script de Test OLED Funciona**

- ☐ Código de ejemplo ejecuta sin errores
- ☐ Pantalla muestra "Hola Mundo" o similar

- ☐ Actualización de pantalla fluida

6.3 Módulos Python de Expansión (Día 19)

☐ **Archivo arduino_i2c_master.py Creado**

- ☐ Clase ArduinoI2C implementada
- ☐ Métodos: read_sensor(), set_relay()
- ☐ Manejo de excepciones robusto
- ☐ Logging configurado

☐ **Archivo oled_display.py Creado**

- ☐ Clase OLEDDisplay implementada
- ☐ Métodos: show_system_info(), show_experiment_status()
- ☐ Funciones de utilidad: get_ip_address(), get_cpu_temp(), get_cpu_usage()
- ☐ Actualización automática de pantalla

☐ **Archivo thermal_experiment.py Mejorado**

- ☐ Clase ThermalExperiment integrada con expansión
- ☐ Métodos de cálculo térmico verificados
- ☐ Integración con Arduino I2C para lectores auxiliares (si aplica)
- ☐ Exportación a CSV funcionando

SEMANA 7: Control de Bomba y Auto-Inicio

7.1 Configuración de Relé para Bomba (Día 20)

☐ **Especificaciones de Bomba Verificadas**

- ☐ Voltaje de operación: _____V
- ☐ Corriente típica: _____A
- ☐ Tipo: ☐ DC ☐ AC
- ☐ Carga compatible con relé ✓

☐ **Cableado del Relé 1 (GPIO 26) para Bomba**

- ☐ Relé (IN) → GPIO 26
- ☐ Relé (GND) → GND del Pi
- ☐ Relé (VCC) → 5V del Pi
- ☐ Contactos del relé → Circuito de bomba
- ☐ Diodo flyback instalado (si bomba es inductiva): 1N4007 en paralelo
- ☐ Sin conexión directa de 300V del DAQ

☐ **Test de Bomba**

- ☐ Comando Python activa bomba: GPIO.output(26, GPIO.LOW)
- ☐ Bomba enciende correctamente
- ☐ Comando desactiva: GPIO.output(26, GPIO.HIGH)
- ☐ Bomba apaga correctamente
- ☐ Ruido mecánico audible del relé

7.2 Servicios systemd para Auto-Inicio (Día 21)

☐ **Servicio OLED Display Creado**

- ☐ Archivo: `/etc/systemd/system/oled-display.service`
- ☐ Inicia automáticamente después de network.target
- ☐ Muestra IP del Pi en pantalla OLED
- ☐ Reinicia automáticamente si falla

☐ **Actualización de labpipanel.service**

- ☐ Dependency agregado: `After=oled-display.service`
- ☐ Order garantizado: Network → OLED → LabPiPanel
- ☐ Gunicorn configurado con workers correctos

☐ **Instalación de Servicios**

- ☐ OLED service copiado: `sudo cp oled-display.service /etc/systemd/system/`
- ☐ Permisos: `sudo chmod 644 /etc/systemd/system/oled-display.service`
- ☐ Daemon recargado: `sudo systemctl daemon-reload`
- ☐ Servicios habilitados: `sudo systemctl enable oled-display.service labpipanel.service`

☐ **Prueba de Auto-Inicio**

- ☐ Sistema reiniciado: `sudo reboot`
- ☐ OLED muestra IP dentro de 15 segundos del boot
- ☐ LabPiPanel Flask accesible en `http://<IP>:5000/` dentro de 30 segundos
- ☐ Ambos servicios muestran: `active (running)` en systemctl status

7.3 Endpoints Flask Actualizados (Día 22)

☐ **Nuevo Endpoint: POST /pump/control**

- ☐ Acepta: `{"state": "on"}` o `{"state": "off"}`
- ☐ Controla: GPIO 26 (Relé 1)
- ☐ Responde: `{"pump_state": "on/off"}`
- ☐ Logging de evento

☐ **Nuevo Endpoint: GET /arduino/sensor**

- ☐ Lee sensor del Arduino vía I2C
- ☐ Responde: `{"sensor_value": 2.34}`
- ☐ Manejo de errores si Arduino no responde

☐ **Nuevo Endpoint: POST /oled/update**

- ☐ Actualiza contenido del display OLED
- ☐ Acepta: `{"message": "Experimento iniciado"}`
- ☐ Muestra en pantalla OLED

☐ **Modificación: GET /system/status**

- ☐ Retorna: IP, CPU temp, CPU usage, estado de servicios
- ☐ Datos mostrados en OLED

SEMANA 8: Pruebas Completas e Integración Final

8.1 Pruebas de Integración Sistema Completo (Día 23)

☐ **Secuencia de Prueba Integrada**

1. ☐ Reboot del Pi
2. ☐ OLED muestra IP dentro de 15s
3. ☐ LabPiPanel Flask accesible
4. ☐ DAQ lee temperatura correctamente
5. ☐ Arduino I2C responde a consultas
6. ☐ Relé 1 (bomba) activable vía web
7. ☐ Relé 2-4 funcionan como backup
8. ☐ Fuente XLN responde a comandos
9. ☐ OLED actualiza con información del sistema
10. ☐ Logs se escriben correctamente

8.2 Ejecución de Experimento Térmico Piloto (Día 24)

☐ **Experimento Base (Sin Nanofluidos)**

- ☐ Heat pipe preparado con agua destilada
- ☐ Termopares instalados correctamente
- ☐ Bomba activa (circulación opcional)
- ☐ Secuencia de potencia: 1W → 2W → 3W (10 min cada una)
- ☐ Datos guardados en CSV: `thermal_experiment_water_20251030_*.csv`
- ☐ Resistencia térmica calculada correctamente

☐ **Validación de Datos**

- ☐ Temperaturas en rango realista: 20-50°C
- ☐ ΔT (evaporador - condensador) > 0°C
- ☐ R_thermal = ΔT/P en rango esperado: 10-50 °C/W
- ☐ CSV exportable a Excel/Python para análisis

8.3 Documentación de Expansión (Día 25)

☐ **Documentación Técnica Completada**

- ☐ API Documentation con Swagger/OpenAPI (opcional pero recomendado)
- ☐ Diagrama de arquitectura actualizado (incluir Arduino + OLED)
- ☐ Diagrama de flujo de experimento térmico
- ☐ Datasheet de componentes nuevos compilado

☐ **Manual de Usuario Expandido**

- ☐ Instrucciones de control de bomba
- ☐ Interpretación de display OLED
- ☐ Guía de uso para experimento térmico completo
- ☐ Procedimiento de calibración

☐ **Troubleshooting Expandido**

- ☐ Problemas de I2C y soluciones
- ☐ Problemas de visualización OLED
- ☐ Problemas de comunicación Arduino
- ☐ Problemas de bomba no funciona

PARTE 3: GUÍA PARA PRÓXIMAS EXPANSIONES

Fase 3 (Futuro): Capacidades Avanzadas

Opciones de Expansión Posterior a Semana 8

☐ **Base de Datos para Históricos**

- Tecnología: SQLite (simple) o PostgreSQL (profesional)
- Función: Almacenar resultados de experimentos, histórico de mediciones
- Estimado: 2-3 semanas

☐ **WebSockets para Tiempo Real**

- Tecnología: Flask-SocketIO
- Función: Actualización en vivo de gráficos de temperatura
- Estimado: 1-2 semanas

☐ **Perfiles de Experimento Preconfigurados**

- Función: Templates de pruebas térmicas comunes
- Estimado: 1 semana

☐ **Sistema de Alarmas Inteligentes**

- Función: Detección de dry-out, sobrecalentamiento
- Notificaciones: Email, SMS, interfaz web
- Estimado: 2 semanas

☐ **Gráficos Científicos en Tiempo Real**

- Tecnología: Matplotlib, Plotly.js
- Función: Visualización de R_thermal vs Potencia
- Estimado: 1-2 semanas

☐ **Integración con LabVIEW/MATLAB**

- Función: Exportar datos en formato compatible
- Estimado: 1 semana

☐ **Multi-dispositivo (Múltiples Fuentes XLN)**

- Función: Experimentar con varios heat pipes simultáneamente
- Estimado: 3-4 semanas

MATRIZ DE DECISIONES CRÍTICAS

Antes de terminar Semana 4, decidir:

Decisión	Opción A	Opción B	Seleccionada
Arduino modelo	Nano 33 IoT 3.3V	Pro Mini 3.3V + convertidor	☐ _____
Control Bomba	Relé GPIO 26	Controlador PWM Arduino	☐ _____
Control Fuente	Comando SCPI "OUT"	Relé de contacto	☐ _____
Presupuesto USB	1.2A actual suficiente	Hub alimentado necesario	☐ _____
OLED frecuencia	Actualiza cada 2s	Actualiza cada 5s	☐ _____
Auto-inicio	systemd (profesional)	rc.local (simple)	☐ _____

TIMELINE CONSOLIDADO (8 SEMANAS)

```
SEMANA 1: Hardware + OS + Drivers DAQ
├─ Día 1-2: Setup inicial
├─ Día 3-4: Instalación drivers MCC
└─ Día 5-7: Primeras verificaciones

SEMANA 2: Configuración Hardware
├─ Día 5-6: Fuente XLN red + protecciones
├─ Día 6-7: Termopares + relés
└─ Día 7: Documentación

SEMANA 3: Desarrollo Software Base
├─ Día 8-9: Estructura + dependencias
├─ Día 9-10: Integración módulos
├─ Día 10: Frontend básico
└─ Día 10: Logging

SEMANA 4: Testing + Deployment
├─ Día 11-12: Pruebas unitarias
├─ Día 13: Endpoints API
├─ Día 14: systemd service
└─ Día 14: Auto-boot verification

SEMANA 5: Preparación Expansión
├─ Día 15: Compra componentes Arduino + OLED
├─ Día 16: Test componentes independientes
└─ Día 16: Presupuesto corriente USB

SEMANA 6: Integración Arduino + OLED
├─ Día 17: Arduino I2C
├─ Día 18: OLED display
└─ Día 19: Módulos Python nuevos

SEMANA 7: Bomba + Auto-inicio Expandido
├─ Día 20: Relé bomba + test
├─ Día 21: systemd OLED + LabPiPanel
└─ Día 22: Endpoints Flask nuevos

SEMANA 8: Testing e Integración Final
├─ Día 23: Pruebas integradas
├─ Día 24: Experimento piloto
└─ Día 25: Documentación final
```

CRITERIOS DE ÉXITO

Fin de Semana 4 (Base operacional):

- ✓ Sistema LabPiPanel operando en modo base
- ✓ Todos los componentes hardware funcionan
- ✓ API endpoints responden correctamente
- ✓ Servicio auto-inicia al reboot

Fin de Semana 8 (Sistema completo):

- ✓ Arduino I2C comunicando
- ✓ OLED mostrando IP y rendimiento
- ✓ Bomba controlable vía relé y web
- ✓ Experimento térmico piloto exitoso
- ✓ Documentación completa

RECURSOS NECESARIOS

- [ ] Acceso administrativo al Raspberry Pi (sudo)
- [ ] Conexión a internet para descargas
- [ ] Laboratorio con fuente de 12-24V para bomba (si aplica)
- [ ] Computadora para desarrollo y testing
- [ ] Cables USB, Ethernet, GPIO jumpers
- [ ] Multímetro para verificaciones
- [ ] Documentación técnica de componentes (impresa o digital)

CONTACTOS Y SOPORTE

En caso de problemas:

- Problemas de hardware: _____
- Problemas de software: _____
- Supervisor de laboratorio: _____
- Responsable de proyecto: _____

DOCUMENTO FIRMADO Y COMPROMETIDO

Responsable del Proyecto: _____ Fecha: _____

Supervisor de Laboratorio: _____ Fecha: _____

FIN DE ROADMAP - VERSIÓN 1.0