Menoufia University

Faculty of Electronic Engineering

Computer Science and Engineering Department

2022 - 2023

Pentor

Personal network inspector

Graduation project documentation

Supervisor

Dr. Sonia Hashish

Prepared by

Mahmoud Nasser Elsayed Mansour

Ahmed Hosny Salah Ahmed

Amira Sami Saad Omar

Aya Islam Gaber Allam

Team Information

Team Name

Fluteeks team

Derived from Flutter Geeks.

Fluteeks Logo

Team Members

Fluteeks team consists of 4 senior CSE students.

Name	**Section**	**Acad. Num.**
Aya Islam Gaber Allam	1	1800073
Amira Sami Saad Omar	1	1700067
Ahmed Hosny Salah Ahmed	1	1600432
Mahmoud Nasser Elsayed Mansour (Team Leader)	4	1700245

Supervisor

Dr. Sonia Hashish

Acknowledgement

We humbly extend our sincerest gratitude to Allah, the Most Merciful and Compassionate, for blessing us with the strength, knowledge, and perseverance to bring this project to fruition. Our faith and trust in His guidance have been the driving force behind our journey.

We would also like to express our deepest appreciation to our families for their unwavering love, encouragement, and support throughout the development of Pentor. Their patience and understanding have been invaluable, and we are grateful for their constant presence in our lives.

Furthermore, we extend our heartfelt thanks to **Dr. Sonia Hashish** for her exceptional mentorship, expertise, and guidance throughout this endeavor. Her wisdom, encouragement, and dedication have inspired us to reach new heights and have contributed immensely to the success of Pentor.

With deep gratitude and humility, we express our thanks to Allah, our families, and Dr. Sonia Hashish for their unwavering support and guidance, which have played a vital role in the creation of Pentor.

Abstract

In today’s digital world, network inspection is an essential tool for businesses and individuals alike. Network inspection apps provide users with the ability to monitor and analyze their networks in real-time, allowing them to identify potential security threats, optimize performance, and troubleshoot any issues that may arise. With the rise of cybercrime and the increasing complexity of networks, network inspection apps are becoming increasingly important for people who need to keep their networks safe and secure.

This is where an app like Pentor is needed to allow users to monitor their networks in real-time, giving them visibility into what is happening on their network at any given time, which allows users to quickly identify potential security threats or performance issues before they become major problems.

Overall, Pentor app provides a valuable service for people who need to keep their networks safe and secure. By providing real-time monitoring and detailed reports about their networks, and allowing users to quickly identify potential security threats or performance issues before they become major problems. Additionally, it can be used to troubleshoot many issues that may arise on the network, helping users resolve them quickly and efficiently

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction

Personal Network Inspector (Pentor) is a cross-platform network monitoring app designed to provide personal network owners with comprehensive information about their networks. With the increasing number of devices connected to personal networks, it has become more important than ever to have a tool that can help prevent unwanted and unauthorized access. Pentor is here to fill that gap by providing real-time monitoring and alerts for any suspicious activity on your network. In this chapter, we will explore project details, software development cycle, features and benefits of Pentor, as well as its compatibility with various operating systems. Let's dive in and discover how Pentor can help you keep your network secure.

Figure 1.1: Pentor Icon

1.2 Project Purpose

Pentor is a network monitoring cross platform app developed to help personal network owners get more information about their networks and prevent unwanted and unauthorized access to the network.

1.3 Problem Definition

Network inspection nowadays is a must because of the fast spread of network hacking. Network inspector scans the current network for security issues and vulnerabilities that may expose the sensitive private data to attacks. It shows the devices currently connected to the home network. So, if any of them is connected without permission, it is detected. Everyone should have enough knowledge about their networks and try to protect them from any attacks.

For example, a person with no knowledge of networks may have a Wi-Fi network set up in their home for personal use. However, if the person does not secure the network with a strong password, an attacker could gain access to the network and use it to distribute malware or carry out illegal activities without the person's knowledge.

In some cases, an attacker may use the hijacked network to engage in illegal file sharing, which could result in the person receiving copyright infringement notices or even being sued for damages.

In another example, an attacker may use the hijacked network to launch phishing attacks against other users on the network, tricking them into sharing sensitive information or downloading malware.

These types of local network hijacking attacks can result in significant losses for individuals who are not familiar with network security or how to secure their local networks. Victims may face legal or financial consequences as a result of the actions carried out on their network by the attacker.

1.4 Project Description

Pentor Project is a software project aimed to provide a cross platform mobile application. Pentor developed to help personal network owners get more information about their networks and prevent unwanted and unauthorized access. Pentor also provides detailed reports on the activity of each device on the network, including IP addresses, MAC addresses, and other data. Additionally, Pentor can detect suspicious activity on the network and alert users to potential security threats. With its easy-to-use interface and powerful features, Pentor is the perfect tool for keeping home network safe from unauthorized access.

1.4.1 Project scope

Pentor project is an application, which enables personal network owners to get more information about their networks and prevent unwanted and unauthorized accesses to the network.

1.4.2 Project target

The ordinary users. Users don't have to be in an advanced level in networks, but everyone who has a little knowledge about networks can use the application and get benefit from it.

1.4.3 Project Challenges

1. Ensuring compatibility with all types of networks: Pentor needs to be able to work with all types of networks.
2. Developing a user-friendly interface: Pentor needs to have an intuitive and easy-to-use interface so that users can easily access the features they need.
3. Integrating security features: Pentor must be able to detect and prevent unauthorized access attempts, as well as protect users' data from malicious actors.
4. Providing real-time monitoring: Pentor must be able to provide real-time monitoring of the network so that users can quickly identify any potential threats or issues.
5. Implementing automated alerts: Pentor should be able to send automated alerts when it detects suspicious activity or unauthorized access attempts on the network.

1.4.4 Project Features

"Pentor" application provides many tasks as the following:

1. Display the basic information of the network like host IP, gateway, MAC address etc.
2. Internet speed test: a quick way to see the speed of the internet. It is the best way to get an idea of how fast the connection is right.
3. Ping test: ping is simply the shortest possible time to send data and receive a response between any given set of hosts. Testing the ping is by sending empty requests to servers. When a response is received, the time taken is the ping.
4. DNS test: DNS are services which inform the device of the IP address of a domain. DNS test is used to find out the DNS a device is using at any given time.
5. Display the information of the devices connected to the network.
6. Display the data usage of the applications.
7. Display the network logger: a network log is a file which contains a record of events that occurred in the network.
8. Scan the network periodically and notify the user in case of intruders.
9. Change Wi-Fi password.
10. Block intruders using MAC filter: every device connected to the network has a unique MAC address. MAC address filtering can check connected devices, block connected devices and allow blocked devices back onto the network.
11. Setup router DNS.
12. Limit router speed.

1.4.5 Expected outcome

The project expected outcome:

- Written documentation for the whole project
- Presentation file concludes the whole project process
- Mobile Application that implements the project features

1.5 Software Development Approach

Pentor project follows the Waterfall approach as each phase is carried sequentially to make the whole team participate in the whole development process.

1.5.1 Waterfall advantages

1. **Easy to understand and use**: The waterfall model is easy to understand and use. It is a linear approach that follows a logical progression from one phase to the next. This makes it easier for project managers and developers to plan, execute, and track progress.
2. **Clear milestones**: The waterfall model provides clear milestones for each phase of the project. This helps project managers monitor progress and ensure that deadlines are met.
3. **Low risk**: The waterfall model is a low-risk approach because it does not require major changes once the project has begun. This makes it easier to manage costs and timelines, as well as reduce the risk of failure due to unforeseen circumstances.

1.5.2 Waterfall disadvantages

1. **Inflexible**: The waterfall model is inflexible because changes cannot be made once the project has begun. This can be problematic if requirements change or if new features need to be added during development.
2. **Poorly suited for complex projects**: The waterfall model is not well-suited for complex projects because it does not allow for iteration or adaptation during development. This can lead to delays or even failure if unexpected issues arise during development.
3. **Poorly suited for long projects**: The waterfall model is also not well-suited for long projects because it does not allow for iteration or adaptation during development, which can lead to delays or even failure if unexpected issues arise during development.

1.5.3 Development phases

When used for a software development process, the waterfall methodology has seven stages (Requirements Gathering, Analysis, Design, Implementation, Testing, Deployment, and Maintenance) each stage is carried out as independently is possible and in a sequential style. The seven Stages are separated as following:

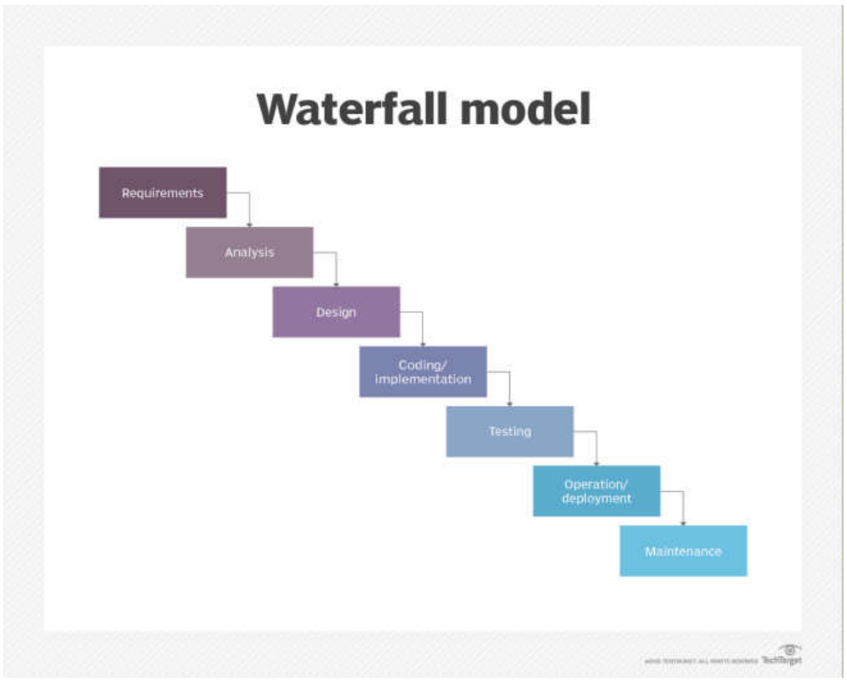


Figure 1.2: Waterfall model

1. **Requirements Gathering**: This is the first phase of the waterfall software development approach. During this phase, the project requirements are gathered from stakeholders or customers.
2. **Analysis**: after requirements is organized it should be analyzed.
3. **Design**: During this phase, a detailed design of the system is created based on the requirements gathered in the previous phase. The design should include all aspects of the system such as user interface, database structure, algorithms, etc.
4. **Implementation**: This is where the actual coding takes place. The code is written according to the design created in the previous phase and tested for correctness and performance.
5. **Testing**: Once all the code has been written, it is tested for correctness and performance against a set of test cases that were created during the design phase. Any bugs or issues found during testing are fixed before moving on to the next phase.
6. **Deployment**: Once all tests have passed successfully, the system is ready for deployment into a production environment where it can be used by end users. This may involve setting up servers, configuring networks, etc., depending on what type of system it is.
7. **Maintenance**: After deployment, there will be ongoing maintenance required to ensure that any new features or bug fixes are implemented correctly and that any changes made do not break existing functionality or introduce new bugs into the system.

1.6 Requirement Gathering

As the project targets the network inspection market, a market search and report were conducted to gather the requirements.

1.6.1 Market search

After performing a wide search on mobile networking apps, the team generated a detailed market report. Following a list of most famous apps that were tested:

1. PingTools Network Utilities
2. Wi-Fi Analyzer
3. IP Tools - Network Utilities
4. *Network Monitor Mini*
5. 3G Watchdog - Data Usage
6. Fing - Network Tools
7. Router Chef
8. Wi-Fi Blocker
9. 3G Watchdog - Data Usage
10. Router Chef
11. Internet Speed Meter Lite
12. Who Uses My WiFi Pro
13. *Network Connections*

1.6.2 Market report conclusion

From the previous list it became clear that most networks monitoring apps contains multiple features that help users monitor and manage their own local networks and protect themselves from unauthorized access to their local network, resources, and information.

Popular network monitoring apps contain the following features:

- Run Wi-Fi and Cellular internet speed tests.
- Scan network to discover all connected devices.
- Inform the user when a new device gets connected to the network.
- Data usage.
- Ping and Calculate Latency
- DNS Lookup
- Basic network information (host IP, gateway, Signal strength, etc.)

1.6.3 What network monitoring apps lack

- *Easy to use and understand app (as most apps usually directed to experts).*
- *Support of multiple languages (Most apps only support English).*
- *App that doesn't require root privileges.*
- *All in one app (provides most needed features in one application).*
- *Open-source app (trusted as source code is readable to all, usable by other developers in the community).*
- *Free app available to everybody with no annoying ads.*

From the previous list the team targeted the project requirements, and the next phase was carried on.

1.7 Development Environment

The Development and implementation Language of Pentor Project is Dart Programming language and the Flutter framework for cross platform development, Code writing, debugging and simulation performed on android Studio. While the database is stored over the cloud of Firebase as a backend implementation. Pentor will be available for Android phones natively written in Java, as well as IOS phones natively written in Objective-c which is automatically generated by Flutter framework tools.

1.8 Project Management

Both Microsoft Planner and onlinegantt tools were used to manage and schedule tasks as following

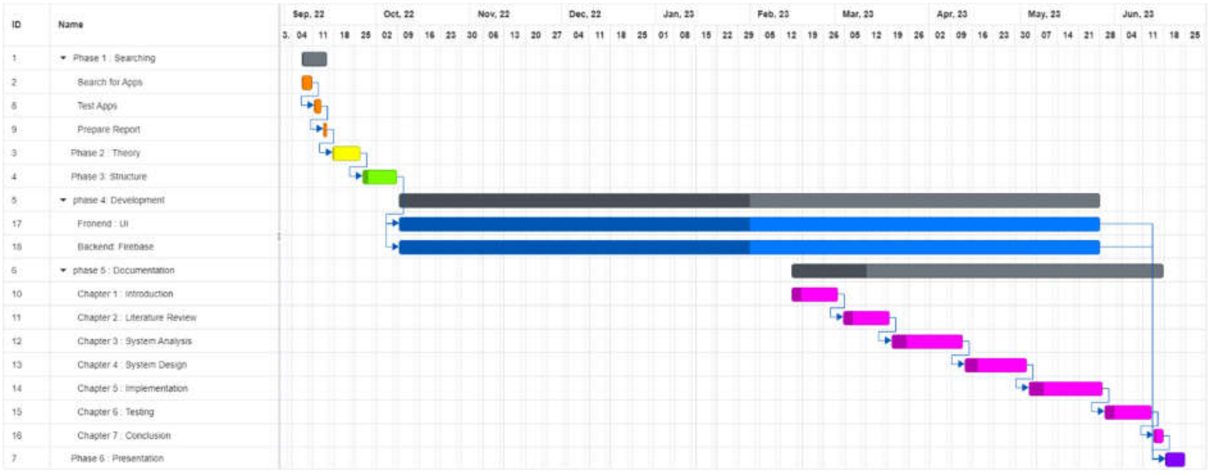


Figure 1.3: Project Gantt chart

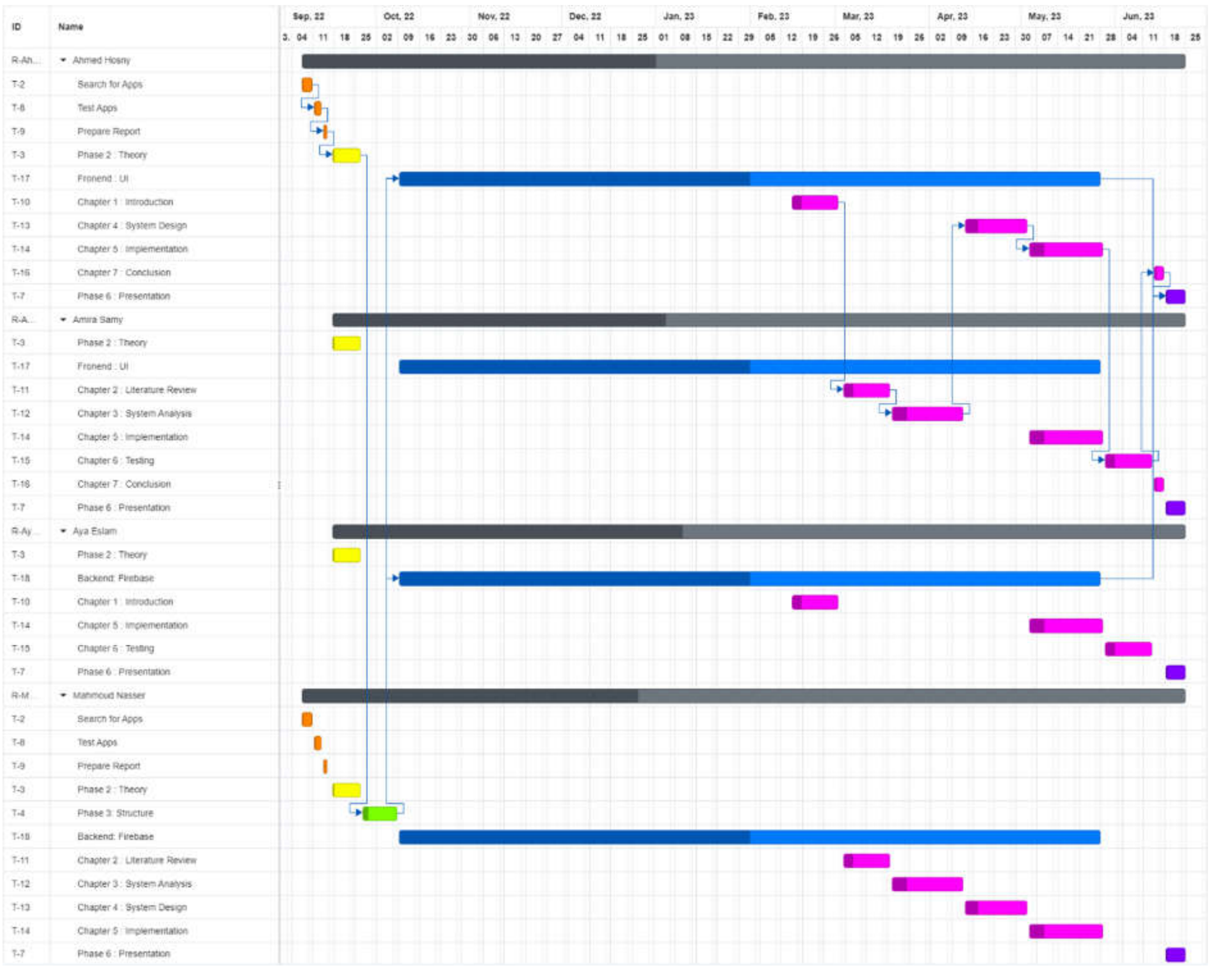


Figure 1.4: Task scheduling chart

1.9 Summary

The introduction chapter outlines the purpose, description, and scope of the project. The problem definition and justification are also discussed, along with the target audience and challenges that may arise during development. The expected outcome of the project is also highlighted. The software development approach is explained, and it is noted that network monitoring apps lack certain features that this project aims to address. Finally, the development environment is discussed.

Chapter 2 Background Review

2.1 Introduction

The Literature Review chapter provides a comprehensive overview of the existing literature on network security and monitoring. This chapter aims to explore the various approaches, techniques, existing systems and tools used in network monitoring and security to help personal network owners get more information about their networks and prevent unwanted and unauthorized access.

2.2 Similar Systems

Android network monitor apps are designed for advanced users who want to control incoming and outgoing traffic on their phones and tablets. Such programs provide information on all Internet connections, services, and apps that utilize Internet traffic, and IP addresses they connect to. Monitoring software displays the amount of data sent and received during each connection. This data is useful for tracking suspicious network activity. Some applications can be configured to send notifications every time the phone establishes an Internet connection. This section contains a list of the most popular android network monitor apps listing their features and issues which discussed briefly.

2.2.1 *PingTools Network Utilities*

PingTools makes it possible to ping the network, get information about its configuration, detect ports and Wi-Fi networks, check whois information, lookup IP addresses, DNS, etc. With PingTools, you can track the use of the network. It also features a wake-on network function

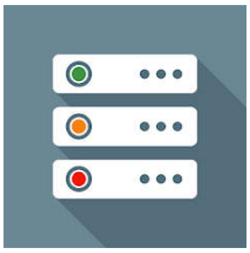


Figure 2.1: PingTools Network Utilities Icon

2.2.2 *Wi-Fi Analyzer*

Introducing a new way to analyze and optimize your Wi-Fi, turn android device into Wi-Fi Analyzer, Recommends the best channel and place for your network, gives you the most useful optimization information to help decrease interference and increase connection speed and stability.

Figure 2.2: Wi-Fi Analyzer icon

2.2.3 *Netcut pro for android 2021*

It can disable (shutdown) the internet connection of other devices connected to the same Wi-Fi network. It is a very useful tool for Wi-Fi internet users using which you can cut (Turn off) other people off form a common Wi-Fi network and allocate all the bandwidth to yourself. The app is for Android only and it requires root access.

Figure 2.3: Netcut pro for android 2021 Icon

2.2.4 *IP Tools - Network Utilities*

IP Tools is a powerful network toolkit for speeding up and setup networks. It allows quick detecting any computer network problems, IP address detection and boosting network performance. This is a must-have app for IT specialists and network administrators.

Figure 2.4: IP Tools - Network Utilities Icon

2.2.5 *Wi-Fi Password Recovery*

Wifi Password Recovery is the app to recover the ever-lost password of WiFi networks that device has connected. "WIFI Password Recovery" will recover wlan passwords of previously connected Wi-Fi networks on android devices.

Figure 2.5: Wi-Fi Password Recovery

2.2.6 *Network Monitor Mini*

This is a mini network monitor for your phone. It monitors the upload and download speed per second. It will always stay in the corner of the phone's screen. Can set the indicator to any corner of the screen, customize the color and transparency of the indicator. Record the live network information for WiFi / 3G / 4G network speed.

Figure 2.6: Network Monitor Mini

2.2.7 *Network Connections*

Powerful tool that displays and monitors (tracks) all inbound and outbound connections from and to your Android device. A low-level connections capture module ensures best performance with minimal battery usage.

Figure 2.7: Network Connections Icon

2.2.8 *3G Watchdog - Data Usage*

The app can count every type of data usage (3G, 4G, WiFi, etc.) and display it in a convenient way. 3G Watchdog demonstrates traffic used by every app on your device. View a net data usage with detailed information about traffic used for certain periods of time (today, per week, per month). Export all data to the CSV file.

Figure 2.8: 3G Watchdog - Data Usage

2.2.9 *Fing - Network Tools*

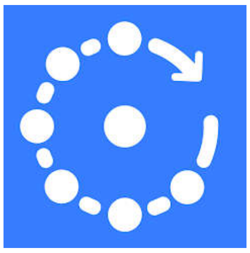


Figure 2.9: Fing - Network Tools

The app can count every type of data usage (3G, 4G, WiFi, etc.) and display it in a convenient way. 3G Watchdog demonstrates traffic used by every app on your device. View a net data usage with detailed information about traffic used for certain periods of time (today, per week, per month). Export all data to the CSV file.

2.2.10 Router Chef

Figure 2.10: Router Chef Icon

Router Chef for Android is a chef app which latest version is released on Jan 16, 2022. Developed as a reset app it is specially designed to give you a unique connected experience. With the help of the Router Chef App, you can control your Router basic settings e.g. - Control Wi-Fi max number of connected devices. - Show the active connected devices on Wi-Fi. - Configure your Router with the ISP account after factory reset. - Reboot and reset Router to factory settings Supported Routers.

2.2.11 Internet Speed Meter Lite

Figure 2.11: Internet Speed Meter Lite Icon

WiFi Internet Speed Meter Lite displays internet speed in status bar and shows the amount of data used in notification pane. This helps to monitor network connection anytime while using your device

2.2.12 Who Uses My WiFi Pro

Figure 2.12: Who Uses My WiFi Pro

The fastest, smartest, and easiest way to control and monitor the number of users connected to WiFi network and get information about the connected devices.

Table 1 : Market apps comparison

App	Show network devices	Internet speed test	Manage router	Data Usage	Require root access	Network tools support
PingTools Network Utilities	✗	✓	✗	✗	✗	✓
Wi-Fi Analyzer	✗	✓	✗	✗	✗	✗
Netcut pro for android 2021	✓	✗	✓	✗	✗	✓
IP Tools - Network Utilities	✓	✗	✗	✗	✗	✓
Wi-Fi Password Recovery	✗	✗	✗	✗	✓	✗
Network Monitor Mini	✓	✗	✗	✗	✗	✗
Network Connections	✓	✗	✗	✗	✗	✗
3G Watchdog - Data Usage	✓	✗	✗	✓	✗	✗
Fing - Network Tools	✓	✓	✓	✓	✗	✓
Router Chef	✗	✗	✓	✗	✗	✗
Internet Speed Meter Lite	✗	✓	✗	✗	✗	✗
Who Uses My WiFi Pro	✓	✗	✓	✗	✗	✗

2.3 Frontend architecture

Mobile application development is the process of creating software applications that run on mobile devices such as smartphones and tablets. Mobile applications can be developed for different platforms, including iOS, Android, and Windows, and can offer a wide range of functionality such as gaming, social media, e-commerce, productivity, and more.

The process of mobile application development involves several stages, including requirements gathering, design and prototyping, development, testing, and deployment to the app stores. Mobile application developers use programming languages such as Java, Kotlin, Swift, and Objective-C, as well as development tools and frameworks such as Android Studio, Xcode, React Native, and Flutter.

Mobile application development is a fast-growing industry, with the demand for mobile applications increasing every year. According to Statista, there were 218 billion app downloads in 2020, and this number is expected to grow to 258 billion by 2022. Businesses of all sizes are leveraging mobile applications to reach their customers and increase engagement, and the trend is expected to continue in the years to come.

Mobile application development offers several benefits for businesses, including increased customer engagement, improved customer experience, increased brand awareness, and increased revenue. Mobile applications can also provide businesses with valuable data insights that can be used to improve their products and services.

However, mobile application development also presents challenges such as fragmentation, security, and performance issues. Developers must ensure that their applications are compatible with different devices and operating systems, and that they are secure and performant.

In conclusion, mobile application development is a complex process that requires specialized knowledge and skills. With the continued growth in the demand for mobile applications, businesses must consider mobile application development as an essential part of their digital strategy.

2.3.1 Flutter Framework

Flutter is an open-source mobile application development framework created by Google. It allows developers to build high-performance, cross-platform mobile applications for iOS and Android using a single codebase. Flutter uses the Dart programming language, which is also developed by Google.

Flutter provides a rich set of pre-built widgets that can be customized to create beautiful and responsive user interfaces. It also has a hot reload feature that allows developers to see the changes they make in real-time, making the development process faster and more efficient.

One of the key benefits of using Flutter is its ability to deliver native-like performance on both iOS and Android platforms. This is achieved through its use of a high-performance rendering engine that allows for smooth animations and transitions. Flutter also has a strong community of developers who contribute to its growth and offer support through forums, blogs, and other resources. This makes it easier for developers to learn and use Flutter effectively.

Overall, Flutter is a powerful tool for building mobile applications that offers many benefits such as fast development time, excellent performance, and cross-platform compatibility.

2.3.2 Flutter Plugins

Flutter plugins are packages of pre-built code that can be added to a Flutter project to extend its functionality. These plugins are designed to provide developers with an easy way to integrate native platform features into their Flutter applications.

Following is a list of flutter packages used in Pentor development:

1. **network_info_plus: ^3.0.2**

   Flutter plugin network_info_plus 3.0.2 is a plugin that provides developers with a way to access network information in their Flutter applications. This plugin provides information about the current network status, including whether the device is connected to a Wi-Fi network or a cellular network, the network name, and the network type.

   With network_info_plus, developers can use this information to create apps that respond to changes in the network status, such as switching to a lower bandwidth mode when the user is on a cellular network or displaying a message when the network connection is lost.

2. **dart_ping: ^7.0.1**

   The dart_ping plugin for Flutter is a package that provides a simple and easy-to-use API for network ping operations. This plugin is based on the ping package, a Dart package that implements ping functionality for both Windows and Unix platforms.

   With dart_ping, developers can easily integrate network ping functionality into their Flutter applications. The plugin allows developers to specify the IP address or hostname to ping, set a custom timeout for the ping request, and customize the payload data sent in the ping packet. Additionally, dart_ping provides real-time feedback on the status of the ping operation, including whether the host is reachable, the time it took for the ping packet to be sent and received, and the number of packets sent and received.

3. dns_client: ^0.2.1

The dns_client plugin for Flutter is a package that provides a simple and easy-to-use API for DNS (Domain Name System) resolution. With this plugin, developers can resolve domain names to IP addresses and vice versa in their Flutter applications.

The dns_client plugin uses the dart:io library to perform DNS resolution, making it compatible with both iOS and Android platforms. The plugin supports various DNS record types, including A, AAAA, CNAME, MX, NS, PTR, SOA, SRV, and TXT.

Using dns_client, developers can easily integrate DNS resolution functionality into their Flutter applications. The plugin allows developers to specify the DNS server to use for resolution, set a custom timeout for the resolution request, and specify the record type to resolve.

4. lan_scanner: ^3.5.0

The lan_scanner plugin for Flutter is a package that provides a simple and easy-to-use API for scanning local area networks for active hosts. With this plugin, developers can easily detect and obtain information about devices that are connected to the same local network as the device running the Flutter application.

The lan_scanner plugin uses a combination of ICMP and ARP protocols to scan the local network for active hosts. The plugin can scan the entire network or a specific IP range and provides real-time feedback on the scanning progress and the status of each scanned host.

Using lan_scanner, developers can easily integrate local network scanning functionality into their Flutter applications. The plugin allows developers to specify the range of IP addresses to scan, set a custom timeout for the scanning operation, and customize the payload data sent in the ICMP packets.

5. network_logger: ^1.0.2

The network_logger plugin for Flutter is a package that provides a simple and easy-to-use API for logging HTTP requests and responses in a Flutter application. With this plugin, developers can easily track and debug network requests and responses in real-time, helping to improve the overall performance and reliability of their applications.

The network_logger plugin intercepts all HTTP requests and responses made by the Flutter application and logs them in a customizable format. The plugin supports logging of request and response headers, request and response bodies, and HTTP method and status code. Developers can customize the logging format and destination, and also filter out requests and responses based on various criteria such as HTTP method, URL, and status code.

Using network_logger, developers can easily integrate network logging functionality into their Flutter applications. The plugin provides a simple and straightforward API that can be easily added to existing codebases.

6. data_usage

The data_usage plugin for Flutter is a package that provides a simple and easy-to-use API for monitoring data usage in a Flutter application. With this plugin, developers can easily track the amount of data being consumed by their application, helping to improve user experience and optimize data usage.

The data_usage plugin allows developers to monitor both cellular and Wi-Fi data usage in real-time. The plugin provides information such as the amount of data used, the amount of data remaining, and the data usage history. Developers can also set data usage limits and receive notifications when these limits are reached.

Using data_usage, developers can easily integrate data usage monitoring functionality into their Flutter applications. The plugin provides a simple and straightforward API that can be easily added to existing codebases.

7. firebase_core: ^2.6.1

The firebase_core plugin for Flutter is a package that provides the core functionality for Firebase integration in a Flutter application. This plugin initializes Firebase and provides access to the Firebase app instance, which can then be used to access other Firebase services such as Firebase Authentication and Firebase Cloud Messaging.

With the firebase_core plugin, developers can easily integrate Firebase into their Flutter applications and benefit from its wide range of features and services such as real-time database, cloud storage, and cloud messaging.

8. **cloud_firestore: ^4.4.2**

   The cloud_firestore plugin for Flutter is a package that provides a simple and easy-to-use API for integrating Cloud Firestore, a NoSQL document database, into a Flutter application. With this plugin, developers can easily store, retrieve and query data in real-time from the cloud.

   With the cloud_firestore plugin, developers can create and manage collections and documents in Cloud Firestore, as well as listen for real-time updates to their data. The plugin also supports offline data persistence, allowing users to access data even when they are offline.

9. **get: ^4.6.5**

   The get package for Flutter is a state management library that provides a simple and intuitive API for managing the state of a Flutter application. With this package, developers can easily create reactive and modular applications without the need for complex state management solutions.

   The get package provides a set of features including Dependency Injection, reactive state management, and routing. It also includes a set of helper functions that simplify the process of building user interfaces in a Flutter application.

10. **shared_preferences: ^2.0.15**

    The shared_preferences plugin for Flutter is a package that provides a simple and easy-to-use API for storing and retrieving user preferences in a Flutter application. With this plugin, developers can easily store key-value pairs on the device, allowing users to personalize their application experience.

    The shared_preferences plugin supports a wide range of data types, including integers, strings, Booleans, doubles, and lists. It also provides a set of convenience methods for working with common data types, making it easy to read and write data.

11. **get_storage: ^2.0.3**

    The get_storage plugin for Flutter is a lightweight and fast key-value storage library for storing data on the device. With this plugin, developers can easily store and retrieve data from the device's persistent storage, including Shared Preferences on Android and User Defaults on iOS.

The get_storage plugin provides a simple and easy-to-use API for storing data, and supports a wide range of data types, including strings, Booleans, integers, doubles, and lists. It also includes a set of convenience methods for working with common data types, making it easy to read and write data.

12. expansion_tile_card: ^2.0.0

The expansion_tile_card plugin for Flutter provides an easy way to create expandable cards in a Flutter app. With this plugin, developers can add a collapsible card widget to their app that expands or collapses when tapped.

The expansion_tile_card widget is highly customizable, with options to change the card's background color, border radius, and elevation. The plugin also includes support for animations and a variety of customization options for the expand and collapse icons.

13. permission_handler: ^10.2.0

The permission_handler plugin for Flutter provides an easy way to request and check permissions in a Flutter app. With this plugin, developers can request permissions for a variety of features, including camera, location, storage, and more.

The permission_handler plugin supports both Android and iOS and provides a consistent API for requesting and checking permissions across both platforms. The plugin also includes support for handling permission status changes and provides helpful utility functions for checking whether a permission is granted or not.

14. connectivity_plus: ^3.0.3

The connectivity_plus plugin for Flutter provides an easy way to monitor the network connectivity status of a Flutter app. With this plugin, developers can easily check whether the device is connected to the internet or not, and can also listen for changes in the network connectivity status.

The connectivity_plus plugin supports both Android and iOS and provides a consistent API for monitoring network connectivity across both platforms. The plugin includes support for checking the type of network connection (e.g. WiFi, cellular) and provides helpful utility functions for checking whether the device is connected to the internet or not.

15. Telephony: ^0.2.0

The telephony plugin for Flutter provides an easy way to interact with the telephony capabilities of a device. With this plugin, developers can access various telephony features, such as making phone calls, sending SMS messages, and listening for incoming call events.

The telephony plugin supports both Android and iOS and provides a consistent API for interacting with the telephony features of the device. The plugin includes functions for making phone calls and sending SMS messages, as well as listeners for receiving incoming call events.

16.firebase_auth: ^4.2.2

The firebase_auth plugin for Flutter provides an easy way to integrate Firebase Authentication into a Flutter app. With this plugin, developers can add authentication to their app using various providers, such as email and password, Google, Facebook, and more.

The firebase_auth plugin supports both Android and iOS and provides a consistent API for authenticating users with Firebase. The plugin includes functions for creating user accounts, signing in with different providers, and managing user sessions.

17. **google_sign_in: ^6.0.0**

The google_sign_in plugin for Flutter provides an easy way to integrate Google Sign-In into a Flutter app. With this plugin, developers can add authentication to their app using a user's Google account.
The google_sign_in plugin supports both Android and iOS and provides a consistent API for signing in with a user's Google account. The plugin includes functions for signing in, signing out, and retrieving information about the authenticated user.

2.4 Backend architecture

Mobile backend development is the process of creating and managing the server-side infrastructure that powers mobile applications. It involves designing and implementing APIs, databases, authentication systems, and other components that enable mobile apps to communicate with servers and access data.

One of the key challenges in mobile backend development is ensuring scalability and performance. Mobile apps can generate a large volume of requests to servers, which can put a strain on resources if not properly managed. To address this challenge, developers often use cloud-based solutions such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure to provide scalable infrastructure.

Another important aspect of mobile backend development is security. Mobile apps often handle sensitive user data such as login credentials, payment information, and personal details. To protect this data from unauthorized access or breaches, developers must implement robust security measures such as encryption, firewalls, and access controls.

Mobile backend development also involves integrating third-party services such as social media platforms, payment gateways, or analytics tools into mobile apps. This requires developers to understand how these services work and how to integrate them seamlessly into their app's architecture.

Overall, mobile backend development plays a critical role in enabling mobile apps to deliver rich user experiences while ensuring scalability, security, and reliability. By leveraging cloud-based solutions and best practices for security and integration, developers can create powerful mobile apps that meet the needs of today's users.

2.4.1 Google Firebase

Firebase is a mobile and web application development platform that provides developers with a range of tools and services to build high-quality applications. It was developed by Google in 2011 and has since become one of the most popular platforms for building mobile and web applications.

Firebase offers a wide range of features including:

1. **Real-time database**: Firebase provides a real-time database that allows developers to store and sync data in real-time across multiple clients. This feature is particularly useful for applications that require real-time updates, such as chat apps or collaborative tools.
2. **Authentication**: Firebase offers a robust authentication system that supports various authentication providers, including email/password, Google, Facebook, Twitter, and more. This feature

makes it easy for developers to add user authentication to their applications without having to build their own authentication system from scratch.

3. **Cloud messaging**: Firebase provides a cloud messaging service that allows developers to send notifications and messages to users across multiple platforms, including iOS, Android, and the web. This feature is particularly useful for applications that need to send push notifications or alerts to users.
4. **Hosting**: Firebase offers hosting services that allow developers to deploy their applications quickly and easily without having to worry about server management or infrastructure setup. This feature makes it easy for developers to get their applications up and running quickly.
5. **Analytics**: Firebase provides analytics tools that allow developers to track user behavior and app performance in real-time. This feature helps developers understand how users are interacting with their application and identify areas for improvement.

2.5 Summary

This chapter has provided insights into the frontend and backend architectures used in mobile application development with a focus on Flutter and Google Firebase. It has also highlighted some existing systems developed using these technologies that can serve as a reference for future developers.

Chapter 3 System Analysis

3.1 Introduction

System analysis refers to the process of studying, designing, and documenting a system's requirements, functions, and operations, the limitations of the system, in addition to understand the interaction between the system components. It involves identifying the goals and objectives of the system, analyzing its current state, defining its scope and boundaries, and determining the resources required to implement it. The system analysis phase is critical in project documentation as it helps to ensure having a clear understanding of what the system is supposed to do and how it will be implemented. This information is then used to develop detailed specifications for the system's design and development.

3.2 System Architecture

The application allows the user to create a new account and to help users monitor and analyze network traffic on their mobile devices. It allows users to view detailed information about the data being sent and received by their device, including the source and destination of the traffic, the protocol being used, and any errors or issues that may be occurring which is in the user device. The application can be useful for troubleshooting network problems, identifying potential security threats, or optimizing network performance. Some network inspector apps may also include additional features such as packet capture and analysis tools, bandwidth monitoring, and device discovery.

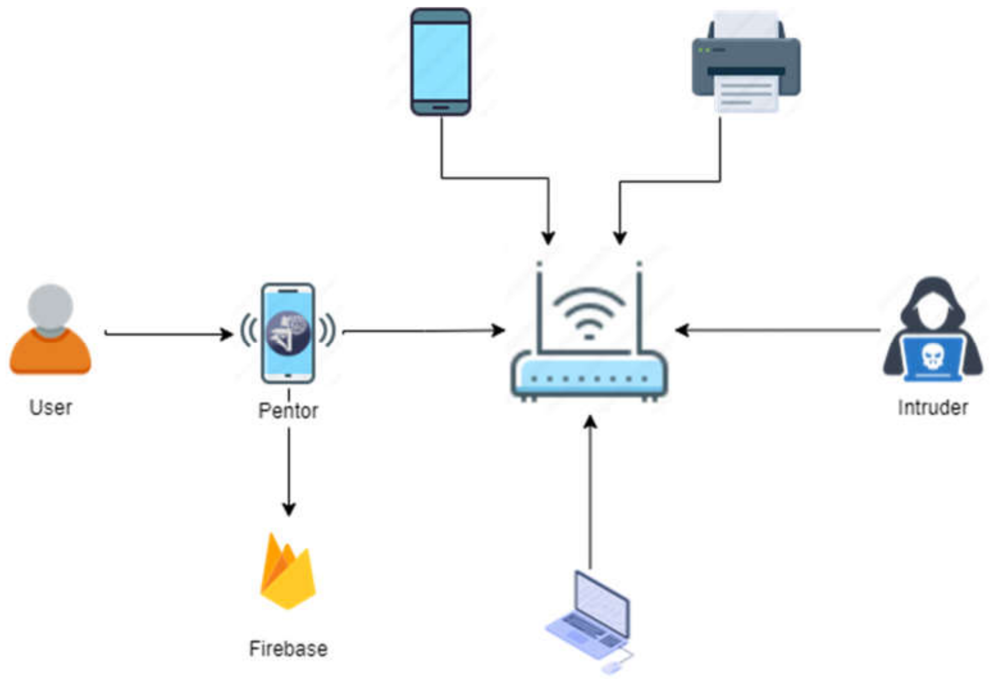


Figure 3.1 : System Architecture

As mobile app's system architecture is designed to provide a seamless user experience while ensuring that data is processed securely and efficiently.

the system architecture of a mobile app typically includes the following components:

1. User Interface (UI): This component is responsible for presenting the app's interface to the user and handling user interactions.
2. Application Logic: This component contains the core functionality of the app, including data processing, business logic, and communication with external services.
3. Data Storage: This component manages data storage and retrieval for the app.
4. Network Communication: This component handles communication between the app and external services over a network connection.
5. Security: This component ensures that user data is protected from unauthorized access or manipulation.
6. Device Compatibility: This component ensures that the app is compatible with different devices and operating systems.

Overall, in our project the UI, application logic, network communication and device compatibility are all provided by Flutter framework, while the data storage and security are provided by Firebase platform.

3.3 System Requirements

The application depends on being connected to a local network to operate its functionality.

3.3.1 System User

The application provides service for ordinary users, who have the minimum enough knowledge of networks, its definitions and concepts.

3.3.2 Functional & Data Requirements

Functional requirements are the ones that define what the app is supposed to do. They are the key functions without which, the app would not function, or do what needed from it. There are the basic functional requirements for the proposed application:

- The application should enable the user to sign in with google account.
- The application should enable the user to display the basic information of the network.
- The application should allow the user to measure the speed of the internet.
- The application should allow the user to do the ping test.
- The application should allow the user to do the DNS test.
- The application should enable the user to get the information of the connected devices on the network.
- The application should enable the user to know the data usage of the applications.
- The application should enable the user to display the network logger.
- The application should allow the user to do a periodic scan for the network and notify them in case of intruders.
- The application should allow the user to block intruders.
- The application should allow the user to change WIFI password.
- The application should enable the user to setup router DNS.
- The application should allow the user to limit the speed of the router.

3.4 Use Cases

A Use Case Diagram is a graph that is used in system analysis to model system requirements. It is made of sequence of interactions between systems and user in a particular environment, the Use Case Diagrams that model the functional requirements of the proposed system are explained in following figures.

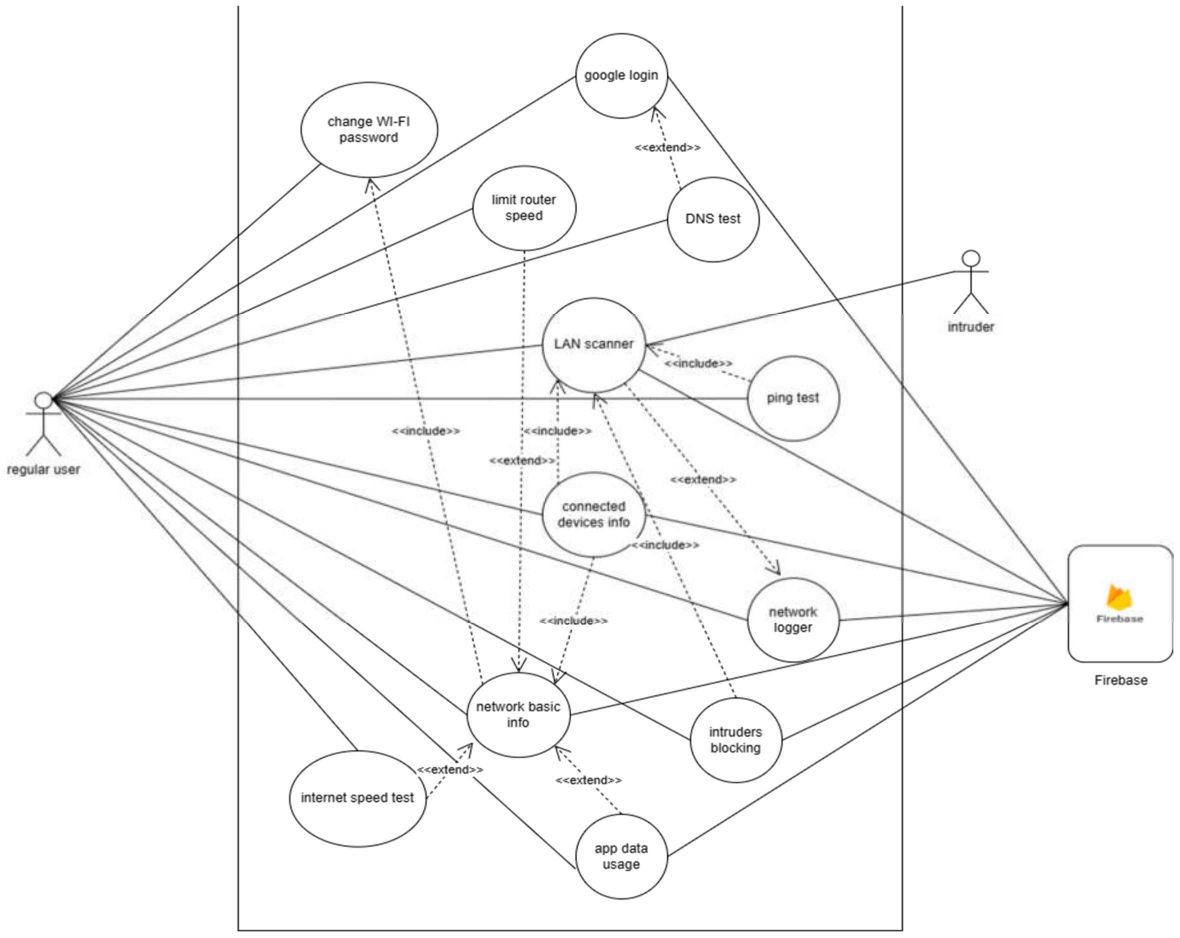


Figure 3.2: Pentor Use Case Diagram

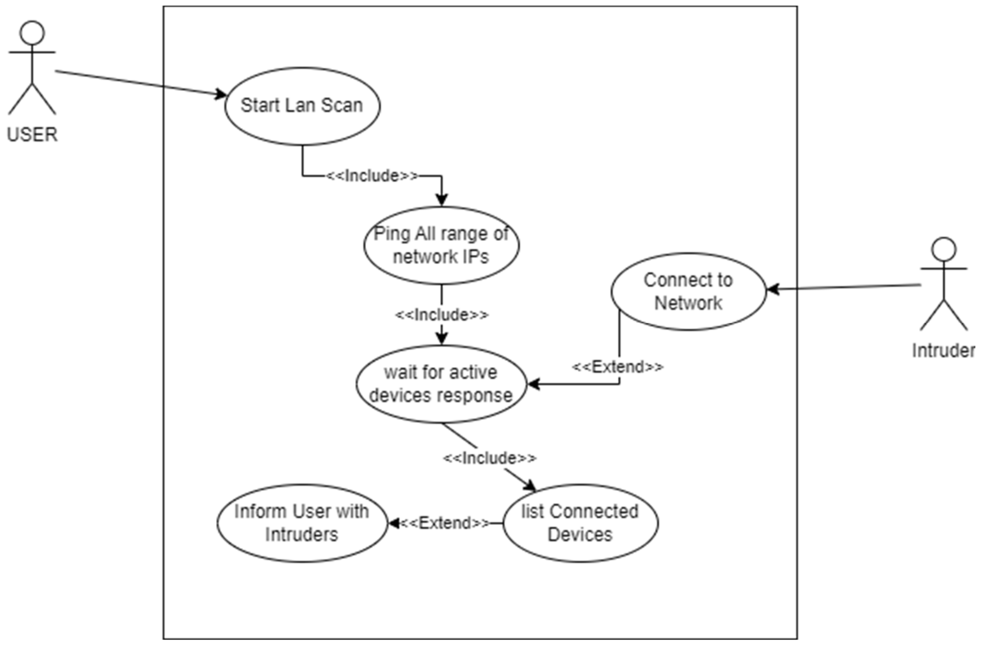


Figure 3.3: Lan Scanner Use Case Diagram

3.4.1 Lan Scanner Scenario

Table 2: Lan Scanner Use Case Scenario

	Scan Local Network
Actor(s)	User, connected devices to network
Description	Allow user to scan local network and determine connected devices.
Steps	1- User Opens the application. 2- Navigates to the Lan scanner page. 3- Press scan now button 4- Application performs scan on the whole range of IPs of the user network. 5- Show Results to the user
Preconditions	• Device Connected to Local Network Via WIFI • Application Granted Location Permissions
Assumptions	User understand basic knowledge of Devices types and what IP and MAC address terms mean

3.4.2 Google Login Scenario

Table 3: Google Login Use Case Scenario

	Sign In using Google Account
Actor(s)	User, Google API, Firebase API
Description	Allow user to login using Google account
Steps	1- User Opens the application 2- User Open App drawer 3- User Press Google Login Button
Preconditions	• Device Connected to Internet • User have Google Account
Assumptions	User have Google account, User want to save his network settings on cloud, google prefer OAuth more than usual email and password login.

3.4.3 DNS Test Scenario

Table 4: DNS Test Use Case Scenario

	Show DNS information
Actor(s)	User
Description	Allow user to get more information about given URL
Steps	1- Open app 2- Choose DNS Test service 3- Enter key word “URL” 4- Get the result

Preconditions	Internet connection required
Assumptions	User knowledge of DNS

3.4.4 Ping Test Scenario

Table 5: Ping Test Use Case Scenario

	Ping Test
Actor(s)	User
Description	Allow user to Test Ping for a specific Server
Steps	1. Open The app. 2. Choose Ping Test Page 3. Choose /Enter the Target 4. Press Ping 5. Get The Result
Preconditions	Internet connection required
Assumptions	User knowledge of Ping

3.4.5 Data usage Use Case Scenario

Table 6: Data Usage Use Case Scenario

	Data Usage
Actor(s)	User
Description	Allow user to Test Ping for a specific Server
Steps	1. Open The app. 2. Choose Data Usage Page 3. Select Data Usage /Wifi Usage 4. Select a period. 5. Get The Results 6. Show More Details
Preconditions	Mobile Data connection or Wifi connection
Assumptions	User can read Statistics

3.4.6 Network basic info Use Case Scenario

Table 7 : Network basic info Use Case Scenario

	Network basic info
Actor(s)	User
Description	Allow user to know basic information (like gatway ip, SSID name, Subnet, Broadcast ip and BSSID) about the local network connected to it.

Steps	1. Open The app. 2. Navigate to home Page. 3. Show network info Details
Preconditions	Device has Pentor installed. Device connected to Cellular or Wi-fi network
Assumptions	User understands what Ip, subnet and gateway means

3.4.7 Network Logger Use Case Scenario

Table 8 : Network Logger Use Case Scenario

	Network Logger
Actor(s)	User
Description	Allow user to view device network communication logs.
Steps	1. Open The app. 2. Choose Network Logger Page 3. Perform some (Get, Post, Delete …) requests. 4. Select log. 5. Show request and response details
Preconditions	Device connected to any network.
Assumptions	User understand what is post, get, delete requests.

3.4.8 Change Wi-fi password Use Case Scenario

Table 9 : Change Wi-fi password Use Case Scenario

	Change Wi-fi password
Actor(s)	User
Description	Allow user to change Wi-fi password
Steps	1. Open The app. 2. Choose Network Setting Page 3. Enter router username and password. 4. Enter Wi-fi password

Preconditions	Device connected to Wi-fi network that user wants to change its Wi-fi password.
Assumptions	User knows Router username and password

3.4.9 Block intruder Use Case Scenario

Table 10 : Block intruder Use Case Scenario

	Block intruder
Actor(s)	User
Description	Allow user to block intruders
Steps	1. Pentor performs periodic LAN scanning. 2. Notify User when new devices connect to the network. 3. User decides whether the devices is trusted or not Trusted. 4. Add Not trusted devices to router block list to prevent intruder from connecting to network.
Preconditions	Device connected to Wi-fi network. Pentor perform periodic Lan scanning and store trusted devices MAC addresses.
Assumptions	User is connected to the network most of time.

3.5 Non-functional requirements:

Non-functional requirements are the ones that define how the app must perform a certain function. They are the quality attributes of an application that define the user experience of the application.

3.5.1 Look and feel requirements

The application should be designed in a user-friendly way which allow ordinary users to deal with the application:

- The application will have quick responses and smooth navigation between different application interfaces.
- Graphical elements in the application interfaces like (buttons, icons...etc.) should be designed to show how they should be used.
- The fonts and buttons size and colors should be suitable and clear with the background, so they don't disturb the user.

- The application will support Multilanguage (English and Arabic).

3.5.2 Security requirements

- The user's data should be secured.
- All the transactions should be secured.
- Authentication and authorization should be implemented by default.

3.5.3 Privacy requirements

The user and network data should be protected, and only authorized user can get them.

3.5.4 Portability requirements

The application will be portable among Android, Windows, Linux and IOS platforms. It will be developed by dart programming language, flutter framework.

3.5.5 Availability requirements

The application will be free and can be downloaded from the suitable store for the platform online. It will be available for everyone without any annoying ads.

3.6 Summary:

In this chapter, we discussed the analysis phase of the system. We presented the system architecture and the general interaction between components. We also described the system requirements in terms of functional and non-functional requirements. The functionality was documented by the Use Case Diagram and Use Case Scenario. Finally, Non-functional requirements were stated with explanations.

Chapter 4
System Design

4.1 Introduction

System design is the process of designing the architecture and interfaces for a system so that it meets the end user requirements. This chapter is the software development stage. It is based on the detailed analysis of the system that was analyzed from the system analysis. It provides converting the requirements into tangible reality. It is a narrative and graphical description of the system design process including use case diagrams, class diagrams and sequence diagrams.

4.2 System Architecture

Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

4.2.1 Major Modules

The main components of the system are Mobile application and Database. The system consists of a mobile application, which is like the human brain, it serves all the requests coming from the client and handles the database connection and the operations, while it integrates with services which the application handles too. Through the mobile application you can access all features, and all the functions can only be accessed for registered users only. The mobile application is responsible for providing nice and easy GUI to use all the features.

4.2.2 Sub Modules

Other sub modules the system consists of, which can be swapped with others, are the external APIs and SDKs

- Google auth
- Fast.com
- Android SDK
- Flutter SDK
- GETX
- FIREBASE

4.2.3 Program Flow

The program starts when the user installs the application for the first time, and the home page is viewed from which he can navigate to all pages.

And the test flow is as following:

1. Network info

After the user logs into the system, they can display the information of their network. The system gets the required data from the database and displays it.

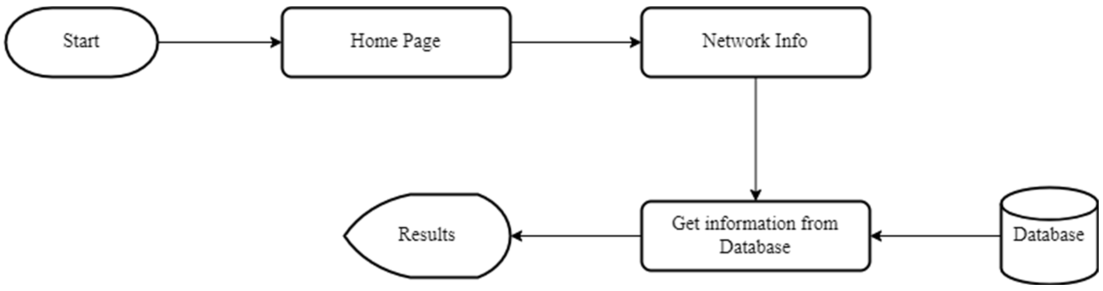


Figure 4.1 : Network info flow diagram

2. Lan Scanner

If the user wants to get a list of connected devices to the network, they can choose the Lan Scanner page and get details about it.

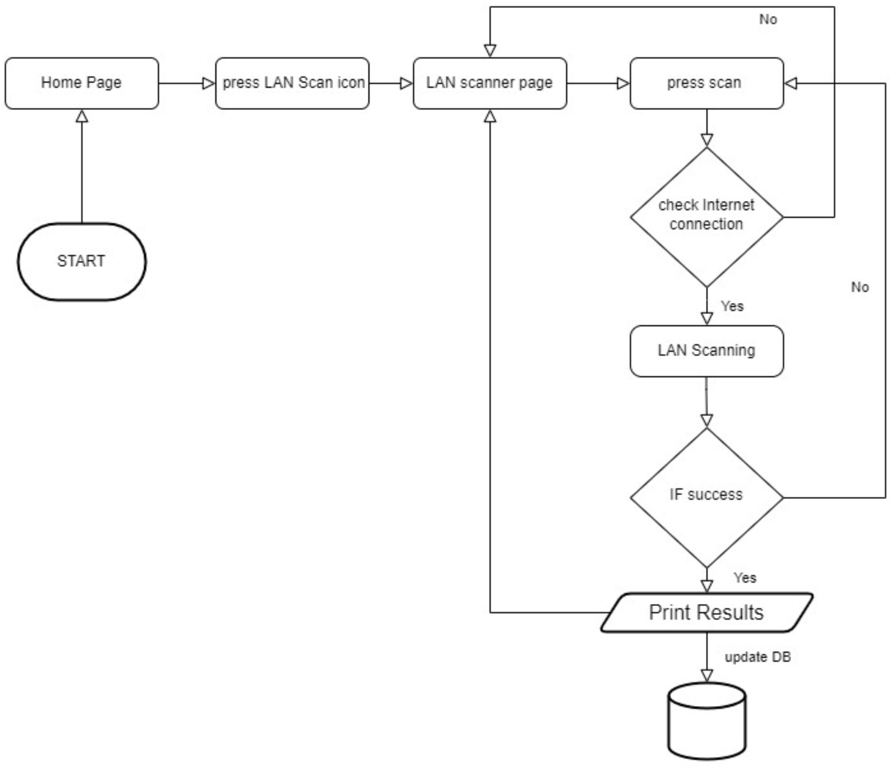


Figure 4.2 : Lan Scanner flow

3. DNS Test

If the user wants to get DNS data about certain IP, they can choose the DNS test page and get details about it.

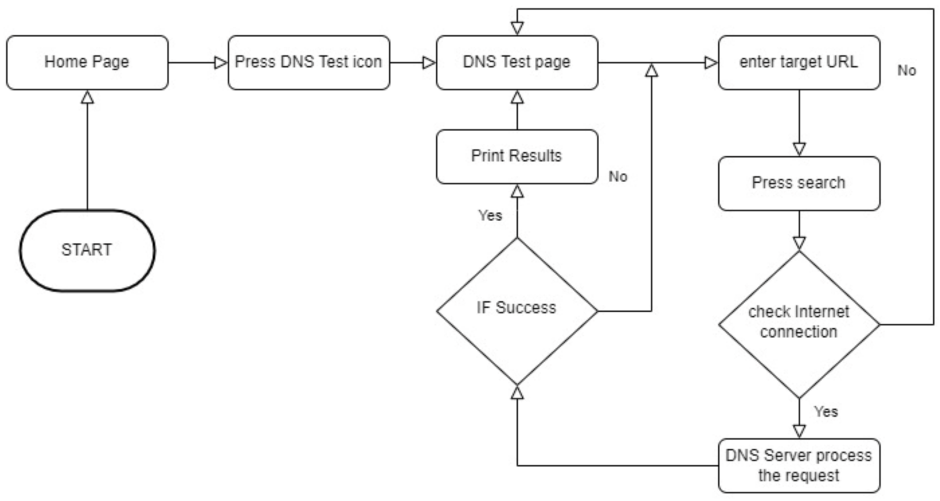


Figure 4.3 : DNS Test flow

4. Ping Test

If the user wants to start ping test, they can choose the Ping test page and get details about it.

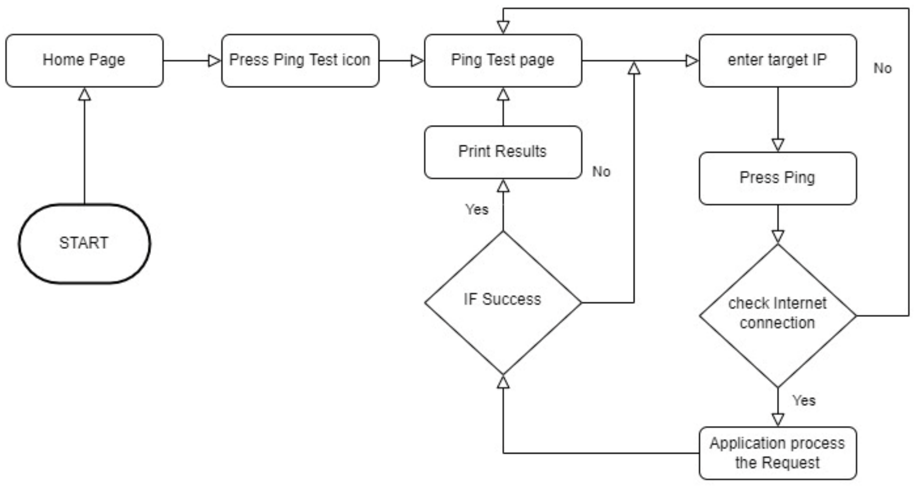


Figure 4.4 : Ping Test flow

5. Internet speed test:

The application enables the user to do the internet speed test. The user chooses the wanted speed either download or upload speed. Calculated speed is displayed.

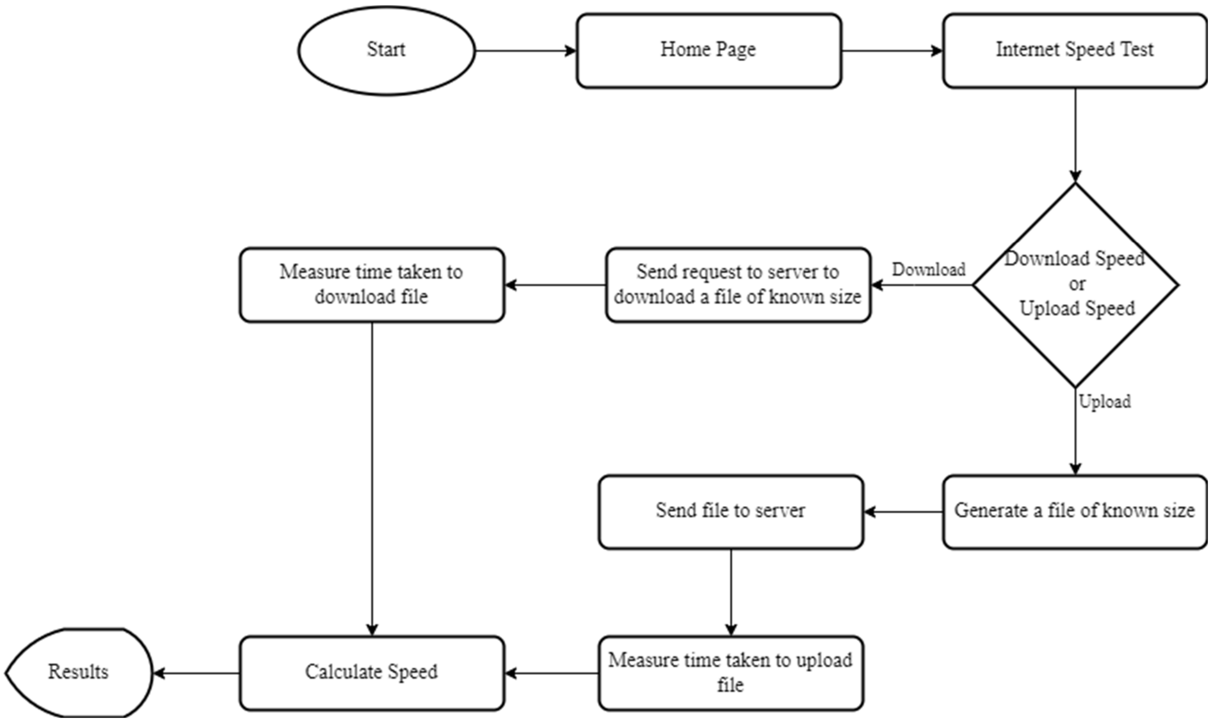

Start
Home Page
Internet Speed Test
Download Speed or Upload Speed
Download
Send request to server to download a file of known size
Measure time taken to download file
Upload
Generate a file of known size
Send file to server
Measure time taken to upload file
Calculate Speed
Results

Figure 4.5 : Speed test flow diagram

6. Apps data usage:

If the user wants to get the data usage for applications, they can choose the Data Usage page and get details about it.

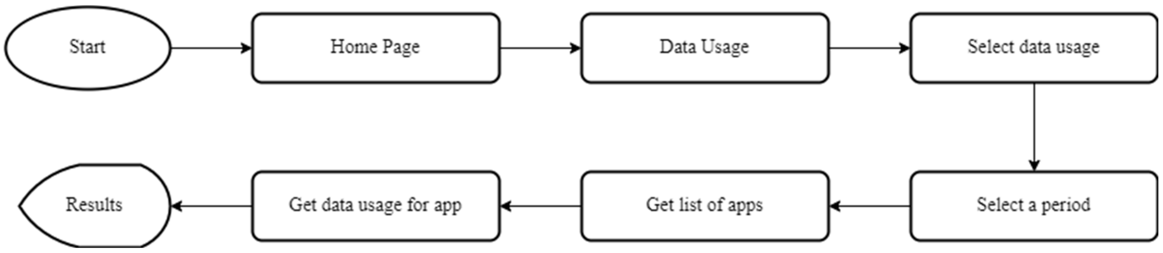


Figure 4.6 : Data usage flow diagram

4.2.4 Sequence Diagrams

Following are some of the important sequence diagrams in the project:

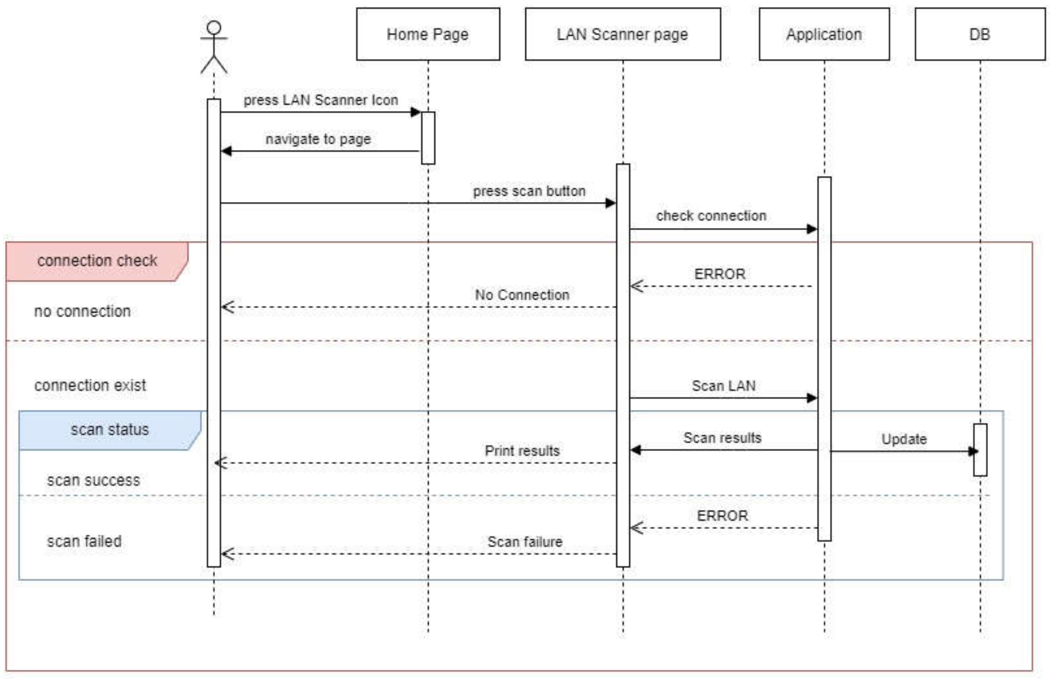


Figure 4.7 : Lan Scanner Sequence diagram

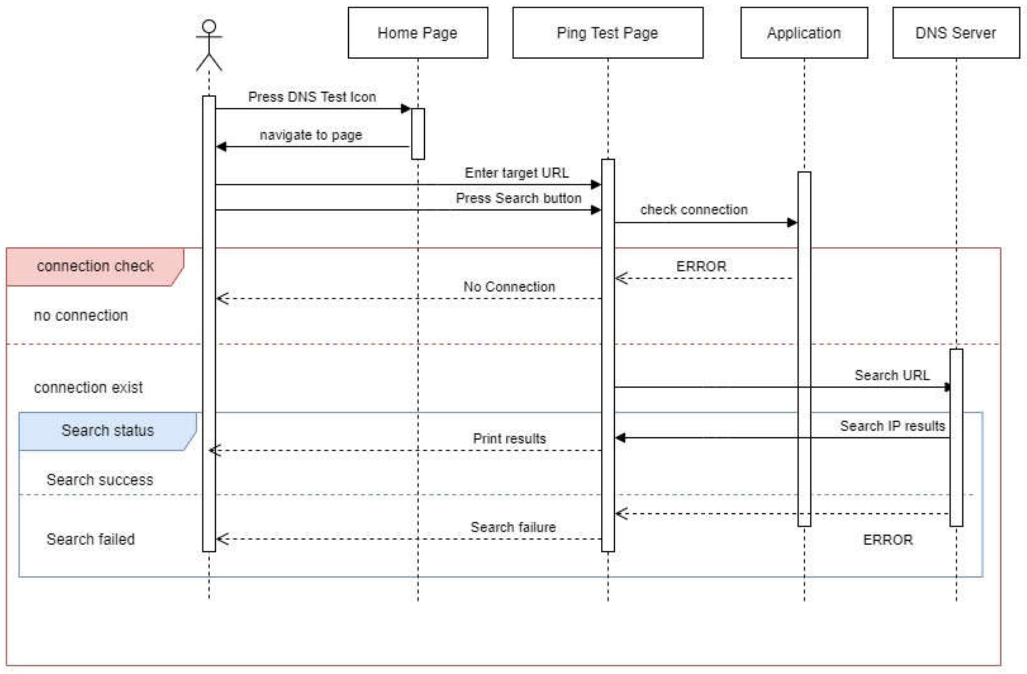


Figure 4.8 : DNS test Sequence Diagram

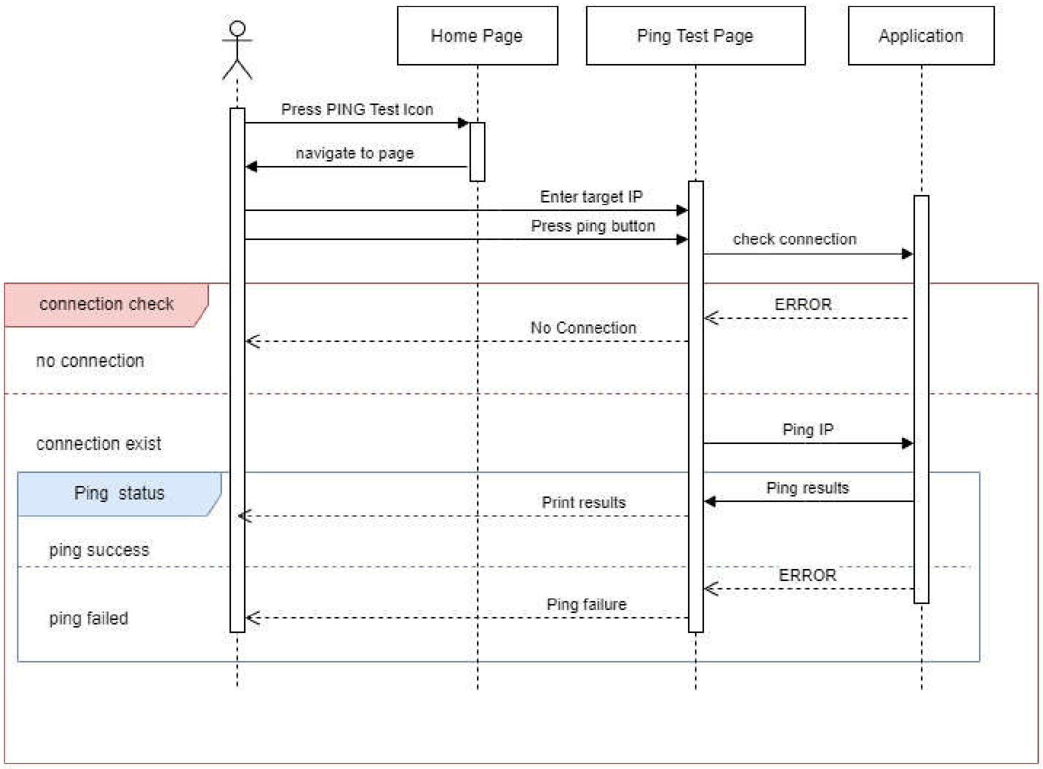


Figure 4.9 : Ping test Sequence diagram

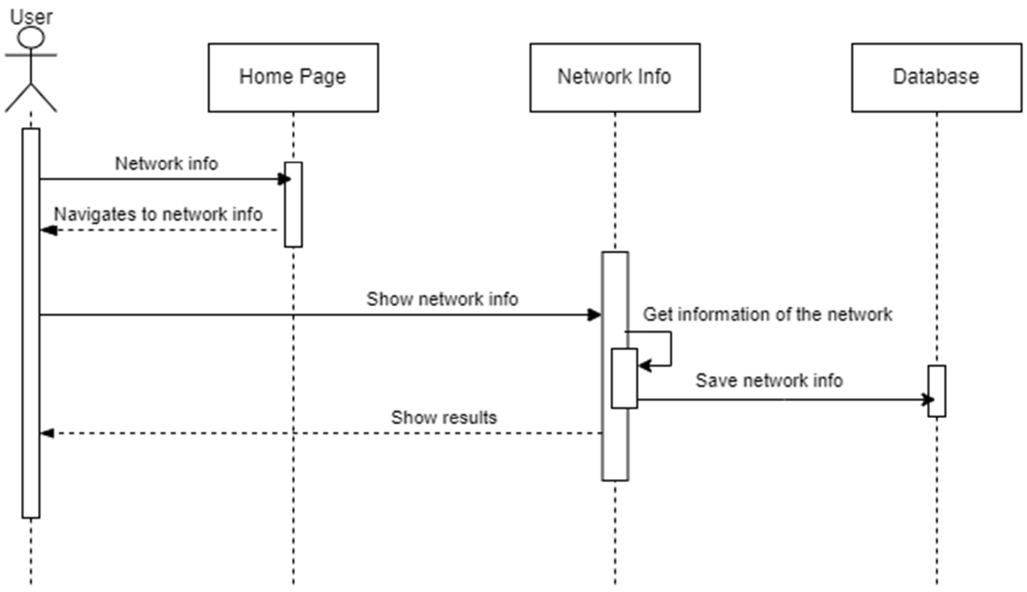


Figure 4.10 : Network info Sequence Diagram

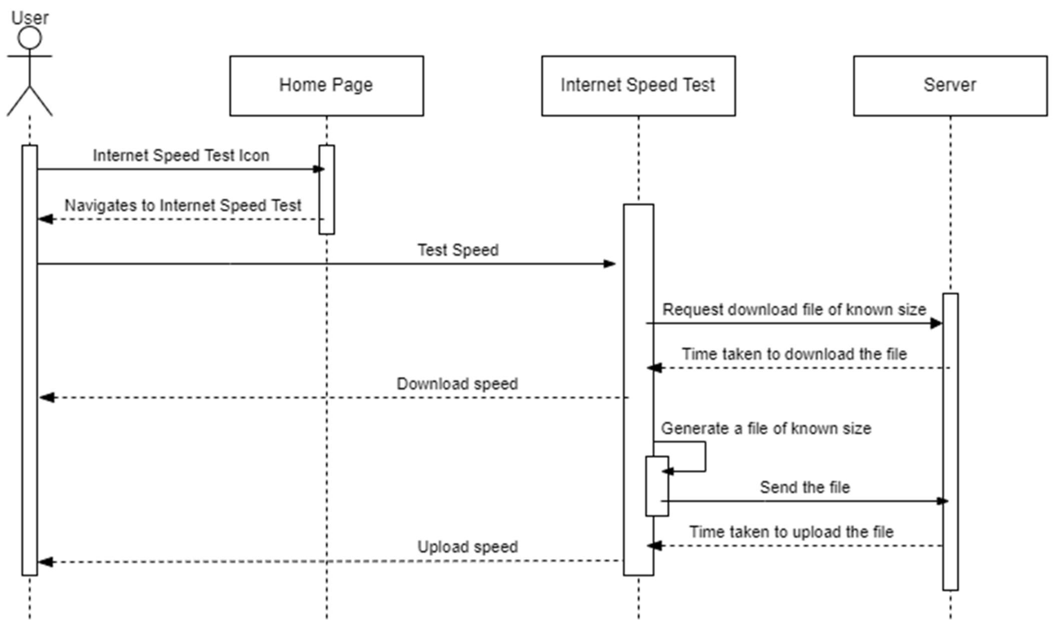

User
Home Page
Internet Speed Test
Server
Internet Speed Test Icon
Navigates to Internet Speed Test
Test Speed
Request download file of known size
Time taken to download the file
Download speed
Generate a file of known size
Send the file
Time taken to upload the file
Upload speed

Figure 4.11 : Speed Test Sequence Diagram

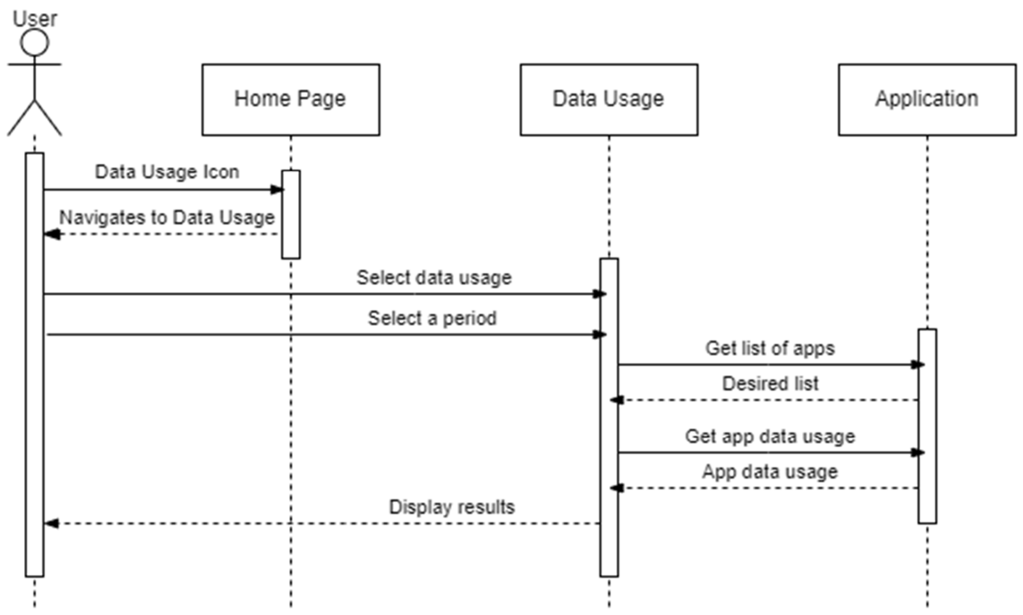


Figure 4.12 : Data Usage Sequence Diagram

4.3 Detailed System Design

4.3.1 User Stories

The user stories are used to define the business needs of the application, and it's required to determine the perfect design of the application.

1. **User**
   - User can sign up/in with Google account.
   - User can know The Devices connected on the network.
   - User can track Wi-Fi usage /Data usage for all applications installed on mobile.
   - User can make DNS test.
   - User can test download and upload Speed of the network.
   - User can ping the network.
   - User can show network logs.
   - User can choose the language of the app (Arabic / English).

2. **Network info**

   It is the action of displaying the information of the network to get more details about the connected network.

   To display the network info:

   1. User open the application.
   2. User navigate to the home page.
   3. The name of the connected network appears.
   4. User click on the network info arrow.
   5. Network information is displayed.

   Network info contains:

   1. The network name.
   2. IPv4 address.
   3. Subnet mask.
   4. IPv6 address.
   5. Gateway.
   6. Broadcast IP.

3. LAN Scanning

The act of scanning the Local Area Network to detect all the connected physical devices to this network helps to have good overview of the network and so it speeds troubleshooting.

In LAN Scanner page user can:

1. Know the number of and list connected devices.
2. Know the IP, MAC address and vendor of connected devices.
3. Set some devices as trusted.
4. Set some devices as intruder and block them.

To Scan a LAN user should:

1. Opens the application.
2. Navigates to the Lan scanner page.
3. Press scan now button.
4. Wait while the App perform the scan on the LAN.
5. Get the results.

4. DNS Testing

Is the act of translating domain names to IP addresses so browsers can load Internet resources and user getting more information about given URL.

In DNS Test page user can:

- ♦ Give a URL to translated and get associated IP address.

To test a DNS for URL user should:

1. Open app
2. Choose DNS Test service.
3. Enter key word "URL".
4. Press search button.
5. Wait while the app processes the request.
6. Get the result.

5. **Ping Testing**

   Is the act of check for a response from a host, Ping is also used to troubleshoot and test connectivity and determine response time.

   In Ping Test page user can:

   - Test and determine how fast a data signal travels from one place, like a computer, to another, like a website.

   To test a DNS for URL user should:

   1. Open app.
   2. Choose Ping Test service.
   3. Enter the target IP address.
   4. Press Ping button.
   5. Wait while the app processes the request.

6. **Internet speed test:**

   It is the action of testing the speed of the internet, the download speed and the upload speed.

   - The average household speed is 25 Mbps to perform basic functions like checking email, browsing, or streaming video.
   - The average internet download speed is between 12 and 25 Mbps.
   - The download speed that ranges from 3 to 8 Mbps is considered basic service, while advanced service exceeds 25 Mbps.
   - Upload speeds usually range from 1Mbps to 15 Mbps.
   - The minimum upload speed for the fast internet is 3Mbps.

   To test the internet speed:

   1. User open the application.
   2. User choose internet speed test.
   3. User click on test button.
   4. Internet speed is displayed.

   Internet speed test page contains:

   1. An indicator that indicates the speed.
   2. Download speed.
   3. Upload speed.

7. **Data usage:**

   It is the action of calculating the data used by applications on the mobile. User must keep track of their data usage to save data as possible.

   To get apps data usage:

   1. User opens the application.
   2. User chooses data usage.
   3. Apps data usage is displayed.

   Data usage page contains:

   1. The whole data used by all applications.
   2. Applications found using data.
   3. Data used by each application.

4.3.2 Class Diagram

A class diagram is a type of UML diagram that provides a visual representation of the classes, interfaces, associations, and other components of a software system, along with their relationships and dependencies. It is a valuable tool for software developers and architects as it helps to visualize the structure of a system and the interactions between its various components.

A class diagram typically includes classes that represent the data and logic of the system, interfaces that define the methods that classes must provide, and relationships that describe how these classes and interfaces are related to each other. These relationships may include inheritance, composition, aggregation, and association.

In this section, we will present the detailed class diagram to show all the methods and the attributes. In Figure 32, shows the project class diagram.

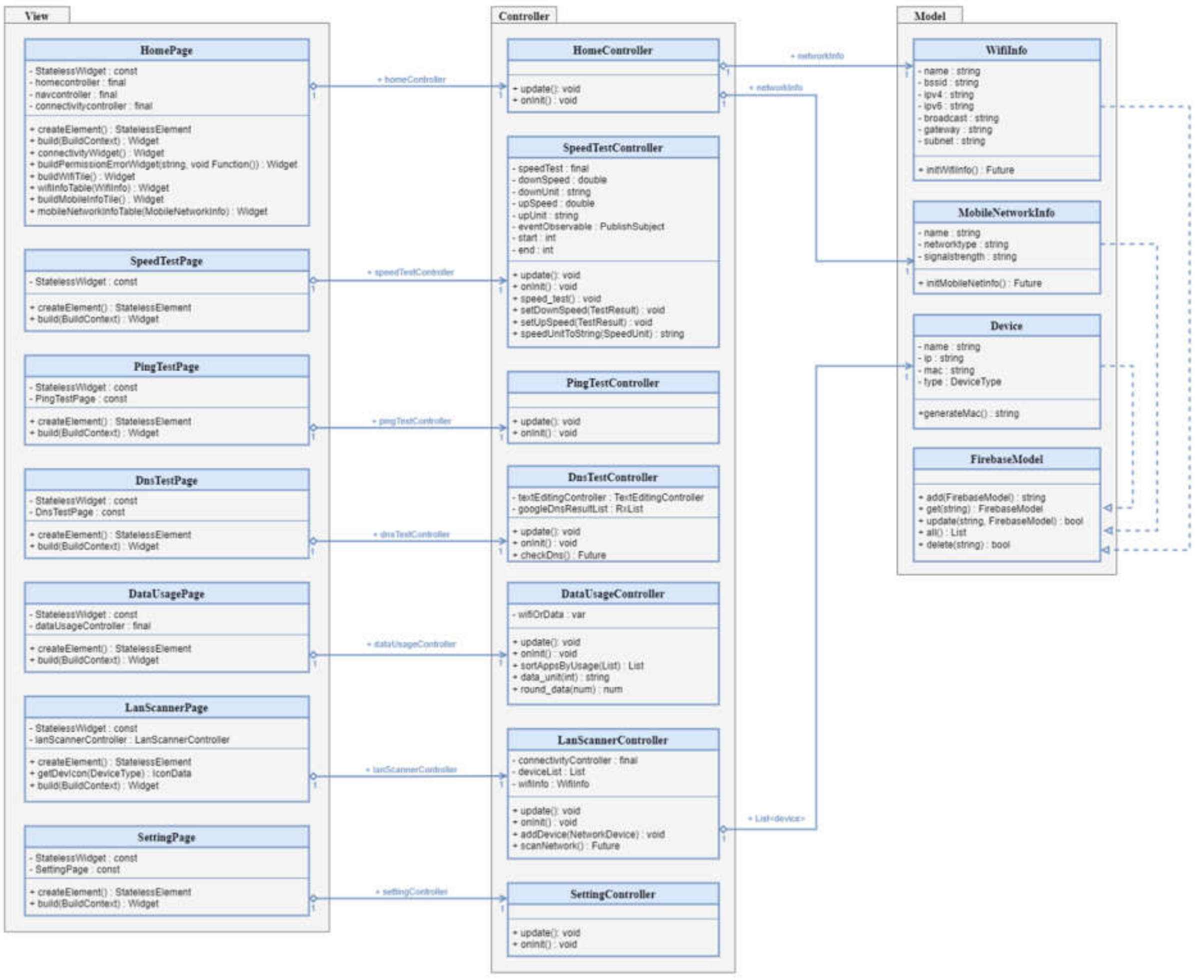

View
HomePage
- StatelessWidget : const
- homecontroller : final
- navcontroller : final
- connectivitycontroller : final
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
+ connectivityWidget() : Widget
+ buildPermissionErrorWidget(string, void Function()) : Widget
+ buildWifiTile() : Widget
+ wifiInfoTable(WifiInfo) : Widget
+ buildMobileInfoTile() : Widget
+ mobileNetworkInfoTable(MobileNetworkInfo) : Widget
SpeedTestPage
- StatelessWidget : const
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
PingTestPage
- StatelessWidget : const
- PingTestPage : const
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
DnsTestPage
- StatelessWidget : const
- DnsTestPage : const
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
DataUsagePage
- StatelessWidget : const
- dataUsageController : final
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
LanScannerPage
- StatelessWidget : const
- lanScannerController : LanScannerController
+ createElement() : StatelessElement
+ getDevIcon(DeviceType) : IconData
+ build(BuildContext) : Widget
SettingPage
- StatelessWidget : const
- SettingPage : const
+ createElement() : StatelessElement
+ build(BuildContext) : Widget
+ homeController
+ speedTestController
+ pingTestController
+ dnsTestController
+ dataUsageController
+ lanScannerController
+ settingController
Controller
HomeController
+ update(): void
+ onInit() : void
SpeedTestController
- speedTest : final
- downSpeed : double
- downUnit : string
- upSpeed : double
- upUnit : string
- eventObservable : PublishSubject
- start : int
- end : int
+ update(): void
+ onInit() : void
+ speed_test() : void
+ setDownSpeed(TestResult) : void
+ setUpSpeed(TestResult) : void
+ speedUnitToString(SpeedUnit) : string
PingTestController
+ update(): void
+ onInit() : void
DnsTestController
- textEditingController : TextEditingController
- googleDnsResultList : RxList
+ update(): void
+ onInit() : void
+ checkDns() : Future
DataUsageController
- wifiOrData : var
+ update(): void
+ onInit() : void
+ sortAppsByUsage(List) : List
+ data_unit(int) : string
+ round_data(num) : num
LanScannerController
- connectivityController : final
- deviceList : List
- wifiInfo : WifiInfo
+ update(): void
+ onInit() : void
+ addDevice(NetworkDevice) : void
+ scanNetwork() : Future
SettingController
+ update(): void
+ onInit() : void
+ networkInfo
+ networkInfo
+ List<device>
Model
WifiInfo
- name : string
- bssid : string
- ipv4 : string
- ipv6 : string
- broadcast : string
- gateway : string
- subnet : string
+ initWifiInfo() : Future
MobileNetworkInfo
- name : string
- networktype : string
- signalstrength : string
+ initMobileNetinfo() : Future
Device
- name : string
- ip : string
- mac : string
- type : DeviceType
+generateMac() : string
FirebaseModel
+ add(FirebaseModel) : string
+ get(string) : FirebaseModel
+ update(string, FirebaseModel) : bool
+ all() : List
+ delete(string) : bool

Figure 4.13 : Pentor Class diagram

The Pentor class diagram implementing the Model-View-Controller (MVC) pattern provides a visual representation of how the various components of the Pentor software system interact with each other according to the MVC architecture.

The diagram includes three main components: the Model, the View, and the Controller. The Model includes classes such as WifiInfo, FirebaseModel, MobileNetworkInfo, and Device, which represent the data and logic of Pentor app. The View includes classes such as HomePage, LanScannerPage, and DNSPage, which represent the user interface components of Pentor app. The Controller includes classes such as HomeController, LanScannerController, and DNSController, which handle user input and manage communication between the Model and View components.

4.3.3 Database Design

The database used in NoSQL is not a typical relational database and has a different way of storing relationship data. Despite this difference, it is still capable of storing related data, albeit in a different way than what is seen in relational databases. Interestingly, some people find modeling relationship data in NoSQL databases to be more straightforward than in relational databases because NoSQL data models allow related data to be nested within a single data structure, unlike relational databases where related data needs to be split between tables.

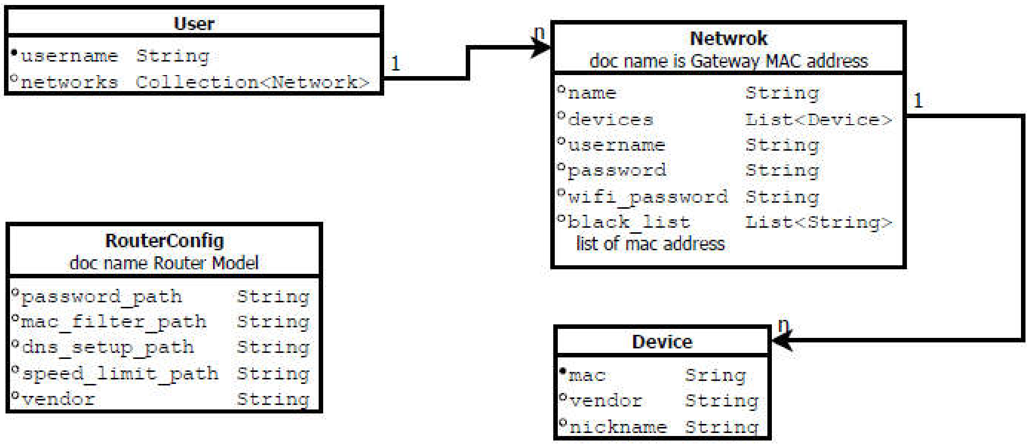


Figure 4.14 : Database ERD

Table 11 : ERD attributes description

Class	attribute	Type	Description
User	username	String	A string to be used as a username
	networks	Collection<Network>	A collection of networks user has connected to.
Network	name	String	SSID of network
	devices	List<Device>	List of devices user has detected by LAN scanning
	username	String	Router username used in changing wifi password
	password	String	Router password used in changing wifi password
	wifi_password	String	Password of Wifi network user connected to
	black_list	List<Device>	List of intruder devices the user listed as untrusted or blocked

Device	mac	String	MAC address of the device
	vendor	String	Vendor of the device
	nickname	String	Nickname user can set to identify the device
Router Config	password_path	String	Path to request Wifi password change
	mac_filter_path	String	Path to add, delete or update router mac filter list
	Dns_setup_path	String	Path to setup router dns
	Speed_limit_path	String	Path to setup bandwidth limit for router
	vendor	String	Router vendor

4.3.4 Interfaces

User interfaces are a crucial aspect of software design, as they provide the means for users to interact with and control software systems. A well-designed user interface can greatly enhance the user experience, making software more intuitive and efficient to use. Following is a list of

UI components of the application:

1. **Home page**
   The home page UI provides users with an at-a-glance view of their network information and easy access to other sections of the application. The home page features a network info section that displays key details about the user's network, such as network SSID, connection type (Wi-fi or cellular) and Gateway IP.
   In addition to the network info section, the home page also includes a navigation grid that provides links to other pages of the application. This includes links to features such as Data usage, DNS test, and Application settings. By clicking on these links, users can quickly navigate to different areas of the application and access the information they need.

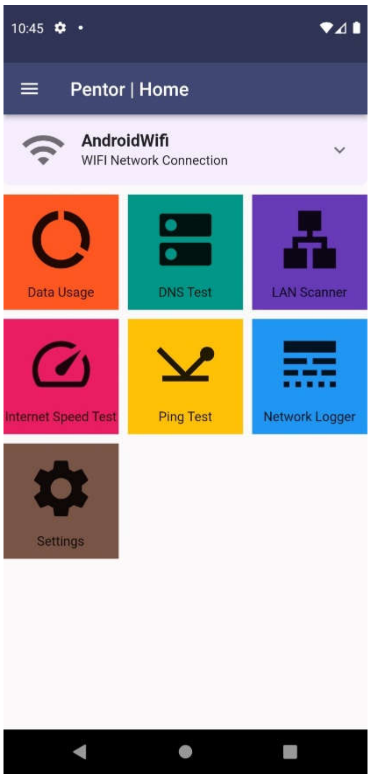


Figure 4.15 : Home page - EN

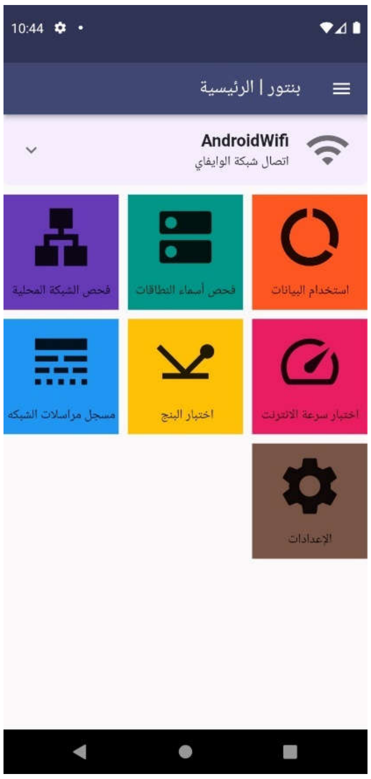


Figure 4.16 : Home page - AR

2. **Navigation Drawer**

The navigation drawer UI provides a convenient and intuitive way for users to access different pages of the application. The navigation drawer is located on the left side (or to the right in case of the application is running in Arabic language) of the screen and can be opened by swiping or clicking on the menu icon.

The navigation drawer contains two main sections: the account info section and the navigation list. The account info section displays the user's profile picture and name, as well as any other relevant account information. This provides users with quick and easy access to their account details.

The navigation list contains links to different pages within the Pentor application. This includes links to features such as Lan Scanner, Internet Speed test and Application Settings. By clicking on these links, users can quickly access the information they need and navigate to different areas of the application.

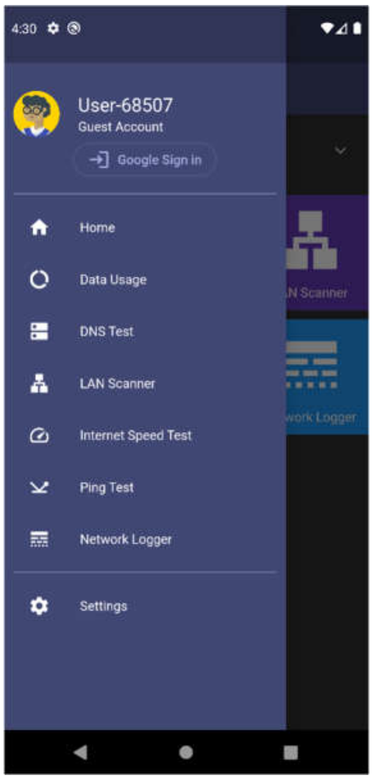


Figure 4.17 : Navigation Drawer – EN

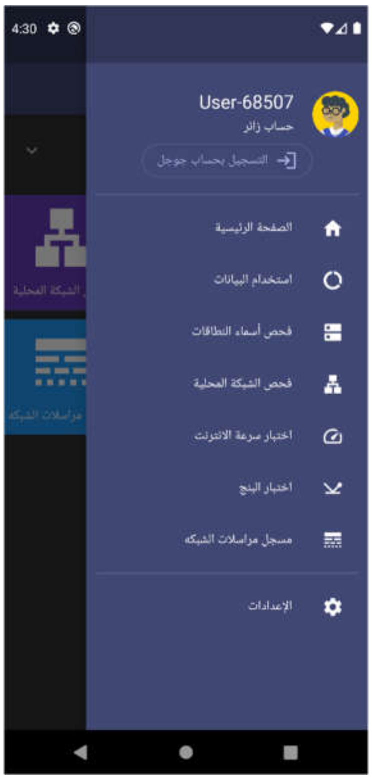
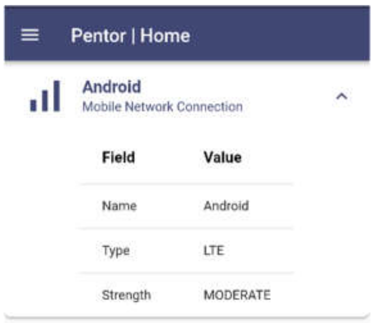


Figure 4.18 : Navigation Drawer - AR

3. Network info Section

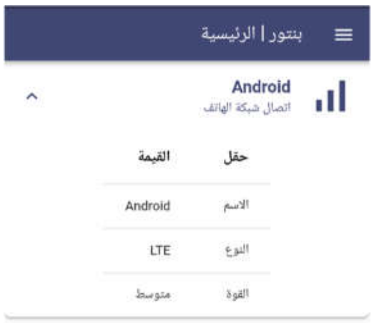
The network info section UI provides users with an overview of their network details in a clear and concise format. This section displays key information such as the network's SSID, gateway IP, and device IP. This information is essential for users to understand the state of their network.

The network SSID is displayed prominently at the top of the section, providing users with a quick and easy way to identify which network they are currently connected to. The gateway IP and device IP are also displayed, giving users the ability to get more insights to the network he is connected to.



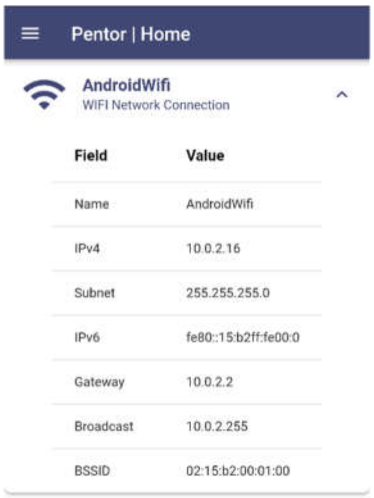
Field	Value
Name	Android
Type	LTE
Strength	MODERATE

Figure 4.19 : Network info section mobile data connection - EN



حقل	القيمة
الاسم	Android
النوع	LTE
القوة	متوسط

Figure 4.20 : Network info section mobile data connection - AR

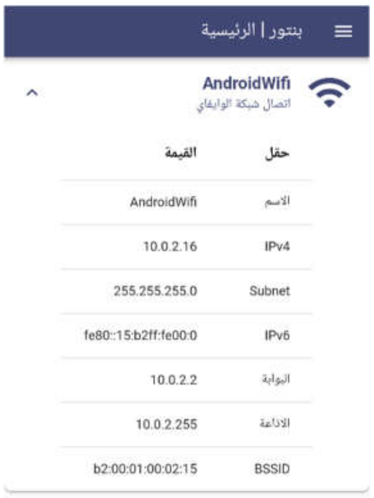


Pentor | Home

AndroidWifi
WIFI Network Connection

Field	Value
Name	AndroidWifi
IPv4	10.0.2.16
Subnet	255.255.255.0
IPv6	fe80::15:b2ff:fe00:0
Gateway	10.0.2.2
Broadcast	10.0.2.255
BSSID	02:15:b2:00:01:00

Figure 4.21 : Network info section wifi Connection - EN



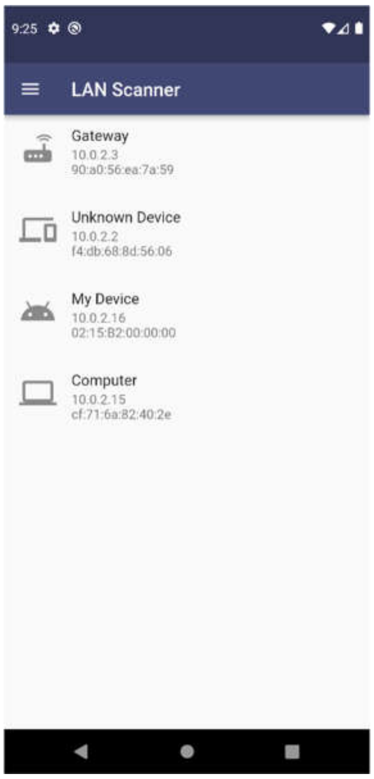
بنتور | الرئيسية

AndroidWifi
اتصال شبكة الوايفاي

حقل	القيمة
الاسم	AndroidWifi
IPv4	10.0.2.16
Subnet	255.255.255.0
IPv6	fe80::15:b2ff:fe00:0
البوابة	10.0.2.2
الاذاعة	10.0.2.255
BSSID	b2:00:01:00:02:15

Figure 4.22 : Network info section wifi Connection - AR

4. **Lan Scanner page**
The LAN scanner page UI provides users with a comprehensive tool for scanning and identifying devices on their local network. The page features a clean and intuitive interface, allowing users to easily start a new scan and view the results.
Users can initiate a new scan by clicking the "Start Scan" button, which triggers a search for all devices connected to the local network. The scan results are displayed in a clear and organized format, with each device listed with its corresponding IP address and MAC address.



9:25

LAN Scanner

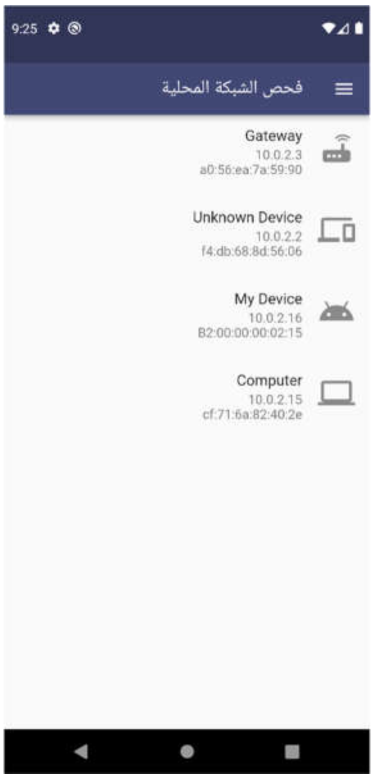
Gateway
10.0.2.3
90:a0:56:ea:7a:59

Unknown Device
10.0.2.2
f4:db:68:8d:56:06

My Device
10.0.2.16
02:15:B2:00:00:00

Computer
10.0.2.15
cf:71:6a:82:40:2e

Figure 4.23 : : Lan Scanner - EN



9:25

فحص الشبكة المحلية

Gateway
10.0.2.3
a0:56:ea:7a:59:90

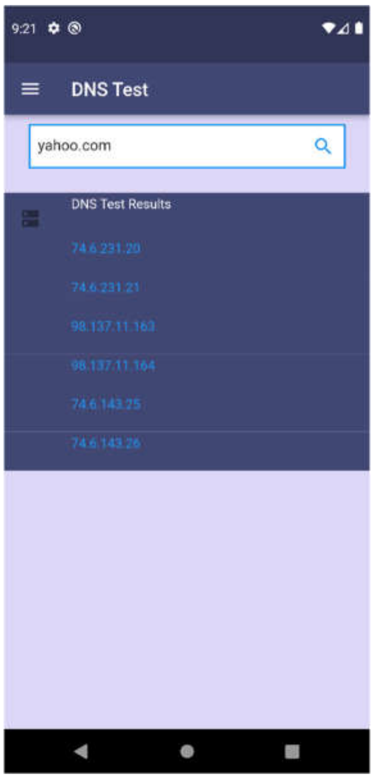
Unknown Device
10.0.2.2
f4:db:68:8d:56:06

My Device
10.0.2.16
B2:00:00:00:02:15

Computer
10.0.2.15
cf:71:6a:82:40:2e

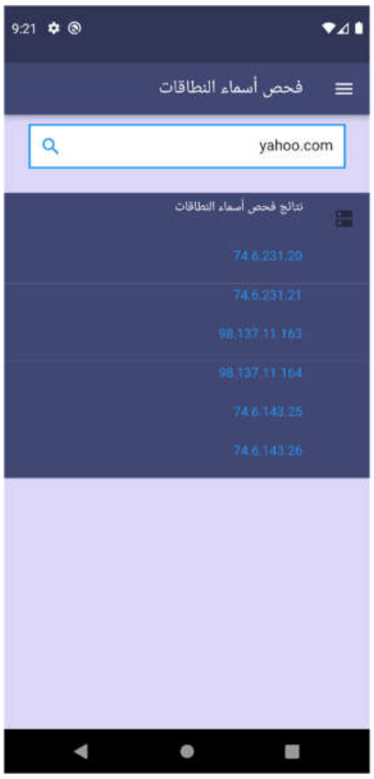
Figure 4.24 : Lan Scanner - AR

5. **DNS test page**
The DNS test page UI provides users with a tool for testing the performance and reliability of their DNS servers. The page features a simple and easy-to-use interface that allows users to quickly run a DNS test and view the results.
To initiate a test, users can simply enter the domain name they want to test and select the DNS server they want to use. The test results are displayed in a clear and organized format.



```
74.6.231.20
74.6.231.21
98.137.11.163
98.137.11.164
74.6.143.25
74.6.143.26
```

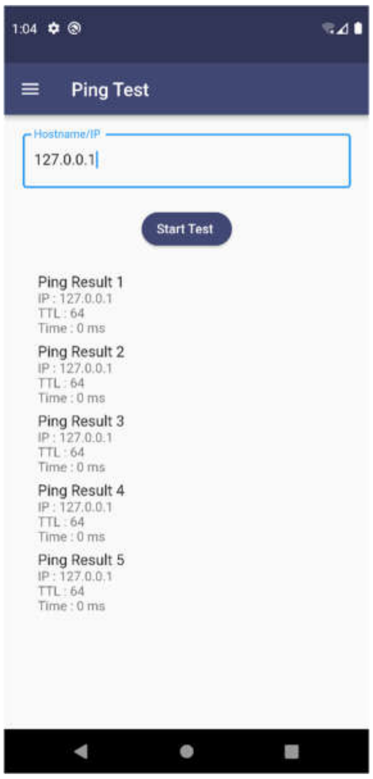
Figure 4.25 : DNS Test - EN



```
74.6.231.20
74.6.231.21
98.137.11.163
98.137.11.164
74.6.143.25
74.6.143.26
```

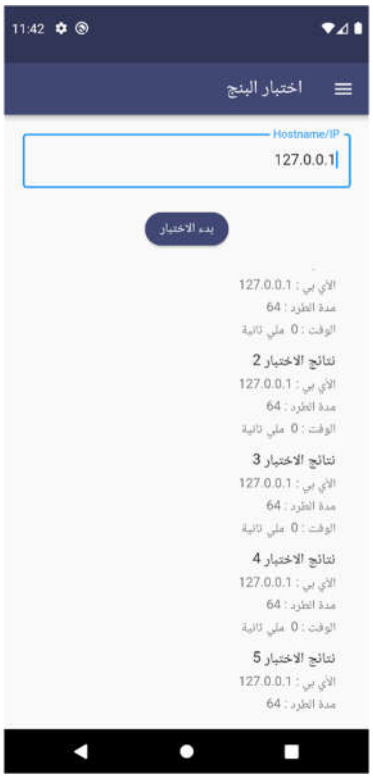
Figure 4.26 : DNS Test - AR

6. **Ping Test page**
The ping test page UI provides users with a tool for testing the connectivity and response time of their network devices. The page features a straightforward and user-friendly interface that allows users to easily perform a ping test and view the results.
To initiate a ping test, users can enter the IP address or domain name of the device they want to test. The test results are displayed in a clear and organized format, showing the response time and status for each packet sent.



127.0.0.1

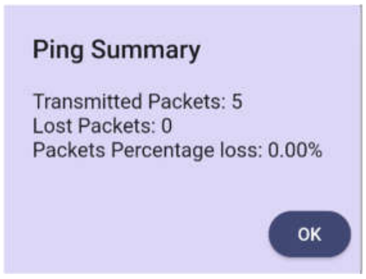
Ping Result 1
IP : 127.0.0.1
TTL : 64
Time : 0 ms
Ping Result 2
IP : 127.0.0.1
TTL : 64
Time : 0 ms
Ping Result 3
IP : 127.0.0.1
TTL : 64
Time : 0 ms
Ping Result 4
IP : 127.0.0.1
TTL : 64
Time : 0 ms
Ping Result 5
IP : 127.0.0.1
TTL : 64
Time : 0 ms



127.0.0.1

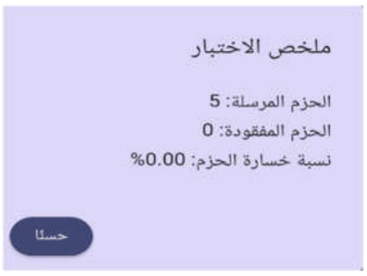
الآي بي : 127.0.0.1
مدة الطرد : 64
الوقت : 0 ملي ثانية
نتائج الاختبار 2
الآي بي : 127.0.0.1
مدة الطرد : 64
الوقت : 0 ملي ثانية
نتائج الاختبار 3
الآي بي : 127.0.0.1
مدة الطرد : 64
الوقت : 0 ملي ثانية
نتائج الاختبار 4
الآي بي : 127.0.0.1
مدة الطرد : 64
الوقت : 0 ملي ثانية
نتائج الاختبار 5
الآي بي : 127.0.0.1
مدة الطرد : 64

Figure 4.27 : Ping Test - EN *Figure 4.28 : Ping Test – AR*



Ping Summary

Transmitted Packets: 5
Lost Packets: 0
Packets Percentage loss: 0.00%



ملخص الاختبار

الحزم المرسلة: 5
الحزم المفقودة: 0
نسبة خسارة الحزم: 0.00%

Figure 4.29 : Ping Summary - EN *Figure 4.30 : Ping Summary - AR*

7. **Speed test page**
   The internet speed test page UI provides users with a tool for measuring the speed and performance of their internet connection. The page features a simple and user-friendly interface that allows users to easily perform a speed test and view the results.
   To initiate a speed test, users can simply click the "Start Test" button on the page, and the test will automatically begin. The test measures both the download and upload speeds of the user's internet connection, and the results are displayed in a clear and organized format.

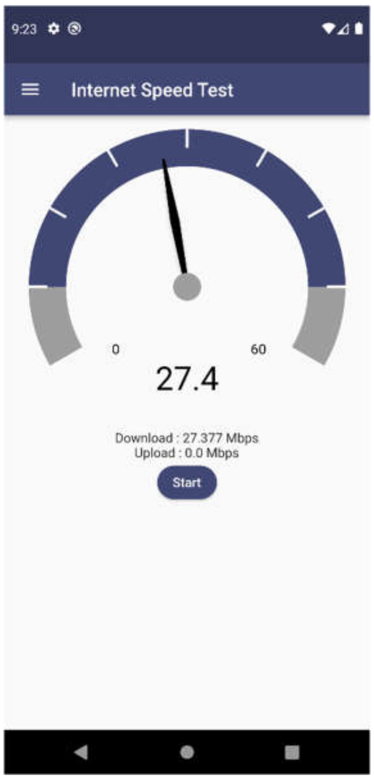


Figure 4.31 : Speed Test - EN

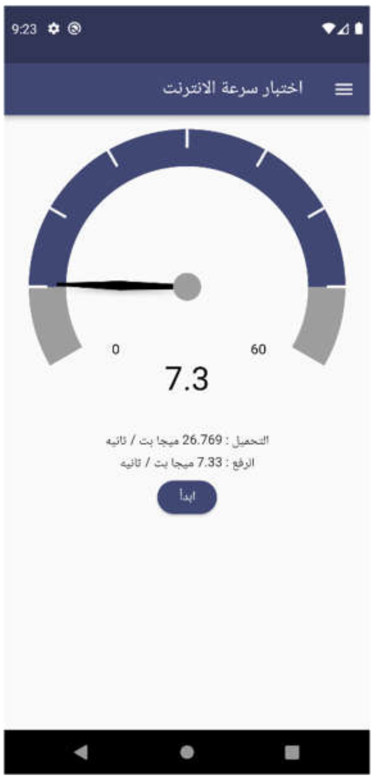
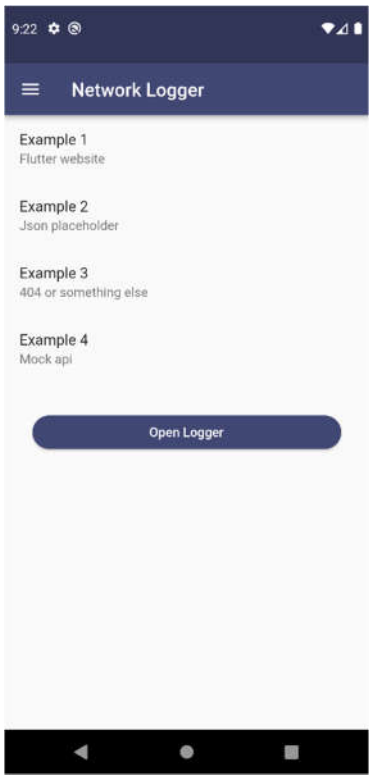

9:23
اختبار سرعة الانترنت
0
60
7.3
التحميل : 26.769 ميجا بت / ثانيه
الرفع : 7.33 ميجا بت / ثانيه
ابدأ

Figure 4.32 : Speed Test - AR

8. **Network Logger Page**

   The network logger page UI provides users with a tool for monitoring and logging network activity. The page features a clean and organized interface that allows users to easily view and analyze network traffic.

   The network logger page displays a detailed log of network activity, including information about the source and destination IP addresses, protocol type, and data size. Users can filter and sort the log data based on various criteria, such as source IP address or protocol type, to quickly identify specific types of network activity.



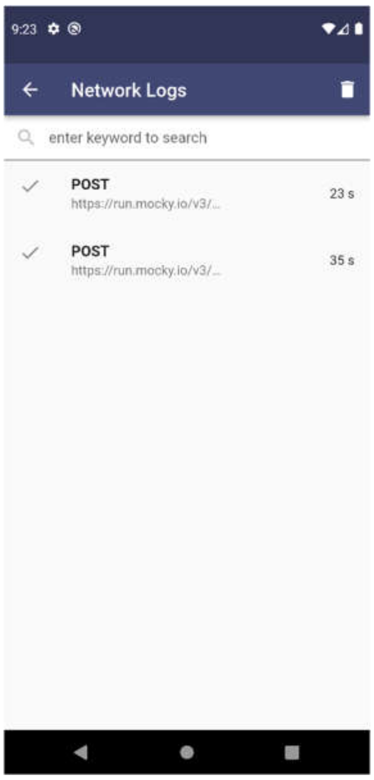
Example 1
Flutter website

Example 2
Json placeholder

Example 3
404 or something else

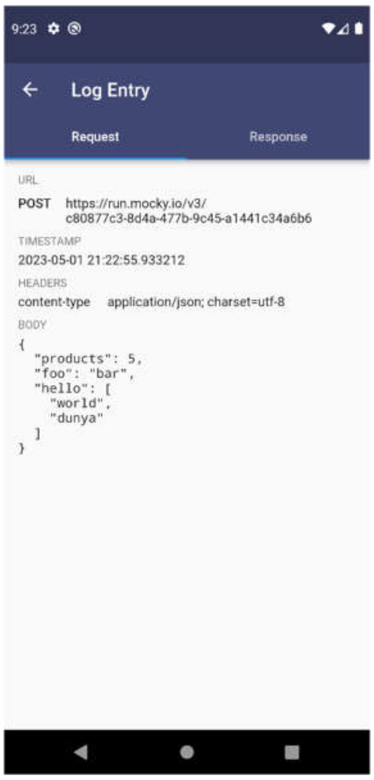
Example 4
Mock api

Figure 4.33: Network Logger page



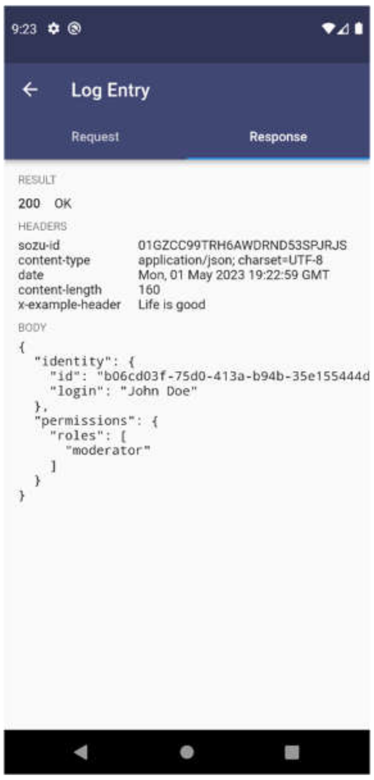
```
POST  https://run.mocky.io/v3/...   23 s
POST  https://run.mocky.io/v3/...   35 s
```

Figure 4.34 : Network logs list



```
URL
POST   https://run.mocky.io/v3/
       c80877c3-8d4a-477b-9c45-a1441c34a6b6
TIMESTAMP
2023-05-01 21:22:55.933212
HEADERS
content-type    application/json; charset=utf-8
BODY
{
  "products": 5,
  "foo": "bar",
  "hello": [
    "world",
    "dunya"
  ]
}
```

Figure 4.35 : Log Entry Request



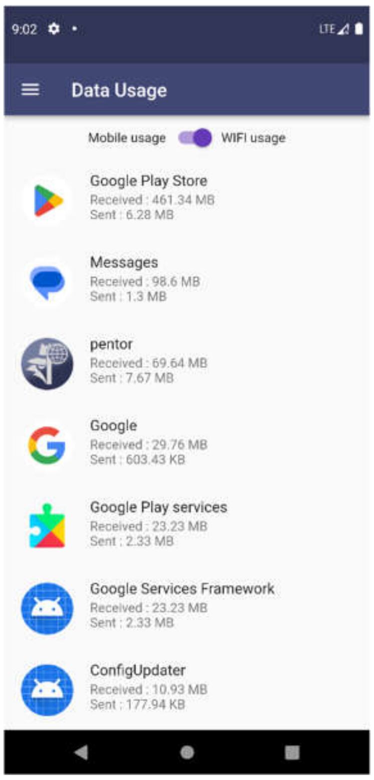
```
RESULT
200   OK
HEADERS
sozu-id            01GZCC99TRH6AWDRND53SPJRJS
content-type       application/json; charset=UTF-8
date               Mon, 01 May 2023 19:22:59 GMT
content-length     160
x-example-header   Life is good
BODY
{
  "identity": {
    "id": "b06cd03f-75d0-413a-b94b-35e155444d
    "login": "John Doe"
  },
  "permissions": {
    "roles": [
      "moderator"
    ]
  }
}
```

Figure 4.36 : Log entry response

9. Data Usage Page

The data usage page UI provides users with a tool for tracking and monitoring their internet data usage. The page features a clean and organized interface that allows users to easily view their installed applications’ data usage.

The data usage page displays a detailed breakdown of the user's current data usage, including each application received and sent data sorted in descending order by received then sent data.



Google Play Store
Received : 461.34 MB
Sent : 6.28 MB

Messages
Received : 98.6 MB
Sent : 1.3 MB

pentor
Received : 69.64 MB
Sent : 7.67 MB

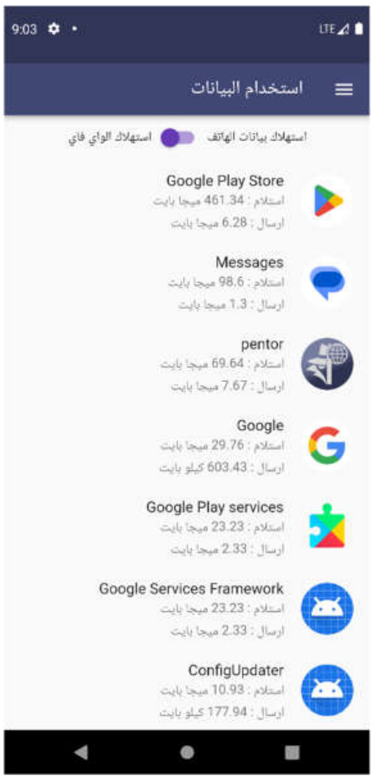
Google
Received : 29.76 MB
Sent : 603.43 KB

Google Play services
Received : 23.23 MB
Sent : 2.33 MB

Google Services Framework
Received : 23.23 MB
Sent : 2.33 MB

ConfigUpdater
Received : 10.93 MB
Sent : 177.94 KB

Figure 4.37 : Data usage - EN



Google Play Store
استلام : 461.34 ميجا بايت
ارسال : 6.28 ميجا بايت

Messages
استلام : 98.6 ميجا بايت
ارسال : 1.3 ميجا بايت

pentor
استلام : 69.64 ميجا بايت
ارسال : 7.67 ميجا بايت

Google
استلام : 29.76 ميجا بايت
ارسال : 603.43 كيلو بايت

Google Play services
استلام : 23.23 ميجا بايت
ارسال : 2.33 ميجا بايت

Google Services Framework
استلام : 23.23 ميجا بايت
ارسال : 2.33 ميجا بايت

ConfigUpdater
استلام : 10.93 ميجا بايت
ارسال : 177.94 كيلو بايت

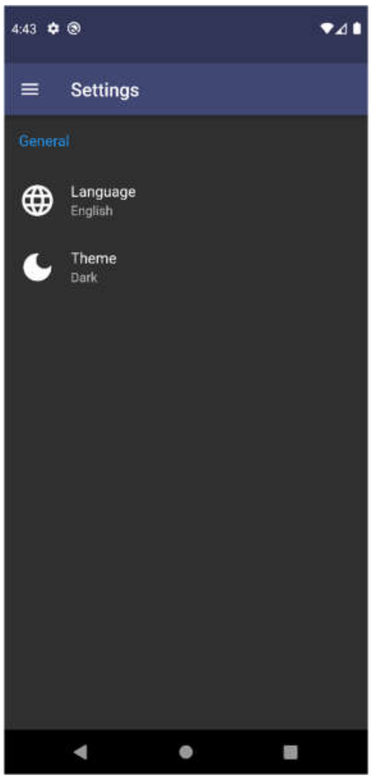
Figure 4.38 : Data usage - AR

10. Settings Page

The Pentor settings page UI provides users with a tool for customizing their Pentor app preferences and configurations. The page features a clean and organized interface that allows users to easily navigate and modify their settings.

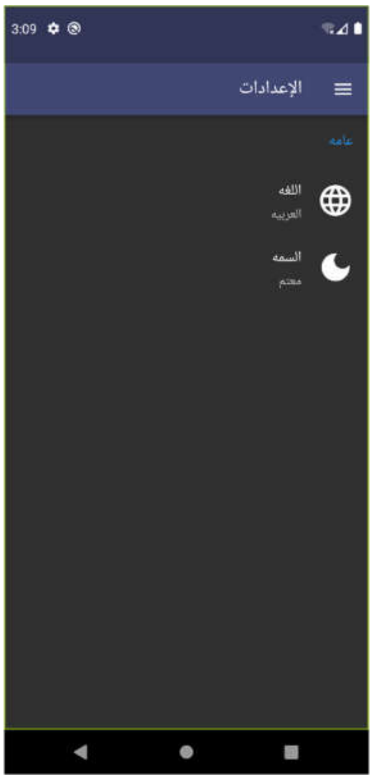
The settings page allows users to customize various aspects of the Pentor app, such as network scanning settings, log configurations, data usage monitoring and changing application language.

The settings page also provides users with advanced options for configuring their app behavior, such as defining custom scan schedules and setting up notification preferences. This allows users to customize their app experience to meet their specific needs.



Language
English

Theme
Dark



اللغه
العربيه

السمة
معتم

Figure 4.39 : Settings Page - EN

Figure 4.40 : Settings Page - AR

4.4 Summary

In this chapter, we proposed the system design details by exploring the main and the subcomponents of the system. Program flow, sequence diagrams and user stories of the system are well described. We explained the class diagram with its relationship, the database relationship and database design. Finally, screens of the interfaces are added with a description about each feature in the project.

Chapter 5
Implementation

5.1 Introduction

This chapter discusses the implementation of the system. The implementation phase is the core stage in software development. It is the process of bringing the strategic plan to life and examine in detail the analysis and design of the system. It is a measurement that determines the success of the project. Without it, the strategic goals and objectives remain unactionable. This chapter provides technical information about the system, including the system and software design decisions taken, and outlines from the structure of the system, showing the various directories and file organization.

5.2 Implementation phases

The various stages of implementation are determined by the specific tools utilized in the project. These tools include: {Database Connection, Authentication, and operating} all of which will be discussed in further detail below.

5.2.1 Database

The used database is Firebase. It is a set of backend cloud computing services and application development platforms provided by Google. It hosts databases, services, authentication, and integration for a variety of applications, including Android, iOS, JavaScript, Node.js, Java, Unity, PHP, and C++.

Some reasons to use Firebase in your project include:

- **Accelerated app development**: Firebase provides fully managed backend infrastructure, allowing you to focus on building your app without worrying about managing servers.
- **Improved app quality**: Firebase offers tools to simplify testing, triaging, and troubleshooting, helping you improve app stability and performance.
- **Increased user engagement**: Firebase provides rich analytics, A/B testing, and messaging campaigns to help you understand your users and boost engagement.
- **Easy integration**: Firebase offers detailed documentation and cross-platform SDKs to help you easily integrate its services into your app.

Overall, Firebase can help you build a better app faster and more efficiently.

5.2.2 Connect Pentor with Database

To connect a Flutter app to Firebase, you need to add the Firebase SDK dependencies to your project, configure Firebase in your app, and add the Firebase configuration files to your project. For Android, this means adding the google-services.json file to the app/ directory of your Flutter project. For iOS, you need to add the GoogleService-Info.plist file to the Runner/ directory of your Flutter project.

Once you have completed the setup process, you can initialize Firebase in your app and start using Firebase services such as Authentication, Cloud Firestore, Realtime Database, and Storage. Connecting your Flutter app to Firebase is essential for leveraging Firebase's powerful backend services and can greatly enhance the functionality and user experience of your app.

Here are the steps for connecting an Android app to Firebase:

Step 1: Install the required command line tools

1. Install the Firebase CLI if not already installed.
2. Log into Firebase using your Google account by running the following command:

```
$ firebase login
```

3. Install the FlutterFire CLI by running the following command from any directory:

```
$ dart pub global activate flutterfire_cli
```

Step 2: Configure app to use Firebase

To configure Flutter apps to connect to Firebase, you can use the FlutterFire CLI. The FlutterFire CLI is a command-line tool that helps you to simplify the process of configuring Firebase for your Flutter apps.

```
$  flutterfire configure
```

Step 3: Initialize Firebase in app

1. From Flutter project directory, run the following command to install the core plugin:

```
$  flutter pub add firebase_core
```

2. From your Flutter project directory, run the following command to ensure that your Flutter app's Firebase configuration is up-to-date:

```
$  flutterfire configure
```

3. In your lib/main.dart file, import the Firebase core plugin and the configuration file you generated earlier:

```
import 'package:firebase_core/firebase_core.dart';
import 'firebase_options.dart';
```

4. Also in your lib/main.dart file, initialize Firebase using the DefaultFirebaseOptions object exported by the configuration file:

```
await Firebase.initializeApp(
  options: DefaultFirebaseOptions.currentPlatform,
);
```

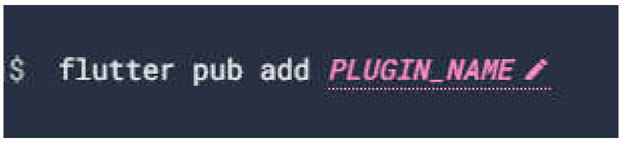
5. Rebuild your Flutter application:

```
$  flutter run
```

Step 4: Add Firebase plugins

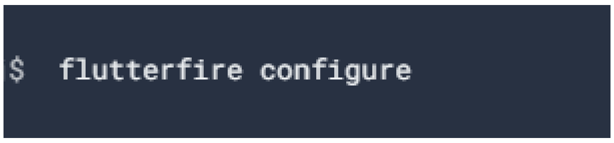
To incorporate Firebase into Flutter app, you can utilize the Firebase Flutter plugins, each corresponding to a specific Firebase product such as Cloud Firestore, Authentication, Analytics, and more. With Flutter being a versatile cross-platform framework, these Firebase plugins can be employed seamlessly across Apple, Android, and web platforms. Consequently, integrating a Firebase plugin into your Flutter app ensures its functionality across all platforms. Here is a step-by-step guide on adding a Firebase Flutter plugin:

1. From your Flutter project directory, run the following command:



```
$  flutter pub add PLUGIN_NAME
```

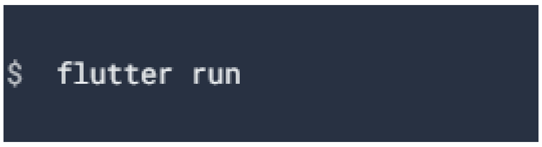
2. From your Flutter project directory, run the following command:



```
$  flutterfire configure
```

By executing this command, you ensure that the Firebase configuration of your Flutter app remains current. Additionally, for Android devices, it automatically incorporates the necessary Gradle plugins for Crashlytics and Performance Monitoring into your app.

3. Once complete, rebuild your Flutter project:



```
$  flutter run
```

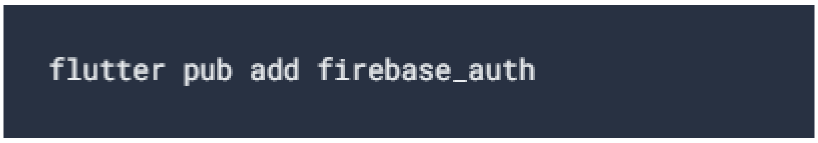
You're all set! Your Flutter apps are registered and configured to use Firebase.

5.2.3 Authentication

Firebase Authentication is a powerful authentication service offered by Firebase that enables developers to authenticate users to their mobile or web applications using secure and easy-to-use mechanisms. Firebase Authentication provides a simple and intuitive API for developers to handle user authentication and authorization, including sign-up and sign-in workflows, password resets, and user profile management. It supports a variety of authentication mechanisms such as email and password, phone number, Google, Facebook, Twitter, and GitHub, among others. Firebase Authentication also offers advanced security features such as two-factor authentication, account linking, and client-side session management. With Firebase Authentication, developers can easily add secure and scalable authentication to their apps, freeing them from the complexities of building and managing authentication systems themselves. Following is how to get started with firebase authentication:

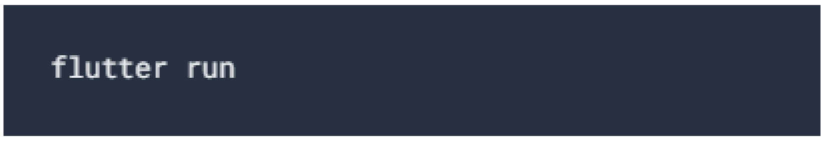
Add Firebase Authentication to your app.

1. From the root of Flutter project, run the following command to install the plugin:



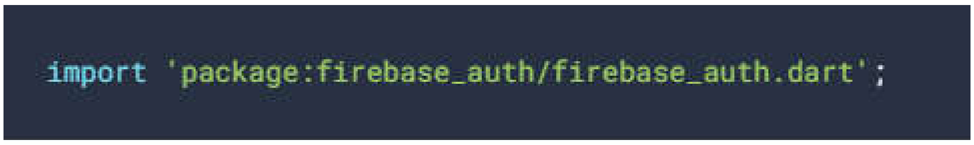
```
flutter pub add firebase_auth
```

2. Once complete, rebuild your Flutter application:



```
flutter run
```

3. Import the plugin in your Dart code:



```
import 'package:firebase_auth/firebase_auth.dart';
```

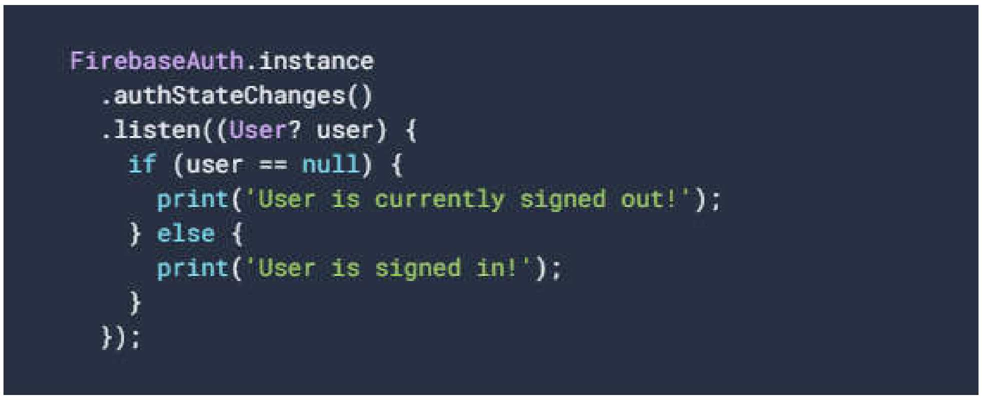
In order to utilize an authentication provider, it is necessary to activate it within the Firebase console. Navigate to the Sign-in Method page located in the Firebase Authentication section to enable features such as Email/Password sign-in, as well as any other identity providers you wish to incorporate into your app.

Check current auth state.

Firebase Auth offers a range of methods and utilities that facilitate seamless integration of secure authentication into your Flutter application, whether it's a new project or an existing one. Monitoring the authentication state of your users is often crucial, including whether they are logged in or logged out.

With Firebase Auth, you can easily subscribe to real-time updates regarding the authentication state through a Stream. Once invoked, the stream instantly delivers the current authentication state of the user, followed by subsequent events whenever there are changes to the authentication state.

To subscribe to these dynamic changes, simply call the authStateChanges() method on your instance of FirebaseAuth:



```
FirebaseAuth.instance
  .authStateChanges()
  .listen((User? user) {
    if (user == null) {
      print('User is currently signed out!');
    } else {
      print('User is signed in!');
    }
  });
```

Events are fired when the following occurs:

- Right after the listener has been registered.
- When a user is signed in.
- When the current user is signed out.

Persisting authentication state

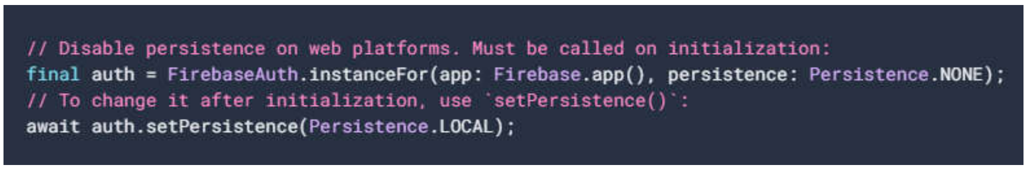
The Firebase SDKs for all supported platforms come with built-in functionality to ensure that the authentication state of your users is preserved even when the app is restarted or the page is reloaded.

On native platforms like Android and iOS, this behavior is not customizable, and the user's authentication state will be automatically stored on the device between app restarts. However, users can choose to clear the app's cached data through their device settings, which will remove any existing stored state.

On web platforms, the authentication state of the user is stored in IndexedDB by default. You have the option to modify the persistence behavior to store the data in local storage instead by using the Persistence.LOCAL parameter. If needed, you can further customize this behavior to persist the authentication state only for the current session or disable persistence entirely. To configure these settings, use the following method:

FirebaseAuth.instanceFor(app:Firebase.app(),persistence: Persistence.LOCAL);

Additionally, you can update the persistence for each instance of Auth by utilizing the setPersistence(Persistence.NONE) method.



```
// Disable persistence on web platforms. Must be called on initialization:
final auth = FirebaseAuth.instanceFor(app: Firebase.app(), persistence: Persistence.NONE);
// To change it after initialization, use `setPersistence()`:
await auth.setPersistence(Persistence.LOCAL);
```

5.2.4 Operating in Database

1- Add data to Cloud Firestore

Cloud Firestore offers multiple methods for writing data:

- Set data for a specific document within a collection by explicitly providing a document identifier.
- Add a new document to a collection, allowing Cloud Firestore to automatically generate a document identifier for you.
- Create an empty document with an automatically generated identifier and assign data to it at a later point.

These approaches provide flexibility when it comes to writing data to Cloud Firestore, allowing you to choose the method that best suits your application's requirements.

Initialize Cloud Firestore

```
db = FirebaseFirestore.instance;
```

Set a document

To create or overwrite a single document, use the following language-specific **set()** methods:

```
final network = <String, String>{
      "name": "Wifi Network Name",
      "router_mac": "DE:73:6E:6D:E9:45",
      };
      db
      .collection("Networks")
      .doc(User.id)
      .set(network)
      .onError((e, _) => print("Error writing document: $e"));
```

When writing data to a document in Cloud Firestore, the behavior depends on whether the document already exists or not. If the document does not exist, a new document will be created. However, if the document already exists, the default behavior is to overwrite its contents with the newly provided data.

To modify this behavior and merge the new data into the existing document instead of overwriting it completely, you can specify the merge option. This allows you to selectively update specific fields while preserving the existing data in the document. By utilizing the merge option, you can ensure that only the specified fields are modified, leaving the rest of the document unaffected.

```
// Update one field, creating the document if it does not already exist.
final data = {"name": "New Wifi Name"};
      db.collection("Networks ").doc(User.id ).set(data, SetOptions(merge:
true));
```

To handle scenarios where you are unsure whether a document exists in Cloud Firestore, you can utilize the option to merge new data with any existing document. By using this merge option, you can prevent the inadvertent overwrite of entire documents.

It is important to note that when working with documents containing maps, if you specify a set operation with a field containing an empty map, it will overwrite the existing map field in the target document. Therefore, exercise caution when updating documents with map fields to ensure that the intended data modifications are applied without unintentionally removing or overwriting existing map entries.

Add a document

When you use **set()** to create a document, you must specify an ID for the document to create. For example:

```
db.collection("Networks").doc(User.id).set({"name": "WIFI Network"});
```

But sometimes there isn't a meaningful ID for the document, and it's more convenient to let Cloud Firestore auto-generate an ID for you. You can do this by calling the following language-specific add() methods:

```
// Add a new document with a generated id.
final data = {"name": "Wifi Network", "router_mac": "
DE:73:6E:6D:E9:45"};

db.collection("Networks").add(data).then((documentSnapshot) =>
      print("Added Data with ID: ${documentSnapshot.id}"));
```

2- Update a document

To update some fields of a document without overwriting the entire document, use the following language-specific update() methods:

Use the update() method:

```
final currentNetwork = db.collection("Networks").doc(User.id);
        currentNetwork.update({"name": "New Wifi name"}).then(
        (value) => print("DocumentSnapshot successfully updated!"),
        onError: (e) => print("Error updating document $e"));
```

Server Timestamp

You can set a field in your document to a server timestamp which tracks when the server receives the update.

```
final docRef = db.collection("Networks").doc(User.id);
final updates = <String, dynamic>{
        "timestamp": FieldValue.serverTimestamp(),
        };

        docRef.update(updates).then(
        (value) => print("DocumentSnapshot successfully updated!"),
        onError: (e) => print("Error updating document $e"));
```

When updating multiple timestamp fields inside of a transaction, each field receives the same server timestamp value.

Update fields in nested objects

If your document contains nested objects, you can use "dot notation" to reference nested fields within the document when you call update():

```
db
        .collection("Networks")
        .doc(User.id )
        .update({"name": "New Wifi name"});
```

3- Delete data from Cloud Firestore

To delete a document, use the following language-specific delete() methods:

Use the delete() method:

```
db.collection("Networks").doc(User.id).delete().then(
      (doc) => print("Document deleted"),
      onError: (e) => print("Error updating document $e"),
      );
```

When you delete a document in Cloud Firestore, it's important to note that the deletion does not automatically remove the documents within its subcollections. Even if you delete the parent document, you can still access the subcollection documents by referencing them directly using their paths. For instance, you can still access the document at path `/mycoll/mydoc/mysubcoll/mysubdoc` even if the ancestor document at `/mycoll/mydoc` has been deleted.

Although non-existent ancestor documents may still be visible in the console, they will not appear in query results or snapshots. Therefore, it's crucial to manually delete both the parent document and all the documents within its subcollections if you wish to remove them entirely. This ensures proper cleanup of the document hierarchy within Cloud Firestore.

Delete fields

To delete specific fields from a document, use the following language-specific **FieldValue.delete()** methods when you update a document:

Use the **FieldValue.delete()** method:

```
final docRef = db.collection("Networks").doc(User.id);

// Remove the name field from the document
final updates = <String, dynamic>{
      "name": FieldValue.delete(),
      };

      docRef.update(updates);
```

4- Retrieve data

In Flutter, to fetch data from Firebase, you have the option to utilize either Firebase's Cloud Firestore or Realtime Database. To begin retrieving data, you need to obtain a reference to the specific Firestore collection from which you wish to retrieve the data.

```
final CollectionReference users =
FirebaseFirestore.instance.collection('users');
```

Use the get() method to retrieve the data from the collection:

```
  Future<void> getData() async {
    final QuerySnapshot snapshot = await users.get();
    final List<DocumentSnapshot> documents = snapshot.docs;
    documents.forEach((doc) {
        print(doc.data());
});
  }
```

5.3 Coding examples

In this section we will discuss some codes as examples of application functionality.

5.3.1 Network info

We use network_info_plus flutter plugin. This plugin allows flutter apps to discover network info and configure themselves accordingly.

1. Add it to the project in dependicies

```
network_info_plus: ^3.0.2
```

2. Download dependency

```
$ flutter pub get
```

3. Rebuild the app

```
$ flutter run
```

Now the package is ready to be used:

```
import 'package:network_info_plus/network_info_plus.dart';
```

And we can get WIFI related information:

```
final info = NetworkInfo();

var wifiBSSID = await info.getWifiBSSID(); // 11:22:33:44:55:66
var wifiIP = await info.getWifiIP(); // 192.168.1.1
var wifiName = await info.getWifiName(); // FooNetwork
```

5.3.2 Lan scanner

lan_scanner is a Dart/Flutter package that allows you to discover network devices in a local network (LAN) via multi-threaded ICMP pings. It is intended to be used on Class C networks and currently does not support the iOS platform due to a compatibility issue with the underlying ping library. This package can help you scan your local network and find connected devices.

Import the library:

```
import 'package:lan_scanner/lan_scanner.dart';
```

Create an instance of the class and call icmpScan() on it:

```
final scanner = LanScanner();

final stream = scanner.icmpScan('192.168.0', progressCallback: (progress) {
    print('Progress: $progress');
});

stream.listen((HostModel device) {
    print("Found host: ${device.ip}");
});
```

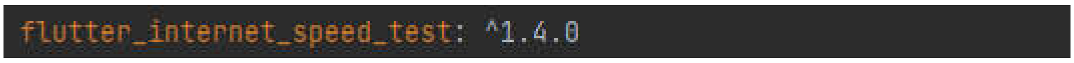
In Pentor we used it in LanScannerController as following

```
import 'package:arp_scanner/arp_scanner.dart';
import 'package:arp_scanner/device.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:get/get.dart';
import 'package:mac_address/mac_address.dart';
import
'package:pentor/controller/ConnectivityController.dart'
;
import 'package:pentor/model/device.dart';
import 'package:pentor/model/wifi_info.dart';

class LanScannerController extends GetxController {
  final _connectivityController =
Get.find<ConnectivityController>();
  final List<NetworkDevice> _devicesList = [];
  WifiInfo _wifiInfo = WifiInfo();
  @override
  void onInit() async {
    super.onInit();
    this._wifiInfo = await
_connectivityController.getWifiInfo;
    scanNetwork();
  }

  void addDevice(NetworkDevice device) {
    _devicesList.add(device);
    update();
  }

  List<NetworkDevice> get getDevicesList {
    return _devicesList;
  }

  Future<void> scanNetwork() async {
    _devicesList.clear();
    String? mac = await GetMac.macAddress;
    ArpScanner.onScanning.listen((Device device) async
{
      if (device.ip == this._wifiInfo.ipv4)
        addDevice(NetworkDevice('My Device',
device.ip!, DeviceType.Mobile,mac!));

      else
        addDevice(NetworkDevice('', device.ip!));

    });
    ArpScanner.onScanFinished.listen((List<Device>
devices) {
      Fluttertoast.showToast(msg: "Found (
${devices.length} ) Devices");
    });
    return ArpScanner.scan().then((value) => Null);
  }
}
```

5.3.3 Internet speed test

We use internet_speed_test flutter plugin. It is a swift library typically used in server and runtime applications. It has no bugs and no vulnerabilities. It has a permissive license and low support. It is used to test internet download and upload speed.

1. Add it to the project in dependencies



```
flutter_internet_speed_test: ^1.4.0
```

2. Download dependency

```
$ flutter pub get
```

3. Rebuild the app

```
$ flutter run
```

Now the package is ready to be used:

```
import 'package:internet_speed_test/internet_speed_test.dart';
```

And we can get the download and upload speed for the internet:

```
final internetSpeedTest = InternetSpeedTest();

internetSpeedTest.startDownloadTesting(
   onDone: (double transferRate, SpeedUnit unit) {
      // TODO: Change UI
   },
   onProgress: (double percent, double transferRate, SpeedUnit unit) {
      // TODO: Change UI
   },
   onError: (String errorMessage, String speedTestError) {
      // TODO: Show toast error
   },
 );
```

```
internetSpeedTest.startUploadTesting(
   onDone: (double transferRate, SpeedUnit unit) {
     print('the transfer rate $transferRate');
     setState(() {
       // TODO: Change UI
     });
   },
   onProgress: (double percent, double transferRate, SpeedUnit unit) {
     print(
         'the transfer rate $transferRate, the percent $percent');
     setState(() {
       // TODO: Change UI
     });
   },
   onError: (String errorMessage, String speedTestError) {
     // TODO: Show toast error
   },
);
```

And we can also configure the test server URL:

```
import 'package:internet_speed_test/internet_speed_test.dart';

final internetSpeedTest = InternetSpeedTest();

internetSpeedTest.startDownloadTesting(
   onDone: (double transferRate, SpeedUnit unit) {
      // TODO: Change UI
   },
   onProgress: (double percent, double transferRate, SpeedUnit unit) {
      // TODO: Change UI
   },
   onError: (String errorMessage, String speedTestError) {
      // TODO: Show toast error
   },
   testServer: //Your test server URL goes here,
 );
```

```
internetSpeedTest.startUploadTesting(
   onDone: (double transferRate, SpeedUnit unit) {
     print('the transfer rate $transferRate');
     setState(() {
       // TODO: Change UI
     });
   },
   onProgress: (double percent, double transferRate, SpeedUnit unit) {
     print(
         'the transfer rate $transferRate, the percent $percent');
     setState(() {
       // TODO: Change UI
     });
   },
   onError: (String errorMessage, String speedTestError) {
     // TODO: Show toast error
   },
   testServer: //Your test server URL goes here,
);
```

5.3.4 DNS Test

The dns_client package is a software component that provides DNS (Domain Name System) resolution services to client applications. It is responsible for translating domain names into IP addresses, which allows clients to connect to servers and other network resources. The dns_client package typically includes a set of libraries and utilities that enable applications to perform DNS queries and manage DNS cache. It is an essential component of any networked system that relies on domain names for communication. It is Dart implementation of DNS-over-HTTPS.

A simple usage example:

```
import 'package:dns_client/dns_client.dart';

main() async {
  final dns = DnsOverHttps.google();
  var response = await dns.lookup('google.com');
  response.forEach((address) {
    print(address.toString());
  });}
```

In Pentor we used it in DNSTestController as following:

```
import 'package:get/get.dart';
import 'package:flutter/material.dart';
import 'package:dns_client/dns_client.dart';

class DnsTestController extends GetxController {
  final TextEditingController textEditingController =
TextEditingController();
  final RxList<String> googleDnsResultList =
<String>[].obs;
  // final RxList<String> cloudflareDnsResultList =
<String>[].obs;

  Future<void> checkDns() async {
    final googleDns = DnsOverHttps.google();
    // final cloudflareDns = DnsOverHttps.cloudflare();

    final queryGoogle = await
googleDns.lookup(textEditingController.text.trim());
    googleDnsResultList.value =
queryGoogle.map((record) => record.address).toList();

    // final queryCloudflare = await
cloudflareDns.lookup(textEditingController.text.trim())
;
    // cloudflareDnsResultList.value =
queryCloudflare.map((record) =>
record.address).toList();
  }
}
```

5.3.5 Ping Test

Dart_ping is a package in the Dart programming language that provides a way to send ICMP echo requests (also known as pings) to a remote host and receive the corresponding ICMP echo replies. This package can be used to test network connectivity and measure network latency between two hosts. It supports both IPv4 and IPv6 addresses and allows customization of the ping request parameters such as payload size, time-to-live (TTL), and number of packets to send.

A simple usage example:

```
import 'package:dart_ping/dart_ping.dart';

void main() async {
  // Create ping object with desired args
  final ping = Ping('google.com', count: 5);

  // [Optional]
  // Preview command that will be run (helpful for debugging)
  print('Running command: ${ping.command}');

  // Begin ping process and listen for output
  ping.stream.listen((event) {
    print(event);
  });
}
```

To print the underlying ping command that will be used (useful for debugging):

```
print('Running command: ${ping.command}')
```

To prematurely halt the process:

```
await ping.stop()
```

To override the character encoding to ignore non-utf characters:

```
final ping = Ping('google.com', encoding: Utf8Codec(allowMalformed: true));
```

In Pentor we used it in PingTestController as following:

```
import 'package:dart_ping/dart_ping.dart';
import 'package:flutter/cupertino.dart';
import 'package:get/get.dart';

class PingTestController extends GetxController {
  final urlController = TextEditingController();
  final pingResults = <PingData>[].obs;
  var transmittedPackets = 0.obs;
  var lostPackets = 0.obs;

  void doPing() async {
    final url = urlController.text.trim();
    final ping = Ping(url, count: 5);
    final stream = ping.stream;

    pingResults.clear();
    transmittedPackets.value = 0;
    lostPackets.value = 0;
    stream.listen((event) {
      if (event.toString().startsWith("PingResponse"))
{
        if (event.response?.time != null) {
          pingResults.add(event);
        } else {
          lostPackets.value++;
        }
        transmittedPackets.value++;
      }
    });
  }
}
```

5.3.6 Apps data usage

We use data_usage flutter plugin. It is a package that provides access to the device network usage information. It allows developers to retrieve data usage statistics for both cellular and Wi-Fi networks, as well as monitor real-time network usage. It is a useful tool for developers who need to track and manage their apps network usage.

It supports both android and ios platforms. On android, it will fetch the data per app but due to current limitations on ios, it can only give the total value of overall data used.

- Usage for android:

After initializing plugin and requests for permission, we request data usage states:

```
DataUsageModel.init() // Only Required for Android
List<DataUsageModel> dataUsage = await DataUsage.dataUsageAndroid(
   withAppIcon: true, // if false `DataUsageModel.appIconBytes` will be null
   dataUsageType: DataUsageType.wifi, // DataUsageType.wifi | DataUsageType.mobile
   oldVersion: false // will be true for Android versions lower than 23 (MARSHMELLOW)
 );
```

This would return:

```
[   ...,
   DataUsageModel({
         String appName; //App's Name
         String packageName; // App's package name
         Uint8List appIconBytes; // Icon in bytes
         int received; // Amount of data Received
         int sent; // Amount of data sent/transferred
   })
]
```

- Usage for ios:

We request for total data usage on ios devices:

```
IOSDataUsageModel dataUsage = await DataUsage.dataUsageIOS();
```

This would return:

```
IOSDataUsageModel({
   int wifiCompelete, // Total Amount of wifi data (received + sent)
   int wifiReceived, // Amount of wifi data Received
   int wifiSent, // Amount of data sent/transferred
   int wwanCompelete, // Total Amount of mobile data (received + sent)
   int wwanReceived, // Amount of mobile data Received
   int wwanSent // Amount of data sent/transferred
});
```

5.3.7 Network logger

We use network_logger flutter plugin. It is a package that allows developers to log network requests and responses in flutter applications. It provides a simple and easy-to-use interface for monitoring network traffic, making it easier to debug and troubleshoot issues related to network connectivity. It works by intercepting all HTTP requests made by the application and logging them to the console. It also logs the corresponding response data, including status codes, headers, and body content.

Usage:

1. Install network_logger by running this command:

```
$ flutter pub add network_logger
```

This will add a line like this to the package pubspec.yaml (and run an implicit flutter pub get):

```
network_logger: ^1.0.2
```

And now the package is ready to be used:

```
import 'package:network_logger/network_logger.dart';
```

2. Add DioNetworkLogger interceptor to dio client:

network_logger comes with Dio interceptor which will intercept traffic from Dio client.

```
var dio = Dio();
dio.interceptors.add(DioNetworkLogger());
```

3. Attach network logger overlay button to UI:

The easiest way to access Network Logger UI is using NetworkLoggerOverlay which will display floating action button over all screens.

```
@override
void initState() {
  NetworkLoggerOverlay.attachTo(context);
  super.initState();
}
```

5.3.8 Expansion Tile Card

A solitary line ListTile with the trailing button expands or collapses the tile to uncover or conceal the children. This widget is normally utilized with ListView to make an "expand/collapse" list section. When utilized with looking over widgets like ListView, an interesting PageStorageKey should be indicated to empower the ExpansionTileCard to save and reestablish its expanded state when it is scrolled all through see.

The Expansion Tile Card works comparably to that of the Flutter SDK's standard expansion tile. However, it utilizes the style utilized by Google itself in its items to raise a tile. It very well may be known as a better version of Flutter's ExpansionTileCard.

Step 1: Add the dependencies

Run this command:

```
$ flutter pub add expansion_tile_card
```

This will add a line like this to your package's pubspec.yaml (and run an implicit flutter pub get*):*

```
dependencies:
 expansion_tile_card: ^2.0.0
```

Step 2: Import

Now in your Dart code, you can use:

```
import 'package:expansion_tile_card/expansion_tile_card.dart';
```

Step 3:

Run flutter packages get in the root directory of your app.

Usage

```
return ExpansionTileCard(
 expandedTextColor: Themes.primaryColor,
 leading: Icon(
  Icons.wifi,
  size: 50.0,
 ),
 title: Text(wifiInfo.name,
   style:
     TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
 subtitle: Text("WIFI Network Connection".tr),
 children: [
  wifiInfoTable(wifiInfo),
 ],
);
```

5.3.9 Flutter UI speedometer widget

Speedometer widget is a visual element that visually represents the speed of a vehicle or machine. It typically includes a circular shape divided into segments and a pointer or needle that indicates the current speed value.

To incorporate a speedometer widget into your Flutter application, you can utilize the CustomPaint widget. This allows you to draw the gauge and pointer using the Canvas API. Alternatively, you can leverage third-party packages like flutter_gauge or gauge_chart, which provide ready-made components for speedometer visualization, simplifying the implementation process.

To ensure real-time updates of the speedometer value, you can employ a stateful widget or utilize a state management solution such as Provider or Bloc. This allows you to manage the speed value and trigger widget rebuilds whenever the value changes, providing a dynamic and responsive speedometer display.

Step 1: Add the dependencies

Run this command:

```
$ flutter pub add Speedometer
```

This will add a line like this to your package's **pubspec.yaml** (and run an implicit *flutter pub get*):

```
dependencies:
 speedometer: ^1.1.2
```

Step 2: Import

Now in your Dart code, you can use:

```
import 'package:speedometer/speedometer.dart';
```

Step 3:

Run flutter packages get in the root directory of your app.

```
child: SpeedOMeter(
   start: controller.start, end: controller.end,
   highlightStart: 0.125, highlightEnd: 0.875,
   themeData: ThemeData(
     primaryColor: Themes.primaryColor,accentColor: Colors.black,
backgroundColor: Colors.grey, ),
   eventObservable: controller.eventObservable),
```

5.3.10 Home page

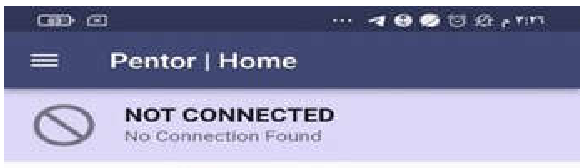
This is the landing page of the application which consisted of:

- Connectivity widget

  Following method return the connectivity widget

```
  Widget connectivityWidget() {
    switch (_connectivityController.connectivityStatus) {
      case ConnectivityResult.none:
        return ListTile(
          tileColor: Themes.backgroundColor,
          leading: Icon(
            Icons.not_interested,
            size: 50.0,
          ),
          title: Text("NOT CONNECTED".tr,
              style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
          subtitle: Text("No Connection Found".tr),
        );
      case ConnectivityResult.wifi:
        return buildWifiTile();
      case ConnectivityResult.mobile:
        return buildMobileInfoTile();
      default:
        return ListTile(
          tileColor: Themes.accentColor,
          leading: Icon(
            Icons.error_outline,
            color: Colors.red,
            size: 50.0,
          ),
          title: Text("ERROR".tr,
              style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
          subtitle: Text("CONNECTION ERROR".tr),
        );
    }
  }
```

Result is show in Figure 5.1


Pentor | Home

NOT CONNECTED

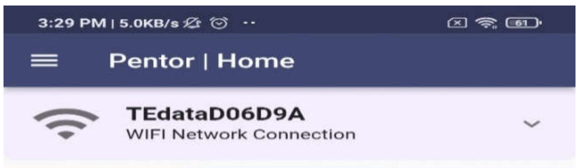
No Connection Found

Figure 5.1 : Connectivity widget

In case of Wifi Connected WifiTile is shown using the following method :

```
  Widget buildWifiTile() {
    return FutureBuilder<WifiInfo>(
        future: _connectivityController.getWifiInfo,
        builder: (_, snapshot) {
          if (snapshot.hasError) {
            return
buildPermissionErrorWidget(snapshot.error.toString(),
                _connectivityController.askForLocationPermission)
;
          } else {
            switch (snapshot.connectionState) {
              case ConnectionState.done:
                WifiInfo wifiInfo = snapshot.data!;
                return ExpansionTileCard(
                  expandedTextColor: Themes.primaryColor,
                  leading: Icon(
                    Icons.wifi,
                    size: 50.0,
                  ),
                  title: Text(wifiInfo.name,
                      style:
                          TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
                  subtitle: Text("WIFI Network Connection".tr),
                  children: [
                    wifiInfoTable(wifiInfo),
                  ],
                );
              default:
                //TODO: Other switch cases
                return Container();
            }
          }
        });
  }
```

Result is show in Figure 5.2



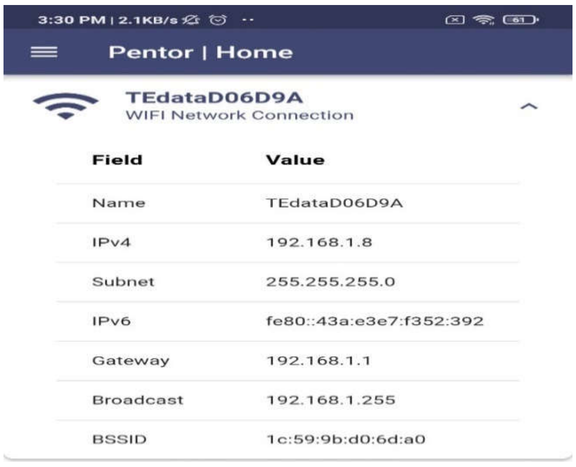
TEdataD06D9A
WIFI Network Connection

Figure 5.2 Wifi Tile

To show more info about the wifi network the wifiInfoTable is build using the following method:

```
  Widget wifiInfoTable(WifiInfo wifiInfo) {
    return DataTable(columns: [
      DataColumn(
          label: Text('Field'.tr,
              style: TextStyle(fontSize: 16, fontWeight:
FontWeight.bold))),
      DataColumn(
          label: Text('Value'.tr,
              style: TextStyle(fontSize: 16, fontWeight:
FontWeight.bold))),
    ], rows: [
      DataRow(cells: [
        DataCell(Text('Name'.tr)),
        DataCell(Text(wifiInfo.name)),
      ]),
      DataRow(cells: [
        DataCell(Text('IPv4')),
        DataCell(Text(wifiInfo.ipv4)),
      ]),
      DataRow(cells: [
        DataCell(Text('Subnet')),
        DataCell(Text(wifiInfo.subnet)),
      ]),
      DataRow(cells: [
        DataCell(Text('IPv6')),
        DataCell(Text(wifiInfo.ipv6)),
      ]),
      DataRow(cells: [
        DataCell(Text('Gateway'.tr)),
        DataCell(Text(wifiInfo.gateway)),
      ]),
      DataRow(cells: [
        DataCell(Text('Broadcast'.tr)),
        DataCell(Text(wifiInfo.broadcast.toString().replaceAll('/
', ''))),
      ]),
      DataRow(cells: [
        DataCell(Text('BSSID')),
        DataCell(Text(wifiInfo.bssid)),
      ]),
    ]);
  }
```

The table is shown in Figure 5.3



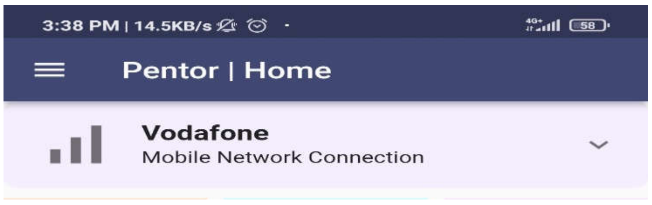
Field	Value
Name	TEdataD06D9A
IPv4	192.168.1.8
Subnet	255.255.255.0
IPv6	fe80::43a:e3e7:f352:392
Gateway	192.168.1.1
Broadcast	192.168.1.255
BSSID	1c:59:9b:d0:6d:a0

Figure 5.3 : Wifi Info Table

In case of Mobile data connected the connectivity tile shows the mobileInfoTile using the following method:

```
  Widget buildMobileInfoTile() {
    return FutureBuilder<MobileNetworkInfo>(
        future: _connectivityController.getMobileNetworkInfo,
        builder: (_, snapshot) {
          if (snapshot.hasError) {
            return
buildPermissionErrorWidget(snapshot.error.toString(),
                _connectivityController.askForPhonePermission);
          } else {
            switch (snapshot.connectionState) {
              case ConnectionState.done:
                MobileNetworkInfo mobileNetworkInfo =
snapshot.data!;
                return ExpansionTileCard(
                  expandedTextColor: Themes.primaryColor,
                  leading: Icon(
                    Icons.signal_cellular_alt, size: 50.0,),
                  title: Text(mobileNetworkInfo.name,
                      style:
                          TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
                  subtitle: Text("Mobile Network Connection".tr),
                  children:
[mobileNetworkInfoTable(mobileNetworkInfo)],
                );
              default:
                return Container();
            }
          }
        });
  }
```

The result is showing in Figure 5.4



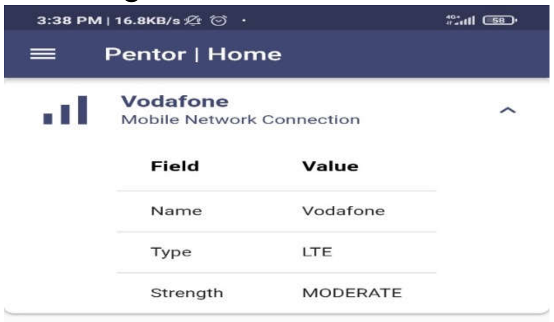
Vodafone
Mobile Network Connection

Figure 5.4 : MobileInfoTile

To show more info about the Mobile data network the mobileNetworkInfoTable is build using the following method:

```
  Widget mobileNetworkInfoTable(MobileNetworkInfo
mobileNetworkInfo) {
    return DataTable(columns: [
      DataColumn(
          label: Text('Field'.tr,
              style: TextStyle(fontSize: 16, fontWeight:
FontWeight.bold))),
      DataColumn(
          label: Text('Value'.tr,
              style: TextStyle(fontSize: 16, fontWeight:
FontWeight.bold))),
    ], rows: [
      DataRow(cells: [
        DataCell(Text('Name'.tr)),
        DataCell(Text(mobileNetworkInfo.name)), ]),
      DataRow(cells: [
        DataCell(Text('Type'.tr)),
        DataCell(Text(mobileNetworkInfo.networkType)),]),
      DataRow(cells: [
        DataCell(Text('Strength'.tr)),
        DataCell(Text(mobileNetworkInfo.signalStrength.tr)),
      ]),
    ]);
  }
}
```

The result is shown in Figure 5.5



Vodafone
Mobile Network Connection

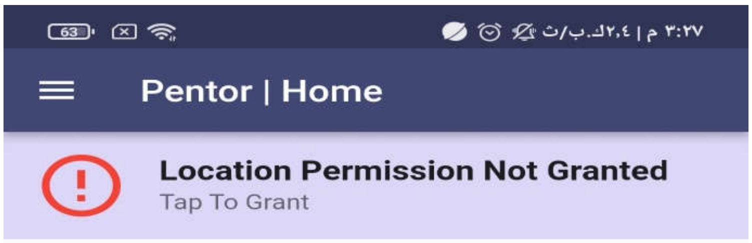
Field	Value
Name	Vodafone
Type	LTE
Strength	MODERATE

Figure 5.5 Mobile Info Table

In case of required permission not granted the following tile is build instead:

```
Widget buildPermissionErrorWidget(String title, void Function()
onTap) {
    return ListTile(
      tileColor: Themes.backgroundColor,
      leading: Icon(
        Icons.error_outline,
        color: Colors.red,
        size: 50.0,
      ),
      onTap: onTap,
      title: Text(title,
          style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold)),
      subtitle: Text("Tap To Grant".tr),
    );
  }
```

The result shown in Figure 5.6



Pentor | Home

Location Permission Not Granted

Tap To Grant

Figure 5.6 : Permission not granted

- Navigation Grid

To Navigate to each page if the system navigation grid is build using the following code:

```
GridView.builder(
                  itemCount: _navController.pages.length,
                  gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
                      crossAxisCount: 3, crossAxisSpacing: 10.0,
                      mainAxisSpacing: 10.0),
                  itemBuilder: (BuildContext context, int index){
                    NavPage page = _navController.pages[index];
                    return InkWell(onTap: page.action,
                      child: Container(color: page.color,
                        child: Column(
                          mainAxisAlignment:
MainAxisAlignment.spaceEvenly,
    children: [ Icon( page.icon,size: 75.0,),Text(page.title.tr)
              ], ), ), ); }, ),
```

The result is shown in Figure 5.7

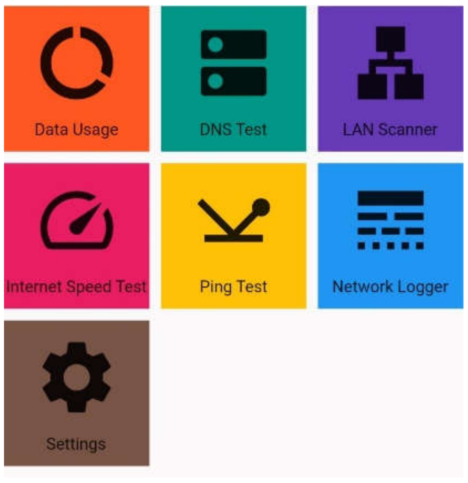


Figure 5.7 : Navigation grid

5.3.11 Navigation Drawer

Has two functions as following:

- Google login tile

The code is as following:

```
  Widget buildHeader({
    required String urlImage,
    required String name,
    required String email,
    required VoidCallback onClicked,
  }) =>
      InkWell(
        child: Container(
          color: Themes.customLightTheme.primaryColor,
          padding: EdgeInsets.only(top: 30, left: 10, right: 10),
          child: Row(
            children: [
              CachedNetworkImage(
                imageUrl: urlImage,
                imageBuilder: (context, imageProvider) =>
Container(
                  width: 50.0,
                  height: 50.0,
                  decoration: BoxDecoration(
                    shape: BoxShape.circle,
                    image: DecorationImage(
                        image: imageProvider, fit: BoxFit.cover),
                  ),
                ),
                placeholder: (context, url) =>
CircularProgressIndicator(),
                errorWidget: (context, url, error) =>
Icon(Icons.error),
              ),
              SizedBox(width: 20),
              Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    name,
                    style: TextStyle(fontSize: 20, color:
Colors.white),
                  ),
                  const SizedBox(height: 4),
                  Text(
                    email,
                    style: textStyle,
                  ),
                ],
              ),
            ],
          ),
        ),
      );
```

The result is in Figure 5.8

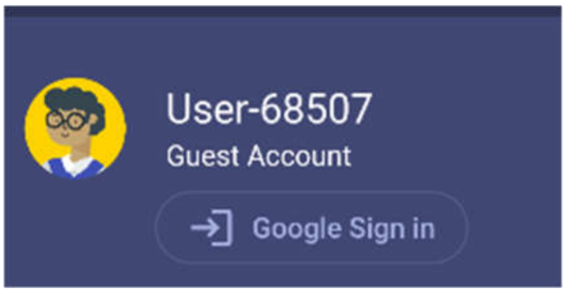


Figure 5.8 : Google Login tile

- Navigation List

  A navigation list to travel through application features from any page was coded as following:

```
  Widget buildMenuItem({
    required String text,required IconData icon,
    VoidCallback? onClicked,
  }) {
    return ListTile(
      leading: Icon(icon, color: Colors.white),
      title: Text(text, style: textStyle),
      onTap: onClicked,); }
```

```
// On Scaffold we put
                  buildMenuItem(
                      text: 'Home'.tr,
                      icon: Icons.home,
                      onClicked: _navController.openHomePage),

                  buildMenuItem(
                      text: 'Data Usage'.tr,
                      icon: Icons.data_usage,
                      onClicked:
_navController.openDataUsagePage),

                  buildMenuItem(
                      text: 'DNS Test'.tr,
                      icon: Icons.dns,
                      onClicked: _navController.openDnsTestPage),

                  buildMenuItem(
                      text: 'LAN Scanner'.tr,
                      icon: Icons.lan,
                      onClicked:
_navController.openLanScannerPage),

                  buildMenuItem(
                      text: 'Internet Speed Test'.tr,
                      icon: Icons.speed,
                      onClicked:
_navController.openSpeedTestPage),

                  buildMenuItem(
                      text: 'Ping Test'.tr,
                      icon: Icons.network_ping,
                      onClicked:
_navController.openPingTestPage),

                  buildMenuItem(
                      text: 'Network Logger'.tr,
                      icon: Icons.line_style,
                      onClicked:
_navController.openNetworkLoggerPage),
                  Divider(color: Colors.white70),
                  buildMenuItem(
                      text: 'Settings'.tr,
                      icon: Icons.settings,
                      onClicked:
_navController.openSettingsPage),
```

the result is shown in Figure 5.9

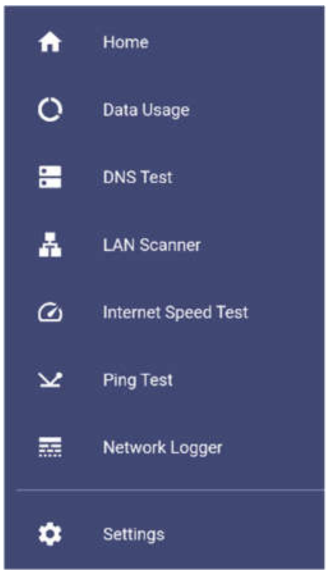


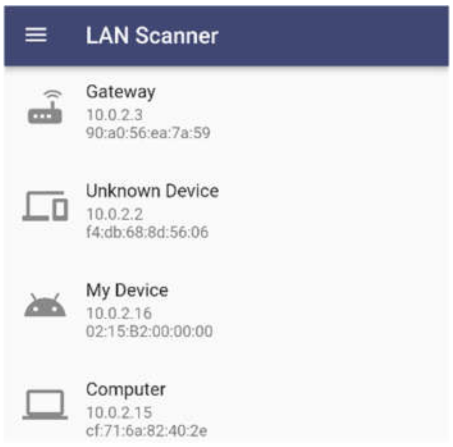
Figure 5.9 : Navigation List

5.3.12 Lan Scanner Page

Most of the code was implemented in the LanScannerController so what was left was only the UI code as following:

```
RefreshIndicator(
        onRefresh: _lanScannerController.scanNetwork,
        child: GetBuilder<LanScannerController>(
          builder: (_) => Container(
            width: double.infinity,
            height: double.infinity,
            child: ListView(
              children: [
                ..._lanScannerController.getDevicesList
                    .map((dev) => ListTile(
                  title: Text(dev.name),
                  subtitle: Text(dev.ip + "\n" + dev.mac),
                  leading: Icon(
                    getDevIcon(dev.type),
                    size: 40.0,
                  ),
                  isThreeLine: true,
                ))
                    .toList(),
              ],
            )
          ),
        ),
      ),
```

The result is shown in Figure 5.10


LAN Scanner

Gateway
10.0.2.3
90:a0:56:ea:7a:59

Unknown Device
10.0.2.2
f4:db:68:8d:56:06

My Device
10.0.2.16
02:15:B2:00:00:00

Computer
10.0.2.15
cf:71:6a:82:40:2e

Figure 5.10 : Lan Scanner Page

5.3.13 Speed test Page

The UI code was as following:

```
GetBuilder<SpeedTestController>(
          init: SpeedTestController(),
          builder: (controller) {
            return Column(
              children: [
                Padding(
                  padding: EdgeInsets.only(top: 35.0,left: 30.0,
right: 30.0,bottom: 5.0),
                  child: SizedBox(
                    width: 300,
                    height: 300,
                    child: SpeedOMeter(
                        start: controller.start,
                        end: controller.end,
                        highlightStart: 0.125,
                        highlightEnd: 0.875,
                        themeData: ThemeData(
                          primaryColor: Themes.primaryColor,
                          // accentColor: Colors.black,
                          backgroundColor: Colors.grey,
                        ),
                        eventObservable:
controller.eventObservable),
                  ), ),
                Text("Download".tr+" : " +
controller.downSpeed.toString() +' '+ controller.downUnit.tr),
                Text("Upload".tr+" : " +
controller.upSpeed.toString() +' '+ controller.upUnit.tr),
                ElevatedButton(onPressed: controller.speed_test,
child: Text("Start".tr)) ], ); }, ),
```

The result is shown in Figure 5.11

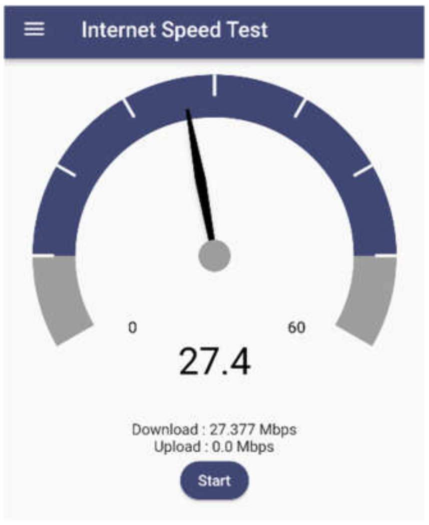


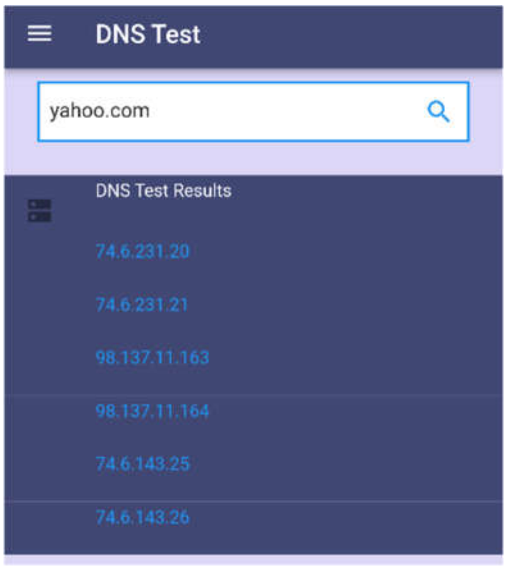
Figure 5.11 : Speed test Page

5.3.14 DNS Test Page

The UI code was as following:

```
ListView.builder(
      shrinkWrap: true,
      itemCount: controller.googleDnsResultList.length,
      itemBuilder: (context, index) {
          return GestureDetector(
                  onTap: () {
                  Clipboard.setData(ClipboardData(
                   text: controller.googleDnsResultList[index], ));
    ScaffoldMessenger.of(context).showSnackBar(
       SnackBar(content: Text("IP Copied to clipboard".tr)), ); },
         child: ListTile(
         tileColor: Themes.primaryColor,
         leading: index == 0?Icon(Icons.dns):SizedBox(width: 24.0),
         title: index == 0 ? Column(
         crossAxisAlignment: CrossAxisAlignment.start,
         children: [
             Text("DNS Test Results".tr+"\n\n",
                style: textStyle, ), ], ) : SizedBox.shrink(),
                subtitle: index >= 0
                        ? Padding(
                          padding:EdgeInsets.only(bottom:18.0,
                                child: Text(
                              controller.googleDnsResultList[index],
                                style: TextStyle(color:Colors.blue),
                                ),    )  : null,
                            ),);
                        },
                      ),
```

The result is shown in Figure 5.12



DNS Test Results

74.6.231.20

74.6.231.21

98.137.11.163

98.137.11.164

74.6.143.25

74.6.143.26

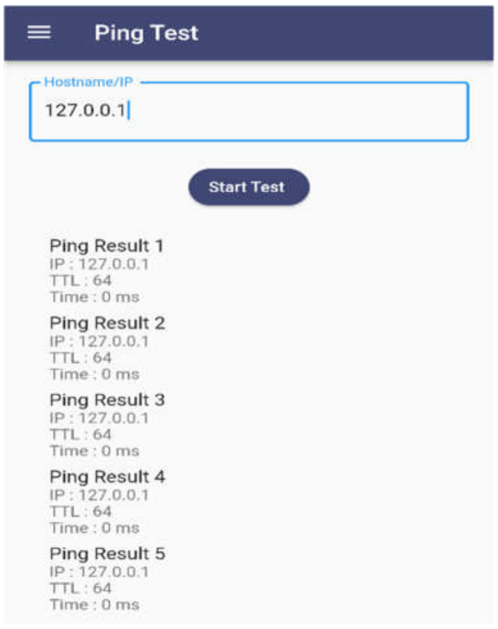
Figure 5.12 : DNS Test Page

5.3.15 Ping Test Page

The UI code is as following:

```
Padding(
        padding: EdgeInsets.all(20),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextField(
              controller: pingController.urlController,
              decoration: InputDecoration(
                labelText: 'Hostname/IP',
                border: OutlineInputBorder(),
              ),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () => pingController.doPing(),
              child: Text('Start Test '.tr),
            ),
            SizedBox(height: 20),
            Obx(() {
              final pingResults = pingController.pingResults;
              if (pingResults.length == 5) {
                final transmitted =
pingController.transmittedPackets;
                final lost = pingController.lostPackets;
                final percentage = lost > 0 ? lost /
transmitted.toInt() * 100 : 0;
                WidgetsBinding.instance.addPostFrameCallback((_) {
                  showDialog(
                    context: context,
                    builder: (_) => AlertDialog(
                      title: Text("Ping Summary".tr),
                      backgroundColor:Themes.backgroundColor ,
                      content: Text(
                       'Transmitted Packets'.tr+ ': $transmitted\n'+
                            'Lost Packets'.tr +
                            ': $lost \n'+
                            'Packets Percentage loss'.tr+
                            ':${percentage.toStringAsFixed(2)}%'.tr,
                      ),
                      actions: <Widget>[
                        ElevatedButton(
                          onPressed: () {
                            Navigator.of(context).pop();
                          },
                          child: Text('OK'.tr),
                        ),
                      ],
                    ),
                  );
                });
              }
              return Expanded(
                child: ListView.builder(
                  itemCount: pingResults.length,
                  itemBuilder: (context, index) {
                    final result = pingResults[index];
                    return ListTile(
                      title: Text('Ping Result'.tr +' ${index +1}'),
                      subtitle: Column(
                        crossAxisAlignment:CrossAxisAlignment.start,
                        children: [
                          Text('IP'.tr +':${result.response?.ip}'),
                          Text('TTL'.tr+':${result.response?.ttl}'),
                          Text('Time'.tr +':
${result.response?.time?.inMilliseconds ?? 'N/A'}' + ' ms'.tr),
                        ],
                      ),
                    );
                  },
                ),
              );
            }),],),),
```

The result is shown in Figure 5.13



127.0.0.1

Ping Result 1
IP : 127.0.0.1
TTL : 64
Time : 0 ms

Ping Result 2
IP : 127.0.0.1
TTL : 64
Time : 0 ms

Ping Result 3
IP : 127.0.0.1
TTL : 64
Time : 0 ms

Ping Result 4
IP : 127.0.0.1
TTL : 64
Time : 0 ms

Ping Result 5
IP : 127.0.0.1
TTL : 64
Time : 0 ms

Figure 5.13 : Ping Test Page

5.3.16 Data Usage Page

The UI code is as Following:

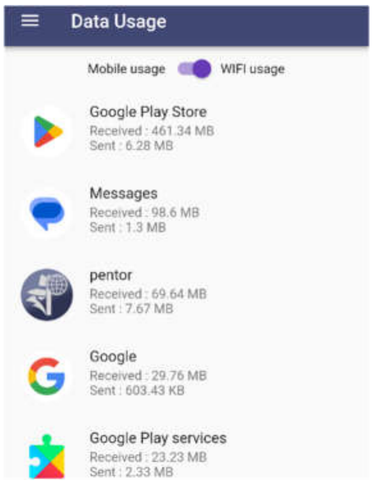
- Selection Switch to select Wifi Usage or Data Usage

```
Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text("Mobile usage".tr),
                ObxValue(
                    (p0) => Switch(
                        value:_dataUsageController.wifiOrData.value,
                        activeColor: Colors.deepPurple,
                        inactiveThumbColor: Colors.blue,
                        inactiveTrackColor: Colors.lightBlueAccent,
                        onChanged: (val) {
                          _dataUsageController.wifiOrData.value=val;
                        }),
                    _dataUsageController.wifiOrData),
                Text("WIFI usage".tr),
              ],
            ),
```

- Actual applications list with its sent and received bytes

```
Obx(() {
  return FutureBuilder(
      future: _dataUsageController.wifiOrData.value
          ? _dataUsageController.wifiAppsList
          : _dataUsageController.mobileAppsList,
      builder: (_, snapshot) {
        if (snapshot.connectionState == ConnectionState.done) {
          if (snapshot.hasData) {
            List<DataUsageModel> appsList =
                snapshot.data as List<DataUsageModel>;
            return SizedBox(
              width: double.infinity,
              height: 620.0,
              child: ListView(
                children: [
                  ...appsList.map((app) {
                    num recvVal = _dataUsageController
                        .round_data(app.received!);
                    String recvUnit = _dataUsageController
                        .data_unit(app.received!);
                    num sentVal =
                        _dataUsageController.round_data(app.sent!);
                    String sentUnit =
                        _dataUsageController.data_unit(app.sent!);

                    return ListTile(
                      title: Text(app.appName!),
                      leading: Container(
                        height: 60,
                        width: 60,
                        decoration: BoxDecoration(
                          image: DecorationImage(
                            image: MemoryImage(app.appIconBytes!),
                          ),
                        ),
                      ),
                      subtitle: Column(
                        crossAxisAlignment:
                            CrossAxisAlignment.start,
                        children: [
                          Text("Received".tr +
                              " : " +
                              recvVal.toString() +
                              " " +
                              recvUnit.tr),
                          Text("Sent".tr +
                              " : " +
                              sentVal.toString() +
                              " " +
                              sentUnit.tr),
                        ],
                      ),
                      isThreeLine: true,
                    );
                  }).toList()
                ],
              ),
            );
          } else {
            return Center(child: Text("Nothing to view".tr));
          }
        } else {
          return Column(
            children: [
              SizedBox(height: 100.0),
              CircularProgressIndicator(),
            ],
          );
        }
      });
}),
```

The result is shown in Figure 5.14



Google Play Store
Received : 461.34 MB
Sent : 6.28 MB

Messages
Received : 98.6 MB
Sent : 1.3 MB

pentor
Received : 69.64 MB
Sent : 7.67 MB

Google
Received : 29.76 MB
Sent : 603.43 KB

Google Play services
Received : 23.23 MB
Sent : 2.33 MB

Figure 5.14 : Data Usage Page

5.3.17 Settings Page

The settings page contains the following:

- Tile to switch the application language

```
ListTile buildLanguageTile() {
  return ListTile(
    title: Text("Language".tr),
    leading: const Icon(Icons.language, size: 40.0),
    subtitle: GetBuilder<LocaleController>(builder: (_) {
      return Text(localeCtrl.language.tr);
    }),
    onTap: localeCtrl.switchLang,
  );
}
```

The result is shown in Figure 5.15

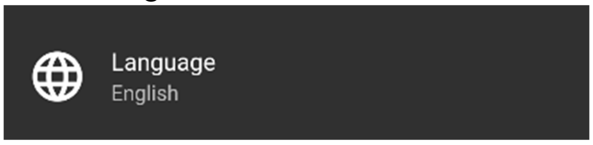


Figure 5.15 : Switch Language Tile

- Tile to switch between light and dark themes

```
GetBuilder<SettingController> buildThemeTile() {
  return GetBuilder<SettingController>(builder: (context) {
    return ListTile(
      title: Text("Theme".tr),
      subtitle: Text(settingController.theme_str.tr),
      leading: Icon(
        settingController.theme_str == 'light'
            ? Icons.light_mode
            : Icons.dark_mode,
        size: 40.0,
      ),
      onTap: settingController.switchTheme,
    );
  });
}
```

The result is shown in Figure 5.16

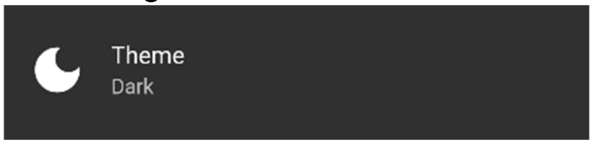


Figure 5.16 : Theme Switch Tile

5.4 Summary

In this chapter, we become closer to the application. We provided a description of the technical side of the project, including its software engineering base and design decisions. The structure of the template of the project was discussed with details of each page structure, and samples of the code were presented.

Chapter 6
Testing

6.1 Introduction

Project testing is the process of verifying the accuracy and completeness of project deliverables before the delivery to the client. It is an essential part of quality assurance and helps to ensure that the final product meets its requirements, performs as expected and is fit for purpose. It helps to identify errors, gaps, and risks in the project so that they can be addressed before the project is completed. It aims not only at finding faults in the software but also at finding measures to improve the software. Project testing should be done throughout the project lifecycle, from the beginning of the planning phase to the end of the project

6.2 Evaluation Requirement

This part will evaluate how the application will meet the required functions

Table 12 : Testcases decription table

Functions	Test description	Expected result	Actual result
Network Info	In Home page, when pressing on the network name, information about it is displayed.	Network information should be displayed.	Network information is displayed.
LAN scanner	When icon is pressed the LAN should be scanned.	A list of all the connected devices is shown.	Devices connected to the LAN are shown.
Speed test	When icon is pressed it tests upload and download speed and compare it with results from fast.com.	Results should be the same.	Results usually are the same or almost.
Data Usage	Data Usage icon is pressed to show data usage of applications.	List of applications should be displayed with their usage of data.	Applications and their usage of data are displayed.
Google sign in	On pressing on Google sign in Button, a window showed up	Get Google Account data after sign in	Got Google account user name, email and profile_pic

Functions	**Test description**	**Expected result**	**Actual result**
DNS Test	Enter a hostname like google.com and compare the output with windows nslookup result	IP address or list of IP addresses will show up.	The IP address or list of IP addresses appears same as nslookup output
Ping Test	Enter a hostname or IP address ,Then compare it with windows ping result	List of Results includes IP address of the target,TTL, and Time it took for the packet to make a round trip. Then a pop up window contains Ping Summery	Perform the ping test and view list of results and the values appears to be close to windows ping command results

6.3 Testing Tools and Environment

The purpose of testing tools is to automate the testing process and increase the efficiency and effectiveness of testing. A good set of testing tools can help to identify defects early in the development cycle, reduce the cost of testing, and improve the quality of the software.

The testing environment, on the other hand, refers to the hardware and software configuration that is used to execute the tests. It is important to ensure that the testing environment is stable and consistent to ensure reliable and repeatable test results. The testing environment should also be representative of the production environment to ensure that the software will perform as expected in the real world.

SW: Android 11

HW: Android phone

Other: stable internet connection

6.4 Running and Testing

Running refers to the process of executing the software to ensure that it performs as expected. It is important to ensure that the software runs smoothly and without errors on various platforms and configurations.

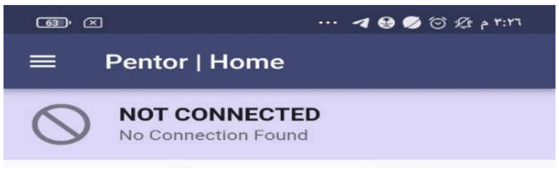
Testing, involves validating the software against a set of requirements or specifications to ensure that it meets the desired quality standards. Testing can be performed at different stages of the development process, including unit testing, integration testing, system testing, and acceptance testing. A well-designed testing strategy that includes a combination of manual and automated testing can help to ensure that the software is of high quality and meets the needs of the end-users.

6.4.1 Running Examples

- **Network Info**

  The user will display network info by pressing on the network name. The user can be connected by either WIFI or mobile data.

  - When the user is not connected to a network:



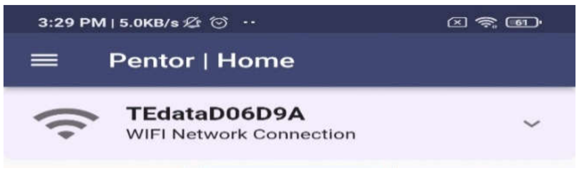
٣:٢٦ م

Pentor | Home

NOT CONNECTED
No Connection Found

Figure 6.1 : Device Not Connected to Network

  - When connected to WIFI:

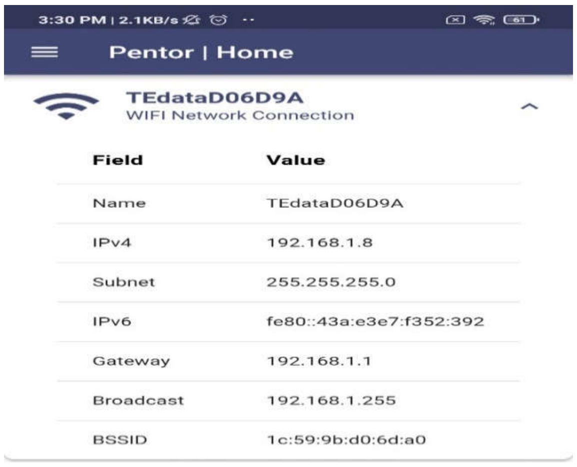


3:29 PM | 5.0KB/s

Pentor | Home

TEdataD06D9A
WIFI Network Connection

Figure 6.2 : Connectivity tile Wifi Connected - Collabsed



3:30 PM | 2.1KB/s

Pentor | Home

TEdataD06D9A
WIFI Network Connection

Field	Value
Name	TEdataD06D9A
IPv4	192.168.1.8
Subnet	255.255.255.0
IPv6	fe80::43a:e3e7:f352:392
Gateway	192.168.1.1
Broadcast	192.168.1.255
BSSID	1c:59:9b:d0:6d:a0

Figure 6.3: Connectivity tile Wifi Connected – Expanded

- When connected to mobile data:

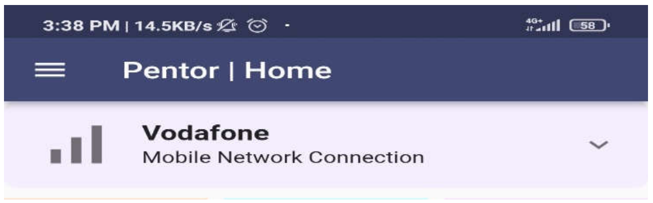
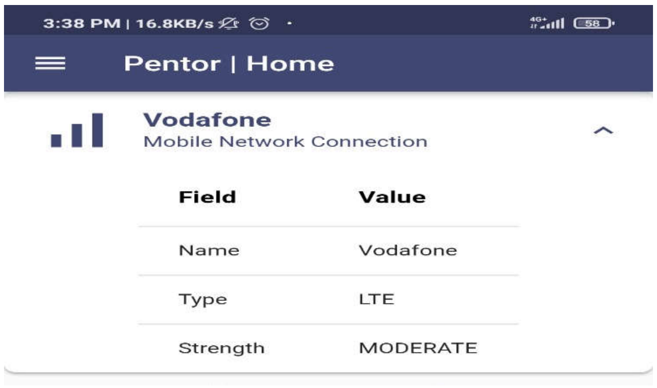


Figure 6.4: Connectivity tile Mobile Data Connected - Collabsed

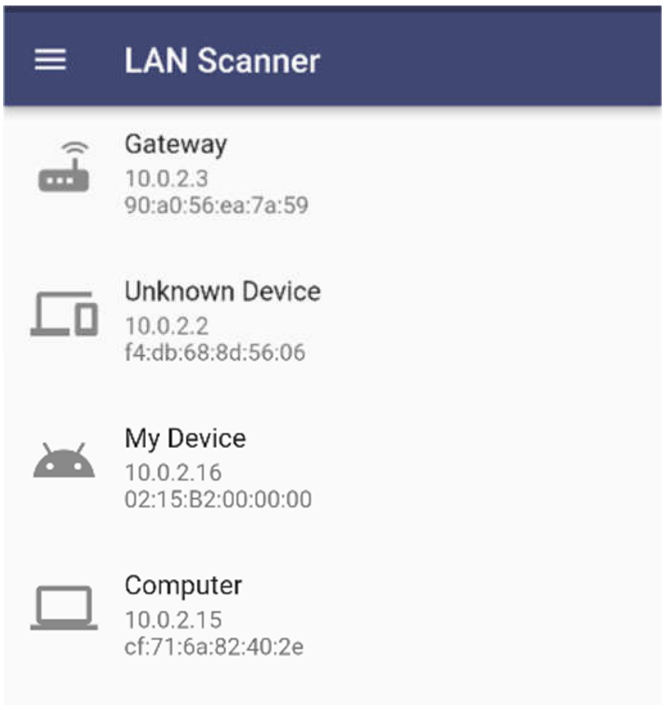


Field	Value
Name	Vodafone
Type	LTE
Strength	MODERATE

Figure 6.5 : Connectivity tile Mobile Data Connected - Expanded

- **LAN Scanner**

  The user gets a list of the connected devices to the LAN after opening the Lan scanner page icon.



Gateway
10.0.2.3
90:a0:56:ea:7a:59

Unknown Device
10.0.2.2
f4:db:68:8d:56:06

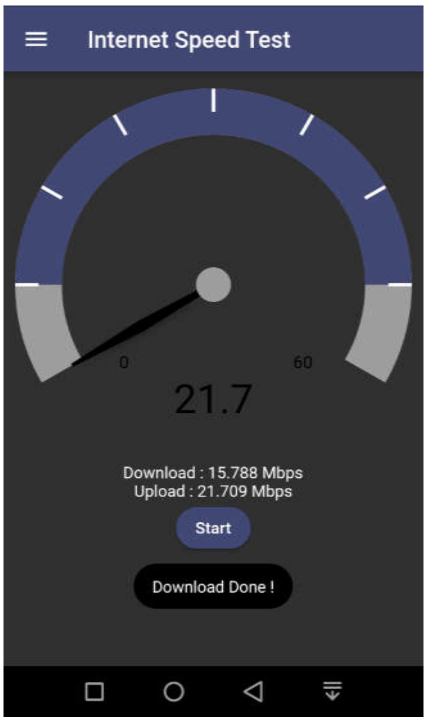
My Device
10.0.2.16
02:15:B2:00:00:00

Computer
10.0.2.15
cf:71:6a:82:40:2e

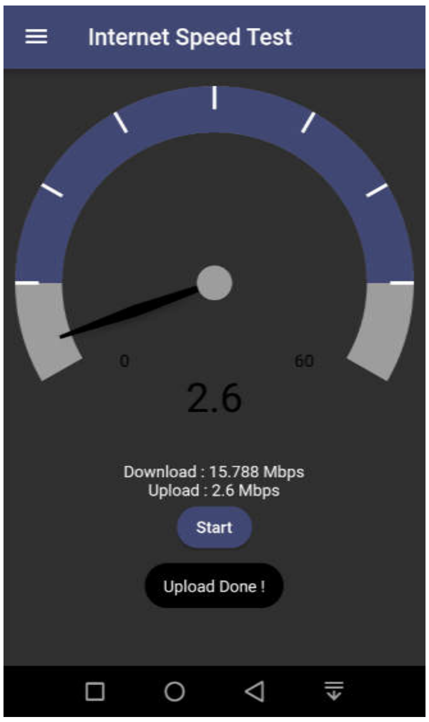
Figure 6.6 : Lan Scanner Results

- **Internet Speed Test**

  The upload and download speeds are tested and shown.



Download : 15.788 Mbps
Upload : 21.709 Mbps

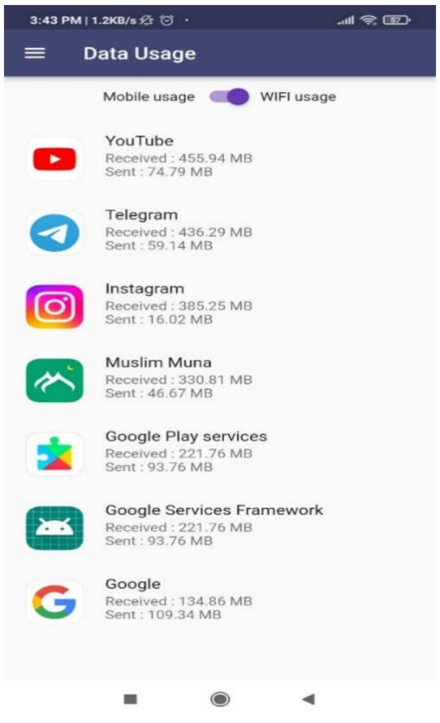


Download : 15.788 Mbps
Upload : 2.6 Mbps

Figure 6.7: Download Speed Test *Figure 6.8 : Upload Speed Test*

- **Data Usage**

  The user press on Data Usage icon and choose WIFI usage or Mobile usage to display applications and their usage of data.



YouTube
Received : 455.94 MB
Sent : 74.79 MB

Telegram
Received : 436.29 MB
Sent : 59.14 MB

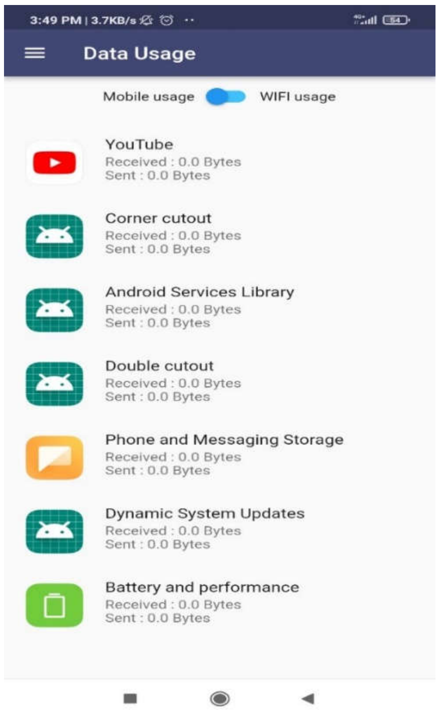
Instagram
Received : 385.25 MB
Sent : 16.02 MB

Muslim Muna
Received : 330.81 MB
Sent : 46.67 MB

Google Play services
Received : 221.76 MB
Sent : 93.76 MB

Google Services Framework
Received : 221.76 MB
Sent : 93.76 MB

Google
Received : 134.86 MB
Sent : 109.34 MB



YouTube
Received : 0.0 Bytes
Sent : 0.0 Bytes

Corner cutout
Received : 0.0 Bytes
Sent : 0.0 Bytes

Android Services Library
Received : 0.0 Bytes
Sent : 0.0 Bytes

Double cutout
Received : 0.0 Bytes
Sent : 0.0 Bytes

Phone and Messaging Storage
Received : 0.0 Bytes
Sent : 0.0 Bytes

Dynamic System Updates
Received : 0.0 Bytes
Sent : 0.0 Bytes

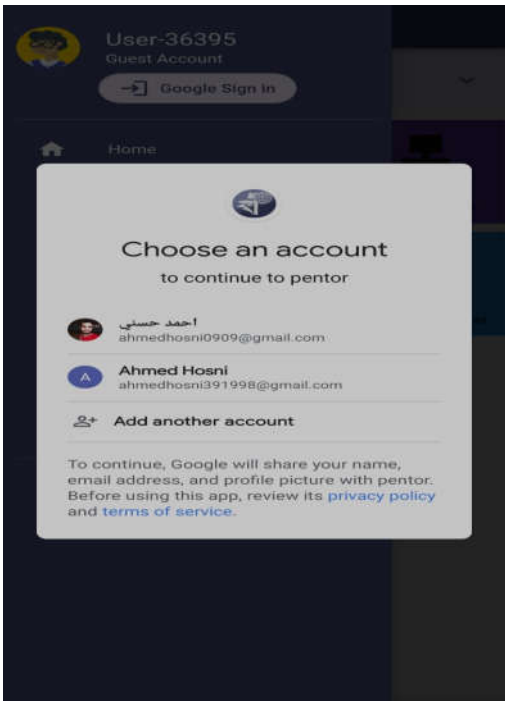
Battery and performance
Received : 0.0 Bytes
Sent : 0.0 Bytes

Figure 6.9 : Wifi Data Usage *Figure 6.10 : Mobile Data Usage*

- **Google sign in**

  The User will open Navigation Drawer then will Press on Google Sign in button.

  - When the user Press on Google sign in button



Choose an account

to continue to pentor

احمد حسني
ahmedhosni0909@gmail.com

Ahmed Hosni
ahmedhosni391998@gmail.com

Add another account

To continue, Google will share your name, email address, and profile picture with pentor. Before using this app, review its privacy policy and terms of service.

Figure 6.11 : Google sign in - google accounts

  - After signing in

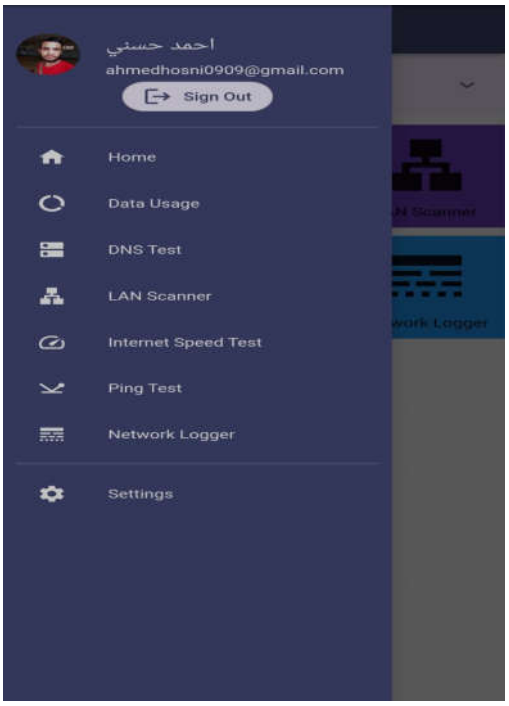
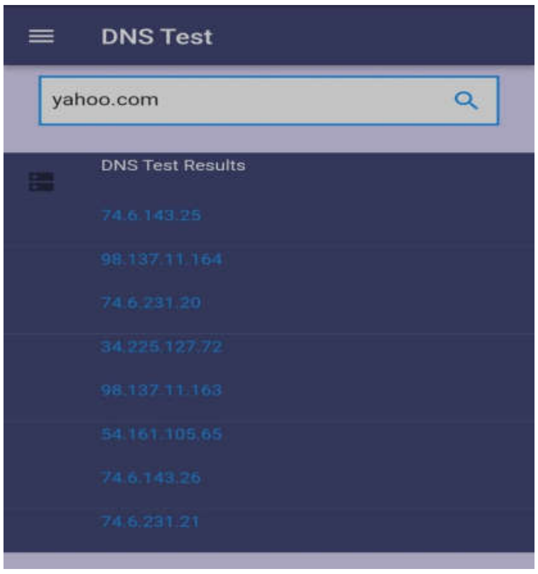


Figure 6.12 : Google Sign in - loged in

- **DNS Test**

The user press on DNS Test icon and enter Hostname then pressing on search icon DNS Test Results.

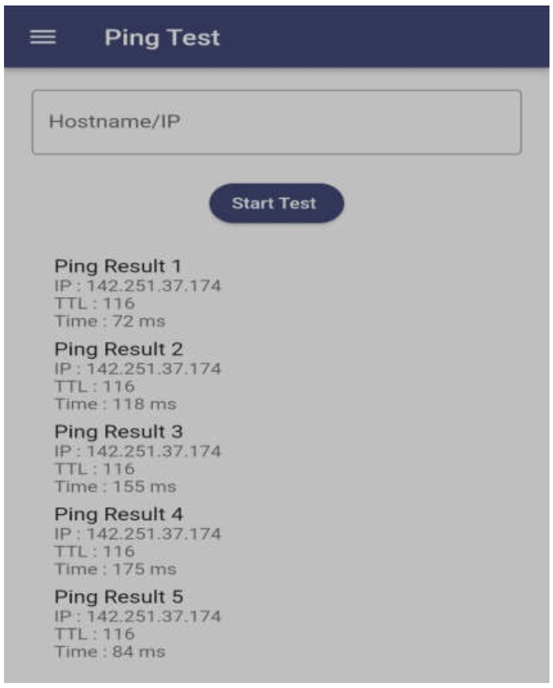


DNS Test Results
74.6.143.25
98.137.11.164
74.6.231.20
34.225.127.72
98.137.11.163
54.161.105.65
74.6.143.26
74.6.231.21

Figure 6.13 : DNS Test Results

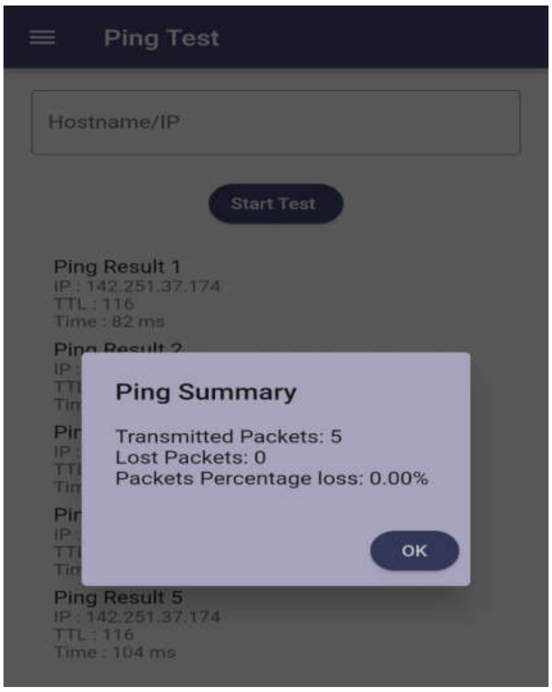
- **Ping Test**

The user press on Ping Test icon and enter Hostname or IP address then pressing on Start Test button to display Test Results.



Ping Result 1
IP : 142.251.37.174
TTL : 116
Time : 72 ms
Ping Result 2
IP : 142.251.37.174
TTL : 116
Time : 118 ms
Ping Result 3
IP : 142.251.37.174
TTL : 116
Time : 155 ms
Ping Result 4
IP : 142.251.37.174
TTL : 116
Time : 175 ms
Ping Result 5
IP : 142.251.37.174
TTL : 116
Time : 84 ms

Figure 6.14 : Ping Test Results



Ping Result 1
IP : 142.251.37.174
TTL : 116
Time : 82 ms

Ping Summary
Transmitted Packets: 5
Lost Packets: 0
Packets Percentage loss: 0.00%

Ping Result 5
IP : 142.251.37.174
TTL : 116
Time : 104 ms

Figure 6.15 : Ping Test Summary

6.4.2 Validation

Following is list of action validations implemented in Pentor.

- **Network Info**
  - When connected to WIFI:

    Location permission is required to get the information about the network.

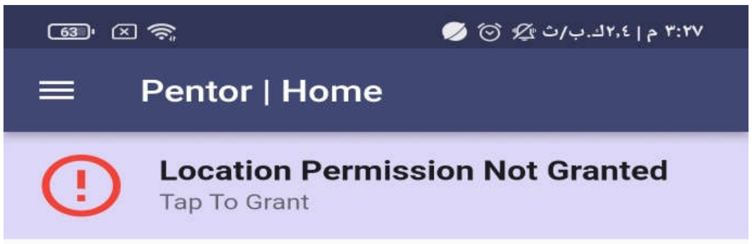


Figure 6.16 : Location Permission Not Granted

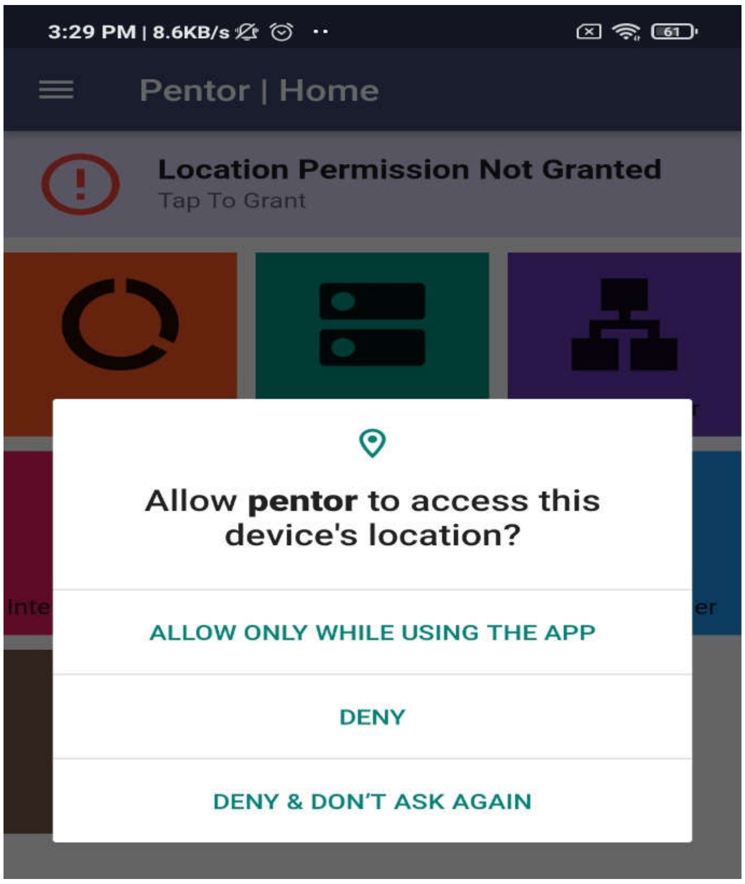
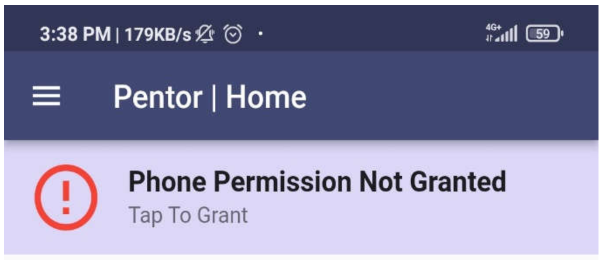

3:29 PM | 8.6KB/s
Pentor | Home
Location Permission Not Granted
Tap To Grant
Allow pentor to access this device's location?
ALLOW ONLY WHILE USING THE APP
DENY
DENY & DON'T ASK AGAIN

Figure 6.17 : Location Permission pop up

  - When connected to mobile data:

    Phone permission is required to get information about the mobile data.



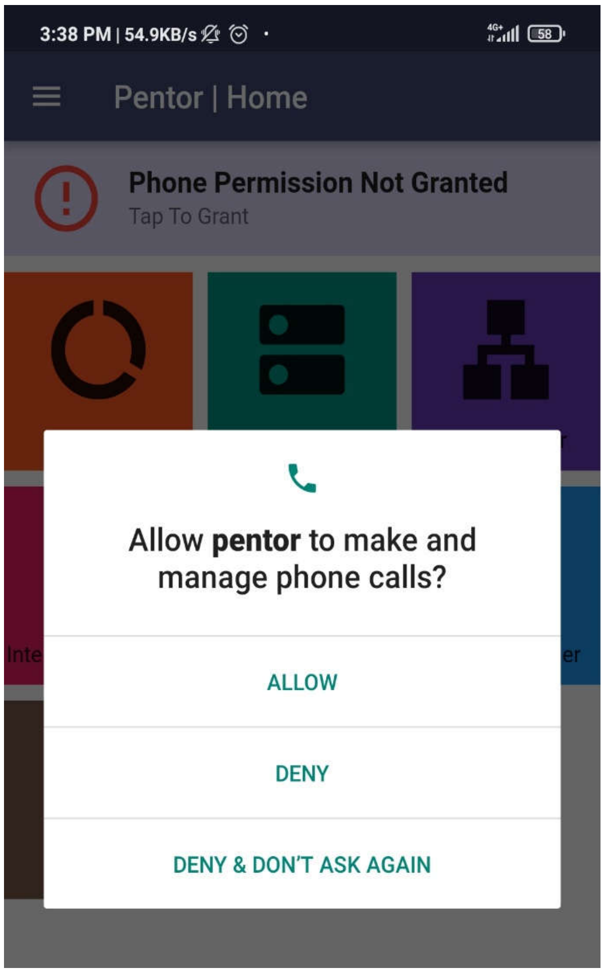
3:38 PM | 179KB/s

Pentor | Home

Phone Permission Not Granted

Tap To Grant

Figure 6.18 : Phone Permission Not Granted



3:38 PM | 54.9KB/s

Pentor | Home

Phone Permission Not Granted

Tap To Grant

Allow **pentor** to make and manage phone calls?

ALLOW

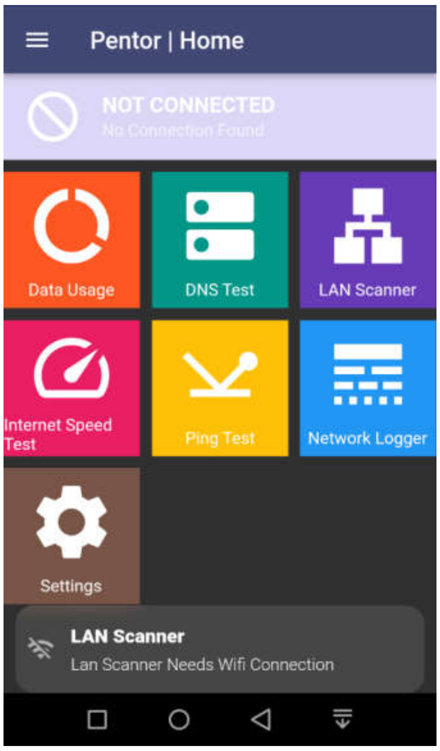
DENY

DENY & DON'T ASK AGAIN

Figure 6.19 : Phone Permission pop up

- **LAN scanner:**

  This need a WIFI connection, If the user tries to use the feature while using mobile data or not Connected a message saying that "Lan Scanner Needs Wifi Connection "will appear as follows.

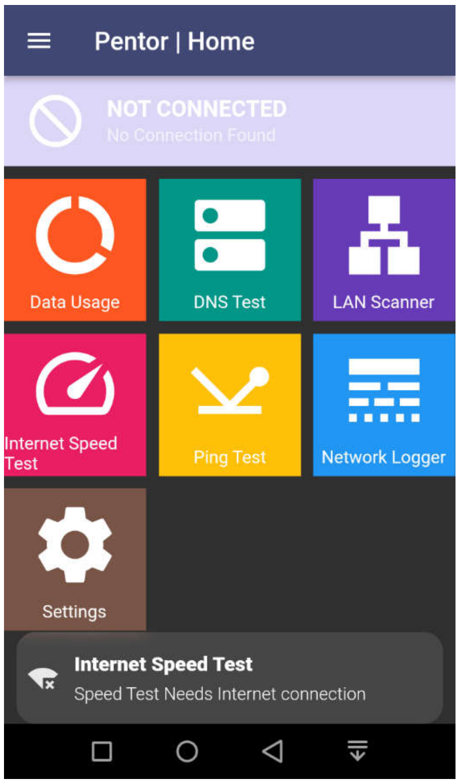


LAN Scanner
Lan Scanner Needs Wifi Connection

Figure 6.20 : Lan Scanner Connection Validation

- **Internet speed test:**

  This needs an internet connection, if the user try to use the feature without having Internet connection a message will appear saying " ".

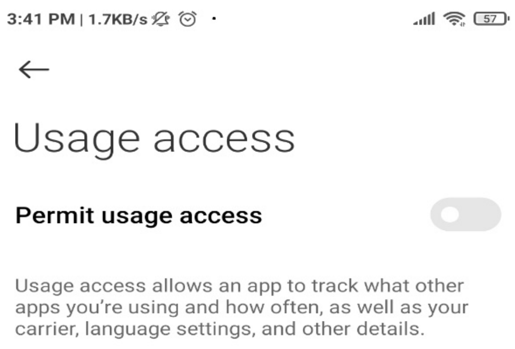


Internet Speed Test
Speed Test Needs Internet connection

Figure 6.21 : Speed Test Connection Validation

- **Data Usage**

  There is a need to permit usage access to "Pentor" application so that the user can use data usage feature.



3:41 PM | 1.7KB/s

Usage access

Permit usage access

Usage access allows an app to track what other apps you're using and how often, as well as your carrier, language settings, and other details.

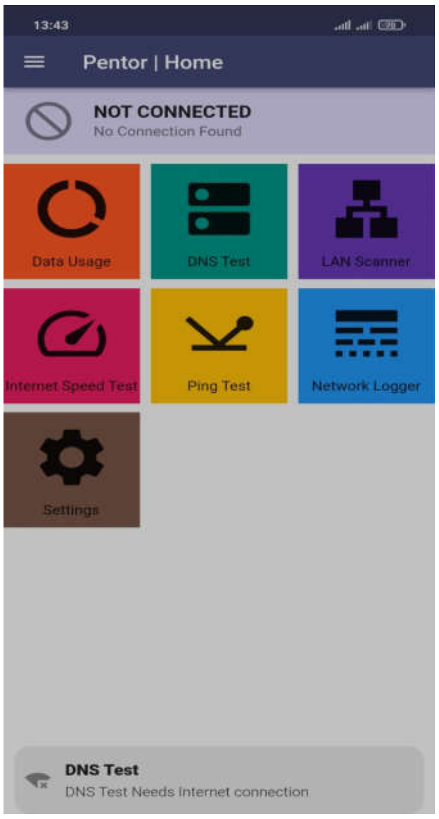
Figure 6.22 : Usage access Permission page

- **Google sign in**

  Internet connection is required to sign in into Google account.

- **DNS Test**

  Internet connection is required to make a DNS Test.



13:43

Pentor | Home

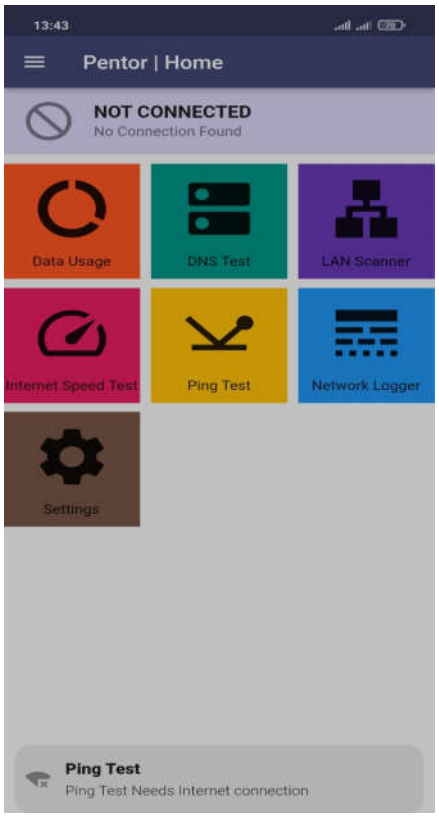
NOT CONNECTED
No Connection Found

DNS Test
DNS Test Needs Internet connection

Figure 6.23 : DNS Internet Connection validation

- **Ping Test**

  Internet connection is required to make a Ping Test.



NOT CONNECTED
No Connection Found

Ping Test
Ping Test Needs Internet connection

Figure 6.24 : Ping Test internet connection validation

6.5 Summary

The importance of software testing is imperative. Software testing is a crucial component of software product development because it increases consistency and performance. It makes the software more reliable and easier to use. The main benefit of testing is the identification and subsequent correction of faults. It also helps to compare actual and expected results to improve quality. In this chapter, screens of running examples are provided and also the validations.

Chapter 7
Conclusion

7.1 Introduction

As we conclude the documentation, we reflect on the journey we have taken to develop this humble network monitoring tool. Throughout this documentation, we have covered the various aspects of Pentor, including its purpose, features, design principles, and development methodologies. We have explored the software development cycle of Pentor and how it follows best practices in software engineering. Additionally, we have discussed the importance of network monitoring in ensuring network security and preventing unauthorized access.

In this final chapter, we provide a summary of the key points covered in the document and highlight the main takeaways for our readers. We believe that Pentor offers a unique solution to personal network owners by providing real-time monitoring and alerts for suspicious activity on their networks. With its intuitive user interface and comprehensive feature set, Pentor is an invaluable tool for safeguarding network security.

7.2 Faced Problems

During the development of Pentor, our team encountered several challenges that we had to overcome. These challenges included:

1. **Designing an intuitive user interface**: We wanted Pentor to be user-friendly and accessible to users with varying levels of technical expertise. Designing an intuitive interface that met this requirement was a significant challenge that required multiple iterations and user testing.
2. **Integrating with different network protocols**: Pentor needed to support various network protocols such as FTP, ARP, and ICMP. Implementing support for these protocols was challenging and required a deep understanding of networking concepts.
3. **Ensuring security**: As a network monitoring tool, security was a top priority for Pentor. We had to ensure that Pentor itself did not pose a security risk and that it was capable of detecting and alerting users to potential security threats on their networks.
4. **Platform Limitations**: as advanced features like LAN scanning on android require additional root privileges which is unsecure, vulnerable and not easy to grant for most of our targeted users it was so hard to find workarounds to cover up these limitations.

Despite these challenges, our team was able to successfully develop Pentor and deliver a robust, user-friendly, and secure network monitoring tool. We remain committed to addressing any future challenges and ensuring that Pentor continues to meet the needs of network owners.

7.3 Pentor Benefits and findings

During the development of Pentor, the team has gained numerous benefits and findings. Listed as following:

- working on a project that aims to improve network security has helped the team to improve their technical skills and knowledge of network security concepts. They have also learned how to work collaboratively on a large-scale project and communicate effectively to ensure that everyone is on the same page.
- **Using Waterfall methodology**. Despite its limitations, the team found several benefits to using this model. One of the main advantages of the waterfall model is its sequential nature, which allowed the team to plan each phase of the project in detail and ensure that each step was completed before moving on to the next one. This approach also enabled the team to identify potential issues early in the project and address them before they became major problems.
- **Using Flutter** for cross-platform development significantly reduced development time and effort. With Flutter's hot reload feature, developers were able to quickly make changes to the code and see the results in real-time, which greatly improved the development process.
- **Integrating Firebase** into the app allowed for easier implementation of features such as authentication, real-time database updates, and cloud storage. This significantly reduced the time and effort required to develop these features from scratch.

Overall, the development of Pentor has been a valuable learning experience for the team, and they are proud to have created a product that can improve network security for personal network owners.

7.4 Recommendations and future enhancements

While Pentor has been designed to provide personal network owners with comprehensive information about their networks, there is always room for improvement. In this section, we will discuss some recommendations for future enhancements to the application.

1. **Router Setting**: Pentor can provide setting up router configuration like changing WIFI password and changing default DNS server for home gateway to make is easier and faster for user to setup his local network.
2. **Advanced Analytics**: Pentor can benefit from advanced analytics capabilities to provide more detailed insights into network activity. This could include machine learning algorithms that detect anomalies and

suspicious activity in real-time, and visualizations that help users understand their network traffic patterns.

3. **Integration with Third-Party Tools**: Pentor could be integrated with other third-party tools to provide more value to users. For example, integration with antivirus software could provide additional protection against malware, while integration with firewalls could help users better manage their network security policies.
4. **Multi-Language Support**: Finally, Pentor could benefit from multi-language support to reach a wider audience. This could be achieved by translating the application into additional languages like Chinese and Spanish, which would make it more accessible to non-English and non-Arabic speaking users.

In conclusion, Pentor is a valuable tool for personal network owners to monitor their networks and ensure security. However, there is always room for improvement, and the above recommendations can help Pentor become even more effective and valuable for its users. By using the waterfall model, the team was able to design, develop and test the application effectively, and future enhancements can be implemented using the same methodology.

7.5 Summary

We hope that this documentation has been informative and insightful for our readers. We remain committed to maintaining and improving Pentor to ensure that it continues to meet the evolving needs of network owners. Thank you for joining us on this journey, and we hope that Pentor proves to be a valuable addition to your network monitoring arsenal.

References

1. Types of Software Development Approaches - acqnotes [Article] [https://acqnotes.com/acqnote/careerfields/software-development-approaches]
2. Waterfall model – techtarget [Article] [https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model]
3. Flutter documentation [https://docs.flutter.dev]
4. Official Flutter package repository [https://pub.dev/]
5. Android Developer Guides [https://developer.android.com/docs]
6. Project Repository on GitHub [https://github.com/CSEMN/Pentor]
7. Understanding Functional and Non-Functional Requirements in App Development [Article][https://www.moveoapps.com/blog/functional-and-non-functional-requirements-in-app-development]
8. Statista. (2021). Mobile app usage - Statistics & Facts [https://www.statista.com/topics/1002/mobile-app-usage/]
9. Tenev, T. (2021). The Pros and Cons of Mobile App Development for Business.[https://www.business.com/articles/pros-and-cons-of-mobile-app-development-for-business/]
10. dart_ping package on pub.dev: [https://pub.dev/packages/dart_ping]
11. dns_client package on pub.dev: [https://pub.dev/packages/dns_client]
12. lan_scanner package on pub.dev: [https://pub.dev/packages/lan_scanner]
13. ICMP protocol documentation: [https://tools.ietf.org/html/rfc792]
14. ARP protocol documentation: [https://tools.ietf.org/html/rfc826]

15. network_logger package on pub.dev:

    [https://pub.dev/packages/network_logger]

16. HTTP protocol documentation: [https://tools.ietf.org/html/rfc7230]

17. data_usage package on pub.dev:

    [https://pub.dev/packages/data_usage]

18. Android NetworkStatsManager documentation:

    [https://developer.android.com/reference/android/app/usage/NetworkStatsManager]

19. firebase_core package on pub.dev:

    [https://pub.dev/packages/firebase_core]

20. Cloud Firestore documentation:

    [https://firebase.google.com/docs/firestore]

21. get package documentation: [https://pub.dev/packages/get]

22. shared_preferences package documentation:

    [https://pub.dev/packages/shared_preferences]

23. get_storage package documentation:

    [https://pub.dev/packages/get_storage]

24. expansion_tile_card package documentation:

    [https://pub.dev/packages/expansion_tile_card]

25. permission_handler package documentation:

    [https://pub.dev/packages/permission_handler]

26. **connectivity_plus** package documentation:

    [https://pub.dev/packages/connectivity_plus]

27. **telephony** package documentation:

    [https://pub.dev/packages/telephony]

28. **firebase_auth** package documentation:

[https://pub.dev/packages/firebase_auth]

29. **google_sign_in** package documentation:

[https://pub.dev/packages/google_sign_in]

30. Mobile Backend as a Service (MBaaS) Market - Growth Trends Forecast (2020 - 2025)

31. "Best Practices for Mobile App Development" - AWS Whitepaper

32. Ambler, S. W. (2002). The elements of UML™ 2.0 style. Cambridge University Press

33. Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The unified modeling language user guide (2nd ed.). Addison-Wesley

34. Firebase documentation:

[https://firebase.google.com/docs/flutter/setup]

35. Firebase Authentication documentation:

[https://firebase.google.com/docs/auth/flutter/start]

36. Software Testing | Basics - GeeksforGeeks

37. Why is Testing Necessary and Important? | ISTQB | ToolsQA

Tools

1. Dart: Programming Language.
2. Flutter Framework: for Cross platform development
3. Android Studio: for Code Editing and simulation.
4. Google Firebase: for cloud storage.
5. Draw.IO: for Diagrams.
6. Git & GitHub: for version control.
7. Microsoft word: for reports and documentation.
8. Microsoft Planner: for task scheduling.
9. Onlinegantt : For Gantt Charts
10. IconFinder : For Icons in Figures