TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Ilja Vasilenko 206136IADB

Application for Monitoring and Administering Network-based Applications on a Windows Operating System

Bachelor's thesis

Supervisor: Mohammad Tariq Meeran

PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Ilja Vasilenko 206136IADB

Rakendus võrgupõhiste rakenduste jälgimiseks ja haldamiseks Windowsi operatsioonisüsteemis

Bakalaureusetöö

Juhendaja: Mohammad Tariq Meeran

PhD

Tallinn 2023

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Ilja Vasilenko

24.04.2023

Abstract

This thesis aims to develop an application for network monitoring and administration on Windows Operating System, designed specifically for non-advanced users. Existing solutions often lack important features or are too complex for beginners. In this study, the author compares and analyses similar third-party and built-in solutions, identifying their limitations, usability, and feature sets. Based on this analysis, the author develops an application with the most relevant features and a simple UI/UX design that is accessible to beginner-level users.

The theoretical part of this work presents a comprehensive analysis of existing solutions and identifies gaps in feature sets and usability. Based on literature review findings and analysis the general and functional requirements for the application are defined.

The practical part of the thesis describes the development process of the application, including code patterns and algorithms used in implementing the major features.

The contribution of this study is an application design with a user-friendly interface and relevant features that address the limitations of existing solutions. It provides insights into the design and development of efficient and effective network monitoring and administration tools and is particularly relevant for non-advanced users.

This thesis is written in English and is 56 pages long, including 6 chapters, 17 figures and 2 tables.

Annotatsioon

Rakendus võrgupõhiste rakenduste jälgimiseks ja haldamiseks Windowsi operatsioonisüsteemis

Selle lõputöö eesmärk on luua rakendus Windowsi operatsioonisüsteemis võrgu jälgimiseks ja haldamiseks, algaja taseme kasutaja jaoks. Olemasolevad lahendused võivad sageli puududa olulistest funktsioonidest või olla liiga keerulised tavakasutajatele. Uuringus võrdleb ja analüüsib autor sarnaseid kolmandate osapoolte ja sisseehitatud lahendusi, tuvastades nende piirangud, kasutatavuse ja funktsioonide kogumi. Analüüsi põhjal töötab autor välja asjakohaste funktsioonide ja lihtsa UI/UX disainiga rakenduse, mis on ligipääsetav tavakasutajatele.

Töö teoreetiline osa esitab olemasolevate lahenduste põhjaliku analüüsi ning tuvastab lüngad funktsioonide kogumis ja kasutatavuses. Kirjanduse ülevaate leidude ja rakenduse üldiste ning funktsionaalsete nõuete analüüsi põhjal on määratletud.

Lõputöö praktilises osas kirjeldatakse rakenduse arendusprotsessi ning koodimustreid ja põhifunktsioonide realiseerimisel kasutatavaid algoritme.

Uuringu panus on kasutajasõbraliku liidese ja asjakohaste funktsioonidega rakenduse disain, mis käsitleb olemasolevate lahenduste piiranguid. See annab ülevaate tõhusate ja tulemuslike võrgujälgimis- ja haldustööriistade kavandamisest ja arendamisest, olles eriti oluline tavakasutajatele.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 56 leheküljel, 6 peatükki, 17 joonist, 2 tabelit.

List of abbreviations and terms

API	Application Programming Interface – the connection between different applications
CPU	Central processing unit – computer processor
DNS	Domain Name System - a hierarchical and distributed naming system for computers, services, and other resources in the Internet
Driver	Computer program that controls a particular type of device - enabling operating systems and other computer programs to access hardware functions
ERD	Entity relationship diagram – visualization of entity relationship sets stored in a database
Ethernet	Technology for connecting devices in a wired local area network (LAN) or wide area network (WAN)
GUI	Graphical user interface
IP address	A unique address that identifies a device on the network
MVVM	Model-View-ViewModel - software design pattern
NIC	Network interface card or network interface controller - a hardware component that connects a computer or other device to a network
ORM	Object Relational Mapping - a technique used to communicate between object-oriented programs and databases
OS	Operating System - system software that manages computer hardware
Pcap	Abbreviation of packet capture - application programming interface (API) for capturing network traffic
Repository	Abstraction, which provides default functionalities and can be selectively changed by the developers
Thread	Single sequential flow of execution of tasks of a process
UI	User interface
UX	User experience
Wi-Fi	Wireless networking technology
XAML	Extensible Application Markup Language - declarative markup language

Table of contents

List of figures

List of tables

1 Introduction

The Internet and network-based applications have become an essential part of our daily lives, making it necessary to manage and monitor network usage efficiently. Monitoring network traffic can provide insights into internet usage patterns, optimize network usage, and improve productivity. Administering network traffic can also be helpful. Users can restrict access to the internet for certain applications, for example to disable ads and automatic updates, or reduce overall internet traffic consumption. Additionally, closely monitoring network traffic can help to detect any malicious software and prevent security breaches and potential data loss.

The market offers a variety of solutions for administrating and monitoring network traffic. For smartphone users, these features often come as built-in functionalities, which provide powerful monitoring and administration capabilities. Users can access detailed statistics and restrict Wi-Fi access to app. Unfortunately, for Windows OS users it is not too simple. Most of the provided solutions are either too basic and offer limited functionality or too complex, designed primarily for advanced users. These solutions are often separated into different settings menus or other parts of the user interface, making it difficult for users to find and access the necessary features. Moreover, third-party solutions are mostly designed for advanced users, with complex or outdated design. This lack of user-friendly and accessible network monitoring and administration solutions for Windows OS users is a significant issue that can lead to overall frustration and unwillingness to deal with their use of the Internet.

This thesis is divided into two main parts. In the first, background part, author will analyse and compare existing solutions with similar functionality to the application being developed in this thesis. Next, author will provide a detailed overview of the chosen frameworks, libraries, and technologies used in the development of the application. The author will explain the reasoning behind the technology choices. In the second, the practical part, author will demonstrate the development progress and implementation of the core application parts. The author will present the implementation of the application's features, discuss any challenges faced during development, and provide solutions to overcome these challenges.

1.1 Problem statement

According to article - “A Methodical Review on Network traffic monitoring and Analysis tools” by Prabhjot Kaur and Neeti Misra [1] third-party network traffic monitoring and administration applications offer a vast and distinctive range of functionality in comparison to built-in solutions. However, these solutions are primarily designed for advanced users, which results in overcomplication and may not be suitable for ordinary users.

In addition, according to multiple articles [2, 3, 4] Windows has been criticized for having inconsistent UI/UX designs, which are characterized by multiple design patterns and outdated designs from previous versions of Windows. Examples of such inconsistent design can be found in Windows Task Manager, Resource Monitor, and Firewall Manager. These inconsistencies can lead to user’s confusion, poor user experience, difficulties in navigating and understanding the interface. In addition, third-party solutions designed for administrators generally have a steep learning curve.

The goal of this thesis is to solve these problems by developing an application for monitoring and administering network traffic for Windows operating systems. Despite the administrative nature of the theme, this project will focus on the development aspects, with a particular emphasis on creating a user-friendly interface and providing only essential features. It should be noted that while some of the functionalities can be found in existing applications or tools, the intention is to create a solution that is both streamlined and approachable for the end user.

2 Background

In this section of the thesis, the author will explore the general concepts of network traffic monitoring and administration. Explaining the general concepts of monitoring and administration is necessary as it provides a foundational understanding of the core functionality that the developed application will aim to implement.

Next, author will analyse and compare existing applications with similar functionality to the application being developed in this thesis. Analyses will assist in guiding the development of the application and will address the thesis problems such as decision of user-interface and user-experience design implementation, as well as selection of the feature set.

Finally, author will overview the analyse and select technologies used to create the desired application.

2.1 Main concepts of network traffic monitoring and administration

In order to develop a desired application with monitoring and administrating functionality, it is essential to understand network traffic and administrating concepts technological basics: how they work, perform, and how to use them. The two concepts are the core features that the application will implement, and the critical role it plays in ensuring applications security and efficiency. In this section author will delve into these concepts in some detail and analyse their importance in the context of the application being developed.

2.1.1 Network traffic monitoring

Network traffic is defined as something arisen from the bidirectional flow from Origin to Destination. Network traffic monitoring is observation of the inflow and outflow of traffic moving in-across the network. Network traffic analysis is the technique of extracting the features from the traffic to understand its behaviour. Through the analysis process, it is possible to gather meaningful data that can be used to draw a wide range of conclusions or generate statistics. [1]

A network is a group of two or more than two connected computers. The Internet is a network of networks — multiple networks around the world that are all interconnected with each other. In networking, a packet is a small segment of a larger message. Data sent over computer networks, such as the Internet, is divided into packets. These packets are then recombined by the computer or device that receives them. Packets consist of two portions: the header and the payload. The header contains information about the packet, such as its origin, destination IP addresses (an IP address is like a computer's mailing address), packet length, priority, and information about the payload or content of the packet. The payload is the actual data. [5, 6].

Network packet capture is the act of recording packets that traverse a computer network, including every packet header and packet payload. Once packets have been captured, analyzation software can view the header and payload of any captured packet [6].

2.1.2 Network administration

Network administration is a process of managing, monitoring, maintaining, and securing a computer network [7]. In Windows Operating Systems network administrating is typically referred to as a Windows Firewall manipulation.

A Firewall is a network security device that monitors and filters incoming and outgoing network traffic based on an organization’s previously established security policies. At its most basic, a firewall is essentially the barrier that sits between a private internal network and the public Internet. A firewall’s main purpose is to allow non-threatening traffic in and to keep dangerous traffic out. [8]

2.2 Existing solutions comparison

In this section, various existing solutions for network monitoring and administration will be explored and compared. The author will select a range of applications to reflect different feature sets, target audiences, and price points. To ensure a fair comparison, the author will establish general criteria for evaluation. The author will analyse each application based on these criteria and summarize its strengths and weaknesses.

Additionally, a table will be created to highlight the differences and similarities among existing solutions. After conducting this comparison, the author will list the observations based on the experience using the different applications. These observations will help guide the development of the application, particularly with user interface and user experience design, and decide the feature set.

2.2.1 Selection of the solutions to compare

The choice of applications to compare is based on the popularity of the app, its target audience, its list of features, and its design. The main focus was to cover a wide range of applications that can potentially have a similar functionality as the solution being developed. The selection is limited to include only those solutions that are widely used and have the potential to offer valuable insights into the best implementation approaches. The selection incudes both, build-in and third-party solutions. Based on the selection criteria 4 application were chosen: Windows Task Manager, Windows Resource Monitor, Wireshark and GlassWire.

2.2.2 Comparison criteria

In order to better compare applications, different criteria have been defined to make it easier to categorize and evaluate network monitoring and administration applications. The criteria for the comparison is based on article - “A Methodical Review on Network traffic monitoring and Analysis tools” by Prabhjot Kaur and Neeti Misra [1] as well importance for the development process.

- Feature set – application functionality and its limitations
- UI/UX design and overall user-friendliness – usability of the application
- Target audience – for whom application is designed to and who uses it

Additionally, the author will provide a personal perspective on the compared applications, including experience using them.

2.2.3 Comparison

This part consists of existing solution comparison based on criteria described above.

Windows Task Manager

Windows Task Manager is a built-in application in Microsoft Windows operating systems that allows users to monitor system performance and manage running applications [9].

The feature set of the Windows Task Manager includes monitoring system performance metrics such as CPU usage, memory usage, disk usage, and network activity. Additionally, the Task Manager provides a list of running applications and their resource usage, allowing users to manage and terminate processes as needed. [9]

The interface of Windows Task Manager is also straightforward, with a tabbed layout that allows users to navigate between different sections of the application easily. While the design may not be visually appealing, it is functional and easy to use. It is worth noting that Windows Task Manger received the major design update in 11th version of Windows Operating System. Despite visual appearance improvement, application core functionality and overall user experience remained the same. [9]

The target audience of Windows Task Manager is primarily individual users and IT professionals who need to monitor system performance and manage running processes. As a built-in application, it is available to all users of Microsoft Windows operating systems at no additional cost [9].

Overall, Windows Task Manager provides a basic set of features for monitoring system performance and managing running processes. While it may not have the advanced features of other third-party applications, its simplicity and accessibility make it a useful tool for many users.

In the author's opinion, Windows Task Manager is a useful and reliable tool for monitoring and managing system resources, but it may lack the additional features and visual appeal of other dedicated applications. However, for users who do not require advanced network monitoring or security features, Windows Task Manager is a suitable option with a straightforward interface and no cost associated with its use.

Windows Resource Monitor

Windows Resource Monitor is a built-in Windows utility that provides more in-depth information about system resources and processes than Windows Task Manager. It displays real-time data on CPU, memory, disk, and network usage, as well as detailed statistics on individual processes. [10]

One of the most useful features of Windows Resource Monitor is the ability to monitor bytes sent and received by individual processes in a given time frame. This is particularly helpful for identifying applications that may be using excessive network bandwidth. Additionally, users can restrict network access for individual processes from within Windows Resource Monitor. However, this restriction is not permanent and will be lifted if the application is opened again. [10]

The interface is similar to Windows Task Manager, but with more detailed information and graphs to aid in analysis. While not as visually appealing as some third-party applications, the design is functional and easy to navigate. Windows Resource Monitor is used by IT professionals and advanced home users who require more in-depth information about system resources and processes. As a built-in application, it is available to all users of Microsoft Windows operating systems at no additional cost. [10, 11]

Overall, Windows Resource Monitor is a powerful tool for monitoring and managing system resources, particularly for advanced users. While it may not have the same visual appeal as third-party applications, its extensive feature set and no-cost availability make it a valuable tool for many users.

In authors opinion application, while Windows Resource Monitor does provide more information about network than Task Manager, it does have its limitations. One limitation is that it does not allow users to see statistics for a specific time range. Additionally, the ability to restrict internet access for a process is not permanent and will be granted again if the application is opened again. From a UI standpoint, the author finds the graph provided by the application to be useful in visualizing network activity. However, the UI design of Windows Resource Monitor is outdated and can be improved.

Wireshark

Wireshark is a network protocol analyser, or an application that captures packets from a network connection, such as from your computer to your home office or the internet. Packet is the name given to a discrete unit of data in a typical Ethernet network [12].

The application has a wide range of features, including live packet capture from a network interface, displaying, filtering and searching packets with very detailed protocol information, statistics generation, and network troubleshooting tools [12, 1].

The design of the application is straightforward and functional, with a focus on providing detailed technical information. However, the interface can be overwhelming for beginners, as there are numerous settings and options available. [12]

In terms of cost, Wireshark is open-source software and is available for free. This makes it a popular choice for network administrators and security professionals who want an effective network analysis tool without the high cost of commercial alternatives. [12, 11]

Overall, Wireshark is a powerful and feature-rich application that is ideal for advanced users who require detailed network analysis. However, its complexity may make it less suitable for casual users or those with limited technical knowledge.

In authors opinion, Wireshark is extremely powerful software that offers extensive network monitoring and analysis capabilities, making it an essential tool for advanced users or network administrators. However, its complex user interface and steep learning curve may be daunting for beginners or casual users. Additionally, its focus on packet-level analysis may not be necessary for all users, and its detailed network data may require some technical knowledge to interpret. Overall, Wireshark is a valuable tool for its intended audience, but may not be the best choice for all users.

GlassWire

GlassWire is a network security tool that visualizes your past and present network activity on an easy-to-understand graph. The GlassWire tool alerts you to possible threats, manages your firewall, monitors remote servers, and helps anyone understand their network activity [13].

GlassWire's feature set includes real-time visual network monitoring, a first network activity alert, network security monitoring, an application usage tracker, firewall profile management, anomaly detection, and a network time machine [13].

The application has an intuitive and user-friendly interface, with clear graphs and charts that make it easy to understand network activity. It has a sleek and modern design that is visually appealing and easy to use.

In terms of the target audience, GlassWire is marketed toward individual users and small businesses [11]. Its cost is relatively affordable, with a free version and paid options that offer additional features [14].

Overall, GlassWire is a powerful network monitoring and security tool with a user-friendly interface and a variety of features for both home and business users.

In the author's opinion, GlassWire is a robust application that achieves a balance between advanced features and is highly effective user-friendliness. In addition, GlassWire stands out for its high level of usability, making it accessible to a broad range of users. However, many features are only accessible through a paid subscription, which may be a disadvantage for some users. Although the design is aesthetically pleasing, some of its features may provide minimal practical information and primarily serve as visual embellishments.

2.2.4 Comparison Summary

Based on the analyses conducted in the previous section, the author has created a table summarizing the most important comparison criteria for Windows Task Manager, Windows Resource Monitor, Wireshark, and GlassWire. It is important to note that the comparison summary does not consider the entire scope of the applications functionality, but only the features that relate to the two thesis problems.

Table 1. Existing applications comparison summary

App Name	**Network Monitoring features**	**Network Administrating features**	**Author's opinion**
Windows Task Manager	Basic network monitoring functionality: Live app network usage monitoring	Basic administration features: Suspend network processes	Useful and reliable tool for, but it lacks the additional features, visual appeal and statistics.
Windows Resource Monitor	Live process network usage monitoring, graph visualization	Block internet access to the process (not permanent)	More advanced than Windows Task Manager, but has it's limitations and UI drawbacks
Wireshark	Extremely advanced network monitoring functionality: Live network packet capture, with the ability to create detailed statistics	No network administration functionality	Feature rich and powerful tool, but designed for advanced users, lacks visual appeal and learning curve is very steep.
GlassWire	Advanced network monitoring functionality: Live app network usage monitoring, live statistics, app network usage history and statistics	Advanced network functionality: Suspend network processes, block DNS addresses, establish group policies	Application aligns with the functionality and design being developed in this thesis; however some graphs and statistics provide minimal practical information

After analysing and comparing Windows Task Manager, Windows Resource Monitor, Wireshark, and GlassWire, it can be concluded that each of these network monitoring applications has its own strengths and weaknesses. Windows Task Manager provides basic functionality and is only useful in certain cases. Windows Resource Monitor is more advanced than Windows Task Manager but has its limitations and UI drawbacks. Wireshark, on the other hand, is an extremely powerful tool but has a complex design and steep learning curve, making it only suitable for advanced users. GlassWire has a user-friendly interface and is highly effective in monitoring and administration network traffic. However, some of its features are only available through a paid subscription.

Additionally, some of its design choices, for example in showing statistics, may provide minimal practical information and, in authors opinion, primarily serve as visual embellishments.

Despite its limitations, GlassWire will be taken as the main example and inspiration in this thesis. The author will try to overcome its issues, such as the limited functionality available in the free version and the visual embellishments that may not provide practical information.

2.3 Choice of technologies

The selection of appropriate technologies for the development is a critical task that significantly affects the success of the project. Information technologies, which are used in the development of digital platforms, are commonly called technology stacks. The system performance depends on the efficiency of each of the components of the technology stack and on the effectiveness of their interaction. [15]

When considering technology stack choices certain criteria should be considered [16]:

- Author expertise – usage of experience and skills that author already have.
- Project requirements – technology effectiveness in solving project problems.
- Trendiness – technology popularity, usability amongst other developers and companies.

2.3.1 Choice of programming language

The choice of programming language is often tied to the choice of framework because many frameworks are designed to work with specific programming languages [17]. Due to that programming languages will be analysed considering their choice of frameworks. Most popular programming languages for the desktop development are C#, Java, C++ and JavaScript [18].

C# is a modern, object-oriented, and type-safe programming language. C# enables developers to build many types of secure and robust applications that run in .NET [19]. .NET is an open-source platform for building desktop, web, and mobile applications that can run natively on any operating system. The .NET system includes tools, libraries, and languages that support modern, scalable, and high-performance software development.

An active developer community maintains and supports the .NET platform. [20] .NET offers a lot of UI frameworks for developing the desktop application such as WPF, WinForms and MAUI [18]. The main advantage of using the C# for Windows desktop application development is the fact that C# has a great integrated Windows OS support out of the box [21].

Java is a multi-platform, object-oriented, and network-centric language that can be used as a platform in itself. It is a fast, secure, reliable programming language for coding everything from mobile apps and enterprise software to big data applications and server-side technologies. Java is a widely-used programming language it has been a popular choice among developers for over two decades, with millions of Java applications in use today. [22] Selection of desktop development frameworks is limited to main 3 ones: Swing, JavaFX and Spring [18].

JavaScript is lightweight scripting language mainly used for creating dynamic web pages and web applications. However, JavaScript can also be used in creating desktop, mobile applications using Node.js. Node.js is a server-side, open-source JavaScript framework that runs JavaScript code outside a browser. [23] Popular framework choices for desktop development include Electron JS, Node GUI, NW.js [18, 24].

Table 2. Programming languages comparison

Language	**Author’s experience**	**Suitability**	**Difficulty**
Java	Average	Average	High
C#	High	High	High
JavaScript	High	Low	Average

Based on the evaluation of different programming languages and their suitability for the development of the application in this thesis, the author has decided to use C# as the primary programming language. This choice was made due to the language's robustness and large community, as well as its close ties to Windows Operating system, which aligns with the application's target platform and will be useful in network features implementations. Furthermore, the author has good experience with C#, making it a suitable choice for the project.

2.3.2 Choice of framework

Framework is a software platform that provides a foundation for building desktop applications. Frameworks simplify the development process by providing developers with pre-built components, libraries, and tools that can be used to build robust and scalable applications. [25]

When it comes to developing in C#, .NET is the main and most suitable choice due to its close integration with the language. C# is designed to work seamlessly with the .NET framework, which provides a wide range of libraries and tools for desktop application development. In addition, .NET offers extensive support for Windows development and is regularly updated and maintained by Microsoft. [26]. .NET have 2 implementations: .NET Framework and .NET Core. .NET framework was the first software framework introduced by Microsoft, and it was built only for Windows. .NET Core is a newer version of the .NET domain that addresses the limitation of the .NET Framework, such as different OS compatibility and performance issues. However, .NET Core focuses mostly on Web applications development and does not provide support for most desktop development UI frameworks, such as WinForms and WPF. [27]

In .NET desktop development, there are several popular UI frameworks, including Windows Presentation Foundation (WPF), Windows Forms (Win Forms), and Multi-platform App UI (MAUI) [18]. Each of these frameworks has its own set of advantages and disadvantages, and choosing the right one for a particular project is crucial for its success. In this section, author will evaluate these frameworks and decide on which one to use for the development of the application in this thesis.

WPF, stands for Windows Presentation Foundation is a development framework and a sub-system of .NET Framework. WPF is used to build Windows client applications that run on Windows operating system. WPF uses XAML as its frontend language and C# as its backend languages. [28] This means that WPF provides more flexibility and control over the layout and appearance of the UI. WPF features great Templating, Binding, and Styling capabilities [26]. WPF is the most popular and stable framework for the building the GUI, it has great community support [29]. However, WPF can be resource-intensive, impacting performance on lower-end systems, and may be more challenging to learn and integrate with legacy systems [30].

Windows Forms or WinForms is graphical user interface (GUI) class library included as a part of Microsoft's .NET Framework. WinForms uses a more traditional approach for designing layout which is based on forms and controls**.** This means that WinForms is simpler to use for basic UI design. WinForms lacks the advanced features of WPF, which can make it difficult to create modern UIs without custom code. However, it is generally more lightweight and performant, and has an established ecosystem that makes it easier to integrate with legacy systems and find resources and support for development. [30]

MAUI stands for Multi-platform App UI, which is a UI framework in .NET 6 for making Windows, iOS, Android, and macOS applications with one project, one codebase. It is an evolution over Xamarin.Forms and takes code reusability to the next level. However, MAUI is a new framework, which currently lacks community support and extensive documentation. [31]

Based on the evaluation of available frameworks for desktop development in .NET, the author has decided to use Windows Presentation Foundation (WPF) as the primary GUI framework for the application, and .NET Framework 7 accordingly. This choice was made due to WPF's great binding functionality and its ability to build complex and aesthetically pleasing user interfaces. Additionally, the author does not require cross-platform compatibility as the application is intended to be developed solely for the Windows operating system. Therefore, utilizing the specific features of WPF and .NET Framework 7 will help ensure a robust and efficient application that meets the project's requirements.

2.3.3 Choice of network capture library

The project requires the functionality of capturing network packets. This is essential to provide the network traffic monitoring and administration features that are the focus of the application. Capturing packets requires low-level access to network traffic [32]. For this reason, the author will be utilizing a special driver API for capturing packets.

Npcap is the packet capture and sending library for Microsoft Windows. It implements the open Pcap API using a custom Windows kernel driver alongside libpcap library. This allows Windows software to capture raw network traffic (including wireless networks, wired ethernet, localhost traffic) using a simple, portable API. [32]

To use Npcap API in .NET desktop application, author will be using the SharpPcap library. SharpPcap is a fully managed, cross platform (Windows, Mac, Linux) .NET library for capturing packets from live and file-based devices. [33]

3 Methodology

The research method has three components. The first part consists of the application requirements gathering. Throughout the process of comparing the existing solutions, a comparative analysis has been made based on the articles, surveys, and statistics which were gathered by the author. The criteria for the comparison is based on article - “A Methodical Review on Network traffic monitoring and Analysis tools” by Prabhjot Kaur and Neeti Misra [1] as well importance for the development process. The development of the application consists of a functional analysis and a UI analysis. The functional analysis helps to define the features and requirements necessary for the application to meet the needs of the users. The UI analysis, on the other hand, helps to ensure that the application is user-friendly and provides a good user experience.

In the second part, based on the findings of the comparative analysis, the functional and non-functional requirements for the application are defined. The analysis provides insights into the strengths, weaknesses, and limitations of existing applications with similar functionality, which in turn helps to identify the necessary features and design elements for the new application. The application requirements are defined in such a way so that it addresses the identified issues and shortcomings, as well as to provide solutions to the identified problems.

Three main features have been defined for the application based on the findings of the comparative analysis. The first feature is the ability to view network statistics over a specific period of time, showing how much data was sent and received by all applications installed on the system. The second feature is the ability to view network statistics for specific applications over a specific period of time. None of the compared solutions offered this feature, making it an important addition to the new application. The third feature is the ability to block internet usage for specific applications, a paid function in some compared solutions and generally difficult to achieve in Windows without third-party tools. This feature is necessary for users who wish to restrict an application's access to the internet.

In terms of the user interface, data visualization is an important aspect for better user experience. Graphs are one of the best visualization methods [34], making it an important design element for the new application. A table view with detailed statistics will also be included in the UI, along with a list of all applications that use the internet for easy reference.

Third part describes application development process. To solve the problems described in the thesis, the author developed a prototype for the desktop application. The technology stack was decided through the process of technologies comparison. The development part also presents detailed overview of implementation of the core application functionality and user interface, outlines the architectural design, coding practices and addresses identified problems and the corresponding solutions.

Application will be tested using the functional testing approach. Each function and UI element of the application will be tested, by providing appropriate input, verifying the output against the Functional requirements.

In the end, the results of the application development will be analysed, and a summary of the development results will be provided.

3.1 Application Requirements

In this section, the author defines the functional and non-functional requirements of the application being developed in this thesis, based on the background analysis. Functional requirements are the features and capabilities that the application must have, while non-functional requirements refer to the performance, security, and other aspects that are not directly related to functionality.

Application main functions are receiving user's network traffic and visualize it; block and unblock application access to network.

Network traffic information is received by intercepting the user's main network interface traffic flow.

Restricting application access to the network is achieved via writing the firewall rules.

3.2 Functional requirements

The following are application functional requirements:

- As a user I want to see overall network statistics over a specific period of time: how much data was sent and received by all application installed on the system.
- As a user I want to specific application network statistics over a specific period of time: how much data was sent and received by specific application installed on the system.
- As a user I want to block access to the internet to the specific application installed on the system.
- As a user I want to unblock access to the internet to the specific application previously blocked by this application.

3.2.1 Non-functional requirements

The following are application functional requirements:

- Network statistics is displayed in the form of graphs
- Network statistics is displayed in the form of tables
- The user can use the application without authorization
- Traffic interception does not modify the packet in any form
- Application does not read packet payload, but only information in traffic header
- Application works in Windows 10 and Windows 11 Operating Systems
- Captured data in stored in database

3.2.2 Limitations

Due to the author's limited time frame to complete the thesis, certain requirements mentioned earlier have some limitations as follows:

- Application will collect network traffic only while application is running
- Application requires admin permission access
- Application won’t collect data in the background

4 Development of the application

In this section, the process of creating the app is described in depth. This section is divided into three parts: project structure, the backend and the frontend. Project structure describes architectural pattern usage and general application structure. Backend part describes the process of implementing features functionality such as working with the active network processes, methods for capturing packets and retrieving necessary information, and linking captured packets with the active processes. Frontend part describes the process of creating the UI components, showing aggerated statistics and information, and communicating with backend.

4.1 Project structure

The project architectural structure follows MVVM pattern. MVVM (Model – View - ViewModel) is an architectural pattern that cleanly separates the business logic of an application from the user interface. The goal of MVVM is to make the view completely independent from the application logic.

The model represents the app’s domain model, which can include a data model as well as business and validation logic. It communicates with the ViewModel and lacks awareness of the View.

The View represents the user interface of the application and holds limited, purely presentational logic that implements visual behaviour. The View is completely agnostic to the business logic. View never contains data, nor manipulates it directly. It communicates with the ViewModel through data binding and is unaware of the Model.

The ViewModel is the link between the View and the Model. It implements and exposes public properties and commands that the View uses by way of data binding. If any state changes occur, the ViewModel notifies the View through notification events. [35]

A visual explanation of MVVM pattern can be seen in Figure 1:

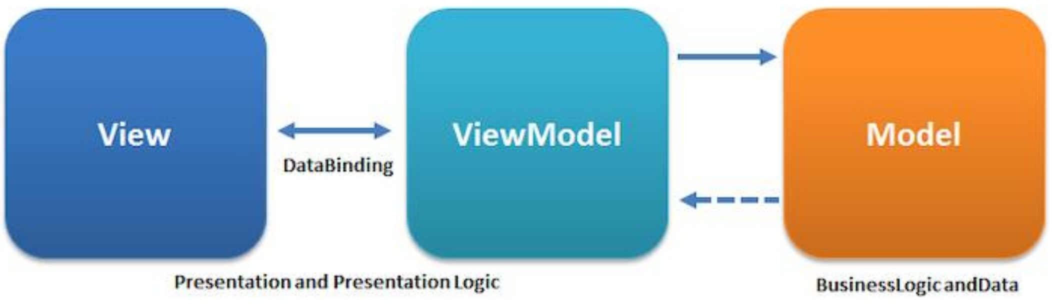


Figure 1. MVVM Architecture communication pathways [36].

The project folder structure follows the packaging by feature layout logic. Packages contain all classes that are required for a feature. The independence of the package is ensured by placing closely related classes in the same package. This guarantees the high cohesion within packages and low coupling between packages.

Cohesion refers to the degree of logical relationship of package members to each other. High relationship between members ensures package independence. Low cohesion not only reduces independence but also significantly reduces reusability and understandability.

Coupling refers to the degree of interdependence between packages or classes. Low coupling significantly increases maintainability. [37]

A simplified visualisation of the project folder structure can be seen in Figure 2:

```
.
└── net-traffic-monitor
      ├── Components
      │    ├── AppCard
      │    │    ├── Styles
      │    │    ├── AppCardModel.cs
      │    │    ├── AppCardView.xaml
      │    │    └── AppCardVM.cs
      │    ├── AppStatistics
      │    │    ├── Styles
      │    │    ├── AppStatisticsModel.cs
      │    │    ├── AppStatisticsView.xaml
      │    │    └── AppStatisticsVM.cs
      │    └── Graph
      │      ├── Styles
      │      ├── GraphModel.cs
      │      ├── GraphView.xaml
      │      └── GraphVM.cs
      ├── Services
      │    ├── ProcessCapture
      │    │    ├── ProcessDataModel.cs
      │    │    └── ProcessCaptureService.cs
      │    ├── PacketCapture
      │    │    ├── PacketDataModel.cs
      │    │    └── PacketCaptureService.cs
      │    └── NetworkBlock
      │      └── NetworkBlockService.cs
      ├── Images
      ├── Themes
      ├── Util
      ├── App.xaml
      └── MainWindow.xaml
```

Figure 2. Simplified project folder structure

4.2 Backend

In the context of this project, backed refers to main features implementation in the form of services. Main features of the project are getting information about active processes and application behind them, capturing application packets, disabling and enabling application access to the internet.

4.2.1 Active network processes capture

When an any installed application needs to connect to the network or communicate with other devices on the network, it creates a network process. This network process is responsible for handling network traffic, sending and receiving data packets, and managing network connections. The network process is created by the application when it calls the appropriate network APIs provided by the operating system. [38]

In the context of this thesis, retrieving information about the active processes on a system is useful for two main reasons. Firstly, it allows for the display of all the currently active applications that utilize the network at any given moment, providing a comprehensive view of network activity. Secondly, this information is necessary for binding captured packets to the actual applications responsible for generating them.

For retrieving information about active network processes, application runs the PowerShell command “netstat -on”. Netstat is a command-line tool that displays active network connections, routing tables, and other network interface information [39].

- Parameter “o” displays active network connections and their status, as well as the process ID of the program that is using each connection [39].
- Parameter “n” displays active network connections and their status without resolving hostnames or port names [39].
- Parameter “on” is a conjunction of these two parameters [39].

Appendix 2 displays sample output of “netstat -on” command.

The result of the command execution gives basic information about the process, such as

- PID - process identification code [39].
- Local address - the network address of the endpoint that the application is listening on or connected to. It includes the IP address of the local machine and the port number of the application. [39]

From PID the information about parent process and associated application name and icon could be retrieved to display it in the UI. Local address is useful for the packet and process binding, which would be shown in the next sections.

To get the necessary information about the process, firstly process is found based on the PID. Then, using the *FileViresionInfo* class the application name is retrieved, see figure 3. Using the *Icon* class application icon is retrieved, see figure 4.

```
private string? GetProcessAppName(Process process)
 {
  var processModule = process.MainModule;
  var filename = processModule?.FileName;
  if (filename == null) return null;
  var versionInfo = FileVersionInfo.GetVersionInfo(filename);
  return versionInfo.ProductName ?? versionInfo.FileDescription ??
  Path.GetFileNameWithoutExtension(filename);

 }
```

Figure 3. Application name extraction from Process

```
private BitmapSource GetProcessIconPath(Process process)
{
 var processModule = process.MainModule;
 if (processModule?.FileName == null) return null;
 Icon? icon =
 Icon.ExtractAssociatedIcon(processModule.FileName);
 BitmapSource? res = null;

 if (icon != null)
 {
  res = Imaging.CreateBitmapSourceFromHIcon(icon.Handle,
  Int32Rect.Empty,
  BitmapSizeOptions.FromEmptyOptions());
 }
 else
 {
  res = new BitmapImage
  (new Uri(@"/Images/noimage.png", UriKind.Relative));
 }
 res.Freeze();
 return res;
}
```

Figure 4. Application icon extraction from Process

Appendix 3 displays active network process capture implementation in code.

4.2.2 Packet capture

For capturing the network packets, the author is using SharpPcap library.

Firstly, the main network interface that the Windows Operating system is using to send and receive network packages is found. The NIC provides the physical interface between the device and the network, enabling the device to send and receive data over the network. [40]

Next, using the SharpPcap library device is marked as open to be read by Pcap API. For receiving the packages, the callback function is assigned to the on packet receive event. Callback handles the received network packets; it filters only the necessary ones and passes it for the information retrieval function.

Next, from the network packet extracted only necessary information: total packet size and source or destination port. Then, the packet port binds together the process port, which is received during the active network processes capture. Process port is extracted from the Local Address.

Appendix 4 displays the packet capture implementation in code.

4.2.3 Using the active network processes capture and packet capture

This section explains how the active network processes capture and packet capture is used in the application.

Those two features run as a continuous background task through the process of the application lifespan. Running background processes or tasks in a separate thread from the UI is important because it can help prevent the application from becoming unresponsive or freezing while the process is running. This is because the UI (Foreground) thread is responsible for handling user input and updating the display, so if a long-running process is executed on the UI thread, it can cause delays or crashes. By executing the process in a separate thread, the UI remains responsive, and the user can continue to interact with the application while the process is running. [41]

The process of tasks execution starts with crating the Timer with a set interval. When the timer ticks, the new Task sequence instance is created, and the task execution starts immediately. First, executed the task in Background thread that retrieves necessary information. Once the task completes, another task is created in the Foreground thread to update the UI with the results of the previous task execution. The created tasks are managed using *CancellationToken* to ensure proper execution and cancellation if needed. Cancellation token is responsible for cancelling the task in any given moment [42].

Appendix 5 displays the execution of active network processes capture as a background task.

4.2.4 Blocking Internet access

In order to block Internet access for an application, the author is using the *NetFwTypeLib* library. *NetFwTypeLib* enables to communicate with Windows Firewall API [43]. The code creates a firewall rule object and sets its properties, such as the rule name, application name, and direction. The action is set to block access, and the rule is enabled. Finally, the rule is added to the Windows Firewall using the *INetFwPolicy2* interface.

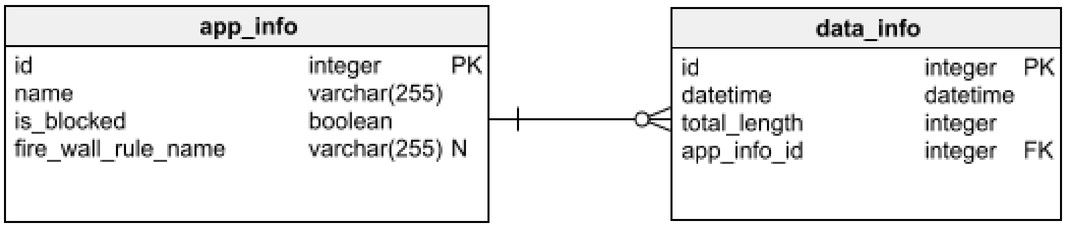
This allows the application to programmatically block Internet access for the specified application.

Appendix 6 displays the implementation of Blocking internet access functionality.

4.2.5 Database

In order to efficiently store and manage information about application internet activity and to check if the user has blocked access to an application, a database is required. For this purpose, the author has chosen SQLite, which is a popular choice for desktop applications. SQLite is an embedded database, which means that it is integrated into the application and does not require a separate server [44]. This provides several advantages, such as simplicity, ease of use, and lower resource consumption [45].

Figure 3 shows database ERD scheme.



app_info		
id	integer	PK
name	varchar(255)	
is_blocked	boolean	
fire_wall_rule_name	varchar(255)	N

data_info		
id	integer	PK
datetime	datetime	
total_length	integer	
app_info_id	integer	FK

Figure 5. Database ERD scheme

The "app_info" table in the database is responsible for storing relevant application information such as the application name, state of application restriction to the internet, and the name of the corresponding firewall rule that blocks internet access for the application.

The "data_info" table is responsible for storing packet capture data, including the date and time of the capture, total packet length, and a reference to the corresponding application table.

The author utilizes an ORM solution, specifically the Entity Framework, for the management and connection to the database. This provides a simplified and efficient approach to interact with the database, enabling the author to easily manipulate data and use the database migrations [46]. The communication between the database and application services is established via repositories. Repository pattern creates an abstraction layer between the data access and the business logic layer of an application. By using it, we are promoting a more loosely coupled approach to access data from the database [47]. Repositories follows pattern which can be seen in figure 6.

```
public interface IRepository<TEntity> where TEntity: class
{
      List<TEntity> GetAll();
      TEntity Get(int id);
      TEntity Add(T entity);
      TEntity Update(T entity);
      void Delete(int id);
}
```

Figure 6. Repository Interface

Appendix 7 displays the implementation of AppInfo Repository.

4.3 Layers communication

The communication between different layers in the application follows a structured approach. The data captured by the application is stored in a database, and the services are responsible for capturing the data and processing the information about the packets and processes. The database is automatically updated with the new information received by the services through repositories.

The model views can communicate with the services to start the capture, perform an action or get the required information. The services act as an intermediary between the model views and the database, providing the necessary data to the views for processing.

Once the data is retrieved from the services, it is bound to the view for display to the user. This approach ensures that the layers of the application are loosely coupled and can communicate with each other efficiently. Additionally, it enables the application be easily maintainable and extensible in the future.

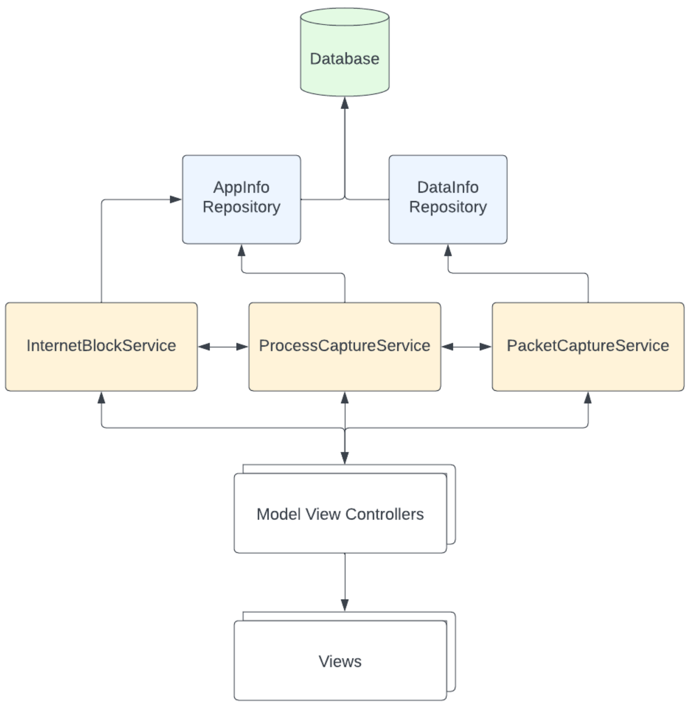


Figure 7. Layers communication

4.4 Frontend

Frontend displays backend information in a form of visual representation. Throughout the application requirements gathering it was decided to first create a graph view representation of user's network as well as detailed statistics about the specific application.

To implement the user interface in the developed application, the author utilized WPF XAML. XAML is a declarative markup language that allows developers to define the user interface of a WPF application. The XAML file contains a hierarchical structure of elements represented by tags, which can include attributes and nested elements. These elements can define various parts of the UI, such as controls, layouts, and styles [26]. Figure 8 shows the example of XAML tag.

```
<StackPanel Grid.Row="2" HorizontalAlignment="Center"
Orientation="Horizontal">
 <Label FontSize="15" FontWeight="Bold" Content="Total: "/>
 <Label FontSize="15" Content="{Binding Total}"/>
</StackPanel>
```

Figure 8. XAML tags

One of the advantages of using XAML is that it supports data binding, allowing for the separation of UI design and code-behind logic. Data binding enables the UI elements to be automatically updated when the data changes, and vice versa, which enhances the application's responsiveness and reduces the amount of code required [26]. Figure 8 shows the example of data binding.

4.4.1 Application Views

On the left-hand side of the application, a navigation bar is located, which displays the list of active applications that are currently using network connections, see figure 9. This information is immediately available to the user upon opening the application, as the process capture tasks are initiated automatically. By default, no application is selected, but the user can choose an application to perform specific activities in the view layout located on the right-hand side of the application.

This design allows users to easily access and navigate the application's features without any unnecessary steps. The navigation bar's intuitive layout ensures that the user can quickly locate and select the application they wish to monitor or administer, while the view layout provides a clear and concise overview of the selected application's network activity.

On the content page, the user can switch between the graph view and the detailed statistics view. The buttons to switch between the views are located at the top of the content page.

The graph view displays a real-time visualization of the network activity of all applications. The y-axis of the graph represents the amount of data sent or received by the application. The x-axis represents the time of the capture. The graph is automatically updated every minute, and the user can zoom in or out by using mouse scroll wheel. Detailed information about the capture time is displayed when the user clicks on a specific graph point. In the detailed information, there is information about every application captured during that specific period of time, including the amount of data sent or received. The graph is initiated automatically when the capture process tasks are started. Additionally, the user can enter a specific date and jump to that date on the graph.

Appendix 8 displays the implementation of Graph View in XAML.

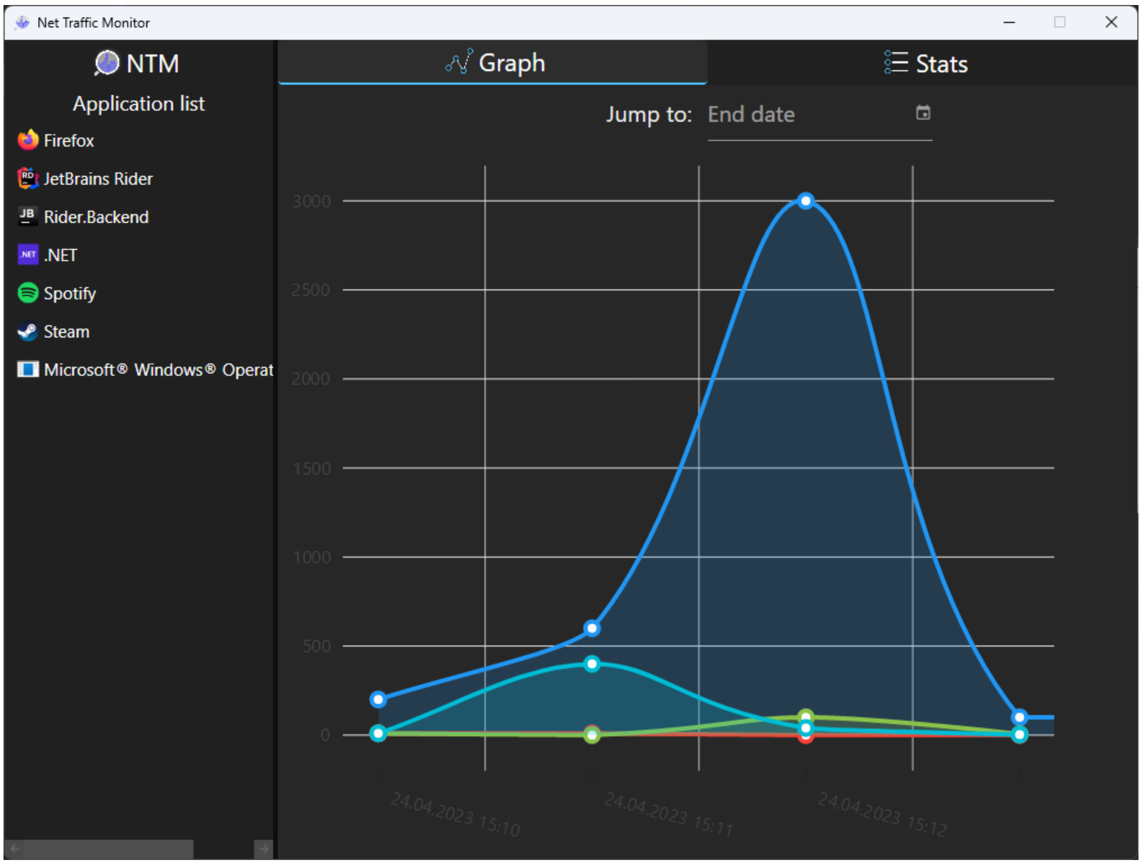
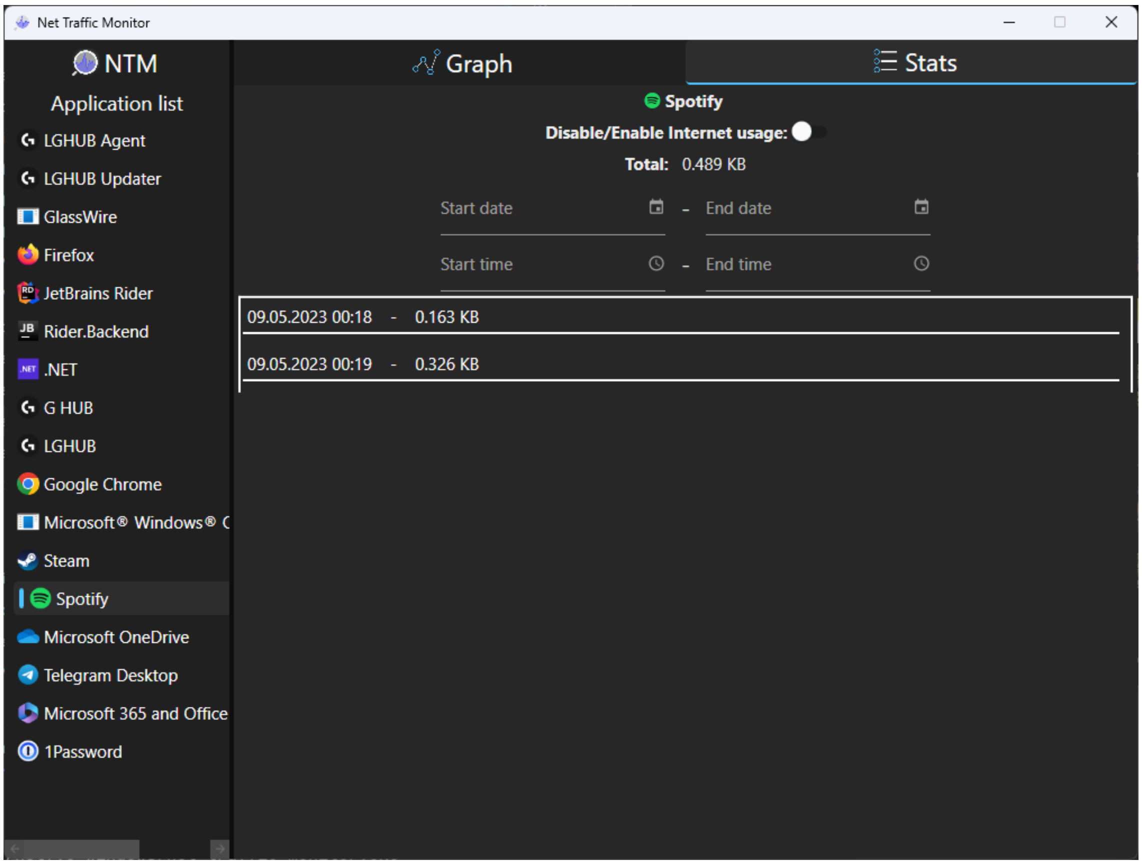


Figure 9. Application Graph view

On the statistics content view, detailed statistics are presented to display valuable information, see figure 10. The user must first select an application before the statistics are shown. If the user has not yet chosen an application, the text "Choose the application first" will be displayed instead of the statistics. The statistics view includes a button that allows the user to enable or disable the internet usage of the selected application. The total amount of captured data is displayed, as well as a data selection feature where the user can choose a specific date range to view the data usage within that time frame.


Net Traffic Monitor

Spotify

Disable/Enable Internet usage:

Total: 0.489 KB

Start date - End date

Start time - End time

```
09.05.2023 00:18  -  0.163 KB
09.05.2023 00:19  -  0.326 KB
```

Figure 10. Application Statistics View

4.5 Results and analysis

In result of the development, using C# programming language, .NET and WPF frameworks author successfully created a prototype of application that fulfils the objectives of monitoring and administering network traffic.

To capture network traffic, the author implemented a dedicated service that utilizes low-level network APIs to intercept and analyse packets. To capture active processes author integrated usage of the PowerShell application functionality. To block internet access author used library that provides API access for communicating with Windows Firewall.

To store and manage the captured data, a database was designed and implemented using SQLite. The database serves as a reliable and efficient repository for storing information about application identification, its network usage, and other related metadata.

The user interface of the application consists of various views and components to present the captured information effectively. This includes graphical representations such as graphs, providing visual insights into network statistics over time. Additionally, detailed tables are utilized to present specific application details, allowing users to explore and analyse network usage in a structured manner.

4.5.1 Testing

To conduct a review, the author launched the application locally on both Windows 10 and Windows 11 devices. The author was able to successfully see how much data was sent and received by specific application installed on the system, navigate between the application views, and block internet access to selected applications. The application met the functional and non-functional requirements specified in the previous chapters, including the ability to capture network statistics over a specific period of time, block internet usage for selected applications, and display data in a visually appealing manner using graphs and tables.

In addition to locally testing the application's functionality and user interface, the author also verified the validity of the captured data by comparing it to data obtained from other applications such as GlassWire and Windows Task Manager. The captured data was consistent and accurate, confirming the effectiveness of the implemented capture process and database management.

4.5.2 Possible improvements

Initially, the author had contemplated incorporating additional functionalities; however, they had to be postponed for later for the sake of time management. Possible future improvements are as follows:

- Enhance the viewing of total network usage by providing an alternative to the current method of combining graph data.
- Implement a feature that displays the total network load.
- Provide users with the ability to remove unused apps from the app list, which will contribute to a more streamlined interface.
- Implement a feature that displays the real-time network status of applications, indicating whether they are currently active or not. This would enhance the user's ability to manage and monitor their network usage.
- Implement a feature that allows users disabling the DNS addresses.

6 Conclusion

After conducting a thorough comparative analysis and defining the functional and non-functional requirements, the author developed an application for monitoring and administering network traffic for Windows operating systems. The developed application provides essential features and a user-friendly design, which was achieved by implementing graph and table views, that aims to make network monitoring and administration easier for users.

The development process included implementing the necessary functionalities such as capturing network traffic data, providing users with the ability to view network statistics over a specific period of time, blocking the internet access to the installed applications.

In authors opinion, the user interface was designed to be intuitive and easy to use, featuring both graph and table views for displaying network statistics. The graph view provides a visual representation of network activity, allowing users to easily identify trends and patterns in their network usage.

Although the application meets the defined requirements, there is still potential for further improvements and features, such as implementing more advanced network analysis tools, adding the ability to remove unused applications from the app list, and displaying the network status of specific applications.

In conclusion, the thesis contributes by introducing a functional and user-friendly application for monitoring and administering network traffic on Windows operating systems. The application addresses the identified issues and shortcomings of existing solutions while providing essential features and a straightforward user interface.

References

[1] N. M. Prabhjot Kaur, “A Methodical Review on Network Traffic Monitoring & Analysis Tools,” *A Journal of Composition Theory,* vol. 12, no. 9, p. 5, 2019.

[2] M. K. A. S. Mehwish Umer, “Usability and Accessibility Evaluation of,” *International Journal of Innovative Research in Computer,* vol. 5, no. 11, p. 17, 2017.

[3] R. KUMAR, “Why User experience of windows 10 is getting bad?,” UX Planet, 2019 Jul 19. [Online]. Available: Why User experience of windows 10 is getting bad?. [Accessed 10 May 2023].

[4] P. Thurrott, “Windows 10 at 3: The Good, the Bad, and the Ugly,” Thurrott, 05 Jul 2023. [Online]. Available: https://www.thurrott.com/windows/windows-10/162506/windows-10-3-good-bad-ugly. [Accessed 10 May 2023].

[5] Cloudflare, Inc., “What is a packet? | Network packet definition,” Cloudflare, Inc., [Online]. Available: https://www.cloudflare.com/learning/network-layer/what-is-a-packet/. [Accessed 21 March 2023].

[6] Endace , “What is Network Packet Capture?,” [Online]. Available: https://www.endace.com/learn/what-is-network-packet-capture. [Accessed 21 March 2023].

[7] Staff Contributor, “Network Administration,” DNSstuff, 22 December 2022. [Online]. Available: https://www.dnsstuff.com/network-administration. [Accessed 21 March 2023].

[8] Check Point, “What is a Firewall?,” [Online]. Available: https://www.checkpoint.com/cyber-hub/network-security/what-is-firewall/. [Accessed 21 March 2023].

[9] Minitool, “Introduction to Task Manager [MiniTool Wiki],” [Online]. Available: https://www.minitool.com/lib/task-manager.html. [Accessed 8 April 2023].

[10] D. Parchisanu, “How to use the Resource Monitor in Windows,” Digital Cititzen, 17 January 2019. [Online]. Available: https://www.digitalcitizen.life/how-use-resource-monitor-windows-7/. [Accessed 4 May 2023].

[11] G. Bidasaria, “6 Best Network Monitoring Tools for Windows 10/11,” 8 December 2022. [Online]. Available: https://techwiser.com/network-monitoring-tools-for-windows/. [Accessed 1 May 2023].

[12] Wireshark Foundation, “Wireshark User’s Guide,” [Online]. Available: https://www.wireshark.org/docs/wsug_html_chunked/. [Accessed 23 March 2023].

[13] GlassWire, “The official user guide for GlassWire.,” [Online]. Available: https://www.glasswire.com/userguide/. [Accessed 23 March 2023].

[14] GlassWire, “Glasswire Pricing,” [Online]. Available: https://www.glasswire.com/pricing/. [Accessed 2 April 2023].

[15] Evgeny Nikulchev, Dmitry Ilin and Alexander Gusev, “Technology Stack Selection Model for Software Design of Digital Platforms,” *mdpi,* vol. 1, pp. 1-2, 2020.

[16] E. Altynpara, “How to Choose Technology Stack for Web Application Development: Tips To Follow,” CLEVERROAD, [Online]. Available: https://www.cleveroad.com/blog/web-development-stacks/. [Accessed 10 April 2023].

[17] K. Horne and M. Levanduski, “Languages and Frameworks for Programming in 2023,” Digital, 3 February 2023. [Online]. Available: https://digital.com/best-web-hosting/languages-and-frameworks/. [Accessed 10 April 2023].

[18] P. Khatun, “A Guide to Desktop Application Development in 2021,” Squash apps, 4 September 2021. [Online]. Available: https://squashapps.com/blog/desktop-application-development-guide-2021/. [Accessed 11 April 2023].

[19] Microsoft, “A tour of the C# language,” February 02 2023. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Accessed 8 April 2023].

[20] Amazon, “What Is .Net?,” [Online]. Available: https://aws.amazon.com/what-is/net/. [Accessed 2023 April 2023].

[21] D. Karczewski, “The State Of C# Development In 2022,” Ideamotive, 18 October 2021. [Online]. Available: https://www.ideamotive.co/blog/the-state-of-csharp-development. [Accessed 10 April 2023].

[22] Amazon, “What Is Java?,” [Online]. Available: https://aws.amazon.com/what-is/java/. [Accessed 9 April 2023].

[23] Mozilla Foundation, “JavaScript,” Mozilla Corporation, [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript. [Accessed 11 April 2023].

[24] R. Senaratne, “JavaScript Frameworks for Building Desktop Applications,” JavaScript in Plain English, 11 May 2020. [Online]. Available: https://javascript.plainenglish.io/javascript-frameworks-for-building-desktop-applications-35ee2370f25d. [Accessed 11 April 2023].

[25] B. O'Grady, “What is a Framework? Why We Use Software Frameworks,” [Online]. Available: https://codeinstitute.net/global/blog/what-is-a-framework/. [Accessed 11 April 2023].

[26] Soft source solutions, “.Net Desktop Application Development: Things To Know,” [Online]. Available: https://www.sourcesoftsolutions.com/net-desktop-application-development-things-to-know/. [Accessed 10 April 2023].

[27] A. Lomas, “.NET Core vs .NET Framework: An In-depth Comparison,” net solutions, 27 June 2022. [Online]. Available: https://www.netsolutions.com/insights/net-core-vs-net-framework/. [Accessed 10 April 2023].

[28] M. Chand, “What Is WPF,” C# Corner, 13 June 2019. [Online]. Available: https://www.c-sharpcorner.com/blogs/what-wpf-is1. [Accessed 10 April 2023].

[29] Microsoft, “Why modern desktop applications,” [Online]. Available: https://learn.microsoft.com/en-us/dotnet/architecture/modernize-desktop/why-modern-applications. [Accessed 12 April 2023].

[30] ByteHide blog, “WPF vs WinForms – Which One is Right for Your Project?,” 13 January 2023. [Online]. Available: https://www.bytehide.com/blog/wpf-vs-winforms. [Accessed 11 April 2023].

[31] M. Delleci, “Meet .NET MAUI, the Technology Replacing Xamarin.Forms,” cellenza blog, 6 April 2022. [Online]. Available: https://blog.cellenza.com/en/mobile/meet-net-maui-the-technology-replacing-xamarin-forms/. [Accessed 10 April 2023].

[32] Npcap, “Npcap Reference Guide,” [Online]. Available: https://npcap.com/guide/index.html#npcap-description. [Accessed 9 April 2023].

[33] C. Morgan, “sharppcap,” [Online]. Available: https://github.com/dotpcap/sharppcap. [Accessed 9 April 2023].

[34] M. M. Gomes, “Data Visualization: Best Practices and Foundations,” Designers, [Online]. Available: https://www.toptal.com/designers/data-visualization/data-visualization-best-practices. [Accessed 1 May 2023].

[35] E. G. Gallardo, 09 January 2023. [Online]. Available: https://builtin.com/software-engineering-perspectives/mvvm-architecture. [Accessed 14 April 2023].

[36] Wikipedia, “Model–view–viewmodel,” [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel. [Accessed 10 May 2023].

[37] M. E. Oral, “Package by Layer vs Package by Feature,” Medium, 1 June 2021. [Online]. Available: https://medium.com/sahibinden-technology/package-by-layer-vs-package-by-feature-7e89cde2ae3a. [Accessed 14 April 2023].

[38] C. Tucker, “Free Code Camp,” [Online]. Available: https://www.freecodecamp.org/news/osi-model-networking-layers-explained-in-plain-english/. [Accessed 12 April 2023].

[39] Daisy, “What Is Netstat Command and How to Use It,” EaseUS, 22 March 2023 . [Online]. Available: What Is Netstat Command and How to Use It. [Accessed 16 April 2023].

[40] tutorialspoint, “What is network interface card (NIC)?,” [Online]. Available: https://www.tutorialspoint.com/what-is-network-interface-card-nic. [Accessed 11 April 2023].

[41] Microsoft, “Foreground and background threads,” [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/threading/foreground-and-background-threads. [Accessed 11 April 2023].

[42] Microsoft, “CancellationToken Struct,” [Online]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken?view=net-8.0. [Accessed 3 May 2023].

[43] H. DuPreez, “Windows Firewalls and .NET,” CodeGuru, 28 June 2018. [Online]. Available: https://www.codeguru.com/dotnet/windows-firewalls-and-net/. [Accessed 4 May 2023].

[44] GreenRobot, “Embedded databases explained,” [Online]. Available: https://greenrobot.org/database/embedded-database/. [Accessed 4 May 2023].

[45] SQlite, “Appropriate Uses For SQLite,” 16 12 2022. [Online]. Available: https://www.sqlite.org/whentouse.html. [Accessed 4 May 2023].

[46] I. V. Abba, “What is an ORM – The Meaning of Object Relational Mapping Database Tools,” 21 October 2022. [Online]. Available: https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/. [Accessed 1 May 2023].

[47] M. Spasojevic, “ASP.NET Core Web API – Repository Pattern,” CodeMaze, [Online]. Available: https://code-maze.com/net-core-web-development-part4/. [Accessed 2 May 2023].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – “netstat -on” output sample

```
Active Connections

Proto  Local Address      Foreign Address        State          PID
TCP    127.0.0.1:9010     127.0.0.1:58616    ESTABLISHED      27484
TCP    127.0.0.1:9010     127.0.0.1:58654    ESTABLISHED      27484
TCP    127.0.0.1:9010     127.0.0.1:58695    ESTABLISHED      27484
TCP    127.0.0.1:9100     127.0.0.1:58643    ESTABLISHED      36488
TCP    127.0.0.1:49823    127.0.0.1:49842    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:49856    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:49867    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:49868    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:58594    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:58595    ESTABLISHED      14780
TCP    127.0.0.1:49823    127.0.0.1:62771    ESTABLISHED      14780
TCP    127.0.0.1:49828    127.0.0.1:49864    ESTABLISHED      15068
```

Figure 11. “netstat -on” output sample

Appendix 3 – Active network process capture implementation

```
public List<AppInfo> CaptureNetworkProcesses()
{
 var proc = new Process
 {
  StartInfo = new ProcessStartInfo
   {
    FileName = "netstat",
    Arguments = "-on",
    UseShellExecute = false,
    RedirectStandardOutput = true,
    CreateNoWindow = true
   }
 };

 proc.Start();
 var result = new List<AppInfo>();
 while (proc.StandardOutput.ReadLine() is { } output)
 {
  var outputMatch = _netstatOutputRegex.Match(output);
  if (!outputMatch.Success) continue;

  var subProcessPid = int.Parse(outputMatch.Groups["pid"].Value);
  var address = outputMatch.Groups["address"].Value;
  var portMatch = int.TryParse(address.Split(":")[1], out var port);
  if (!portMatch) continue;

  var subProcess = Process.GetProcessById(subProcessPid);
  if (subProcess.ProcessName == "Idle" || subProcess.ProcessName ==
  "svchost") continue;

  var parent = FindProcessParent(subProcess);
  if (parent != null)
  {
   subProcess = parent;
  }

  var appName = GetProcessAppName(subProcess);
  if (appName != null)
  {
   if (result.Any(ai => ai.Name == appName))
   {
    result.Find(ai => ai.Name == appName)?
    .Pids.Add(subProcess.Id);
   }
   else
   {
    GetProcessIconPath(subProcess);
    result.Add(new AppInfo(appName,
    subProcess.ProcessName,
    GetProcessIconPath(subProcess),
      subProcessPid));
    }
   }
  }
  proc.Close();
  return result;
}
```
Figure 12. Active network process capture implementation

Appendix 4 – Active network process capture implementation

```
public class PacketCaptureService
{
 public void Start()
 {
  var devices = LibPcapLiveDeviceList.Instance;
  var mainInterface = NetworkInterface.GetAllNetworkInterfaces()
  .FirstOrDefault(ni => ni.OperationalStatus == OperationalStatus.Up
  && ni.NetworkInterfaceType != NetworkInterfaceType.Loopback);

  using var device = devices.FirstOrDefault(x => x.Description ==
  mainInterface!.Description);

  if (device != null)
  {
   device.OnPacketArrival +=
   new PacketArrivalEventHandler(device_OnPacketArrival);
   int readTimeoutMilliseconds = 1000;
   device.Open(mode: DeviceModes.Promiscuous
    | DeviceModes.DataTransferUdp
    | DeviceModes.NoCaptureLocal,
   read_timeout: readTimeoutMilliseconds);
   device.StartCapture();
  }

 }

 public void device_OnPacketArrival(object sender,
 SharpPcap.PacketCapture e)
 {
  var rawPacket = e.GetPacket();
  Console.WriteLine(rawPacket.PacketLength);
  if (rawPacket.LinkLayerType == LinkLayers.Ethernet)

  {
  var tcp = Packet.ParsePacket(rawPacket.LinkLayerType,
  rawPacket.Data).Extract<PacketDotNet.TcpPacket>();
  }
 }
}
```

Figure 13. Active network process capture implementation

Appendix 5 – Active network processes capture as a background task

```
public ICommand StartWorkCommand => new RelayCommand(async o =>
{
 timer = new DispatcherTimer();
 timer.Interval = TimeSpan.FromSeconds(5);
 timer.Tick += OnTimerTick;
 timer.Start();
});

public DispatcherTimer timer;

private void OnTimerTick(object? sender, EventArgs e)
{
 RunTask();
}

public CancellationToken token = new CancellationToken();

public void RunTask()
{
 var complete = true;
 var bgTask = new Task(CaptureThread, token);
 var uiTask = task.ContinueWith(x =>
 {
  foreach (var appInfo in res)
  {
   if (AppCards.Any(x => x.Name == appInfo.Name))
   {
    var appCard = AppCards.First(x => x.Name ==
    appInfo.Name);
    foreach (var appInfoPid in appInfo.Pids)
    appCard.AppInfo.Pids.Add(appInfoPid);
   }
   else
   {
    BitmapSource? icon = null;
    Dispatcher.CurrentDispatcher.Invoke(()=> icon = appInfo.Icon);
    var model = new AppCardModel(appInfo.Name, appInfo,
    appInfo.Icon);
    Dispatcher.CurrentDispatcher.Invoke(() =>AppCards.Add(model));
    Application.Current.Dispatcher.Invoke(() =>OnPropertyChanged());
   }
  }
 }, CancellationToken.None
    TaskContinuationOptions.OnlyOnRanToCompletion,
     TaskScheduler.FromCurrentSynchronizationContext());
  bgtask.Start();
}
```

Figure 14. Active network processes capture as a background task

Appendix 6 – Implementation of Blocking internet access functionality

```
public class BlockInternetService
{
 private readonly AppInfoRepository _appInfoRepository;
 public BlockInternetService(AppInfoRepository appInfoRepository)
 {
  this.appInfoRepository = appInfoRepository;
 }

 public void Block(AppInfo appInfo)
 {
  var firewallRule = (INetFwRule)Activator
  .CreateInstance(Type.GetTypeFromProgID("HNetCfg.FwRule"));

  firewallRule.Name = $"NTM - {appInfo.path} Block Internet Access";
  firewallRule.ApplicationName = appInfo.path;
  firewallRule.Action = NET_FW_ACTION_.NET_FW_ACTION_BLOCK;
  firewallRule.Direction=NET_FW_RULE_DIRECTION_.NET_FW_RULE_DIR_OUT;
  firewallRule.Enabled = true;

  var firewallPolicy = (INetFwPolicy2)Activator
  .CreateInstance(Type.GetTypeFromProgID("HNetCfg.FwPolicy2"));

  firewallPolicy.Rules.Add(firewallRule);
  _appInfoRepository.AddFireWallRule(appInfo, firewallRule.Name);
 }

 public void Unblock(AppInfo appInfo)
 {
  var ruleName = _appInfoRepository
  .FindByName(appInfo).FireWallRuleName;

  var firewallPolicy = (INetFwPolicy2)Activator
  .CreateInstance(Type.GetTypeFromProgID("HNetCfg.FwPolicy2"));
  var firewallRule = firewallPolicy.Rules.OfType<INetFwRule>()
  .FirstOrDefault(r => r.Name == ruleName);

  if (firewallRule != null)
  {
   firewallPolicy.Rules.Remove(ruleName);
  }
 }
}
```

Figure 15. Implementation of Blocking internet access functionality

Appendix 7 – Implementation of AppInfo Repository

```
public class Repository : IRepository<AppInfo>
{
 private readonly DbContext _dbContext;
 protected readonly DbSet<AppInfo> RepoDbSet;

 protected virtual IQueryable<AppInfo> CreateQuery
 (bool noTracking = true)
 {
  var query = RepoDbSet.AsQueryable();
  if (noTracking)
  {
   query = query.AsNoTracking();
  }
  return query;
 }

 public Repository(AppDbContext appDbContext)
 {
  _dbContext = appDbContext;
  RepoDbSet = _dbContext.Set<AppInfo>();
 }

 public List<AppInfo> GetAll(bool noTracking = false)
 {
  return CreateQuery(noTracking).ToList();
 }

 public AppInfo? Get(int id, bool noTracking = false)
 {
  return CreateQuery(noTracking).FirstOrDefaultAsync
  (a => a.Id.Equals(id));
 }

 public AppInfo Add(AppInfo appInfo)
 {
  return RepoDbSet.Add(appInfo);
 }

 public AppInfo Update(AppInfo entity)
 {
  return RepoDbSet.Update(entity);
 }

 public void Delete(int id)
 {
  var appInfo = Get(id);
  if (appInfo != null)
  {
   throw new NullReferenceException($"Entity {typeof(AppInfo).Name}
   with id {id} does not exist");
```

```
    }
    RepoDbSet.Remove(appInfo);
  }
}
```

Figure 16. Implementation of AppInfo Repository

Appendix 8 – Implementation of Graph View

```
<UserControl x:Class="net_traffic_monitor.Components.Graph.GraphView"
             xmlns:materialDesign="http://materialdesigninxaml.net/win
             fx/xaml/themes"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/prese
             ntation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
             compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/20
             08"
             xmlns:lvc="clr-
             namespace:LiveChartsCore.SkiaSharpView.WPF;assembly=LiveC
             hartsCore.SkiaSharpView.WPF"
             xmlns:local="clr-
             namespace:net_traffic_monitor.Components.Graph"
             xmlns:b="http://schemas.microsoft.com/xaml/behaviors"
             mc:Ignorable="d"
             d:DesignHeight="300" d:DesignWidth="300">
 <UserControl.DataContext>
  <local:GraphVM/>
 </UserControl.DataContext>

 <b:Interaction.Triggers>
  <b:EventTrigger EventName="Loaded">
   <b:InvokeCommandAction Command="{Binding StartWorkCommand}" />
  </b:EventTrigger>
 </b:Interaction.Triggers>

 <Grid Background="{DynamicResource Secondary}">
  <Grid.RowDefinitions>
   <RowDefinition Height="50"/>
   <RowDefinition/>
  </Grid.RowDefinitions>

  <lvc:CartesianChart Grid.Row="1" Series="{Binding Series}"
  ZoomMode="X" XAxes="{Binding XAxes}"></lvc:CartesianChart>

  <Grid Grid.Row="0">
   <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition />
   </Grid.ColumnDefinitions>

   <Label Grid.Column="0" Content="{Binding AppCardModel.Name}"
    VerticalAlignment="Center" HorizontalAlignment="Right"
    FontSize="20" Margin="10, 0"/>

   <DatePicker Grid.Column="1" Grid.Row="0" Width="200" Height="50"
    HorizontalAlignment="Left"
    materialDesign:HintAssist.Hint="End date"
    materialDesign:HintAssist.Foreground="white"
    FontSize="20"
    materialDesign:HintAssist.IsFloating="False"
    Foreground="White"
    Style="{StaticResource MaterialDesignFloatingHintDatePicker}" />
  </Grid>
 </Grid>
</UserControl>
```

Figure 17. Implementation of Graph View