Documentación Técnica – Escáner de Red en Java

Nombre del alumno: Fabrizio Bruno
Curso: 5to 1ra
Materia: Redes
Escuela Técnica N°36
Tecnología elegida: Java
Fecha: 18/08/2025

1. Propósito del programa

El sistema desarrollado es un **escáner de red gráfico** que permite a un usuario ingresar un rango de direcciones IP (inicio y fin) y verificar qué dispositivos están activos en esa red local.

La aplicación muestra:

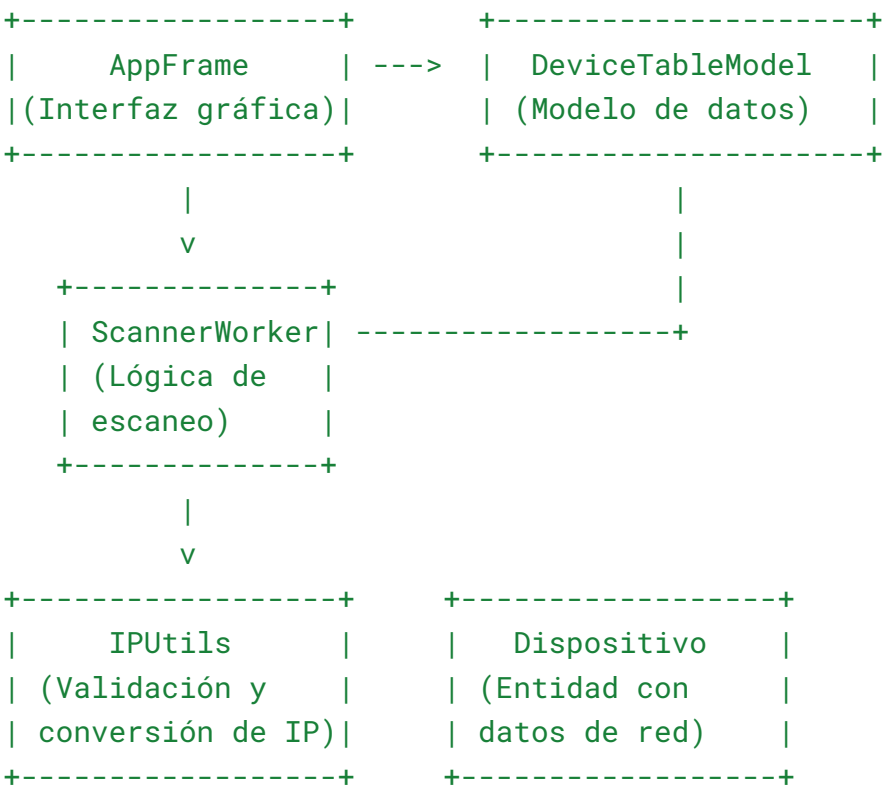
- IP del dispositivo.
- Nombre del host (si está disponible).
- Estado de conexión (disponible o no).
- Tiempo de respuesta en milisegundos.

Este tipo de herramienta es fundamental en entornos de administración de redes, ya que permite detectar dispositivos conectados, validar disponibilidad y tener un diagnóstico rápido del estado de la infraestructura.

2. Arquitectura y armado del sistema

El proyecto está dividido en **módulos/clases con responsabilidades claras**, aplicando un diseño orientado a objetos.

Diagrama de arquitectura



- **AppFrame**: interfaz gráfica principal, construida con Swing.
- **DeviceTableModel**: modelo de tabla personalizado para manejar los datos de los dispositivos detectados.
- **Dispositivo**: clase entidad que representa cada host de la red.
- **IPUtils**: utilidades para validación y transformación de direcciones IP.
- **ScannerWorker**: componente en segundo plano que ejecuta el escaneo (multitarea mediante `SwingWorker`).
- **Main**: clase principal que inicializa la aplicación.

3. Métodos utilizados y su justificación

- **Swing y SwingWorker**: se eligió Swing por ser un framework gráfico integrado en Java, sin dependencias externas. `SwingWorker` permite mantener la interfaz responsiva mientras se ejecuta el escaneo en segundo plano.
- **Regex para validación de IP**: se utilizó `Pattern` y `Matcher` para asegurar que las direcciones IP ingresadas por el usuario sean válidas.

- **Conversión IP ↔ Long**: se emplea para recorrer rangos de direcciones IP de manera eficiente.

- **Ping mediante ProcessBuilder**: se usa para ejecutar el comando del sistema operativo (`ping`) y determinar disponibilidad. Se eligió este enfoque ya que garantiza compatibilidad multiplataforma (Windows/Linux).

- **Resolución de nombres con nslookup**: permite obtener el nombre del host asociado a una IP, facilitando la identificación de dispositivos.

- **Modelo MVC**: aunque no se implementa de forma estricta, se sigue la filosofía de separar la vista (AppFrame), la lógica (ScannerWorker, IPUtils) y los datos (DeviceTableModel, Dispositivo).

4. Tecnologías elegidas y justificación

- **Java SE**: lenguaje multiplataforma, robusto, con bibliotecas estándar para concurrencia, GUI y networking.

- **Swing**: elegido por simplicidad y disponibilidad inmediata sin dependencias externas.

- **Regex en Java**: solución confiable y eficiente para validación de IPs.

- **ProcessBuilder**: alternativa flexible para interactuar con procesos del sistema (ping/nslookup).

Estas elecciones buscan un balance entre **simplicidad de implementación, portabilidad y robustez**.

5. Problemas encontrados y soluciones

1. **Bloqueo de la interfaz durante el escaneo**

    - Problema: si el escaneo se ejecutaba en el mismo hilo que la interfaz, la ventana quedaba congelada.

    - Solución: implementación de `SwingWorker` para manejar el escaneo en un hilo secundario.

2. **Diferencias entre comandos de Windows y Linux**

    - Problema: el comando `ping` utiliza flags diferentes según el sistema operativo.

    - Solución: detección automática del SO con `System.getProperty("os.name")` y construcción del comando adecuado.

3. **IPs inválidas ingresadas por el usuario**
    - Problema: la aplicación fallaba si se ingresaban IPs fuera de rango.
    - Solución: uso de expresiones regulares y validación en la clase `IPUtils`.
4. **Resolución de nombres lenta**
    - Problema: `nslookup` en algunos entornos generaba retrasos.
    - Solución: se estableció un `timeout` controlado con `waitFor`.

6. Posibles mejoras futuras

- **Escaneo paralelo/multihilo**: actualmente se escanean las IPs de manera secuencial, lo que puede ser lento en rangos grandes. Podría mejorarse con un `ExecutorService` o `ForkJoinPool`.
- **Interfaz más moderna (JavaFX)**: migrar de Swing a JavaFX para lograr una UI más intuitiva y atractiva.
- **Detección de puertos abiertos**: ampliar la funcionalidad para incluir un escaneo básico de puertos.
- **Exportación de resultados**: permitir guardar los resultados en archivos CSV o JSON.
- **Integración con SNMP**: para obtener más información sobre los dispositivos conectados.