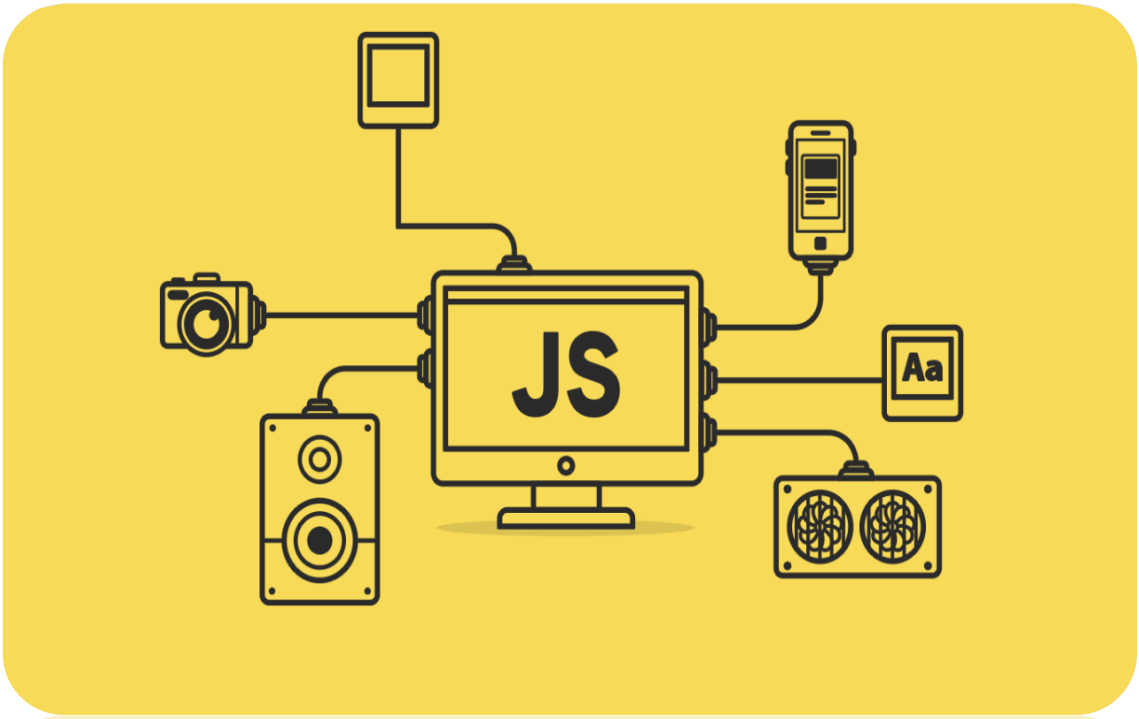
Documentación del Código JAVASCRIPT



Ismael Mariscal Santos

Proyecto de administración de sistemas informáticos en red

Índice

Gráfico de Donut para CPU

```
// Función para actualizar el gráfico de donut de la CPU
function updateCPULoadChart(cpuLoad) {
    // Formatear la carga de la CPU para mostrar solo dos decimales
    const formattedCpuLoad = cpuLoad.toFixed(0);

    // Actualizar el texto de la carga de la CPU
    const cpuLoadText = formattedCpuLoad + '%';
    document.querySelector('.cpu-load').textContent = cpuLoadText;

    // Actualizar los datos del gráfico de donut de la CPU
    cpuChart.data.datasets[0].data[0] = formattedCpuLoad;
    cpuChart.data.datasets[0].data[1] = 100 - formattedCpuLoad;

    // Verificar si la carga de la CPU supera el umbral (por ejemplo, 50%)
    if (cpuLoad > 50) {
        // Agregar una animación al gráfico
        cpuChart.options.animation = {
            duration: 1000, // Duración de la animación en milisegundos
            easing: 'easeInOutQuad' // Tipo de curva de la animación (opcional)
        };
    } else {
        // Si la carga de la CPU está por debajo del umbral, eliminar la animación
        cpuChart.options.animation = false;
    }

    // Actualizar el gráfico
    cpuChart.update();
}

// Función para obtener y mostrar la carga de la CPU en el gráfico de donut
function getCPULoadChartData() {
    // Construir una URL con un parámetro de consulta único para evitar la caché del navegador
    const timestamp = Date.now();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=system.cpu&after=-1&points=1&
    timestamp=${timestamp}`;

    // Realizar la solicitud GET a la API de Netdata
    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener la carga de la CPU
        const cpuLoad = parseFloat(data.data[0][6]);

        // Actualizar el gráfico de donut
        updateCPULoadChart(cpuLoad);
    })
    .catch(error => {
        console.error('Error al obtener la carga de la CPU del sistema:', error);
    });
}
```

- **Descripción:** Representa la carga de la CPU en forma de un gráfico de donut.
- **Funciones:**
    - updateCPULoadChart(cpuLoad): Actualiza el gráfico de donut con la carga de la CPU.

Parámetros:

- cpuLoad: Carga actual de la CPU en porcentaje.

Funcionalidad:

- Formatea la carga de la CPU para mostrar solo dos decimales.
- Actualiza el texto de la carga de la CPU en la interfaz.
- Actualiza los datos del gráfico de donut con la carga de la CPU.
- Agrega una animación al gráfico si la carga de la CPU supera un umbral definido.
- Actualiza el gráfico.

    - getCPULoadChartData(): Obtiene y muestra datos de carga de CPU en el gráfico de donut.

Descripción: Obtiene y muestra datos de carga de CPU en el gráfico de donut.
Funcionalidad:

- Construye una URL para obtener datos de carga de CPU desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener la carga de la CPU.
- Llama a updateCPULoadChart() para actualizar el gráfico con los nuevos datos.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Gráfico de Donut para RAM

```
// Función para actualizar el gráfico de donut de la RAM
function updateRAMChart(usedRAM, freeRAM) {
    // Formatear la carga de RAM para mostrar solo un decimal
    const formattedUsedRAM = usedRAM.toFixed(1);
    const formattedFreeRAM = freeRAM.toFixed(1);

    // Actualizar el texto de la carga de RAM
    const ramLoadText = formattedUsedRAM + ' MB';
    document.querySelector('.ram-load').textContent = ramLoadText;

    // Actualizar los datos del gráfico de donut de la RAM
    ramChart.data.datasets[0].data[0] = formattedUsedRAM;
    ramChart.data.datasets[0].data[1] = formattedFreeRAM;

    // Actualizar el gráfico
    ramChart.update();
}

// Función para obtener y mostrar la carga de RAM en el gráfico de donut
function getRAMChartData() {
    // Construir la URL de la API de Netdata para obtener datos de RAM
    const timestamp = Date.now();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=system.ram&after=-1&points=1&
    timestamp=${timestamp}`;

    // Realizar una solicitud GET a la API de Netdata
    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener los valores de RAM utilizada y libre
        const usedRAM = parseFloat(data.data[0][2]);
        const freeRAM = parseFloat(data.data[0][1]);

        // Actualizar el gráfico de donut de la RAM
        updateRAMChart(usedRAM, freeRAM);
    })
    .catch(error => {
        console.error('Error al obtener la carga de RAM del sistema:', error);
    });
}

// Llamar a la función para obtener la carga de RAM inicialmente
getRAMChartData();
```

- **Descripción:** Representa el uso de la memoria RAM en forma de un gráfico de donut.
- **Funciones:**
    - updateRAMChart(usedRAM, freeRAM): Actualiza el gráfico de donut con el uso de RAM.

Parámetros:

- usedRAM: Cantidad de RAM utilizada.

- freeRAM: Cantidad de RAM libre.

Funcionalidad:

- Formatea el uso de RAM para mostrar solo un decimal.
- Actualiza el texto de uso de RAM en la interfaz.
- Actualiza los datos del gráfico de donut con el uso de RAM.
- Actualiza el gráfico.

    -
    - getRAMChartData(): Obtiene y muestra datos de uso de RAM en el gráfico de donut.

Funcionalidad:

- Construye una URL para obtener datos de uso de RAM desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener el uso de RAM.
- Llama a updateRAMChart() para actualizar el gráfico con los nuevos datos.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Gráfico de Donut para Disco

```
// Función para obtener y mostrar el uso de Disco en el gráfico de donut
function getDiskUsageData() {
    // Construir la URL de la API de Netdata para obtener datos de Disco
    const timestamp = Date.now();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=disk_space._&after=-1&points=1&
    timestamp=${timestamp}`;

    // Realizar una solicitud GET a la API de Netdata
    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener los valores de Disco disponibles y utilizados
        const availDisk = parseFloat(data.data[0][1]);
        const usedDisk = parseFloat(data.data[0][2]);

        // Calcular el total del Disco sumando los valores disponibles y utilizados
        const totalDisk = Math.round((availDisk + usedDisk) * 10) / 10; // Redondear el
        resultado a 1 decimal

        // Actualizar el gráfico de donut de Disco
        updateDiskChart(usedDisk, totalDisk);
    })
    .catch(error => {
        console.error('Error al obtener el uso de Disco del sistema:', error);
    });
}

// Función para actualizar el gráfico de donut de Disco
function updateDiskChart(usedDisk, totalDisk) {
    const availDisk = totalDisk - usedDisk; // Calcular el Disco disponible restando el Disco
    utilizado del total

    // Actualizar los datos del gráfico de donut de Disco
    diskChart.data.datasets[0].data[0] = Math.round(usedDisk * 10) / 10; // Redondear el uso de
    Disco a 1 decimal
    diskChart.data.datasets[0].data[1] = Math.round(availDisk * 10) / 10; // Redondear el Disco
    disponible a 1 decimal

    // Actualizar el gráfico
    diskChart.update();
}

// Llamar a la función para obtener el uso de Disco inicialmente
getDiskUsageData();

// Actualizar el uso de Disco cada segundo
setInterval(getDiskUsageData, 1000);
```

- **Descripción:** Representa el uso del disco en forma de un gráfico de donut.
- **Funciones:**
    - updateDiskChart(usedDisk, totalDisk): Actualiza el gráfico de donut con el uso de disco.

Parámetros:

- usedDisk: Espacio de disco utilizado.

- totalDisk: Espacio total de disco.

Funcionalidad:

- Calcula el espacio de disco disponible.
- Actualiza los datos del gráfico de donut con el uso de disco.
- Actualiza el gráfico.

    -
    - getDiskUsageData(): Obtiene y muestra datos de uso de disco en el gráfico de donut.

Funcionalidad:

- Construye una URL para obtener datos de uso de disco desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener el uso de disco.
- Llama a updateDiskChart() para actualizar el gráfico con los nuevos datos.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Gráfico de Líneas para Lectura de Disco

```
// Función para agregar datos al gráfico de líneas de la lectura de disco
function addDataToDiskReadChart(timestamp, diskReadUsageMB) {
    // Agregar el punto de datos al conjunto de datos del gráfico
    diskReadChart.data.labels.push(timestamp);
    diskReadChart.data.datasets[0].data.push(diskReadUsageMB);

    // Limitar la cantidad de puntos de datos mostrados en el gráfico para mantenerlo limpio
    const maxDataPoints = 10;
    if (diskReadChart.data.labels.length > maxDataPoints) {
        diskReadChart.data.labels.shift();
        diskReadChart.data.datasets[0].data.shift();
    }

    // Actualizar el gráfico
    diskReadChart.update();
}

/// Función para obtener y mostrar la lectura de disco en el gráfico de líneas
function getDiskReadChartData() {
    const timestamp = new Date();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=disk.sda&after=-1&points=1&
    timestamp=${timestamp}`;

    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener los valores de lectura de disco
        const diskReadUsageKB = parseFloat(data.data[0][1]);
        const diskReadUsageMB = diskReadUsageKB / 1024;

        // Actualizar el gráfico de líneas de la lectura de disco
        addDataToDiskReadChart(timestamp, diskReadUsageMB);
    })
    .catch(error => {
        console.error('Error al obtener la lectura de disco del sistema:', error);
    });
}

// Llamar a la función inicialmente y luego establecer una actualización periódica
getDiskReadChartData(); // Obtener datos inicialmente
setInterval(getDiskReadChartData, 3000); // Actualizar cada 3 segundos
```

- **Descripción:** Representa la velocidad de lectura del disco en forma de un gráfico de líneas.
- **Funciones:**
    - addDataToDiskReadChart(timestamp, diskReadUsageMB): Agrega datos al gráfico de líneas de la lectura de disco.

Parámetros:

- timestamp: Marca de tiempo de la lectura.
- diskReadUsageMB: Velocidad de lectura de disco en megabytes por segundo.

Funcionalidad:

- Agrega el punto de datos al conjunto de datos del gráfico.
- Limita la cantidad de puntos de datos mostrados en el gráfico.
- Actualiza el gráfico.

    - getDiskReadChartData(): Obtiene y muestra datos de lectura de disco en el gráfico de líneas.

Funcionalidad:

- Construye una URL para obtener datos de lectura de disco desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener la velocidad de lectura de disco.
- Llama a addDataToDiskReadChart() para agregar datos al gráfico.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Gráfico de Líneas para Escritura de Disco

```
// Función para agregar datos al gráfico de líneas de la escritura de disco
function addDataToDiskWriteChart(timestamp, diskWriteUsageMB) {
    // Agregar el punto de datos al conjunto de datos del gráfico
    diskWriteChart.data.labels.push(timestamp);
    diskWriteChart.data.datasets[0].data.push(diskWriteUsageMB);

    // Limitar la cantidad de puntos de datos mostrados en el gráfico para mantenerlo limpio
    const maxDataPoints = 10;
    if (diskWriteChart.data.labels.length > maxDataPoints) {
        diskWriteChart.data.labels.shift();
        diskWriteChart.data.datasets[0].data.shift();
    }

    // Actualizar el gráfico
    diskWriteChart.update();
}
// Función para eliminar el signo negativo de los valores de escritura
function eliminarSignoNegativo(valor) {
    return Math.abs(valor);
}

// Ejemplo de uso
let valorEscritura = -20.37715;
let valorEscrituraPositivo = eliminarSignoNegativo(valorEscritura);

// Función para obtener y mostrar la escritura de disco en el gráfico de líneas
function getDiskWriteChartData() {
    const timestamp = new Date();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=disk.sda&after=-1&points=1&
    timestamp=${timestamp}`;

    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener los valores de escritura de disco
        let diskWriteUsageKB = parseFloat(data.data[0][2]);

        // Eliminar el signo negativo si lo hay
        diskWriteUsageKB = eliminarSignoNegativo(diskWriteUsageKB);

        // Convertir a megabytes
        const diskWriteUsageMB = diskWriteUsageKB / 1024;

        // Actualizar el gráfico de líneas de la escritura de disco
        addDataToDiskWriteChart(timestamp, diskWriteUsageMB);
    })
    .catch(error => {
        console.error('Error al obtener la escritura de disco del sistema:', error);
    });
}
```

- **Descripción:** Representa la velocidad de escritura del disco en forma de un gráfico de líneas.
- **Funciones:**
    - addDataToDiskWriteChart(timestamp, diskWriteUsageMB): Agrega datos al gráfico de líneas de la escritura de disco.

Parámetros:

- timestamp: Marca de tiempo de la escritura.
- diskWriteUsageMB: Velocidad de escritura de disco en megabytes por segundo.

Funcionalidad:

- Agrega el punto de datos al conjunto de datos del gráfico.
- Limita la cantidad de puntos de datos mostrados en el gráfico.
- Actualiza el gráfico.

  - getDiskWriteChartData(): Obtiene y muestra datos de escritura de disco en el gráfico de líneas.

Funcionalidad:

- Construye una URL para obtener datos de escritura de disco desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener la velocidad de escritura de disco.
- Llama a addDataToDiskWriteChart() para agregar datos al gráfico.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Gráfico de Líneas para Uso de la Red

```
// Función para agregar datos al gráfico de líneas del uso de la red
function addDataToNetworkChart(timestamp, networkUsageKB) {
    // Agregar el punto de datos al conjunto de datos del gráfico
    networkChart.data.labels.push(timestamp);
    networkChart.data.datasets[0].data.push(networkUsageKB);

    // Limitar la cantidad de puntos de datos mostrados en el gráfico para mantenerlo limpio
    const maxDataPoints = 10;
    if (networkChart.data.labels.length > maxDataPoints) {
        networkChart.data.labels.shift();
        networkChart.data.datasets[0].data.shift();
    }

    // Actualizar el gráfico
    networkChart.update();
}

// Función para obtener y mostrar el uso de la red en el gráfico de líneas
function getNetworkChartData() {
    // Construir la URL de la API de Netdata para obtener datos de uso de la red
    const timestamp = Date.now();
    const apiUrl = `http://localhost:19999/api/v1/data?chart=system.net&after=-1&points=1&
    timestamp=${timestamp}`;

    // Realizar una solicitud GET a la API de Netdata
    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Obtener el valor del uso de la red
        const networkUsageKB = parseFloat(data.data[0][1]) / 1024; // Convertir de B/s a KB/s

        // Llamar a la función para agregar datos al gráfico de líneas del uso de la red
        addDataToNetworkChart(timestamp, networkUsageKB);
    })
    .catch(error => {
        console.error('Error al obtener el uso de la red del sistema:', error);
    });
}

// Llamar a la función inicialmente y luego establecer una actualización periódica
getNetworkChartData(); // Obtener datos inicialmente
setInterval(getNetworkChartData, 3000); // Actualizar cada 3 segundos
```

- **Descripción:** Representa el uso de la red en forma de un gráfico de líneas.
- **Funciones:**
    - addDataToNetworkChart(timestamp, networkUsageKB): Agrega datos al gráfico de líneas del uso de la red.

Parámetros:

- timestamp: Marca de tiempo del uso de la red.
- networkUsageKB: Uso de la red en kilobytes por segundo.

Funcionalidad:

- Agrega el punto de datos al conjunto de datos del gráfico.
- Limita la cantidad de puntos de datos mostrados en el gráfico.
- Actualiza el gráfico.

    - getNetworkChartData(): Obtiene y muestra datos de uso de la red en el gráfico de líneas.

Funcionalidad:

- Construye una URL para obtener datos de uso de la red desde una API.
- Realiza una solicitud GET a la API de Netdata.
- Procesa la respuesta para obtener el uso de la red.
- Llama a addDataToNetworkChart() para agregar datos al gráfico.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

Alertas y Monitoreo de Objetivos

```
// Función para obtener las alertas del Alertmanager
function getAlerts() {
    const apiUrl = 'http://192.168.2.50:9093/api/v2/alerts';

    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Llamar a la función para mostrar las alertas
        displayAlerts(data);
    })
    .catch(error => {
        console.error('Error al obtener las alertas de Alertmanager:', error);
    });
}

// Función para mostrar las alertas en el dashboard
function displayAlerts(alerts) {
    // Verificar si hay alertas y mostrarlas en algún contenedor HTML
    const alertsContainer = document.getElementById('alertsContainer');
    alertsContainer.innerHTML = ''; // Limpiar el contenedor antes de agregar nuevas alertas

    if (alerts && alerts.length > 0) {
        alerts.forEach(alert => {
            // Construir la estructura HTML para mostrar la alerta
            const alertHTML = `
                <div class="alert ${alert.labels.severity.toLowerCase()}">
                    <h3>Nombre de la alerta: ${alert.labels.alertname}</h3>
                    <p>Instancia: ${alert.labels.instance}</p>
                    <p>Severidad: ${alert.labels.severity}</p>
                </div>
            `;
            // Agregar la alerta al contenedor
            alertsContainer.insertAdjacentHTML('beforeend', alertHTML);
        });
    } else {
        // Si no hay alertas, mostrar un mensaje indicando que no hay alertas activas
        alertsContainer.innerHTML = '<p>No hay alertas activas en este momento.</p>';
    }
}

// Llamar a la función para obtener las alertas inicialmente y luego establecer una
actualización periódica
getAlerts(); // Obtener datos inicialmente
setInterval(getAlerts, 5000); // Actualizar cada 5 segundos
```

Alertas

- **Descripción:** Obtiene y muestra alertas del Alertmanager.
- **Funciones:**
  - getAlerts(): Obtiene alertas del Alertmanager.

Funcionalidad:

- Realiza una solicitud GET a la API del Alertmanager para obtener las alertas.
- Procesa la respuesta y llama a displayAlerts() para mostrar las alertas.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

- displayAlerts(alerts): Muestra las alertas en el dashboard.

Monitoreo de Objetivos

```
// Función para obtener los targets y mostrar su estado
function getTargets() {
    const apiUrl = 'http://192.168.2.50:9090/api/v1/targets';

    fetch(apiUrl)
    .then(response => response.json())
    .then(data => {
        // Llamar a la función para mostrar los targets y su estado
        displayTargets(data.data.activeTargets);
    })
    .catch(error => {
        console.error('Error al obtener los targets:', error);
    });
}

// Función para mostrar la lista de targets y su estado
function displayTargets(targets) {
    const targetsListContainer = document.getElementById('targetsList');

    // Limpiar el contenedor antes de agregar los nuevos elementos
    targetsListContainer.innerHTML = '';

    // Recorrer cada target y mostrarlo en la lista
    targets.forEach(target => {
        const targetItem = document.createElement('li');
        const targetIp = target.labels.instance.split(":")[0]; // Obtener solo la dirección IP
        targetItem.textContent = `Target: ${target.labels.instance}, Estado: ${target.health}`;
        targetItem.dataset.url = `http://${targetIp}:80`; // Formar la URL completa con la
        dirección IP y el puerto 80

        // Añadir clases de estilo según el estado del target
        targetItem.classList.add(target.health === 'up' ? 'up' : 'down');

        // Agregar evento de clic para abrir la URL en la misma página
        targetItem.addEventListener('click', () => {
            window.location.href = targetItem.dataset.url;
        });

        targetsListContainer.appendChild(targetItem);
    });
}

// Llamar a la función para obtener los targets inicialmente y luego establecer una
actualización periódica
getTargets(); // Obtener datos inicialmente
setInterval(getTargets, 5000); // Actualizar cada 5 segundos
```

- **Descripción:** Obtiene y muestra el estado de los objetivos de monitoreo.
- **Funciones:**
    - getTargets(): Obtiene el estado de los objetivos de monitoreo.

Funcionalidad:

- Realiza una solicitud GET a la API Prometheus para obtener los targets.

- Procesa la respuesta y llama a displayTargets() para mostrar los targets y su estado.
- Maneja errores en caso de fallo en la solicitud o procesamiento de datos.

    - displayTargets(targets): Muestra los objetivos y su estado en el dashboard.

Interfaz de Usuario

```
// Obtener elementos del DOM
const toggleMenuBtn = document.getElementById('toggleMenu');
const menu = document.getElementById('menu');

// Mostrar el menú al pasar el ratón sobre el botón o el menú
toggleMenuBtn.addEventListener('mouseenter', () => {
    menu.classList.add('show');
});

// Mantener el menú visible al pasar el ratón sobre él
menu.addEventListener('mouseenter', () => {
    menu.classList.add('show');
});

// Ocultar el menú al sacar el ratón del botón o el menú
toggleMenuBtn.addEventListener('mouseleave', () => {
    menu.classList.remove('show');
});

menu.addEventListener('mouseleave', () => {
    menu.classList.remove('show');
});

// Función para obtener la IP de la máquina
function getMachineIP() {
    return "localhost";
}

// Función para actualizar dinámicamente la información de la instancia con la IP de la máquina
function updateInstanceInfo() {
    const instanceIP = getMachineIP(); // Obtener la IP de la máquina
    const instanceInfo = document.getElementById('instanceIP'); // Obtener el elemento span
    donde se mostrará la IP

    // Actualizar el contenido del span con la IP de la máquina
    instanceInfo.textContent = instanceIP;
}

// Llamar a la función para actualizar la información de la instancia inicialmente
updateInstanceInfo();
/////
document.addEventListener('DOMContentLoaded', function() {
    const body = document.querySelector('body');
    const themeToggle = document.querySelector('.theme-toggle');

    themeToggle.addEventListener('click', function() {
        body.classList.toggle('dark-theme');
        body.classList.toggle('light-theme');
    });
});
```

- **Descripción:** Gestiona la interfaz de usuario y la interacción con el usuario.
- **Funciones:**
  - Gestiona eventos de clic para abrir enlaces.
  - Actualiza la información de la instancia con la dirección IP.

- Gestiona el cambio de tema entre claro y oscuro.

Utilidades

- **Descripción:** Proporciona funciones de utilidad reutilizables en el código.
- **Funciones:**
  - eliminarSignoNegativo(valor): Elimina el signo negativo de un valor.

Parámetros:

- valor: Valor numérico que puede ser positivo o negativo.

Funcionalidad:

- Devuelve el valor absoluto del parámetro valor.

```
// Función para eliminar el signo negativo de los valores de escritura
function eliminarSignoNegativo(valor) {
    return Math.abs(valor);
}
```