GUÍA DE CONTINUIDAD - Próximas Implementaciones en Laboratorio

Table of Contents

- 1. ESTADO ESPERADO AL FINALIZAR SEMANA 8
- 2. OPCIONES DE EXPANSIÓN (ROADMAP FUTURO)
- models.py
- websocket_handler.py
- alarm_system.py
- fuentes_multiples.py
  - 3. MATRIZ DE PRIORIDADES RECOMENDADAS
  - 4. CHECKLIST PARA INICIAR CADA OPCIÓN
  - 5. CONSIDERACIONES DE PRODUCCIÓN
  - 6. PLANIFICACIÓN A LARGO PLAZO (6-12 MESES)
  - 7. RECURSOS DE APRENDIZAJE RECOMENDADOS
  - 8. EQUIPO RECOMENDADO PARA EXPANSIONES
  - 9. PRESUPUESTO ESTIMADO PARA EXPANSIONES
  - 10. MÉTRICAS DE ÉXITO
  - CONCLUSIÓN

Proyecto: LabPiPanel - Sistema de Control de Laboratorio
Objetivo: Proporcionar hoja de ruta para expansiones futuras post-Semana 8
Fecha: 30 de Octubre de 2025

1. ESTADO ESPERADO AL FINALIZAR SEMANA 8

Sistema Base Completamente Funcional

Al término de la Semana 8, tu sistema LabPiPanel habrá alcanzado:

Hardware Operacional:

- Raspberry Pi 3/4 con Raspbian Bullseye
- Fuente XLN30052 controlable remotamente por Telnet (puerto 5024)
- DAQ USB-5203 con 8 termopares tipo K conectados
- 4 relés GPIO funcionales (1 para bomba, 3 disponibles)
- Arduino Nano 33 IoT comunicando por I2C (dirección 0x08)
- Display OLED SSD1306 mostrando IP y métricas del sistema (dirección 0x3C)

Software Implementado:

- Servidor Flask en Gunicorn escuchando puerto 5000
- API REST con 8+ endpoints funcionales
- Módulos de control: fuente_xln.py, daq_usb5203.py, relay_controller.py
- Módulos de expansión: arduino_i2c_master.py, oled_display.py, thermal_experiment.py
- Servicios systemd para auto-inicio: labpipanel.service, oled-display.service
- Logging estructurado a `/var/log/labpipanel/`

Capacidades Funcionales:

- Experimento térmico piloto ejecutable: secuencia 1-3W, duración 10 min/nivel
- Datos de resistencia térmica exportables a CSV
- Control manual de bomba, relés y fuente desde web e I2C
- Monitoreo en vivo de temperatura desde 8 canales DAQ
- Visualización local en OLED de: IP, CPU, temperatura, hora

2. OPCIONES DE EXPANSIÓN (ROADMAP FUTURO)

Opción A: Base de Datos Histórica (Recomendado - Semana 9-11)

Objetivo: Persistencia de datos y análisis histórico de experimentos

Por qué es importante:

- Actualmente pierdes datos al reiniciar
- No hay comparación entre experimentos antiguos y nuevos
- Falta trazabilidad de cambios en propiedades térmicas

Tecnología elegida: SQLite (inicio simple) → PostgreSQL (profesional futuro)

Implementación:

1. **Crear modelo de datos:**

```
# models.py<a></a>
from datetime import datetime
import sqlite3

class ExperimentDB:
    def __init__(self, db_path='/home/pi/LabPiPanel/data/experiments.db'):
        self.db_path = db_path
        self.init_db()

    def init_db(self):
        conn = sqlite3.connect(self.db_path)
        c = conn.cursor()

        # Tabla de experimentos
        c.execute('''CREATE TABLE IF NOT EXISTS experiments
                     (id INTEGER PRIMARY KEY,
                      fluid_name TEXT,
                      date_started TIMESTAMP,
                      date_finished TIMESTAMP,
                      status TEXT)''')

        # Tabla de mediciones
        c.execute('''CREATE TABLE IF NOT EXISTS measurements
                     (id INTEGER PRIMARY KEY,
                      experiment_id INTEGER,
                      power_w REAL,
                      T_evap REAL,
                      T_cond REAL,
                      R_thermal REAL,
                      timestamp TIMESTAMP,
                      FOREIGN KEY(experiment_id) REFERENCES experiments(id))''')

        conn.commit()
        conn.close()

    def save_measurement(self, exp_id, power, T_e, T_c, R_th):
        conn = sqlite3.connect(self.db_path)
        c = conn.cursor()
        c.execute('''INSERT INTO measurements
                     (experiment_id, power_w, T_evap, T_cond, R_thermal, timestamp)
```

```
                    VALUES (?, ?, ?, ?, ?, ?)''',
                  (exp_id, power, T_e, T_c, R_th, datetime.now()))
        conn.commit()
        conn.close()

    def get_experiment_history(self, fluid_name):
        conn = sqlite3.connect(self.db_path)
        c = conn.cursor()
        c.execute('''SELECT * FROM measurements
                     WHERE experiment_id IN
                     (SELECT id FROM experiments WHERE fluid_name = ?)
                     ORDER BY timestamp DESC''', (fluid_name,))
        results = c.fetchall()
        conn.close()
        return results
```

2. **Integrar con thermal_experiment.py:**
   - Guardar cada medición en BD
   - Cargar históricos para comparación
   - Generar reportes con estadísticas (media, std dev)
3. **Nuevo endpoint Flask:**

```
@app.route('/api/experiments/history/&lt;fluid&gt;')
def get_history(fluid):
    db = ExperimentDB()
    data = db.get_experiment_history(fluid)
    return jsonify({'data': data})
```

Estimado de tiempo: 2-3 semanas

Herramientas:

- SQLAlchemy ORM (opcional, abstracción de BD)
- Alembic para migraciones de esquema

Opción B: WebSockets para Gráficos en Tiempo Real (Semana 9-10)

Objetivo: Visualización en vivo de temperatura y resistencia térmica durante experimento

Por qué es importante:

- Actualmente solo ves números en tabla
- No hay retroalimentación visual durante experimento
- Imposible detectar anomalías en tiempo real (dry-out del heat pipe)

Tecnología: Flask-SocketIO

Implementación:

1. **Instalar dependencia:**

```
pip install flask-socketio==5.3.0
pip install python-engineio==4.5.1
pip install python-socketio==5.9.0
```

2. **Módulo de WebSocket:**

```
# websocket_handler.py<a></a>
from flask_socketio import SocketIO, emit
import threading
import time

def setup_socketio(app):
```

```
    socketio = SocketIO(app, cors_allowed_origins="*")

    @socketio.on('connect')
    def handle_connect():
        print('Cliente conectado')

    @socketio.on('start_experiment')
    def handle_start_exp(data):
        # Función que emite datos en tiempo real
        thread = threading.Thread(
            target=stream_experiment_data,
            args=(socketio, data)
        )
        thread.start()

    return socketio

def stream_experiment_data(socketio, exp_config):
    """Emite datos de experimento cada segundo a clientes conectados"""

    for power_level in exp_config['power_levels']:
        for minute in range(exp_config['duration_min']):
            # Leer datos
            temps = read_all_thermocouples()
            R_th = calculate_resistance(temps, power_level)

            # Emitir a cliente
            socketio.emit('experiment_data', {
                'power': power_level,
                'temps': temps,
                'R_thermal': R_th,
                'timestamp': time.time()
            })

            time.sleep(60)  # 1 minuto
```

3. **Frontend con gráfico en vivo:**

```
&lt;script src="https://cdn.socket.io/4.5.4/socket.io.min.js"&gt;&lt;/script&gt;
&lt;script src="https://cdn.jsdelivr.net/npm/chart.js"&gt;&lt;/script&gt;

&lt;canvas id="temperatureChart"&gt;&lt;/canvas&gt;

&lt;script&gt;
const socket = io();
const ctx = document.getElementById('temperatureChart').getContext('2d');
const chart = new Chart(ctx, {
    type: 'line',
    data: {
        labels: [],
        datasets: [
            {label: 'T Evap', data: [], borderColor: 'red'},
            {label: 'T Cond', data: [], borderColor: 'blue'}
        ]
    }
});

socket.on('experiment_data', (data) =&gt; {
    chart.data.labels.push(new Date().toLocaleTimeString());
    chart.data.datasets[0].data.push(data.temps.evap_avg);
    chart.data.datasets[1].data.push(data.temps.cond_avg);
    chart.update();
});
&lt;/script&gt;
```

Estimado de tiempo: 1-2 semanas

Opción C: Perfiles de Experimento (Semana 11-12)

Objetivo: Templates predefinidos de pruebas térmicas comunes

Por qué es importante:

- Actualmente cada experimento requiere configuración manual
- Sin templates, es fácil cometer errores de parámetros
- No hay estandardización entre experimentos

Implementación:

1. **Archivo de configuración de templates:**

```
// experiment_templates.json
{
  "templates": [
    {
      "name": "Base Water Test",
      "fluid": "Deionized Water",
      "power_levels": [1, 2, 3, 4, 5],
      "duration_per_level_min": 40,
      "description": "Baseline test with pure water"
    },
    {
      "name": "CQD Nanofluid Test",
      "fluid": "Water-Butanol 4% + CQD 500mg/L",
      "power_levels": [1, 2, 3, 4, 5, 6, 7],
      "duration_per_level_min": 40,
      "description": "Full power range for nanofluid"
    }
  ]
}
```

2. **Interfaz web para seleccionar y ejecutar:**
    - Dropdown con templates disponibles
    - Botón "Start Experiment"
    - Opción de personalizar parámetros

Estimado de tiempo: 1 semana

Opción D: Sistema de Alarmas Inteligentes (Semana 12-13)

Objetivo: Detección automática de condiciones anormales y alertas

Condiciones a detectar:

1. **Dry-out del heat pipe:**
    - Indicador: R_thermal aumenta abruptamente (>20% en 1 minuto)
    - Acción: Parar experimento, notificar operador, registrar en log
2. **Sobrecalentamiento:**
    - Indicador: T_evap > 100°C (umbral configurable)
    - Acción: Reducir potencia automáticamente, alerta audible
3. **Falla del DAQ:**
    - Indicador: Timeout en lectura de termopares >3 veces consecutivas
    - Acción: Parar experimento, desactivar fuente de seguridad
4. **Pérdida de comunicación con Arduino:**
    - Indicador: I2C no responde
    - Acción: Log warning, continuar sin Arduino

Implementación:

```
# alarm_system.py<a></a>
class AlarmSystem:
    def __init__(self, oled_display, email_alert=None):
        self.oled = oled_display
        self.email = email_alert
        self.alarm_state = 'normal'

    def check_dry_out(self, R_thermal_history):
        """Detecta secado del heat pipe"""
        if len(R_thermal_history) &lt; 2:
            return False

        R_prev = R_thermal_history[-2]
        R_curr = R_thermal_history[-1]

        if (R_curr - R_prev) / R_prev &gt; 0.20:  # Aumento &gt;20%
            self.trigger_alarm('DRY_OUT', 'Heat pipe dry-out detected')
            return True
        return False

    def trigger_alarm(self, alarm_type, message):
        """Activa alarma multicanal"""
        self.alarm_state = alarm_type

        # Mostrar en OLED
        self.oled.draw_alert(message)

        # Enviar email si está configurado
        if self.email:
            self.email.send_alert(alarm_type, message)

        # Log
        logging.error(f"ALARM: {alarm_type} - {message}")
```

Estimado de tiempo: 2 semanas

Opción E: Integración con LabVIEW/MATLAB (Semana 14-15)

Objetivo: Análisis avanzado de datos en software profesional

Métodos de integración:

1. **Exportar CSV:**
   - Ya implementado en thermal_experiment.py
   - Importar en MATLAB: `data = readtable('experiment.csv')`
   - Usar herramientas de análisis built-in
2. **API REST completa:**
   - Endpoint `GET /api/experiments/<exp_id>/data`
   - LabVIEW conecta por HTTP y lee JSON
   - MATLAB usa `webread()` para descargar datos
3. **Conexión TCP/IP directa (avanzado):**
   - Socket en puerto 5555 específico para clientes LabVIEW
   - Protocolo binario optimizado
   - Baja latencia para análisis en tiempo real

Estimado de tiempo: 2-3 semanas

Opción F: Multi-dispositivo - Múltiples Fuentes XLN (Semana 16-18)

Objetivo: Ejecutar múltiples experimentos simultáneamente

Desafíos:

- Raspberry Pi tiene 1 puerto Telnet
- Múltiples direcciones IP para múltiples fuentes
- Sincronización de datos

Solución:

1. **Adquirir múltiples fuentes XLN:**
   - Cada una con IP diferente en red local
   - Configurar desde panel frontal
2. **Modificar código para multi-conexión:**

```
# fuentes_multiples.py<a></a>
class MultiPowerSupply:
    def __init__(self, ips_config):
        """ips_config = {'exp1': '192.168.1.100', 'exp2': '192.168.1.101'}"""
        self.supplies = {}
        for exp_id, ip in ips_config.items():
            self.supplies[exp_id] = FuenteXLN(host=ip)

    def set_all_voltages(self, voltage_dict):
        """voltage_dict = {'exp1': 5.0, 'exp2': 7.0}"""
        for exp_id, voltage in voltage_dict.items():
            self.supplies[exp_id].enviar_comando(f"VSET {voltage}")
```

3. **Usar múltiples DAQs:**
   - 1 DAQ por Raspberry Pi (subnets diferentes)
   - Raspberry Pi maestro coordina a esclavos por red
   - Agregar nuevos Pi según sea necesario

Estimado de tiempo: 3-4 semanas

Opción G: Integración LIMS (Semana 19-20)

Objetivo: Conectar con sistemas profesionales de gestión de laboratorio

LIMS: Laboratory Information Management System

Funcionalidad:

- Enviar resultados de experimentos automáticamente a LIMS central
- Consultar configuraciones de experimento desde LIMS
- Generar reportes profesionales con trazabilidad completa

Protocolo: HTTP REST o SFTP

Estimado de tiempo: 2-3 semanas

3. MATRIZ DE PRIORIDADES RECOMENDADAS

Opción	Complejidad	Tiempo	Impacto	Prioridad	Dependencias
A. Base de Datos	Media	2-3s	Alto	1	Ninguna
B. WebSockets	Media	1-2s	Alto	2	Base de Datos

Opción	Complejidad	Tiempo	Impacto	Prioridad	Dependencias
C. Perfiles	Baja	1s	Medio	3	Ninguna
D. Alarmas	Media	2s	Alto	4	Base de Datos
E. LabVIEW/MATLAB	Baja	2-3s	Medio	5	Ninguna
F. Multi-dispositivo	Alta	3-4s	Muy Alto	6	A, D
G. LIMS	Media	2-3s	Muy Alto	7	A, B

Recomendación: Implementar en orden **A** → **B** → **C** → **D** → **E** en los siguientes 4-5 meses para sistema profesional completo

4. CHECKLIST PARA INICIAR CADA OPCIÓN

Antes de comenzar Opción A (Base de Datos)

- ☐ Sistema base operando perfectamente (Semana 8 completada)
- ☐ 2+ experimentos exitosos realizados
- ☐ CSV de datos exportados correctamente
- ☐ Acceso sudo al Raspberry Pi
- ☐ 2GB libres en almacenamiento

Antes de comenzar Opción B (WebSockets)

- ☐ Opción A completada (BD funcionando)
- ☐ Knowledge de JavaScript/HTML (o tutorial previo)
- ☐ Librería Chart.js descargada

Antes de comenzar Opción C (Perfiles)

- ☐ Mínimo 5 experimentos completados
- ☐ Identificados parámetros comunes entre experimentos
- ☐ Documentación de procedimiento estándar

Antes de comenzar Opción D (Alarmas)

- ☐ Opción A completada (registro de datos necesario)
- ☐ Identificados umbrales críticos en sistema
- ☐ Email o notificación de alerta configurada

5. CONSIDERACIONES DE PRODUCCIÓN

Antes de escalar a uso profesional/publicación:

✓ **Seguridad:**

- ☐ Autenticación de usuarios implementada (Flask-Login)
- ☐ Base de datos cifrada
- ☐ HTTPS habilitado (certificado SSL Let's Encrypt)
- ☐ Rate limiting en API endpoints

✓ **Confiabilidad:**

- ☐ Backup automático de BD (script cron diario)

- ☐ Monitoreo de espacio en disco
- ☐ Alertas de fallo del sistema
- ☐ Procedimiento de recovery documentado

✓ **Rendimiento:**

- ☐ Índices en BD para queries comunes
- ☐ Caché implementado para datos históricos
- ☐ Límite de tamaño de datos por experimento
- ☐ Archivado de experimentos antiguos

✓ **Documentación:**

- ☐ API Swagger completamente documentada
- ☐ Guía de administrador del sistema
- ☐ Guía de troubleshooting detallada
- ☐ Video tutoriales de operación

6. PLANIFICACIÓN A LARGO PLAZO (6-12 MESES)

Trimestre 1 (Semanas 9-13)

- Implementar Opciones A, B, C
- Beta testing con grupo pequeño de usuarios
- Correcciones basadas en feedback

Trimestre 2 (Semanas 14-26)

- Implementar Opciones D, E
- Publicación en Hackaday/Hackster.io
- Revisión académica del manuscrito

Trimestre 3 (Semanas 27-39)

- Implementar Opción F (multi-dispositivo)
- Escribir artículo científico
- Presentar en conferencia de investigación

Trimestre 4+ (Post-39 semanas)

- Implementar Opción G (LIMS)
- Comercialización/Open Source
- Mantenimiento y mejoras continuas

7. RECURSOS DE APRENDIZAJE RECOMENDADOS

Para SQLite/Base de Datos:

- "Database Design Manual" - Andrew Booz
- SQLAlchemy documentation: sqlalchemy.org
- Real Python: "Python SQLite Tutorial"

Para WebSockets/Tiempo Real:

- Flask-SocketIO docs: flask-socketio.readthedocs.io
- Socket.IO tutorial: socket.io
- Real Python: "Getting Started with Flask-SocketIO"

Para MATLAB/LabVIEW Integration:

- MathWorks: "Web Services in MATLAB"
- NI LabVIEW: HTTP Client tutorial
- Tutorial: REST API calls from LabVIEW

Para Seguridad/Producción:

- OWASP Top 10 for Python web apps
- Flask Security: flask-security-too.readthedocs.io
- "Production-Ready Microservices" - Noah Gift

8. EQUIPO RECOMENDADO PARA EXPANSIONES

Considerar agregar al equipo:

- **Data Scientist/Analyst:** Para análisis estadístico de datos térmicos
- **DevOps Engineer:** Para CI/CD, deployment, monitoreo
- **Frontend Developer:** Si se desean interfaces avanzadas
- **Electrical Engineer:** Para integraciones de hardware adicionales

9. PRESUPUESTO ESTIMADO PARA EXPANSIONES

Opción	Hardware	Software	Total
A. BD	$0	$0	$0
B. WebSockets	$0	$0	$0
C. Perfiles	$0	$0	$0
D. Alarmas	$50-100 (buzzer, etc)	$0	$50-100
E. MATLAB	$0	Existente	$0
F. Multi XLN	$300-500/fuente	$0	$300-500+
G. LIMS	Depende vendor	$0-5000	$0-5000

10. MÉTRICAS DE ÉXITO

Para cada implementación, medir:

1. **Funcionalidad:**
   - ☐ 100% de funciones documentadas trabajando
   - ☐ 0 errores críticos en logs
   - ☐ Uptime >99.5%
2. **Rendimiento:**

- ☐ Tiempo de respuesta API <500ms
- ☐ Uso de memoria <80% de disponible
- ☐ Ningún memory leak después de 24h continuas

3. **Confiabilidad:**
   - ☐ Datos persistentes tras reboot
   - ☐ Recuperación automática de fallos
   - ☐ Consistencia de datos >99.99%
4. **Usabilidad:**
   - ☐ Interfaz intuitiva sin capacitación
   - ☐ <5 clicks para operación común
   - ☐ Error messages claros y accionables

CONCLUSIÓN

El sistema LabPiPanel base es una **plataforma excepcional** para investigación térmica. Las expansiones sugeridas transformarían este sistema en una **solución empresarial profesional** comparable a sistemas comerciales costosos, pero con la ventaja de ser **personalizable y escalable**.

La implementación sugerida en 8 semanas para base + expansiones futuras permite:

- Investigación reproducible de propiedades térmicas de nanofluidos
- Datos históricos para análisis longitudinal
- Integración con herramientas profesionales (MATLAB, LabVIEW, LIMS)
- Potencial publicación en journals científicos

Próximo paso: Después de completar Semana 8, agendar reunión para seleccionar prioridades de expansión.

Documento preparado por: Sistema de Análisis LabPiPanel
Fecha: 30 de Octubre de 2025
Versión: 1.0

Para actualizaciones futuras: `git commit -m "Doc: Update roadmap version X.Y"`