Seknet

PROYECTO FIN DE GRADO

2023

ÁLVARO HERNANTES ANTON
DANIEL IONUT FECHETE
HECTOR TABLADO SANCHEZ

1. INTRODUCCIÓN

1.1 Motivación

Actualmente con el desarrollo e integración de nuevos dispositivos inteligentes conectados a nuestras redes Wifi en los hogares (IoT, Internet Of Things) se está produciendo una transformación en la manera de interactuar y vivir.

La utilización de estos aparatos para todo tipo de tareas ya sea conocer información sobre la temperatura y la humedad del ambiente, controlar robots de limpieza, o interactuar con los dispositivos multimedia y de audio, revelan la necesidad de disponer de una aplicación capaz de auditar nuestra propia red localizando posibles peligros desde el punto de vista de la seguridad, evaluando el rendimiento o posibles problemas de conectividad o incluso registrando los dispositivos conectados a él.

De este modo se tiene toda la información de lo que sucede dentro de la red privada.

1.2 Objetivos

1. Estudiar Kotlin como nuevo estándar para desarrollo de app Android.
2. Diseñar una arquitectura de programación basada en MVVM.
3. Gestionar el proyecto de acuerdo con la metodología Agile.

Para la consecución de estos objetivos se ha creado una aplicación con las siguientes funcionalidades:.

1.2.1. Información general del dispositivo: buscándose la información básica relativa al router al que se esté conectando en ese momento.

1.2.2. PortScan: donde se analizan los puertos abiertos.

1.2.3. Análisis de red: se mostrarán parámetros de red relacionados con exposiciones de riesgo.

1.2.4. SpeedTest: mostrará la velocidad máxima de subida y bajada de la red.

1.2.5. Ping: de una red, se da la medida en ms y se ha permitido que el usuario decida el número de peticiones.

Los datos mostrados serán siempre instantáneos y, al no ser almacenados, no podrán ser consultados en accesos posteriores ni en otros dispositivos que no sea el de uso actual.

2. PALABRAS CLAVE

Activity: representa una pantalla con la cual el usuario interactúa. Sirve como punto de entrada y salida para la interacción del usuario, gestionando la interfaz de usuario y el ciclo de vida de la pantalla. Cada Activity está asociada a una clase que define su comportamiento y puede contener elementos como botones, campos de texto, imágenes y otros componentes visuales.

Fragment: representa una parte reutilizable de la IU de una app. Un fragmento define y administra su propio diseño, tiene su propio ciclo de vida y puede administrar sus propios eventos de entrada. Los fragmentos no pueden existir por sí solos. Deben estar *alojados* por una actividad u otro fragmento.

ViewBinding: es una vinculación con las vistas que contienen ID, enlazándolas mediante la creación de una clase de vinculación. En general, y en el caso de este trabajo, sustituyen de manera efectiva a FindViewById.

Ping: El ping es una utilidad de red para verificar la conectividad entre dispositivos en una red IP. Envía un paquete de solicitud y espera una respuesta para medir el tiempo de ida y vuelta (RTT). Es útil para diagnosticar problemas de conectividad y evaluar la calidad de la red.

Scanner_de puertos: Un escáner de puertos es una herramienta para detectar puertos abiertos en un dispositivo o red. Ayuda a identificar servicios activos y vulnerabilidades de seguridad. Envía solicitudes y analiza respuestas para determinar el estado de los puertos.

Snackbar: Un escáner de puertos es una herramienta para detectar puertos abiertos en un dispositivo o red. Ayuda a identificar servicios activos y vulnerabilidades de seguridad. Envía solicitudes y analiza respuestas para determinar el estado de los puertos.

Corrutina: Una corrutina es una estructura de programación concurrente que permite la ejecución asíncrona de tareas sin bloquear el hilo principal, permitiendo la concurrencia de aplicaciones Android y mejorando el rendimiento.

3. ÍNDICE GENERAL

4. MÓDULOS FORMATIVOS APLICADOS

A lo largo del desarrollo del proyecto se hará uso de las competencias adquiridas en los siguientes módulos:

Programación multimedia y dispositivos móviles:

- Entornos de desarrollo y emuladores en Android (1.3): Comprende todo lo relacionado con los IDEs que se utilizan y la ejecución de las pruebas y test desde emuladores o dispositivos físicos.
- Compilación y Ejecución (1.7): Procedimientos para realizar el "Build" de la APK y realizar "Debug" para depurar comportamientos no esperados en tiempo de ejecución.

Desarrollo de interfaces:

- Interfaces de usuario multiplataforma (2.1): Desarrollo de las UI con las que interactúan los usuarios de la aplicación desde cualquier dispositivo.
- Usabilidad (5.1)
- UX Medida de usabilidad de aplicaciones (5.2): Conjunto de normas de usabilidad y mejora de la experiencia de usuario que permiten una óptima utilización de la aplicación desde el punto de vista de la usabilidad y la accesibilidad.

Sistemas informáticos:

- Componentes y protocolos de una red (1.4): Conocimientos necesarios para realizar la implementación de los protocolos de red necesarios para proveer de funcionalidad a nuestra aplicación, en nuestro caso concreto mediante el uso de "RetroFit".

5. HERRAMIENTAS Y LENGUAJES

5.1 Herramienta DAFO:

Es una aplicación gratuita dependiente del Ministerio de industria, comercio y turismo. Permite realizar matrices DAFO y los análisis derivados de ella.

5.2 Moqups:

Es una aplicación gratuita online. Permite realizar todo tipo de diagramas ofreciendo diferentes plantillas para facilitar la tarea. Se ha utilizado tanto para hacer diagramas UML como diagramas de gant.

5.3 Material Designs

Es una librería de diseño de android. Permite realizar diseños interactivos de la interfaz de una app, pudiendo importar imágenes y otros recursos para personalizar aún más los diseños.

5.4 Kotlin

Es un lenguaje de programación estático de código abierto que admite la programación funcional y orientada a objetos. Proporciona una sintaxis y conceptos similares a los de otros lenguajes, como C#, Java y Scala, entre muchos otros.. Cuenta con variantes que se orientan a la JVM (Kotlin/JVM), JavaScript (Kotlin/JS) y el código nativo (Kotlin/Native). Por último cabe destacar que ha sido elegido por Google como el lenguaje estándar para desarrollar aplicaciones Android, razón por la cual se elije para el desarrollo de esta aplicación.

Algunas de sus características son:

Interoperabilidad con Java: Kotlin es un lenguaje que puede interoperar con Java, el lenguaje de programación en el que se basa el sistema operativo Android. Esto significa que se pueden utilizar todas las bibliotecas, herramientas y frameworks existentes en Java, lo que amplía el ecosistema disponible para desarrollar aplicaciones de Android.

Seguridad del tipo de datos: Kotlin es un lenguaje con un sistema de tipos estáticos que detecta errores de tipo en tiempo de compilación. Esto evita errores en tiempo de

ejecución que pueden causar problemas en las aplicaciones, como NullPointerExceptions y ClassCastExceptions.

Multiplataforma: si bien para el desarrollo de este proyecto no es realmente importante, sí es cierto que permitiría con relativa facilidad crear proyectos para ecosistemas i0S web y desktop.

Para el desarrollo de la aplicación se han utilizado las siguientes librerías:

android.os.Bundle: Esta librería proporciona la clase Bundle, que se utiliza para pasar datos entre actividades y fragmentos.

androidx.appcompat.app.AppCompatActivity: Proporciona características y funcionalidades adicionales en comparación con la clase base Activity estándar, como la compatibilidad con las barras de acción (ActionBar) y las versiones anteriores de Android.

androidx.drawerlayout.widget.DrawerLayout: Esta librería proporciona la clase DrawerLayout, que es un contenedor de diseño que permite mostrar un cajón de navegación (menú lateral) en una aplicación de Android. Se utiliza en combinación con otros componentes de interfaz de usuario, como la barra de acción y los fragmentos, para crear una experiencia de navegación intuitiva y fácil de usar.

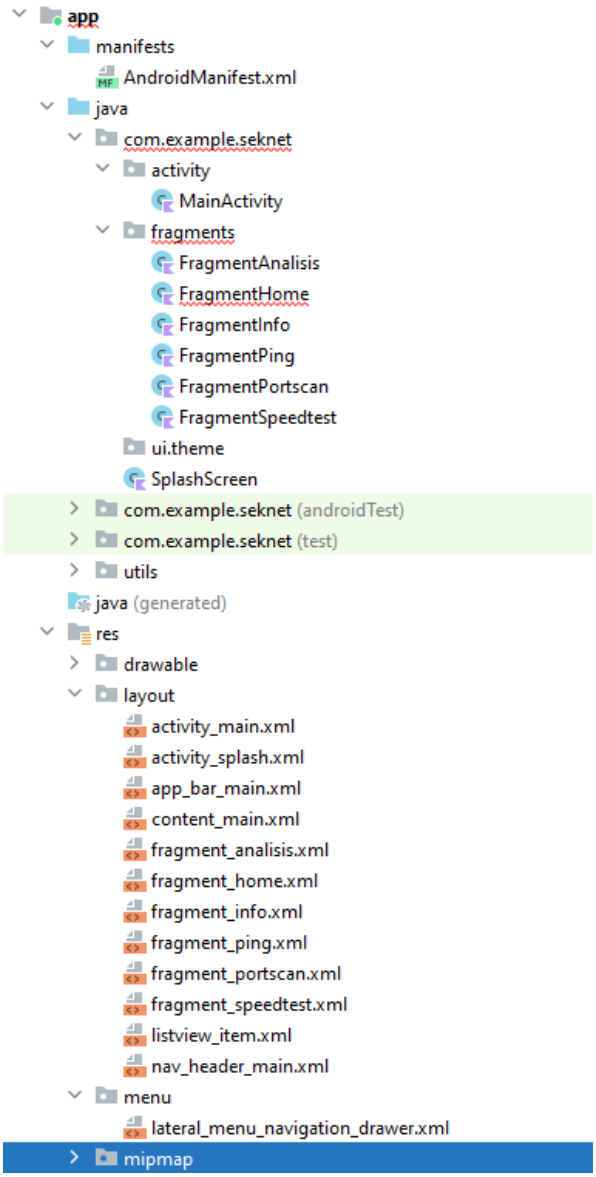
androidx.fragment.app.add y **androidx.fragment.app.commit**: Estas librerías proporcionan métodos para realizar operaciones de fragmentos, como agregar, reemplazar o quitar fragmentos en un contenedor de fragmentos.

5.5 Android Studio

Es el entorno oficial de desarrollo integrado (IDE) que se usa para el desarrollo de aplicaciones Android. Está basado en el editor de código IntlliJ IDEA, pero ofrece más funciones que aumentan la productividad, como un sistema de compilación flexible basado en Gradle, un emulador rápido y con múltiples funciones, un entorno unificado para todos los sistemas Android, una aplicación para insertar cambios de código y recursos en la aplicación en ejecución sin reiniciarla, integración con GitHub, herramientas de marcos de trabajo y herramientas de prueba entre otras muchas.

Todos los proyectos de Android tienen una estructura de archivos común basada en los siguientes módulos:

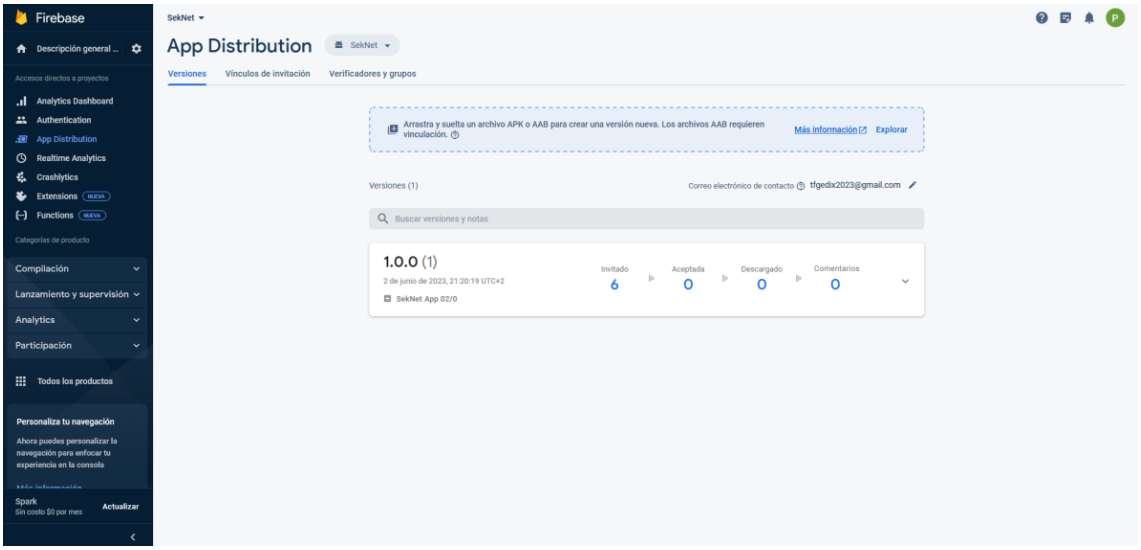
- manifest: contiene el archivo AndroidManifest.xml
- java: contiene los archivos de código fuente (Kotlin).
- res: contiene recursos sin código (diseños XML, imágenes…)



5.6 Firebase

Es una plataforma que permite el desarrollo de aplicaciones web y móvil. Su funcionalidad se divide en 3 componentes:

- Distribución: Firebase App Distribution facilita la distribución de las apps a verificadores de confianza. Si las apps llegan a los dispositivos de los verificadores con rapidez, se puede obtener un feedback más ágil..

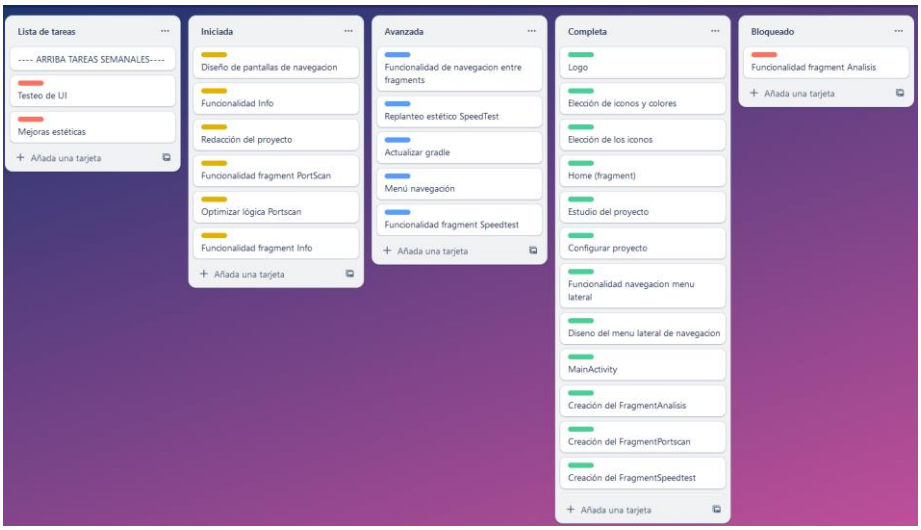


- Desarrollo: es una base de datos no SQL en tiempo real y como sistema de autenticación de usuarios, aportando seguridad y simplicidad en la gestión del registro y autenticación. Además, tiene otras funciones importantes como por ejemplo con el uso de Crashlytics, que genera métricas en tiempo real de los errores de producción. Ayuda así a priorizar los errores más importantes en función del impacto que generan en usuarios reales.
- Notificaciones: no se ha usado, pero FireBase permite informar al usuario de eventos en forma de notificaciones push.
- Analítica y monetización: otra de las componentes no usadas. Permite obtener mediciones y realizar análisis tanto de ventas (ecommerce) como de otros eventos relacionados con las interacciones de los usuarios.

5.7 Trello

Es una herramienta tipo kanban que permite gestionar proyectos y flujos de trabajo. Se organiza en forma de tableros donde se pueden mantener organizadas las tareas mediante columnas. Las tareas se describen en tarjetas que se pueden ir arrastrando entre diferentes columnas según vaya avanzando su estado en la producción (columna pendiente, en proceso o hecha, por ejemplo).

Además, implementa otras funciones útiles como la posibilidad de asociar checklists o archivos adjuntos a las tareas, facilitando la descripción en detalle de las mismas y la realización de las mismas.

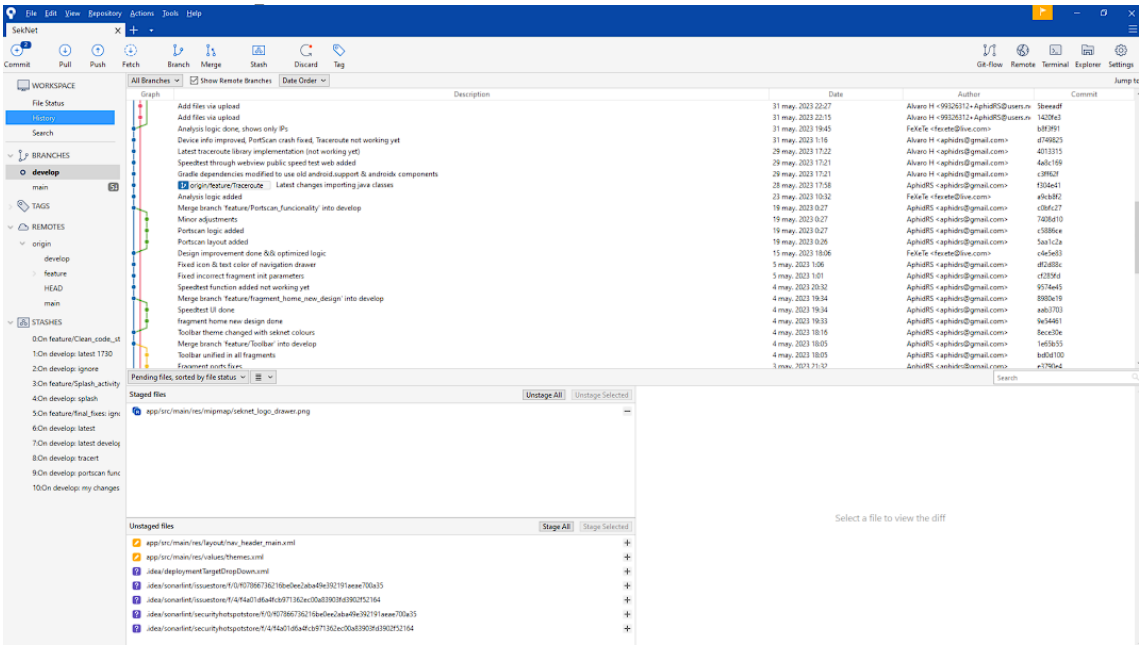

Lista de tareas
---- ARRIBA TAREAS SEMANALES----
Testeo de UI
Mejoras estéticas
Añada una tarjeta
Iniciada
Diseño de pantallas de navegacion
Funcionalidad Info
Redacción del proyecto
Funcionalidad fragment PortScan
Optimizar lógica Portscan
Funcionalidad fragment Info
Añada una tarjeta
Avanzada
Funcionalidad de navegacion entre fragments
Replanteo estético SpeedTest
Actualizar gradle
Menú navegación
Funcionalidad fragment Speedtest
Añada una tarjeta
Completa
Logo
Elección de iconos y colores
Elección de los iconos
Home (fragment)
Estudio del proyecto
Configurar proyecto
Funcionalidad navegacion menu lateral
Diseno del menu lateral de navegacion
MainActivity
Creación del FragmentAnalisis
Creación del FragmentPortscan
Creación del FragmentSpeedtest
Añada una tarjeta
Bloqueado
Funcionalidad fragment Analisis
Añada una tarjeta

5.8 GitHub y SourceTree

Es un portal para alojar código de creadores. Utiliza un sistema de control de versiones Git, lo que permite mantener un control efectivo de cada uno de los cambios implementados en el código.

La forma en la que se ha trabajado con el repositorio ha sido mediante SourceTree, que es un cliente GUI que permite realizar todo tipo de acciones básicas como:

- Crear y clonar repositorios.
- Commit, push, pull y merge de archivos.
- Detectar y resolver conflictos.
- Consultar historial de cambios del repositorio.



Interfaz SourceTree

6. COMPONENTES DEL EQUIPO Y APORTACIONES

El grupo se organizó de acuerdo al equipo típico de un equipo ágil:

Daniel Ionut

Scrum Master: realizándo además tareas de investigación de utilidades y librerías.

Álvaro Hernantes

Porduct Owner: realizando tareas de backlog, investigación y desarrollo.

Héctor Tablado

Desarrollador: realizando tareas de desarrollo.

7. FASES DEL PROYECTO

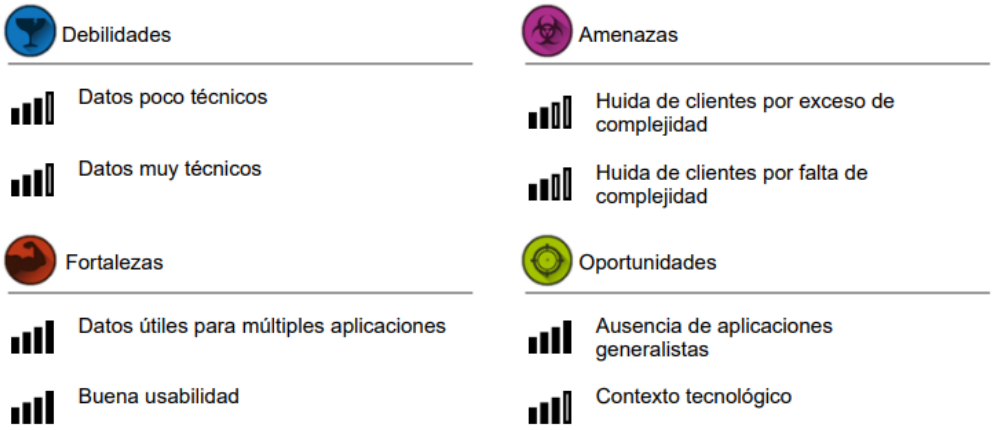
7.1 Análisis

Esta aplicación se ha concebido esencialmente como un producto generalista y está dirigida a un público sin conocimientos extensos del uso de la consola de mandos. Aun así se ha tenido en cuenta que debería poder ser útil para usuarios interesados en obtener de una manera sencilla información útil y de una cierta complejidad de su red.

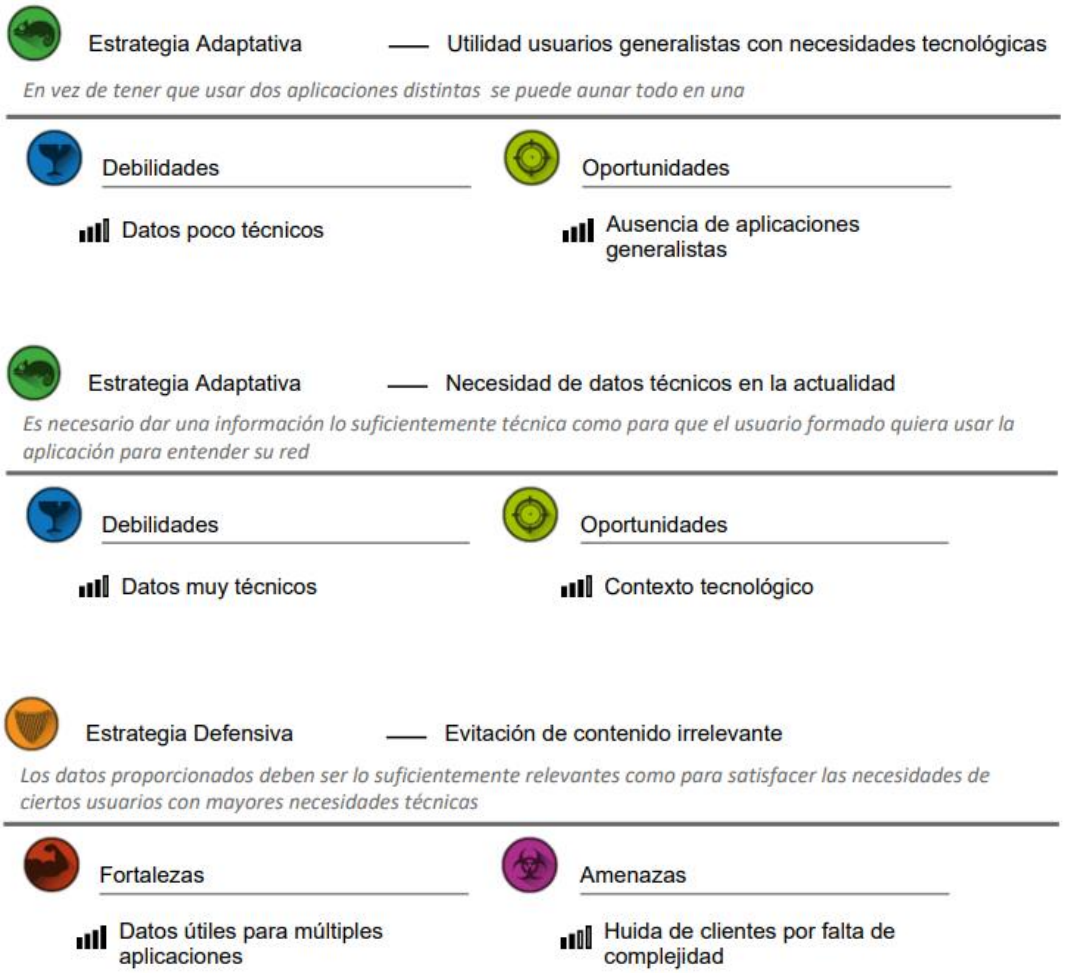
Si bien en el mercado ya hay muchos productos que realizan estas funciones, no son muchos los que realizan todos los análisis. Entre las aplicaciones comerciales más utilizadas se encuentran:

- Speedtest: da información muy detallada de ciertos parámetros como la velocidad de bajada, subida y ping. Ofrece otros servicios como la capacidad de la red para hacer video streaming, mapa de cobertura 5G, distintos servidores de medición, servicio de VPN o monitorización del tráfico de las app instaladas.
- nPerf: una aplicación en línea con la anterior, pero con una interfaz un poco más complicada.
- PingTools Network Utilities: escanea la red en busca de IP's activas, da información detallada sobre la red, conexiones y configuraciones del router. Además escanea puertos y calcula las DNS más rápidas. Carece de información acerca de la velocidad de la red o ping.
- Network Analyzer: permite conocer bandas de conexión de wifi, potencia de la señal wifi, búsquedas de servidores DNS, escaneo de puertos… pero tampoco ofrece información acerca de la velocidad de la red.

Estos son solo algunos de los ejemplos que se encontraron, pero en conjunto permite realizar la siguiente matriz DAFO



A través de ella se pueden extraer las diferentes estrategias para crear una aplicación con impacto en el usuario final:



Estrategia Defensiva — Evitar exceso de pasos para conseguir datos

Aplicaciones muy complejas en su uso terminan expulsando a los usuarios

Fortalezas	Amenazas
Buena usabilidad	Huida de clientes por exceso de complejidad

Estrategia Ofensiva — Ausencia de generalistas

No hay en el mercado aplicaciones generalistas para público poco especializado y bastante especializado

Fortalezas	Oportunidades
Datos útiles para múltiples aplicaciones	Ausencia de aplicaciones generalistas

Estrategia Ofensiva — Rápida y completa

Una buena usabilidad en el contexto tecnológico actual es necesaria, ya que de esta forma las app's se integran mejor en el uso cotidiano

Fortalezas	Oportunidades
Buena usabilidad	Contexto tecnológico

Estrategia Supervivencia — Importancia datos complejos

Los datos deben ser lo suficientemente complejos como para que usuarios más enfocados a las tecnologías los consideren útiles

Debilidades	Amenazas
Datos poco técnicos	Huida de clientes por falta de complejidad

Estrategia Supervivencia — Importancia datos sencillos

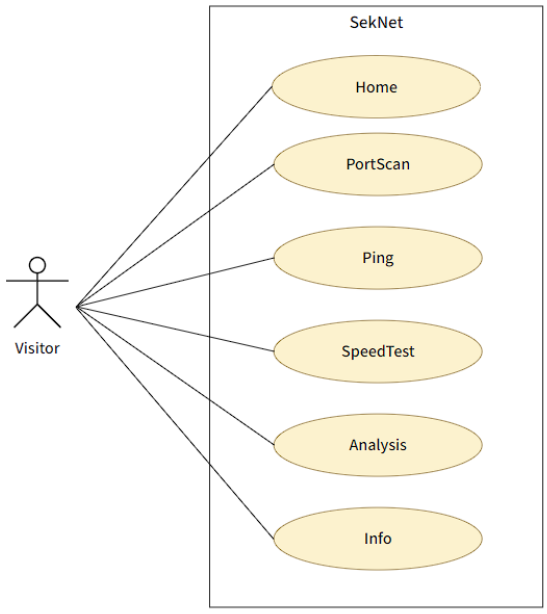
Los datos proporcionados deben ser de uso común y fáciles de encontrar

Debilidades	Amenazas
Datos muy técnicos	Huida de clientes por exceso de complejidad

Gracias al análisis de la matriz DAFO se idean una serie de funciones mínimas que habrá de proporcionar la aplicación:

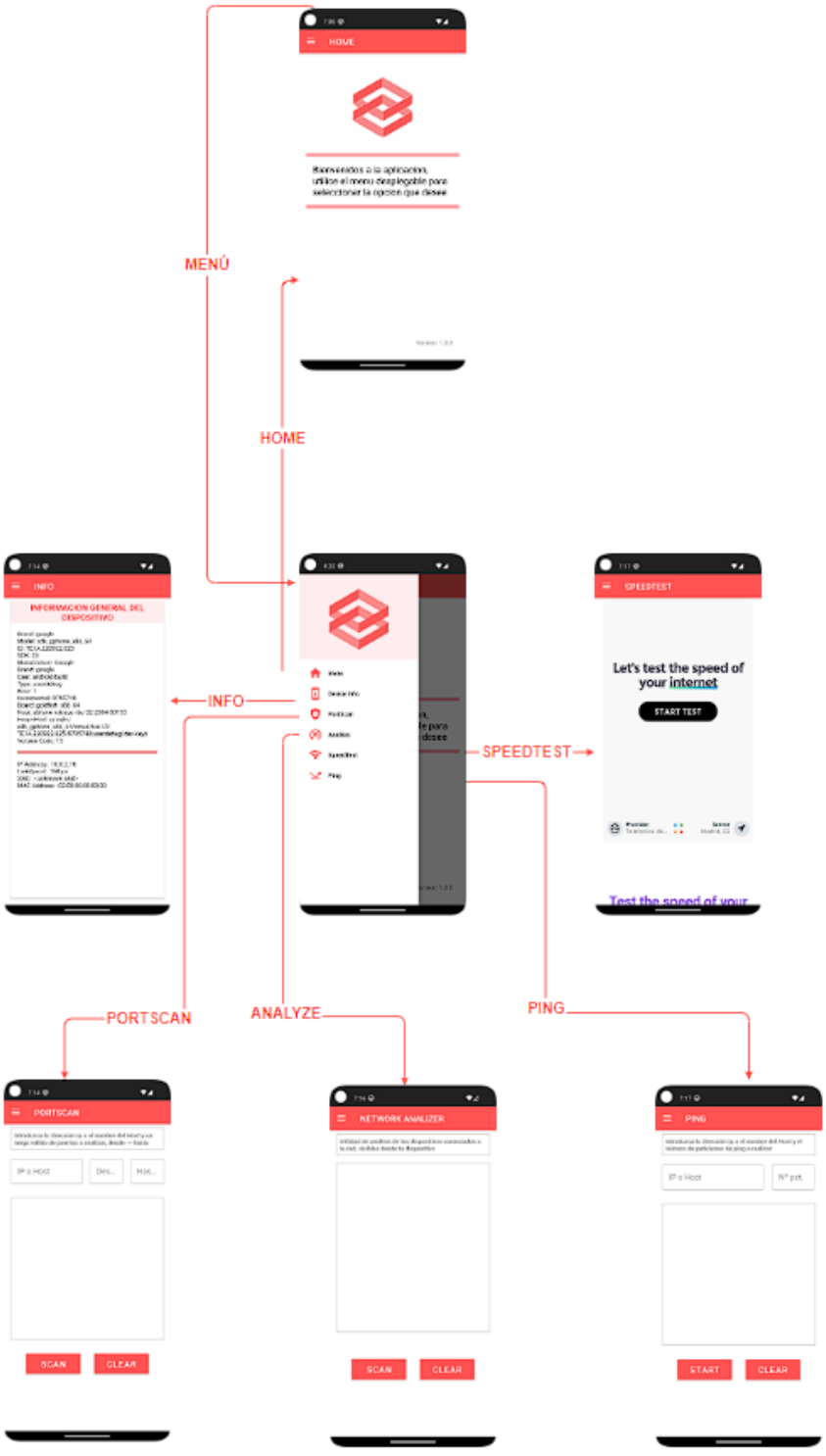
- Información del dispositivo: se darán los siguientes datos: marca, modelo, ID, SDK, fabricante, propietario del dispositivo, tipo de compilación del sistema (versión del usuario final o de ingeniería), número de versión base del sitema Android, versión del sistema, nombre de la placa base del dispositivo, nombre del host en el que se generó la compilación del sistema, huella digital y la versión en la que se está ejecutando la aplicación.
- Escaneo de puertos: permitirá elegir un host en el que hacer el escaneo y el rango de puertos en el que hacerlo.
- Análisis: búsqueda de dispositivos conectados a una red mostrando sus direcciones IP.
- Test de velocidad: da información de subida y bajada, de ping, jitter y datos de la transacción.
- Ping: proporciona el número de paquetes transimitos, porcentaje de pérdida y ping.
- Home: desde aquí se distribuiría el flujo de trabajo.

Tras concretar qué debe hacer la aplicación se puede elaborar el diagrama de casos de uso:



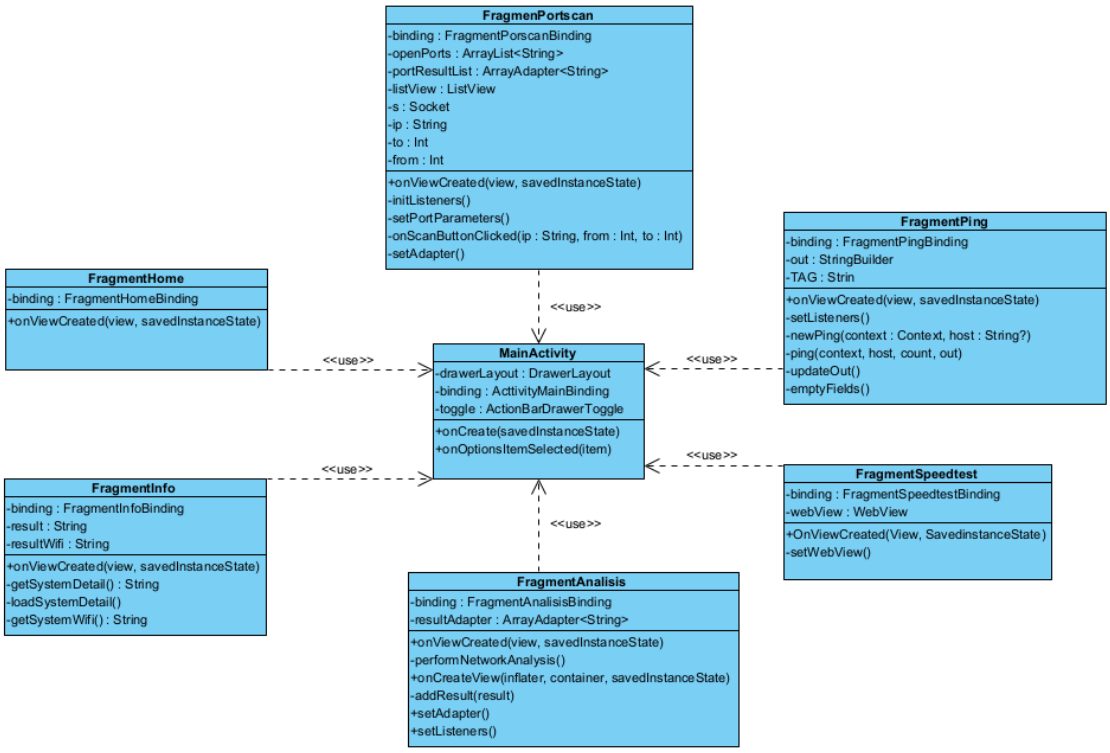
7.2 Diseño

En el punto anterior se establecieron las características que debía cumplir la aplicación a desarrollar, por lo que, de acuerdo a los criterios de sencillez y usabilidad se realizó el siguiente diseño de la aplicación:



Con la información generada se decide usar un sistema de diseño de fragmentos. Con el uso de fragmentos ya se ha visto que la escalabilidad de los proyectos aumenta, por lo que permitiría expandir este proyecto a otros dispositivos con muy pocos cambios.

Se idea por tanto el siguiente diagrama de clases UML:



Modelo de datos: diagrama de las colecciones, documentos (clave:valor)

7.3 Planificación del desarrollo

Durante esta fase se usaron todos los recursos anteriormente creados para establecer la hoja de ruta del desarrollo.

Dado que se había acordado desde el principio trabajar conforme a la metodología ágil con sprints semanales, se decidió hacer uso de Trello, organizando las tareas en sprints semanales. Debido a la poca experiencia del equipo, se trató de dejar semanas con poca carga de trabajo para permitir adaptarse a los cambios imprevistos en el proyecto. A la postre resultaría un acierto pues fueron varios los problemas a los que hubo que enfrentarse.

A continuación, se presenta una tabla en la que se contrastan las tareas programadas frente a las tareas realizadas:

SEMANA	PREVISTO	REAL
Semana 3-9 abril	Estudio del proyecto	Estudio del proyecto
Semana 10-16 abril	Estudio del proyecto	Estudio del proyecto
Semana 17-23 abril	Configurar proyecto	Configurar proyecto
	Menú navegación	Menú navegación
	Elección de iconos y paleta de colores	Elección de iconos y colores
		Creación del FragmentHome
Semana 24-30 abril	Creación de todos los Fragments	Creación del FragmentAnalisis
	Implementación navigation panel	Creación del FragmentInfo
		Creación del FragmentPortscan
		Creación del FragmentSpeedtest
Semana 1-7 mayo	Funcionalidad PortScan	Implementación navigation panel
	Funcionalidad Speedtest	Creación menú navigation drawer
		Creación FragmentHome
		Funcionalidad PortScan
		Funcionalidad Speedtest
Semana 8-14 mayo		Optimizar lógica Portscan
		Mejoras estéticas
	Funcionalidad Info	Funcionalidad Info
Semana 15-21 mayo	Funcionalidad Análisis	Funcionalidad Análisis
		Replanteo estético SpeedTest
Semana 21-28 mayo	Retoques estéticos finales	Implementar webview para SpeedTest
	Redacción del proyecto	Redacción del proyecto

7.4 Explicaciones de la funcionalidad del proyecto

Al iniciar la aplicación el proyecto tiene una activity principal (MainActivity) que establece la interfaz de usuario de la aplicación, maneja la navegación entre los

fragmentos utilizando el drawerLayout y el menú hamburguesa que, al desplegarse, permite seleccionar la funcionalidad deseada.

El código de la MainActivity está compuesto de los siguientes componentes:

- En el método onCreate, se realiza la configuración inicial de la activity. Se infla el diseño mediante el uso de **ActivityMainBinding** y se establece como contenido de la actividad mediante **setContentView(binding.root)**. También se configura la barra de herramientas (**Toolbar**) como la barra de acción de la actividad.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    setSupportActionBar(binding.appBarMain.toolbar)
```

- Se inicializan y configuran los elementos necesarios para el menú lateral **(drawerLayout)**. Se crea una instancia de **ActionBarDrawerToggle**, que actúa como un controlador para abrir y cerrar el drawerLayout en respuesta al botón de navegación en la barra de acción. La instancia de ActionBarDrawerToggle se sincroniza con el drawerLayout mediante **toggle.syncState()**.

```
drawerLayout = binding.lateralMenu
val toggle = ActionBarDrawerToggle( activity: this,binding.lateralMenu,binding.appBarMain.toolbar,
    "Open navigation drawer", "Close navigation drawer")
drawerLayout.addDrawerListener(toggle)
toggle.syncState()
```

- A continuación, se comprueba si savedInstanceState es nulo. Si es así, significa que es la primera vez que se crea la actividad, por lo que se agrega el fragmento **FragmentHome** al área de contenido principal (fragment_content_main) utilizando **supportFragmentManager.commit**.

```
if (savedInstanceState == null) {
    supportFragmentManager.commit { this: FragmentTransaction
        setReorderingAllowed(true)
        add<FragmentHome>(R.id.fragment_content_main)
    }
}
```

- Se configura el listener del menú de navegación (**NavigationView**). Mediante el uso de un switch y **supportFragmentManager.commit** se permite instanciar fragmentos en el menú hamburguesa.

```
binding.navigationView.setNavigationItemSelectedListener { it: MenuItem
    when (it.itemId) {
        R.id.item_menu_home -> {
            supportFragmentManager.commit { this: FragmentTransaction
                setReorderingAllowed(true)
                replace<FragmentHome>(R.id.fragment_content_main)
                drawerLayout.closeDrawers()
            }
        }
```

- Por último, en el método **onOptionsItemSelected** se le pasa como parámetro el objeto MenuItem que es un elemento del menú seleccionado. Esto permite que se pueda elegir un elemento del menú hamburguesa antes de que se pueda llevar a cabo cualquier otra actividad y que, además, se realicen las acciones asociadas una vez elegido el evento.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if (toggle.onOptionsItemSelected(item)){
        return true
    }
    return super.onOptionsItemSelected(item)
}
```

Tras desplegar las opciones de menú hamburguesa se puede elegir una de las funciones descritas. Todas ellas se corresponden con distintos fragmentos que realizarán su propia función.

El comportamiento básico de todos los fragmentos es prácticamente el mismo, variando la lógica asociada a su función. Como ejemplo general se explica el FragmentHome:

- Se utiliza un constructor que toma como argumento el diseño del fragmento "R.layout.fragment_home", consiguiendo así dejar establecido el diseño del fragmento.

```
class FragmentHome : Fragment(R.layout.fragment_home) {
```

- Se declara una variable binding de tipo FragmentHomeBinding utilizando **lateinit var**. Lateinit se utiliza para indicar que la propiedad **binding** se inicializará más adelante en el código antes de ser utilizada. Con este código se podrá acceder a las vistas y elementos del diseño del fragmento.

```
private lateinit var binding : FragmentHomeBinding
```

- En el método onViewCreated es donde se realiza la configuración inicial de las vistas y se asocian con las instancias correspondientes del diseño.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding = FragmentHomeBinding.bind(view)
    activity?.title = "HOME";
```

Dentro de esta función se llama al método **bind(view)** que crea una instancia del fragmento asociado a binding (FragmentHomeBinding en este caso) y permite acceder a las vistas del diseño del fragmento.

A continuación, se establece el título de la actividad ("HOME") utilizando la propiedad activity?.title. Se utiliza el operador de seguridad de llamada (?) para evitar una excepción en caso de que la actividad sea nula.

Tal y cómo se ha visto en los puntos anteriores la aplicación tiene diferentes funcionalidades y lógicas asociadas a ellas. A continuación se detalla la lógica asociada a los distintos fragmentos relacionados con funciones de la aplicación:

1. FragmentHome: hace que la app vuelva al inicio desde la pantalla de cualquier fragmento.
2. FragmentInfo: permite mostrar información detallada sobre el dispositivo en el que se ejecuta la función.
    - Se declara una variable **result** para almacenar la información del sistema:
      ```
      private var result : String = ""
      ```
    - En el método **onViewCreated**, se llama al método **getSystemDetail()** para obtener la información detallada del sistema y se asigna el resultado a la variable result.
      ```
      result = getSystemDetail()
      ```
    - Se llama al método **loadSystemDetail()** para cargar la información del sistema en la vista.
      ```
      loadSystemDetail()
      ```
    - El método **getSystemDetail()** devuelve una cadena de texto que contiene varios detalles del sistema, como la marca (Build.BRAND), el modelo (Build.MODEL) o el ID (Build.ID). La información se obtiene utilizando las propiedades y métodos proporcionados por la clase **Build** de Android

```
private fun getSystemDetail(): String {
    return  "Brand: ${Build.BRAND} \n" +
            "Model: ${Build.MODEL} \n" +
            "ID: ${Build.ID} \n" +
            "SDK: ${Build.VERSION.SDK_INT} \n" +
            "Manufacture: ${Build.MANUFACTURER} \n" +
            "Brand: ${Build.BRAND} \n" +
            "User: ${Build.USER} \n" +
            "Type: ${Build.TYPE} \n" +
            "Base: ${Build.VERSION_CODES.BASE} \n" +
            "Incremental: ${Build.VERSION.INCREMENTAL} \n" +
            "Board: ${Build.BOARD} \n" +
            "Host: ${Build.HOST} \n" +
            "FingerPrint: ${Build.FINGERPRINT} \n" +
            "Version Code: ${Build.VERSION.RELEASE}"
}
```

- El método **loadSystemDetail()** asigna el valor de **result** al campo de texto **infoDeviceNameValue** en la vista, mostrando así la información del sistema en la interfaz de usuario.

```
private fun loadSystemDetail() {
    binding.infoDeviceNameValue.text = result
}
```

3. FragmentPortScan: se utiliza para escanear los puertos de un host específico dentro de un rango.

    - En el método **onViewCreated** se llama a los métodos **setAdapter()** e **initListeners()**.
    - En el método **setAdapter()**, se crea un **ArrayAdapter** para mostrar los resultados del escaneo de puertos en un **ListView**. El adaptador se inicializa con la lista **openPorts** y se asigna al **ListView** **portscanResultList**.

```
private fun setAdapter() {

    portResultList = ArrayAdapter(requireActivity(), R.layout.listview_item, openPorts)
    listView = binding.portscanResultList
    listView.adapter = portResultList

}
```

- El método **initListeners()** configura los listeners de eventos para los botones de escaneo y limpieza.

```
@SuppressLint("SetTextI18n")
private fun initListeners() {
    binding.portscanButtonScan.setOnClickListener { it: View!
        setPortParameters()
    }
    binding.portscanButtonClear.setOnClickListener { it: View!
        portResultList.clear()
    }
}
```

- Cuando se hace clic en el botón de escaneo (**portscanButtonScan**), se llama al método **setPortParameters()** que obtiene la dirección IP, el puerto inicial y el puerto final a través de la interfaz de usuario. Por último, inicia una corrutina utilizando **lifecycleScope.launch** para realizar el escaneo de puertos.

```
private fun setPortParameters() {
    ip = binding.portscanTargetHost.text.toString()
    from = binding.portscanFromPort.text.toString().toInt()
    to = binding.portscanToPort.text.toString().toInt()
    lifecycleScope.launch { this: CoroutineScope
        onScanButtonClicked(ip, from, to)
    }
}
```

- Dentro de la corrutina se ejecuta el método **onScanButtonClicked()** utilizando el contexto de **Dispatchers.IO**. Este método realiza el escaneo de puertos en un bucle **for** desde el puerto inicial hasta el puerto final especificado.

```
private suspend fun onScanButtonClicked(ip: String, from: Int, to: Int) =
    withContext(Dispatchers.IO) { this: CoroutineScope
        for (port in from ≤ .. ≤ to) {
            try {
                s = Socket()
                s.connect(InetSocketAddress(ip, port), timeout: 2000)
                if (s.isConnected) {
                    openPorts.add("Host $ip is OPEN on port $port")
                    requireActivity().runOnUiThread {
                        portResultList.notifyDataSetChanged()
                    }
                    s.close()
                }
            } catch (e: IOException) {
                openPorts.add("Host $ip is CLOSED on port $port")
                requireActivity().runOnUiThread {
                    portResultList.notifyDataSetChanged()
                }
                s.close()
            } catch (e: NullPointerException) {
                e.printStackTrace()
            }
        }
    }
```

- Dentro del bucle **for**, se crea una instancia de **Socket** y se conecta al host con cada puerto utilizando el método .**connect(InetSocketAddress(ip, port), 2000)**. Si la conexión tiene éxito, se agrega un mensaje a la lista **openPorts** indicando que el host está abierto en ese puerto. Luego se notifica a **portResultList** para actualizar la lista en la interfaz de usuario. Finalmente, se cierra el socket.
- Si ocurre una excepción **IOException**, se agrega un mensaje a la lista **openPorts** indicando que el host está cerrado en ese puerto. De nuevo se actualiza la lista a través de **portResultList** y se cierra el socket.
- Si se produjese una excepción **NullPointerException**, se imprime la traza de la excepción.

FragmentAnálisis: este código implementa la funcionalidad de análisis de red buscando dispositivos activos en una red y mostrando los resultados en forma de lista.

- El bloque de variables de instancia declara una variable **binding** de tipo FragmentAnalisisBinding, que representa el enlace de vista para este fragmento, y una variable **resultAdapter** de tipo ArrayAdapter para mostrar los resultados del análisis de red.

```
class FragmentAnalisis : Fragment() {
    private lateinit var binding: FragmentAnalisisBinding
    private lateinit var resultAdapter: ArrayAdapter<String>
```

- En el método **onCreateView** se crea la vista del fragmento y se ingla el diseño usando **FragmentAnalisisBinding**.
- El método **onViewCreated** se llama después de que se haya creado la vista del fragmento y se realiza el enlace con la vista mediante binding.
- El método **setAdapter** crea un nuevo **ArrayAdapter** utilizando el contexto actual y el diseño android.R.layout.simple_list_item_1, y lo asigna al ListView lvAnalyzeResults del fragmento.

```
private fun setAdapter() {
    resultAdapter = ArrayAdapter(requireContext(), android.R.layout.simple_list_item_1)
    binding.lvAnalyzeResults.adapter = resultAdapter
}
```

- El método **setListeners** configura los listeners de los botones **btnAnalyze** y **btnClear**. Al hacer clic en el botón "**Analyze**", se borran los resultados existentes y se realiza un análisis de red mediante la función **performNetworkAnalysis**(). Al hacer clic en el botón

"Clear", se borran los resultados existentes sin realizar ningún análisis adicional.

```
private fun setListeners() {
    binding.btnAnalyze.setOnClickListener { it: View!
        resultAdapter.clear()
        performNetworkAnalysis()
    }
```

- El método **performNetworkAnalysis** realiza un análisis de red en segundo plano utilizando corrutinas (CoroutineScope) y el Dispatchers.IO. Se utiliza por defecto un solo hilo para que no sature la capacidad de la máquina. Luego, hace un bucle sobre las direcciones de interfaz y obtiene la dirección IP correspondiente. Finalmente, llama a la función addResult para agregar la dirección IP al adaptador de resultados.

4. FragmentSpeedTest: se utiliza para realizar una prueba de velocidad de red.
    - **WebView** es una clase utilizada para mostrar contenido web dentro de este fragmento.
    - En el método **onViewCreated** se configura la WebView llamando al método **setWebView().**

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?)
    super.onViewCreated(view, savedInstanceState)
    binding = FragmentSpeedtestBinding.bind(view)
    activity?.title = "SPEEDTEST"
    setWebView()
}
```

    - El método setWebView() obtiene la referencia a la WebView desde el **binding.webWindow**, habilita el soporte para JavaScript y carga la URL "https://speedsmart.net/" en la WebView.

```
private fun setWebView(){
    webView = binding.webWindow
    webView.settings.javaScriptEnabled = true
    webView.loadUrl( url: "https://speedsmart.net/")
}
```

6. FragmentPing: se utiliza para conocer el ping de la red. El código es extremadamente largo, así que se ha optado por comentar qué hace cada bloque de código:

- El primer bucle for realiza las siguientes operaciones:
    - Envía un ping al host (mediante método **ping.ping**).
    - Verifica si se ha enviado el ping y, si es así lo agrega la cadena PING host(ip) al StringBuilder **out**.
    - Agrega una nueva línea y se actualiza la vista.

  Cada iteración da información del host, la dirección IP, el tiempo de respuesta y el estado del ping (si fue exitoso o no o si ocurrió un timeout). También calcula el tiempo promedio, mínimo y máximo y la desviación estándar.

  Al finalizar el bucle el StringBuilder se actualiza para mostrar los resultados en la vista **output**
- El segundo bloque de for realiza una serie de operaciones para procesar los datos. En caso de fallo el error queda registrado en el bloque catch.
- Finalmente se llama **a updateOut()** para mostrar los resultados en la vista **output** y, mediante un Sanackbar se indica que la ejecución ha finalizado.

7. Estructura de navegación: mediante el código escrito en menu_navigation se consigue la navegación entre diferentes fragmentos en una aplicación. A cada fragment se le pasa un id, que servirá para referenciarlo la estructura en otros lugares de la aplicación, y al fragmento de inicio se le asigna una etiqueta del tipo **app:startDestination**.Además a cada fragment se le puede asignar un nombre con la etiqueta **android:label**.

```
app:startDestination="@+id/fragmentHome">

<fragment
    android:id="@+id/fragmentHome"
    android:name="com.example.seknet.FragmentHome"
    android:label="@string/item_menu_init" />
```

Por otro lado, desde MainActivity se ha creado una estructura **when** que redirige la actividad al fragmento correspondiente en función del icono seleccionado por el usuario.

```
binding.navigationView.setNavigationItemSelectedListener { it: MenuItem
    when (it.itemId) {
        R.id.item_menu_home -> {
            supportFragmentManager.commit { this: FragmentTransaction
                setReorderingAllowed(true)
                replace<FragmentHome>(R.id.fragment_content_main)
                drawerLayout.closeDrawers()
            }
        }

        R.id.item_menu_info -> {
            supportFragmentManager.commit { this: FragmentTransaction
```

8. CONCLUSIONES Y MEJORAS DEL PROYECTO

Durante el desarrollo del proyecto han sido varios los problemas a superar, hasta tal punto que la fase de investigación se ha prolongado hasta casi el final.

Además, en ciertos momentos ha habido errores insalvables que han obligado a cambiar el planteamiento inicial y reorientarlo en la misma dirección, pero en otro sentido. Ejemplos claros han sido el test de velocidad, donde se descubrió la imposibildad de realizarlo sin usar un servidor. La solución llegó de la mano del uso de un webView que hospedaría el servicio que se requería.

En el caso del ping se tuvo que hacer un downgrade del classpad debido a que las librerías asociadas, o estaban deprecadas, o dejaron de actualizarse al llegar androidX.

Cuando se realizó el desarrollo de análisis se intentó que se diese vista de las direcciones MAC conectadas a una red doméstica, pero usando Android no fue posible descubrir cómo hacerlo, por lo que se decidió reorientar el servicio.

Por último el uso de las corrutinas y su capacidad para gestionar tareas asíncronas permitió hacer análisis en rangos amplios y actualizar listviews que, de otra manera no habría sido posible popular.

Entre las posibles mejoras del proyecto estarían:

- Crear un login, un signup y un signout.
- Mejorar diseño de la aplicación, aprovechando mejor los espacios.
- Implementación de Firebase Crashlytics para mejorar usabilidad por medio de sus méticas.

Por último se ha visto que todos los proyectos deben estar abiertos a implementar cambios por eventos inesperados y en ocasiones seguir el plan trazado requiere de muchos desvíos en el camino.

9. BIBLIOGRAFÍA

Información general sobre clases: https://developer.android.com/

Matriz dafo: https://dafo.ipyme.org/dafos

Moqups: https://app.moqups.com/

Firebase: https://firebase.google.com/?hl=es

Documentación AndroidStudio https://developer.android.com/studio/intro?hl=es-419

Trello: https://trello.com/es/tour