Propósito y Funcionalidad

Dashboard comprehensivo que muestra el estado de todos los dispositivos con indicadores visuales y detección automática de condiciones críticas.

Características Técnicas

Arquitectura de Monitoreo

```
javascript
```

```
export async function generalStatusWidget() {
  const { devices } = getState();


  // Carga inicial de dispositivos
  if (devices.length === 0) {
    try {
      await deviceService.getAllDevices();
    } catch (error) {
      console.error('Error cargando dispositivos:', error);
    }
  }
}
```

Algoritmo de Detección de Estado Crítico

```
javascript
```

```
async function getDeviceCriticalStatus(device) {
```

```
  try {
    // Obtener datos recientes (últimos 10 registros)
    const sensorData = await
deviceService.getDeviceSensorData(device.id_dispositivo, 10);

    if (!sensorData || sensorData.length === 0) {
      return {
        status: 'no_data',
        color: '#FF9800', // Naranja para sin datos
        message: 'Sin datos recientes'
      };
    }

    // Sistema de umbrales configurables
    const criticalThresholds = {
      'temperatura': { min: 0, max: 50 }, // °C
      'humedad': { min: 20, max: 80 }, // %
      'co2': { min: 300, max: 1000 }, // ppm
      'pm25': { min: 0, max: 25 }, // µg/m³
      'presion': { min: 900, max: 1100 }, // hPa
      'luminosidad': { min: 0, max: 10000 }, // lux
      'vibracion': { min: 0, max: 2 }, // g
```

```
  'sonido': { min: 0, max: 100 } // dB
};

let hasCriticalValues = false;
let criticalSensors = [];

// Análisis de cada dato de sensor
for (const data of sensorData) {
  const threshold = criticalThresholds[data.tipo_sensor];
  if (threshold) {
    const value = parseFloat(data.valor);
    if (value < threshold.min || value > threshold.max) {
      hasCriticalValues = true;
      criticalSensors.push({
        sensor: data.tipo_sensor,
        value: value,
        unit: data.unidad,
        threshold: threshold
      });
    }
  }
}
```

```
    // Determinación del estado final
    if (hasCriticalValues) {
      return {
        status: 'critical',
        color: '#F44336', // Rojo
        message: `${criticalSensors.length} sensor(es) con valores críticos`,
        criticalSensors: criticalSensors
      };
    } else {
      return {
        status: 'normal',
        color: '#4CAF50', // Verde
        message: 'Todos los sensores funcionando normalmente'
      };
    }
  } catch (error) {
    return {
      status: 'error',
      color: '#9E9E9E', // Gris
      message: 'Error verificando estado'
```

```
        };
      }
}
```

Sistema de Actualización Automática

```
javascript
```

```
    // Función de actualización completa
    async function updateDevicesStatus() {
      devicesContainer.innerHTML = "";

      // Indicador de carga
      const loadingDiv = el("div", {
        style: "text-align: center; padding: 20px; color: #666;
    grid-column: 1 / -1;"
      }, "Verificando estado de dispositivos...");
      devicesContainer.appendChild(loadingDiv);

      // Procesamiento paralelo de dispositivos
      const statusPromises = currentDevices.map(async (device) => {
        const criticalStatus = await getDeviceCriticalStatus(device);
        return { device, criticalStatus };
      });
```

```
        try {
          const deviceStatuses = await Promise.all(statusPromises);
          devicesContainer.innerHTML = "";

          // Renderizado de cards
          deviceStatuses.forEach(({ device, criticalStatus }) => {
            const deviceCard = createDeviceCard(device, criticalStatus);
            devicesContainer.appendChild(deviceCard);
          });
        } catch (error) {
          console.error('Error actualizando estado:', error);
          devicesContainer.innerHTML = "";
          devicesContainer.appendChild(el("div", {
            style: "text-align: center; padding: 20px; color: #d32f2f;"
          }, "Error verificando estado de dispositivos"));
        }
      }

      // Actualización automática cada 30 segundos
const autoRefreshInterval = setInterval(updateDevicesStatus, 30000);
```

Componente Visual de Dispositivo

```
javascript
```

```
function createDeviceCard(device, criticalStatus) {
  const deviceCard = el("div", {
    style: `
      border: 1px solid #ddd; border-radius: 8px; padding: 15px;
      background: white; box-shadow: 0 2px 4px rgba(0,0,0,0.1);
      transition: transform 0.2s, box-shadow 0.2s;
    `,
    onmouseover: (e) => {
      e.target.style.transform = 'translateY(-2px)';
      e.target.style.boxShadow = '0 4px 8px rgba(0,0,0,0.15)';
    },
    onmouseout: (e) => {
      e.target.style.transform = 'translateY(0)';
      e.target.style.boxShadow = '0 2px 4px rgba(0,0,0,0.1)';
    }
  });

  // LED Indicator con animación
  const ledIndicator = el("div", {
    style: `
      width: 12px; height: 12px; border-radius: 50%;
      background-color: ${criticalStatus.color};
```

```
      box-shadow: 0 0 8px ${criticalStatus.color}40; /* 40 = 25%
    opacity */

      margin-right: 10px;

      animation: ${criticalStatus.status === 'critical' ? 'pulse 2s
    infinite' : 'none'};

    `

  });



  return deviceCard;
}
```

Sistema de Grid Responsive

```
javascript

    const devicesContainer = el("div", {

      id: "devices-status-container",

      style: "display: grid; grid-template-columns: repeat(auto-fit,
    minmax(300px, 1fr)); gap: 15px;"
});
```

Características

- Monitoreo automático: Actualización cada 30 segundos
- Indicadores visuales: LEDs con animación pulse
- Detección de críticos: Analiza datos de sensores
- Refresh manual: Botón de actualización

Algoritmo de Estado Crítico

1. Obtiene últimos 10 registros del dispositivo
2. Aplica umbrales por tipo de sensor
3. Calcula estado: normal, warning, critical
4. Actualiza indicadores LED