☀️ SISTEMA INVERSOR HÍBRIDO - MANUAL TÉCNICO COMPLETO

Parte 2: Backend y APIs

---

5. BACKEND FASTAPI

5.1 Arquitectura Backend

```
backend/
├── main.py                    # Aplicación principal
├── config.py                  # Configuración y settings
├── database.py                # Modelos y base de datos
├── schemas.py                 # Schemas Pydantic
├── routers/
│   ├── esp32_router.py       # Gestión dispositivos ESP32
│   ├── dimensionamiento_router.py  # Cálculos sistema
│   ├── ml_router.py          # Machine Learning
│   └── status_router.py      # Estado servicios
├── services/
│   ├── nasa_power_service.py        # NASA POWER API
│   ├── dimensionamiento_service.py  # Cálculos
│   ├── ml_predictor_service.py      # ML modelos
│   ├── cargas_service.py            # Cargas inductivas
│   └── weather_confidence_service.py # Multi-fuente
└── .env                       # Variables de entorno
```

5.2 Endpoints Principales

Gestión ESP32:

```
POST /api/esp32/register
POST /api/esp32/heartbeat
GET  /api/esp32/devices
GET  /api/esp32/config/{device_id}
POST /api/esp32/config/{device_id}
```

Dimensionamiento:

```
POST /api/dimensionamiento/opcion1  # Desde consumo
POST /api/dimensionamiento/opcion2  # Desde recursos
GET  /api/dimensionamiento/clima/{lat}/{lon}
```

Machine Learning:

```
POST /api/ml/train
GET  /api/ml/metrics
POST /api/ml/predict/{month}
```

Estado del Sistema:

```
GET /api/status/health
GET /api/status/forecast
```

5.3 Base de Datos

Modelos SQLite:

```
class EnergyRecord:
    - timestamp
    - solar_power
    - wind_power
    - battery_soc
    - consumption

class WeatherData:
    - timestamp
    - temperature
    - humidity
    - wind_speed
    - solar_radiation

class Prediction:
    - timestamp
    - predicted_solar
    - predicted_wind
    - confidence

class Alert:
    - timestamp
    - severity
    - message
```

---

6. NASA POWER API

6.1 Características

- **Fuente**: NASA Langley Research Center
- **Datos**: 40 años históricos (1981-presente)
- **Frecuencia**: Diaria, mensual, climatológica
- **Cobertura**: Global
- **Costo**: GRATIS, sin límite de requests
- **Actualización**: Mensual

6.2 Parámetros Disponibles

Energía Solar:

```
ALLSKY_SFC_SW_DWN     # Irradiancia total (kWh/m²/día)
CLRSKY_SFC_SW_DWN     # Cielo despejado (kWh/m²/día)
```

Energía Eólica:

```
WS50M    # Velocidad viento 50m (m/s)
WS10M    # Velocidad viento 10m (m/s)
WD50M    # Dirección viento 50m (°)
```

Meteorología:

```
T2M       # Temperatura 2m (°C)
T2M_MAX   # Temperatura máxima (°C)
T2M_MIN   # Temperatura mínima (°C)
RH2M      # Humedad relativa (%)
PS        # Presión superficie (kPa)
```

6.3 Ejemplo de Request

```
import httpx

url = "https://power.larc.nasa.gov/api/temporal/monthly/point"
params = {
    "parameters": "ALLSKY_SFC_SW_DWN,WS50M,T2M",
    "community": "RE",  # Renewable Energy
    "longitude": -62.2663,
    "latitude": -38.7183,
    "start": 2014,
    "end": 2024,
    "format": "JSON"
}

response = httpx.get(url, params=params)
data = response.json()
```

6.4 Procesamiento de Datos

Cálculo de Promedios:

```
def calcular_promedios(data, years):
    solar_values = []
    wind_values = []

    for year in range(years):
        for month in range(1, 13):
```

```
            key = f"{year}{month:02d}"
            solar = data["ALLSKY_SFC_SW_DWN"][key]
            wind = data["WS50M"][key]

            solar_values.append(solar)
            wind_values.append(wind)

    return {
        "solar_avg": sum(solar_values) / len(solar_values),
        "wind_avg": sum(wind_values) / len(wind_values)
    }
```

---

7. OPENWEATHER API

7.1 Características

- **Fuente**: OpenWeather
- **Datos**: Tiempo actual + pronóstico 5 días
- **Frecuencia**: Actualización cada 10 minutos
- **Cobertura**: Global
- **Costo**: 60 requests/minuto gratis
- **Precisión**: Alta para 0-48 horas

7.2 Endpoints Usados

Clima Actual:

```
GET https://api.openweathermap.org/data/2.5/weather
Params:
  - lat: latitud
  - lon: longitud
  - appid: API_KEY
  - units: metric
```

Pronóstico 5 Días:

```
GET https://api.openweathermap.org/data/2.5/forecast
Params:
  - lat: latitud
  - lon: longitud
  - appid: API_KEY
  - units: metric
```

7.3 Ejemplo de Response

```
{
  "weather": [{
    "main": "Clear",
    "description": "cielo despejado"
```

```
  }],
  "main": {
    "temp": 22.5,
    "humidity": 65,
    "pressure": 1013
  },
  "wind": {
    "speed": 5.2,
    "deg": 180
  },
  "clouds": {
    "all": 20
  },
  "dt": 1737464400
}
```

7.4 Procesamiento para ML

Cálculo de Factor Solar:

```python
def calcular_factor_solar(clouds_percent):
    # Reducción por nubes (70% de impacto)
    factor = 1.0 - (clouds_percent / 100) * 0.7
    return max(0, min(1, factor))

# Ejemplo:
# 20% nubes → factor 0.86 (muy bueno)
# 50% nubes → factor 0.65 (regular)
# 80% nubes → factor 0.44 (malo)
```

Cálculo de Factor Eólico:

```python
def calcular_factor_eolico(wind_speed_ms):
    # Normalizado a velocidad nominal (10 m/s)
    factor = wind_speed_ms / 10.0
    return min(1.5, max(0, factor))

# Ejemplo:
# 5.0 m/s → factor 0.50
# 10.0 m/s → factor 1.00
# 15.0 m/s → factor 1.50 (limitado)
```

---

8. COMUNICACIÓN ESP32-BACKEND

8.1 Registro de Dispositivo

Request (ESP32 → Backend):

```
POST /api/esp32/register
Content-Type: application/json

{
  "device_id": "ESP32_INVERSOR_001",
  "ip_local": "192.168.0.150",
  "mac_address": "AA:BB:CC:DD:EE:FF",
  "firmware_version": "2.0",
  "latitude": -38.7183,
  "longitude": -62.2663
}
```

Response (Backend → ESP32):

```
{
  "status": "registered",
  "device_id": "ESP32_INVERSOR_001",
  "message": "Dispositivo registrado correctamente",
  "timestamp": "2025-01-21T13:45:00"
}
```

8.2 Telemetría

Request (ESP32 → Backend cada 5 seg):

```
POST /api/telemetry
Content-Type: application/json

{
  "device_id": "ESP32_INVERSOR_001",
  "timestamp": 123456789,
  "voltajes": [52.4, 51.8, 52.1],
  "corrientes": {
    "solar": 12.5,
    "eolica": 8.3,
    "consumo": 15.2
  },
  "temperatura": 25.6,
  "velocidad_viento": 6.2,
  "irradiancia": 850,
  "relays": {
    "solar": true,
    "eolica": true,
    "red": false,
    "carga": true,
    "freno": false
  }
}
```

8.3 Heartbeat

Request (ESP32 → Backend cada 30 seg):

```
POST /api/esp32/heartbeat
Content-Type: application/json

{
  "device_id": "ESP32_INVERSOR_001",
  "uptime": 3600,
  "free_heap": 245000,
  "rssi": -45
}
```

Response:

```
{
  "status": "ok",
  "timestamp": "2025-01-21T13:45:30"
}
```

8.4 Comandos

Request (ESP32 → Backend cada 10 seg):

```
GET /api/esp32/commands/ESP32_INVERSOR_001
```

Response:

```
{
  "commands": [
    {
      "type": "set_relay",
      "relay": "solar",
      "state": true
    },
    {
      "type": "set_mode",
      "mode": "auto"
    }
  ],
  "config_changed": false
}
```

FIN PARTE 2

Continúa en MANUAL_COMPLETO_PARTE_3.md