E.T. Nº 36

ALMIRANTE GUILLERMO BROWN

Redes

TP N°14: Escáner de red: Documentación del Desarrollo

Año: 5º	División: 1º	Turno:	Tarde
Autor:		Sofía Power	
Docente: Oscar Obregón			

Fecha de entrega: 19/08/2025

INTRODUCCION DEL PROYECTO

El presente trabajo consiste en el desarrollo de una herramienta de red destinada al escaneo de direcciones IP, ya sea entre un rango predeterminado o por una dirección DNS.

Ante esto, la aplicación identificará los dispositivos de red encontrados y se los mostrará al usuario como información importante, brindando una solución accesible para el análisis básico de conectividad de una red local o externa.

OBJETIVOS Y FINALIDAD DEL SISTEMA

La aplicación permite hacer comunicaciones de red usando los protocolos **ICMP** (**ping**) y **DNS** con el fin de obtener información de la disponibilidad de los dispositivos. Para eso, el usuario deberá proporcionar el uso de los siguientes parámetros:

- **IP inicial:** Dirección que vamos a utilizar como el comienzo de un rango de equipos de red (ya sea si agregamos una IP final).

- **IP final:** Dirección que va a marcar el límite del rango de búsqueda. En el caso de que solo se quiera calcular una sola dirección, este campo puede dejarse vacío.

- **Dirección DNS:** Direcciones de dominio que sean compatibles. No está permitido el ingreso de direcciones IP, y si se requiere calcular solo este campo las IP inicial y final deben estar vacías. Se agrega este campo de texto

para poder permitir dos direcciones de dominio distintas en la aplicación.

- **Tiempo de espera (timeout):** El usuario puede ingresar el tiempo de espera en milisegundos definido al ejecutar el comando ping. Está configurado de forma predeterminada con un tiempo de 1000 ms, aunque el usuario puede modificarlo según sus necesidades.

Cuando el usuario termine de proporcionar los parámetros requeridos sin ningún error de validación, podrá iniciar el proceso de escaneo presionando el botón "**Escanear**".

Durante la ejecución se desplegará una ventana secundaria con una barra de progreso indicando el **estado de la operación**. Cuando llegue a su fin, todos los resultados encontrados se **mostrarán en la tabla principal** de la aplicación, junto con un cuadro de diálogo que indica la cantidad de equipos de red detectados por la solicitud.

Cada registro en la tabla incluye información como:

- **Dirección IP**
- El **nombre del equipo**
- Estado de **conectividad**
- **Tiempo de espera** calculado en milisegundos mediante el comando ping.

Además, la aplicación ofrece al usuario distintas funcionalidades para gestionar los resultados:

- Eliminar los registros que se muestren en pantalla presionando el botón "**Limpiar**".

- Guardar los resultados, por lo que se permite exportarlos en un archivo, ya sea en formato **CSV** (**Excel**) o en archivo de **texto plano** (.**txt**) con solo presionar el botón "**Guardar**".

- **Ordenar los registros** de menor a mayor utilizando cualquiera de las columnas disponibles.

- **Filtrar los datos** mostrando únicamente los dispositivos con conectividad, los que no respondieron, o todos los resultados disponibles.

De esta manera, la aplicación ofrece un conjunto de herramientas flexibles para el análisis y la organización de la información obtenida, ya sea mediante un rango de direcciones IP o por el uso de dominios específicos.

COMO ESTA ARMADO EL SISTEMA

El sistema fue organizado siguiendo el patrón de diseño Modelo-VistaControlador (MVC), por lo que permite una separación clara entre la lógica de información, la interfaz gráfica y la gestión de interacción con el usuario.

La organización del código se divide en diferentes paquetes:

- **Paquete "main":** Contiene la clase principal ***Main.java***, la cual se encarga de iniciar la aplicación y de que comience el controlador.

- **Paquete "controlador":** Incluye la clase ***Controlador.java***, que va a funcionar como un intermediario entre la interfaz gráfica (vistas) y la sección lógica de la aplicación (modelo). Administra las interacciones del usuario y coordina la ejecución de los procesos de red.

- **Paquete "modelo":** Aquí se encuentran todas las clases que van a servir para la lógica de datos del sistema:

  - ***ComienzoScanner.java***: Esta clase se encarga de conservar las clases principales que van a servir

para la ejecución del programa (validar IP, ping, nslookup, escaneo en un rango y más).

- ***listaEquiposRed.java***: Estructura que nos va a permitir guardar todos los resultados en una lista interna. Contiene funciones que se encargan de agregar y eliminar datos, y devolver los datos almacenados.

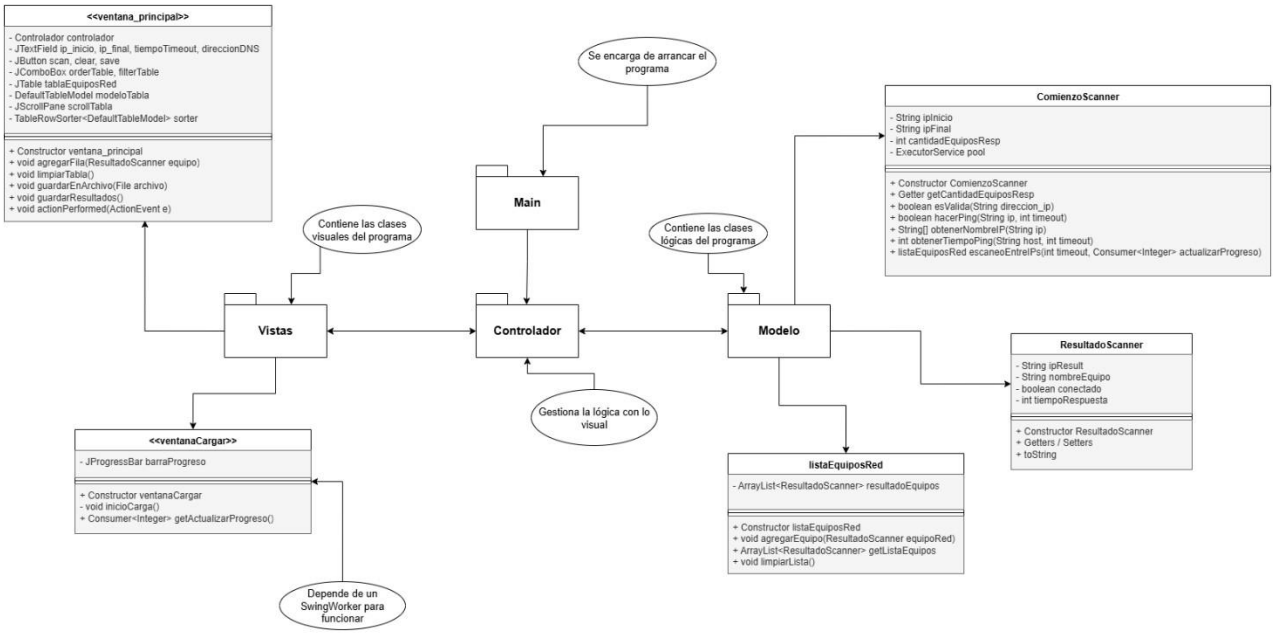
- ***ResultadoScanner.java***: Se encarga de obtener una representación individual de cada dispositivo encontrado en la red. De esta forma se van a poder agregar a la lista de resultados de a poco.

- **Paquete "vistas":** Contiene las clases que corresponden a la interfaz gráfica:

- ***ventana_principal.java***: Es la vista principal que permite al usuario ingresar parámetros y visualizar los resultados.

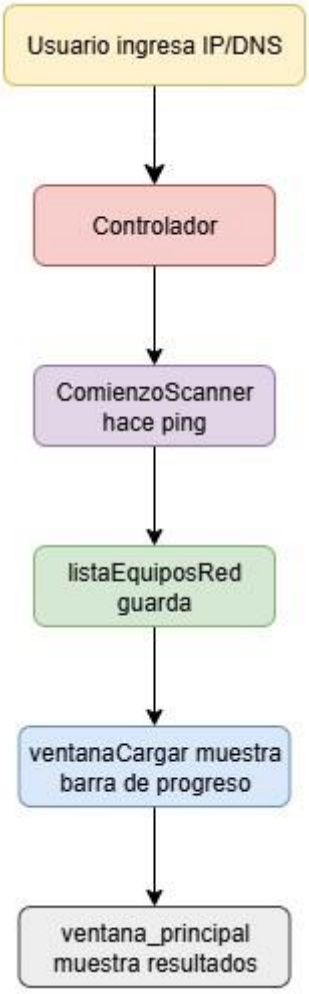
- ***ventanaCargar.java***: Ventana secundaria que contiene la barra de progreso, indicando el estado del escaneo.

Diagrama UML de la organización del sistema:



En este diagrama UML están todos los paquetes ordenados uno por uno, cada uno con sus clases importantes. Por ejemplo, se muestran las dos clases visuales en el paquete Vistas (a la izquierda), y por la derecha están las clases lógicas en el paquete Modelo (a la derecha).

Procedimiento del programa:



METODOLOGIAS UTILIZADAS

En esta aplicación se utilizaron distintas metodologías y buenas prácticas que facilitaron la organización del sistema y su implementación. En este apartado se van a explicar algunos de ellos:

Primero lo que se hizo fue organizar una estructura de paquetes que dividen la parte lógica (modelo), la parte visual (vistas) y la parte que conecta ambos paquetes de clases (controlador). Luego se organizó cómo sería la vista principal de la aplicación en un boceto que sirve como "primera estructura", y se comenzó a dividir en funciones cada clase específica, ya sea con conocimientos previos de Programación Orientada a Objetos (POO) y con otras investigaciones, ya sea por la web o mediante la Inteligencia Artificial.

Ya con toda la estructura organizada, se comenzaron a realizar otras prácticas, como el manejo de excepciones para capturar errores de red o direcciones inválidas, la documentación del código con comentarios y la separación entre la parte lógica con la interfaz gráfica, evitando sobrecargar una clase.

¿POR QUE SE ELIGIERON CIERTAS TECNOLOGIAS?

En esta sección se van a explicar tanto el lenguaje de programación como las librerías elegidas para la creación de esta aplicación.

Esta aplicación fue creada con el lenguaje **Java**, ya que tiene una programación orientada a objetos que es más simple y entendible para crear una aplicación, o al menos ese es mi caso, ya que lo estuvimos utilizando últimamente.
Además, se eligió esta tecnología ya que otorga librerías de red para calcular los comandos requeridos para el objetivo principal, y porque se puede crear una interfaz gráfica rápida y completa.

Para comprender mejor las librerías utilizadas, se nombrarán a continuación:

- **Swing**: Es importante en este proyecto ya que otorga los componentes necesarios para crear una interfaz gráfica de usuario (**GUI**), que es en pocas palabras, la parte visual de la aplicación. También se utilizó **Swing.table** para desarrollar el modelo de la tabla principal. Además, fue de utilidad para mostrar recuadros de información **JOptionPane** para mostrar errores o si se cumplieron las operaciones, y hace posible ejecutar operaciones mediante hilos (**SwingUtilities**).

- **AWT**: Esta tecnología de Java otorga al código un conjunto de herramientas que van a servir para **configurar la GUI**, como botones, etiquetas, cuadros de texto y más. Gracias a esta también se implementó un sistema de eventos para manejar mejor las interacciones del usuario (pulsar botones o teclas y más), los cuales generan eventos procesados en “listeners”.

- **IO**: Este paquete es importante, ya que sirvió para realizar operaciones de entrada y de salida, que en este caso fue de utilidad para guardar los resultados mostrados en un archivo interno del dispositivo. Además, identifica los errores de entrada y salida que vayan a ocurrir.

- **Util**: Esta otra librería utilizada sirvió principalmente para proporcionar **un conjunto de clases e interfaces** que ofrecen funcionalidades esenciales para desarrollar aplicaciones en Java. De esta forma ofrece colecciones de datos (**ArrayList y List**), manejo de hilos con sincronización y ejecución de tareas (**Concurrent**), manejar números enteros de forma segura en hilos, y utilizar interfaces funcionales para operaciones comunes y expresiones lambda.

- **NET**: Este paquete proporciona clases para **implementar distintas aplicaciones de red**, dividido en dos secciones principales: una **API de bajo nivel** que maneja direcciones (como direcciones IP) y **sockets** para la comunicación bidireccional. Las clases importadas para hacer posible este trabajo fueron **InetAddress**, para usar el protocolo IP correctamente, y **UnknownHostException**, importante para indicar cuando una dirección IP del host no fue encontrada.

Gracias a todas estas librerías de Java, fue posible la implementación de comandos, operaciones de almacenamiento internas y más, para que de esa forma la aplicación pueda cumplir con sus objetivos tal como se mencionó anteriormente.

PROBLEMAS ENCONTRADOS Y SUS SOLUCIONES

En la organización de este proyecto, hubo varios errores e inconvenientes que fueron deteniendo el flujo de información y también al exportar la aplicación. En este apartado voy a explicar algunas de estas interrupciones que hubo al codificar la aplicación:

1- Diferenciar una dirección IP con una DNS

Este caso fue uno de los más rápidos de resolver, ya que una de sus soluciones fue crear dos funciones distintas en el controlador del programa, siendo una para comenzar el escaneo entre dos direcciones IP y una para escanear la dirección DNS. Pero el problema estuvo en que, como se podían ingresar ambas direcciones en el campo de IP inicial, hubo confusiones para saber si era necesario realizar esto. Por eso la segunda solución fue agregar otro campo de texto justamente indicado únicamente para ingresar por pantalla una dirección DNS.

Otra solución fue que, si no se ingresaba ninguna dirección en ningún campo, se mostrara por pantalla la información del equipo de red local, representando al equipo que utiliza el usuario (localhost).

2- Problemas de coordinación con la barra de progreso

Para esta sección tuve algunas dudas pequeñas que tienen que ver con el orden de procedimientos para agregar la información. Al principio pensé que, mientras se mostraba por pantalla la ventana secundaria (clase ventanaCargar), que se vayan cargando los datos a la tabla

principal, mostrándole al usuario cada resultado calculado en tiempo real, uno por uno. Pero no pude resolver esto porque, como se utilizaban muchos hilos simultáneos, la aplicación se fue trabando poco a poco para que al final no se mostraran los resultados después de un tiempo.

Este inconveniente se pudo resolver haciendo el proceso por partes: Primero se comprueban las direcciones IP, luego aparece la barra de proceso calculando el tiempo de espera para visualizar los resultados, y por último se muestran por pantalla la información de cada uno de los resultados calculados.

3- Problemas de comprobación de la ArrayList de resultados

Este pequeño error se trataba de calcular una de las primeras partes lógicas y probar mostrarlas por consola, antes de hacerlo en la tabla. Pero el tema estaba en que, aunque la lista estaba creada perfectamente, en consola sólo me aparecía una vinculación que mencionaba el directorio que estaba creada la ArrayList.

Para solucionar este problema se creó el toString en la clase ResultadoScanner, así se indicaba por consola la representación de un objeto de forma legible. De esta forma se obtuvo el IP, el nombre del equipo, el estado de conectividad y el tiempo de respuesta de ese objeto, sin ningún otro inconveniente.

4- Estilo de la aplicación principal

Este fue un inconveniente muy común para armar la estructura del proyecto, ya que para armar el estilo de la interfaz gráfica se fue diferenciando en columnas y tablas, y en cada una se fueron agregando los componentes. Mi problema fue que no era una interfaz amigable para el usuario al principio; Una de las primeras ideas fue mostrar la información en otra ventana secundaria que aparezca cuando la barra de progreso termine. Pero se terminó descartando porque no pude figurar la posibilidad para hacer eso.

Además tuve problemas mínimos de desorden al usar un

GridBagLayout y GridBagConstraints, pero se logró solucionar. Ahora la aplicación cumple con una interfaz que puede resultar amigable para el usuario.

COSAS PARA MEJORAR EN UN FUTURO

En este último apartado de la documentación se explicarán cosas a fondo que no se lograron emplear en la aplicación, pero que pueden actualizarse en un futuro para que de esta forma sea más completo y seguro.

Una de las ideas principales sería poder tener menos tiempo de espera para mostrar algunos resultados en la tabla. Por ejemplo, si ponemos dos IP con un valor más largo para calcular un rango de dispositivos (como 127.0.0.1 y 127.0.0.35), cuando termine la barra del progreso no se van a mostrar los resultados al momento, sino después de 3 o 5 segundos. Esta posible mejora puede servir para evitar confusiones de cálculo en el usuario, y para que ese proceso se realice sin agregar un tiempo adicional.

Otra cosa para recalcar es que se espera mejorar la precisión del progreso de escaneo, ya que a veces la barra que va del 0% al 100% se traba y vuelve para atrás o para adelante, como si repitiera los números. Ante esto la mejora esperada es que, por la ejecución de hilos, no haya ningún inconveniente ni confusión.

Ante esto otra posible mejora es poder mencionar los paquetes enviados a dicha dirección específica, haciendo que suceda con alguna función que los indique. De esa forma, el usuario sabrá si todos los paquetes fueron enviados correctamente a su destinatario sin ninguna pérdida de información en el camino. Igualmente esto es una posible idea que se puede mostrar como información adicional, además de la cantidad de equipos que respondieron a solicitudes de ping.

Por último, una posible mejora en el futuro puede ser emplear un rango de direcciones IP que llegue a calcularse gracias a más dígitos. Por ejemplo, en este sistema se utilizan solo para el último dígito, como por ejemplo, si hacemos ping desde la dirección 8.8.8.8 hasta la dirección

127.0.0.11, el rango de dispositivos sería de 8 a 11 pero calculándolos mediante la IP inicial (o sea que se calculan 8.8.8.8, 8.8.8.9, etc.). La idea sería que se pueda pasar tanto al tercer dígito como al segundo, pero que eso sea a elección del usuario.

Ante esto puede haber una opción en la aplicación que sea directamente para el usuario, de que, si quiere calcular los equipos de red en un rango de números, puede decidir esto con una opción para elegir un dígito específico de ambas direcciones IP, para poner un principio y final. Pero esto es una idea extra para hacer más completa nuestra aplicación.

CONCLUSIÓN DEL PROYECTO

En este proyecto se utilizaron muchas prácticas, tanto visuales como lógicas del programa, pero se pudo implementar una buena herramienta práctica para poder realizar un escaneo de redes gracias a los protocolos ICMP y DNS. Este sistema en general cumple con su objetivo principal de detectar los dispositivos que estén disponibles y luego brindarle esta información al usuario en tiempo real. Además, se lograron aplicar varios conceptos de la programación orientada a objetos, ya sea el manejo de hilos y la separación de clases mediante el patrón MVC.

Este trabajo se abre a una gran posibilidad de mejoras a futuro respecto a rendimiento y utilidad del usuario, permitiendo que la aplicación tenga una base sólida para un análisis de redes.