

PROJECT REPORT
ON
NETADMIN PRO: CENTRALIZED NETWORK MANAGEMENT
SYSTEM FOR EDUCATIONAL INSTITUTIONS

SUBMITTED BY
ANJANI BHAGWAN JHA
&
AMAN RADHESHYAM YADAV
SEAT NO: 05 & 21

PROJECT GUIDE
MRS. KARISHMA JAIN

MSC. (COMPUTER SCIENCE) SEM -IV
2023 – 2024



CONDUCTED AT
CHIKITSAK SAMUHA'S
S. S. & L.S. PATKAR COLLEGE OF ARTS & SCIENCE
AND
V. P. VARDE COLLEGE OF COMMERCE & ECONOMICS
GOREGAON (W). MUMBAI -400062



Chikitsak Samuha's

**Sir Sitaram and Lady Shantabai Patkar College of Arts and Science and
V. P. Varde College of Commerce and Economics**

S.V.Road,Goregaon(West),Mumbai-400062,Maharashtra(India)

AN AUTONOMOUS COLLEGE, University of

**ISO9001:2015 Certified
Best College Award(2016-17)**

**'A+' Grade by NAAC 3rd cycle
DBT Star Scheme Awardee**

India's Education Excellence Award 2018: Berkshire Media LLC, USA

Tel.: 91-022-28723731/28721875

Website.: www.patkarvardecollege.edu.in

E-mail: principal@patkarvardecollege.edu.in & info@patkarvardecollege.edu.in

PROJECT CERTIFICATE

This is to certify that **Mr. Anjani Bhagwan Jha & Aman Radheshyam Yadav** of
S.Y.M.Sc. Computer Science with University Seat no **05 & 21** has completed his project
titled **NETADMIN PRO: CENTRALIZED NETWORK MANAGEMENT SYSTEM FOR**
EDUCATIONAL INSTITUTIONS under the guidance of Project Guide **Mrs. Karishma Jain** as
laid by University of Mumbai in the college during the year 2023-24.

**Project Guide
Co-ordinator**

M.Sc. Computer science

External Examiner

Date: _____

ACKNOWLEDGEMENT

We take this opportunity of submitting this report to express our profound gratitude to the management of “S.S. & L.S. Patkar College of Science and Commerce” for giving us the opportunity to accomplish this project work.

We are also thankful to all the teachers for their kind support and help.

We express our deepest gratitude towards our project guide Mrs. Karishma Jain for her valuable and timely advice during the various phases in my project.

We extend our sincere thanks to our respected Head of Computer Science Department Mrs. Karishma Jain.

We would also like to thank all those unnamed but important people who directly or indirectly helped me in the completion of this project and to our family and friends without whose support, motivation and encouragement this would not have been possible.

**Thanking you,
Anjani Bhagwan jha &
Aman Radheshyam yadav**

Table of Contents

Sr. No	Contents	Page No.
1	Introduction	1
	1.1 Theoretical Background/ history	2
	1.2 Abstract	2
	1.3 Objective of the project	3
2	Requirement Specification	4
	2.1 Hardware Requirement	5
	2.2 Software Requirement	5
3	System planning	6
	Gantt Chart	7
4	System Design	8
	4.1 Methodology Adopted	9
	4.2 Process Model	10
	4.3 UML Diagram-	11
	4.3.1 Use-Case Diagram	11
	4.3.2 System Flowchart	12
	4.3.3 ER Diagram	12
	4.3.4 Class Diagram	13
5	System Implementation: Code implementation	15
6	Results	27
	6.1 Test Cases	28
	6.2 Screen shots	29
7	Conclusion and Future Enhancement	33
8	References	36

**NETADMIN PRO: CENTRALIZED NETWORK MANAGEMENT SYSTEM FOR
EDUCATIONAL INSTITUTIONS**

1. Introduction

1.1 Theoretical Background/ History :-

1.1.1 Evolution of Network Management Systems

Begin by tracing the evolution of network management systems, emphasizing their importance in maintaining and optimizing complex computer networks. Discuss key milestones, such as the development of Simple Network Management Protocol (SNMP) and the emergence of early centralized management solutions.

1.1.2 Challenges in Educational Institutions' Network Management

Examine the unique challenges faced by educational institutions in managing their networks. Explore issues related to scalability, diverse user needs, security concerns, and the increasing complexity of modern educational IT environments.

1.1.3 Centralized Network Management Systems

Define and elaborate on the concept of centralized network management systems. Discuss their role in streamlining network administration tasks, ensuring security, and enhancing overall network performance. Highlight the advantages of a centralized approach for large-scale educational institutions.

1.1.4 Importance of Custom Solutions

Explore the need for tailored network management solutions, especially in the context of educational institutions. Discuss the limitations of generic systems and the benefits of developing a specialized system like NetAdmin Pro to address the specific requirements of educational networks.

1.1.5 Case Studies and Success Stories

Provide examples of successful implementations of centralized network management systems in educational settings. Showcase instances where custom solutions led to improved efficiency, reduced downtime, and enhanced overall network security.

1.2 Abstract :-

In today's technology-driven educational landscape, the efficient management of network infrastructure is paramount to ensure seamless connectivity, security, and optimal performance. The project "NetAdmin Pro" addresses this need by introducing a sophisticated Centralized Network Management System tailored specifically for educational institutions.

NetAdmin Pro offers a comprehensive solution to streamline the complexities associated with managing diverse network components across educational campuses. The system provides a centralized platform that empowers administrators to monitor, configure, and troubleshoot network devices and services from a unified interface. This centralized approach enhances overall network efficiency, reduces downtime, and ensures a secure and reliable connectivity environment. NetAdmin Pro aims to revolutionize network management in educational institutions by providing a centralized, user-friendly, and feature-rich solution. By implementing this system, institutions can ensure a reliable, secure, and high-performance network environment conducive to effective teaching, learning, and administrative operations.

1.3 Objective of the Project :-

The vision behind NetAdmin Pro is to create a secure and conducive digital ecosystem within educational institutions. By fostering an environment where students utilize digital resources ethically and focus on academic pursuits, NetAdmin Pro aims to contribute to the overall enhancement of the learning experience.

Ethical Resource Use: NetAdmin Pro is rooted in the principles of promoting ethical resource use, ensuring that students leverage digital resources for academic purposes.

Innovation: The project reflects a commitment to innovation in network management systems, aiming to set a new standard for centralized control in educational institutions.

Continuous Improvement: NetAdmin Pro is not just a static solution; it is a commitment to continuous improvement. Feedback, collaboration, and future enhancements will drive the evolution of this project.

As we navigate through the documentation, the subsequent sections will delve into the technical aspects, stakeholder considerations, system architecture, and the journey of development. NetAdmin Pro stands as a testament to the fusion of technological innovation with a conscientious approach to education.

Welcome to the exploration of NetAdmin Pro, where secure networks and ethical resource use converge to create a transformative learning environment.

2. Requirement Specification

2. Requirement Specification

2.1 Functional Requirements

1. User Authentication and Authorization:

- Administrators must log in with secure credentials.
- User roles (administrator, support staff) with appropriate permissions should be defined.

2. Centralized Dashboard:

- Provide an intuitive dashboard for administrators to manage network resources and access key features.

3. Remote Access and Support:

- Enable administrators to initiate secure remote connections to client computers.
- Allow real-time screen sharing, remote control, and file transfer during remote sessions.

4. Shared File Transfer:

- Implement a shared folder where administrators can upload and download files to/from client computers.
- Ensure secure file transfer with proper access controls.

2.2 Software & Programming Language:

- **Python:** Python is an excellent choice for desktop application development due to its ease of use, cross-platform compatibility, and a wide range of libraries.

GUI Framework:

- **Tkinter:** Tkinter is a built-in Python library that provides tools for creating graphical user interfaces (GUIs) for desktop applications.

Database:

- **SQLite:** SQLite is a lightweight and embedded database that you can integrate directly into your Python application.

2.3 Hardware Requirement

Processor (CPU): A multi-core processor

Memory (RAM): At least 4 GB of RAM

Operating System: Python is cross-platform, so it can run on Windows, macOS, or Linux.

Storage: Adequate storage space for the operating system, Python interpreter, libraries, and project files. A solid-state drive (SSD) is preferred for faster read/write speeds.

3. System planning

3.1 Gantt Chart :-

A Gantt chart is a project management tool that illustrates work completed over a period of time in relation to the time planned for the work. It typically includes two sections: the left side outlines a list of tasks, while the right side has a timeline with schedule bars that visualize work. The Gantt chart can also include the start and end dates of tasks, milestones, dependencies between tasks, and assignees.



4. System Design

4.1 Methodology Adopted Python:

Python is a high-level, interpreted programming language that emphasizes code readability and simplicity. It supports multiple programming paradigms, including procedural, functional, and object-oriented programming. Python has a vast standard library that provides a wide range of functionality for tasks such as web development, data analysis, and scientific computing. It has a dynamic type system and automatic memory management, making it easy to use and learn. Python is cross-platform and can run on various operating systems such as Windows, macOS, and Linux. It has an active and supportive community of developers and users who contribute to its growth and development. Python is widely used in industries such as finance, education, healthcare, and technology. It is also a popular language for beginners due to its ease of use and readability. Python is open-source and free to use, making it accessible to everyone.

SQLite:

SQLite is a lightweight, serverless relational database management system. It operates as a self-contained, file-based engine, requiring no separate server setup. Key features include cross-platform support, minimal resource consumption, and SQL compatibility. SQLite is commonly used in embedded systems, mobile applications, and desktop software due to its simplicity and versatility. It offers transaction support, is open-source, and is widely adopted for prototyping and development.

Python Libraries :-

Tkinter:- Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications and helps us to interact with code in a simple way.

Socket:- In Python, the socket module provides a way to work with sockets. As we discussed earlier, sockets allow you to establish network connections over various network protocols such as TCP or UDP to send and receive data.

NumPy:- NumPy is a Python library used for working with arrays. NumPy is used to convert our images into some form of an array so that we can store the model that has been trained.

OS:- The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality.

Threading:- This module provides low-level primitives for working with multiple threads (also called light-weight processes or tasks) — multiple threads of control sharing their global data space. For synchronization, simple locks (also called mutexes or binary semaphores) are provided. The threading module provides an easier to use and higher-level threading API built on top of this module.

Random:- This module can be used to perform random actions such as generating random numbers, printing random a value for a list or string, etc. It is an in-built function in Python.

pyqt5:- PyQt5 is cross-platform GUI toolkit, a set of python bindings for Qt v5. One can develop an interactive desktop application with so much ease because of the tools and simplicity provided by this library. A GUI application consists of Front-end and Back-end. PyQt5 has provided a tool called 'QtDesigner' to design the front-end by drag and drop method so that development can become faster and one can give more time on back-end stuff.

4.2 Process Model

The biggest problem we face in the waterfall model is that taking a long duration to complete the product, and the software became outdated. To solve this problem, we have a new approach, which is known as the Spiral model. The spiral model is also known as the cyclic model. In this model, we create the application module by module and handed over to the customer so that they can start using the application at a very early stage. And we prepare this model only when the module is dependent on each other. In this model, we develop the application in the stages because sometimes the client gives the requirements in between the process.

The different phases of the spiral model are as follows:

Requirement analysis

Design

Coding

Testing and risk analysis

Requirement Analysis

The spiral model process starts with collecting business needs. In this, the following spirals will include the documentation of system requirements, unit requirements, and the subsystem needs. In this stage, we can easily understand the system requirements because the business analyst and the client have constant communication. And once the cycle is completed, the application will be deployed in the market.

Design

The second stage of the spiral model is designed, where we will plan the logical design, architectural design, flow charts, decision tree, and so on.

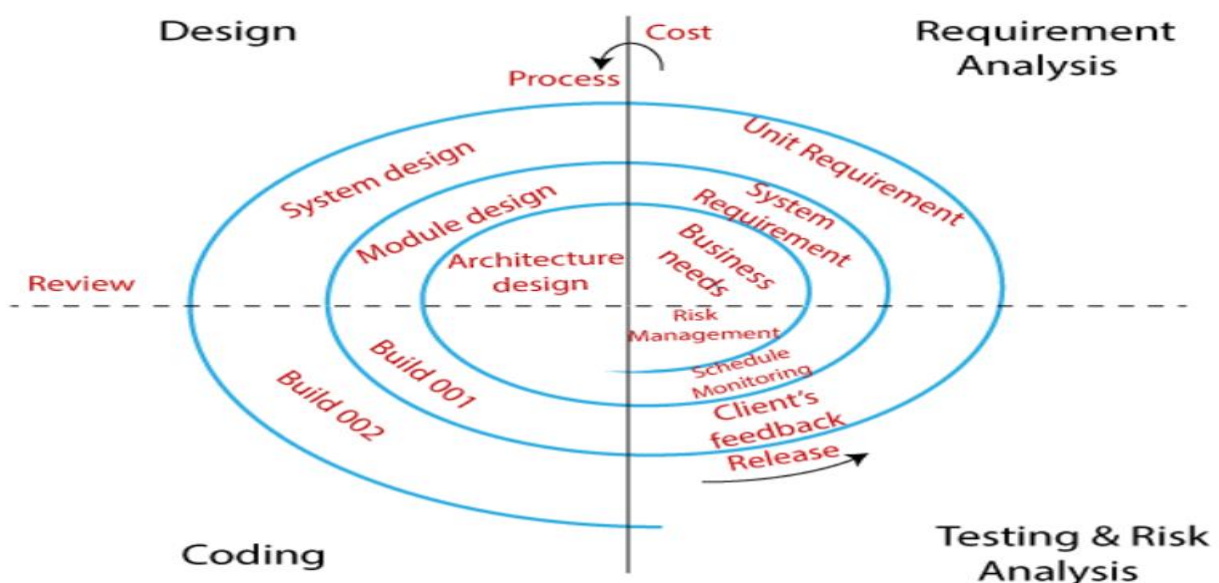
Coding

After the compilation of the design stage, we will move to our next step, which is the coding stage. In this, we will develop the product based on the client's requirement and getting the client's feedback as well. This stage refers to the construction of the real application in every cycle.

And those spirals had an excellent clarity of the requirements, and the design details of an application are known as the build with having version numbers. After that, these builds are transferred to the client for their responses.

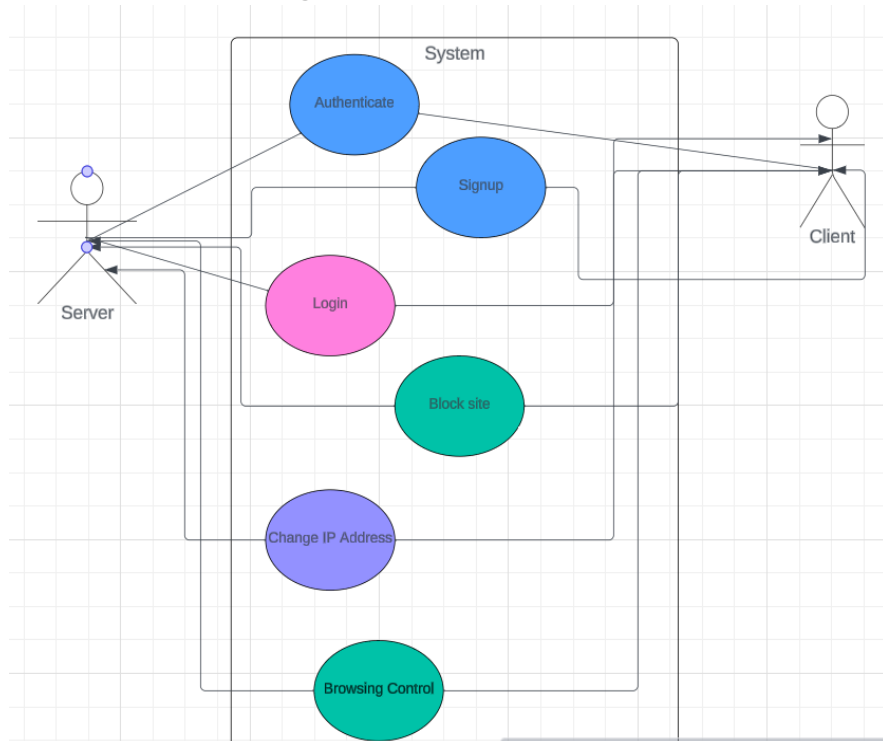
Testing and Risk Analysis

Once the development is completed successfully, we will test the build at the end of the first cycle and also analyze the risk of the software on the different aspects such as managing risks, detecting, and observing the technical feasibility. And after that, the client will test the application and give feedback.

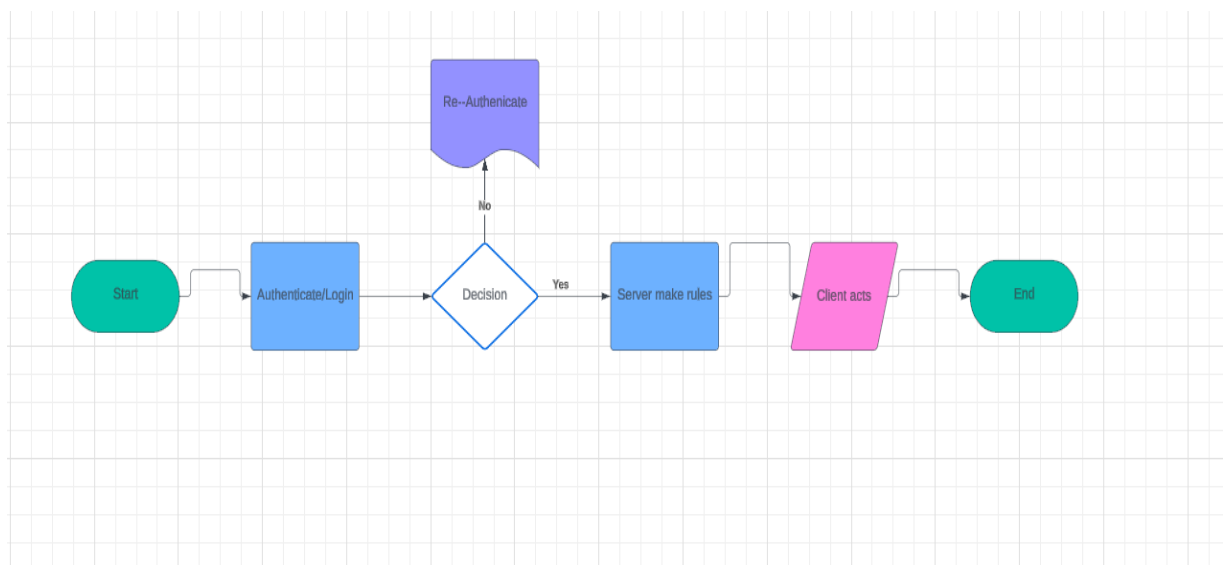


3. UML Diagram

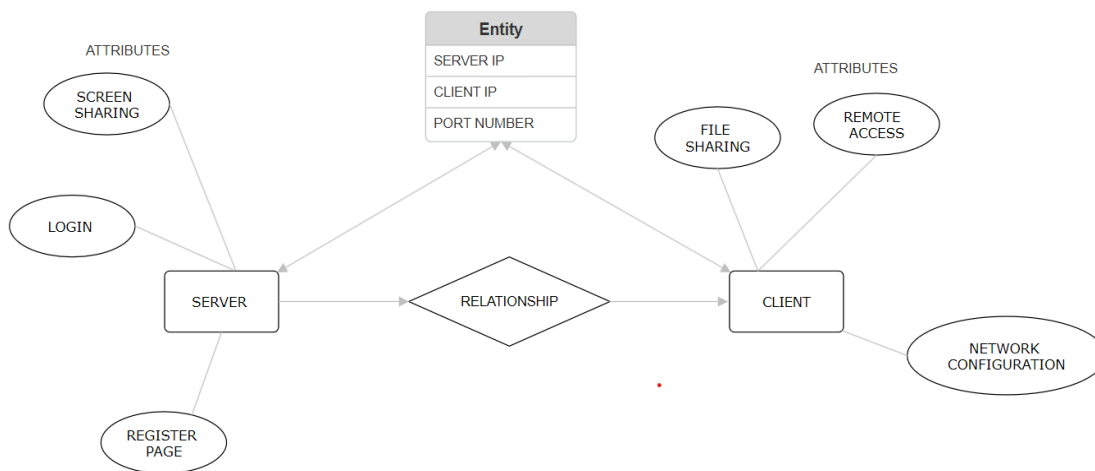
4.3.1 Use-Case Diagram



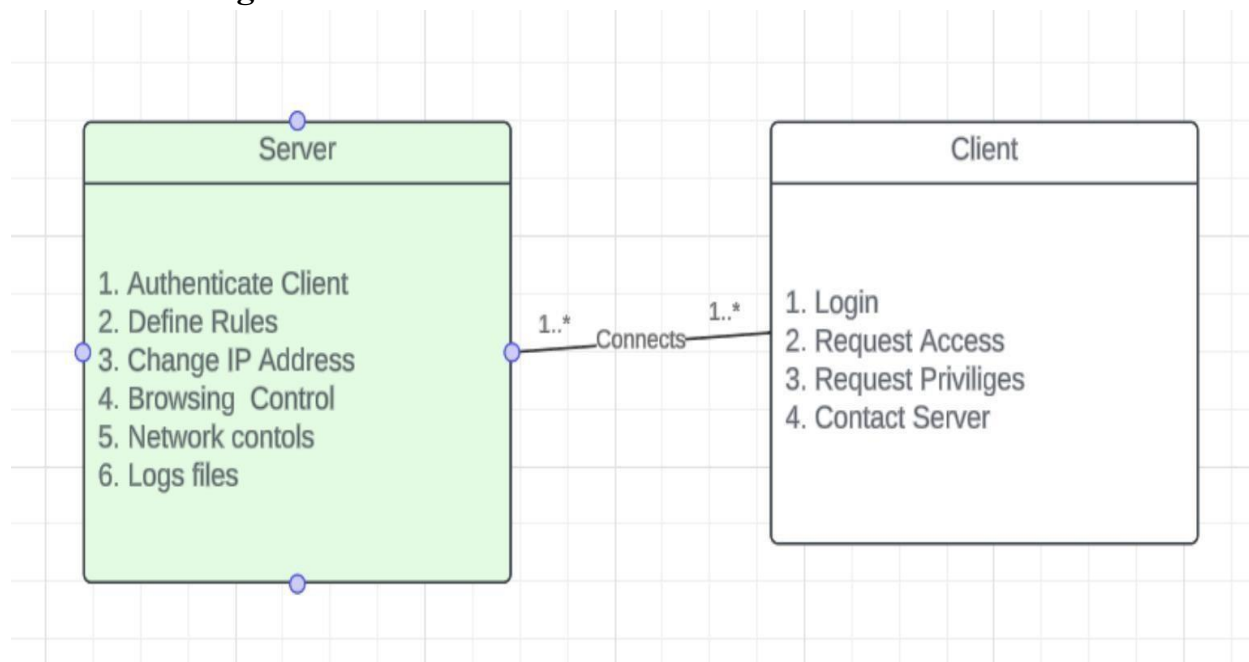
4.3.2 System Flowchart



4.3.3 ER Diagram



4.3.4 Class Diagram



5. System Implementation: Code implementation

Main server page UI Code:-

```
from ui_MainUi import *
from Custom_Widgets.Widgets import *
import socket from os import
getlogin from PIL import
Image import io import
numpy as np from random
import randint import
pyautogui from threading
import Thread import sys
from PySide2.QtWidgets import QMainWindow, QApplication, QLabel, QPushButton, QLineEdit, QAction,
QMessageBox
from PySide2.QtGui import QPixmap
from PySide2.QtCore import Qt from
subprocess import Popen import
subprocess import sqlite3
```

```
class RemoteDesktop(QMainWindow):
    updateImageSignal = Signal(bytes)
```

```
    def __init__(self, conn, addr):
        super().__init__()
        self.conn
        = conn
        self.addr = addr
```

```
        self.initUI()
```

```
    def changeImage(self):
        try:
            print("[SERVER]: CONNECTED: {0}!".format(self.addr[0]))
        while True:
            img_bytes = self.conn.recv(9999999)
            self.updateImageSignal.emit(img_bytes)
        except ConnectionResetError:
            self.conn.close()
        def updateImage(self,
            img_bytes):
            pixmap = QPixmap()
            pixmap.loadFromData(img_bytes)
            self.label.setScaledContents(True)
            self.label.resize(self.width(),
            self.height())
            self.label.setPixmap(pixmap)
```

```
    def initUI(self):
        self.label = QLabel(self)
        self.label.resize(self.width(), self.height())
        self.setGeometry(QRect(pyautogui.size()[0] // 4, pyautogui.size()[1] // 4, 800, 450))
        self.setFixedSize(self.width(), self.height())
```

```

self.setWindowTitle("[SERVER] Remote Desktop: " + str(randint(99999, 999999)))
self.start = Thread(target=self.changeImage, daemon=True)    self.start.start()

self.updateImageSignal.connect(self.updateImage)

class MainUi(QMainWindow):
def __init__(self, parent=None):
    QMainWindow.__init__(self)
    self.ui = Ui_MainWindow()
    self.ui.setupUi(self)
    self.show()

    # Button and LineEdit for Second Server

    # Connect the first server's button to its functionality
    self.ui.RAStartserverBtn.clicked.connect(self.ServerStarted)
    # Connect other buttons to their functionality
    self.ui.RANextBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.SharedFileTransferPage))
    self.ui.RAPrevBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.OtherModulePage))
    self.ui.SFPprevBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.RemoteAccessPage))
    self.ui.SFNextBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.NetworkConfigurationPage))
    self.ui.NCPrevBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.SharedFileTransferPage))
    self.ui.NCNextBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.BrowsingControlPage))
    self.ui.BCPprevBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.NetworkConfigurationPage))
    self.ui.BCNextBtn.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.LogFilePage))
    self.ui.LogPrevBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.BrowsingControlPage))
    self.ui.LogNextBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.OtherModulePage))
    self.ui.OMPrevBtn.clicked.connect(lambda: self.ui.stackedWidget.setCurrentWidget(self.ui.LogFilePage))
    self.ui.OMNextBtn.clicked.connect(lambda:
self.ui.stackedWidget.setCurrentWidget(self.ui.RemoteAccessPage))
    self.ui.RASendCommandBtn.clicked.connect(self.run_second_file)
    self.ui.SFSelectFileBtn.clicked.connect(self.select_file)
    self.ui.SFSendFileBtn.clicked.connect(self.send_file)
    self.ui.OMDocumentBtn.clicked.connect(self.open_pdf)
    self.ui.OMAboutBtn.clicked.connect(self.open_pdf)
    self.ui.OMLogOutBtn.clicked.connect(self.run_second_script)
    self.ui.OMAccountBtn.clicked.connect(self.display_user_info)
    self.ui.LogGenerateBtn.clicked.connect(self.run_third_script) def run_third_script(self):    try:
        # Replace 'second_script.py' with the actual name of your second script
        subprocess.run(['python', 'D:\\NetAdminPro\\logserver.py'], check=True)
    except subprocess.CalledProcessError:    print("Error running second
script")

```

```

def display_user_info(self):
try:

```

```

conn = sqlite3.connect("Account.db")
cursor = conn.cursor() # Use the locally defined 'conn' here

# Fetch user information from the database
cursor.execute("SELECT UserName, Email, MobileNum FROM UsersInfo")
user_info = cursor.fetchone()
if
user_info:
    username, email, mobile_num = user_info
    # Display username, email, and the last 4 digits of the mobile number in a message box
    info_message = f"Username: {username}\nEmail: {email}\nLast 4 digits of Mobile Number: {str(mobile_num)[-4:]}"
    QMessageBox.information(self, 'User Information', info_message)
else:
    QMessageBox.warning(self, 'Warning', 'No user information found in the database.')

except sqlite3.Error as e:
    print(f"Error accessing the database: {e}")
    QMessageBox.critical(self, 'Error', 'Error accessing the database.')
finally:
    conn.close() # Make sure to close the connection when done

def open_pdf(self):
    # Replace 'path/to/your/pdf/file.pdf' with the actual path of your PDF file
    pdf_file_path = "C:\\Users\\sonuj\\Downloads\\sonu_sem5_blackbook.pdf"    url
    = QFile.fromLocalFile(pdf_file_path)
    QDesktopServices.openUrl(url)

def open_pdf2(self):
    # Replace 'path/to/your/pdf/file.pdf' with the actual path of your PDF file
    pdf_file_path2 = 'D:\\NetAdminPro\\project_documentation_softcopy.pdf'    url
    = QFile.fromLocalFile(pdf_file_path2)
    QDesktopServices.openUrl(url)

def run_second_script(self):
    try:
        # Replace 'second_script.py' with the actual name of your second script
        subprocess.run(['python', 'D:\\NetAdminPro\\Login.py'], check=True)    except
        subprocess.CalledProcessError:

    print("Error running second script")

def ServerStarted(self):
    print("[SERVER]: STARTED")
    self.sock = socket.socket() # Assign to the class attribute
    ip_address = self.ui.RAIPLine.text()    port_text =
    self.ui.RAPortLine.text()

    if not ip_address or not port_text:
        QMessageBox.warning(self, "Warning", "Please enter both an IP address and a port number.")
    return

```

```

try:
    # Validate and parse IP address
    socket.inet_pton(socket.AF_INET, ip_address)    except
socket.error:
    QMessageBox.warning(self, "Warning", "Please enter a valid IPv4 address.")
return

try:
    # Validate and parse port number        port_number =
int(port_text)        if not 0 < port_number < 65536:        raise
ValueError("Port number must be between 1 and 65535")    except
ValueError as e:
    QMessageBox.warning(self, "Warning", str(e))
return

try:
    self.sock.bind((ip_address, port_number))
self.sock.listen()        global conn, addr
    conn, addr = self.sock.accept()

    # Start the RemoteDesktop window with the established connection
remote_desktop_window = RemoteDesktop(conn, addr)        remote_desktop_window.show()

except Exception as e:
    QMessageBox.warning(self, "Warning", f"Failed to start server: {e}")

def run_second_file(self):
    ip_address = self.ui.RAIP2Line.text()
port = self.ui.RA2PortLine.text()    if
not ip_address or not port:
    QMessageBox.warning(self, "Warning", "Please enter both an IP address and a port number.")
    Return

try:
    # Validate and parse IP address
    socket.inet_pton(socket.AF_INET, ip_address)    except
socket.error:
    print(f'Invalid IP address: {ip_address}')
    QMessageBox.warning(self, "Warning", "Please enter a valid IPv4 address.")
    Return

try:
    # Validate and parse port number
port_number = int(port)        if not 0
< port_number < 65536:
        raise ValueError("Port number must be between 1 and 65535")
except ValueError as e:        print(f'Invalid port number: {port}')
    QMessageBox.warning(self, "Warning", str(e))
return
    # Run the second file with the provided IP address and port
try:
    print(f'Running with IP: {ip_address}, Port: {port}')
    Popen(['python', 'D:\\NetAdminPro\\SendCommandServer.py', ip_address, port], shell=True)
except Exception as e:
    print(f'Error running second file: {str(e)}')
    QMessageBox.warning(self, "Warning", f'Error running second file: {str(e)}')

```



```

def select_file(self):
    file_path, _ = QFileDialog.getOpenFileName(self, "Select File", "", "All Files (*);;Text Files (*.txt)")
    if
file_path:
    self.ui.SFFileSelectLine.setText(file_path)
self.file_path = file_path

def send_file(self):    try:        if not hasattr(self,
'file_path') or not self.file_path:
    QMessageBox.warning(self, "Warning", "Please select a file.")
return

    with open(self.file_path, 'rb') as file:
file_data = file.read()
    # Replace 'server_ip' and 'server_port' with your server details
server_ip = self.ui.RAIP2Line.text()    server_port =
self.get_server_port()
    print(server_port)

    if not server_ip or not server_port:
        QMessageBox.warning(self, "Warning", "Please enter both an IP address and a port number.")
return

try:
    # Validate and parse IP address
socket.inet_pton(socket.AF_INET, server_ip)
except socket.error:    print(f"Invalid IP
address: {server_ip}")
    QMessageBox.warning(self, "Warning", "Please enter a valid IPv4 address.")
    Return

    # Create a server socket    with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as server_socket:        server_socket.bind((server_ip,
server_port))
        server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.listen(1)
        print(f"Server is listening on {server_ip}:{server_port}")

    # Wait for a client to connect
client_socket, addr = server_socket.accept()
print(f"Accepted connection from {addr}")

    # Extract file name and size from file path
original_file_name = self.file_path.split("/")[-1]    file_size
= os.path.getsize(self.file_path)

    # Send file metadata (original name and size)
client_socket.sendall(original_file_name.encode('utf-8') + b'\x00')
client_socket.sendall(file_size.to_bytes(8, byteorder='big'))

```

```

        # Send file data to the client
client_socket.sendall(file_data)

        print("File sent successfully.")
except Exception as e:
    print(f"Error sending file: {e}")

    def get_server_port(self):
    try:
        port = int(self.ui.RA2PortLine.text())
    if not (0 <= port <= 65535):
        raise ValueError("Port number must be between 0 and 65535")
    except ValueError:
        QMessageBox.warning(self, "Error", "Invalid port number. Using default port.")
    port = None

        # Subtract 1 from the port number if it's a valid port
    if port is not None:
        port -= 1

        return port

def closeEvent(self, event):
    # Implement logic to close the server gracefully
    print("Closing the server...")    try:        if
self.sock:
        self.sock.close() # Close the server socket
        # Perform any additional cleanup operations

    except Exception as e:
        print(f"Error while closing the server: {e}")
if __name__ == "__main__":    app =
QApplication(sys.argv)    window = MainUi()
window.show()    sys.exit(app.exec())

```

Login UI Server code:-

```
import os
import sys
import sqlite3
import subprocess
import threading
from twilio.rest import Client
import random

# Create a SQLite database connection conn
conn = sqlite3.connect("Account.db")
cursor = conn.cursor()

# Create a table to store user information
cursor.execute("""
CREATE TABLE IF NOT EXISTS UsersInfo (
    UserName TEXT PRIMARY KEY,
    Email TEXT,
    MobileNum INT,
    Password TEXT
)
""")
conn.commit()
conn.close()

account_sid = 'AC1afce35a727c6ca1d69d2034c0a62355'
auth_token = 'ae4f9af03bb0651cc74fd7815098c1fd'

# Create a Twilio client
client = Client(account_sid, auth_token)

# Your Twilio phone number
twilio_phone_number = '+14106954344'

def send_verification_code(phone_number):
    # Generate a random six-digit verification code
    verification_code = "".join(random.choices('0123456789', k=6))

    # Send the verification code via SMS
    message = client.messages.create(
        body=f'Your verification code for NetAdminPro software is: {verification_code}',
        from_=twilio_phone_number,
        to=phone_number
    )

    return verification_code

from ui_LoginInterface import *

from Custom_Widgets.Widgets import *
class CodeInputDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.setWindowTitle("Verification Code")
```

```

layout = QVBoxLayout()
self.setLayout(layout)

self.code_input_box = QLineEdit()
self.code_input_box.setPlaceholderText("Enter the verification code")

submit_button = QPushButton("Submit")
submit_button.clicked.connect(self.accept)

layout.addWidget(self.code_input_box)
layout.addWidget(submit_button)
class Login(QMainWindow):
def __init__(self, parent=None):
QMainWindow.__init__(self)
self.ui = Ui_MainWindow()
self.ui.setupUi(self)

loadJsonStyle(self, self.ui, jsonFiles = {
"D:\\NetAdminPro\\RegisterLoginStyle.json"
})

self.show()

self.ui.RegisterBtn.clicked.connect(self.register)
self.ui.LoginBtn.clicked.connect(self.authenticate)

def register(self):
SignUp_UserName_Input = self.ui.SignUpUserNameInput.text()
SignUp_Email_Input = self.ui.SignUpEmailInput.text()
SignUp_Password_Input = self.ui.SignUpPasswordInput.text()
SignUp_ConPassword_Input = self.ui.SignUpConPasswordInput.text()
SignUp_MobileNum_Input = self.ui.SignUpMobileNumInput.text()
self.ui.LoginWidget.setCurrentWidget(self.ui.LoginPage) if not SignUp_UserName_Input or
not SignUp_Email_Input or not SignUp_Password_Input or not SignUp_ConPassword_Input or not
SignUp_MobileNum_Input:
    QMessageBox.warning(self, "Registration Failed", "Please enter all the credentials.")
else:
    # Define a list of allowed email domains
    allowed_domains = ["gmail.com", "outlook.com", "aol.com", "protonmail.com", "proton.me",
    "zohomail.in", "icloud.com", "yahoo.com", "myyahoo.com"]
    # Extract the domain from the provided email address
    email_domain = SignUp_Email_Input.split('@')[-1]

    # Check if the email domain is in the allowed list
    if email_domain not in allowed_domains:
        QMessageBox.warning(self, "Registration Failed", "Email domain is not allowed.")
        Return

try:
    conn = sqlite3.connect("Account.db")
    cursor = conn.cursor()

    # Check if the user already exists
    cursor.execute("SELECT UserName FROM UsersInfo WHERE UserName = ?",

```

```

(SignUp_UserName_Input,))
    existing_user = cursor.fetchone()
    if
existing_user:
    QMessageBox.warning(self, "Registration Failed", "Username already exists.")
elif SignUp_Password_Input != SignUp_ConPassword_Input:
    QMessageBox.warning(self, "Registration Failed", "Password and Confirm Password do not match.")
else:
    # Send the verification code via SMS using Twilio
    verification_code = send_verification_code(SignUp_MobileNum_Input)

    code_input_dialog = CodeInputDialog(self)
result = code_input_dialog.exec_()

    if result == QDialog.Accepted:
        # Get the user's input from the QLineEdit
        user_input = code_input_dialog.code_input_box.text()

        # Check if the user-entered code matches the generated code
if user_input == verification_code:
    # Insert the new user
    cursor.execute("INSERT INTO UsersInfo (UserName, Email, Password, MobileNum) VALUES
(?, ?, ?, ?)",
                    (SignUp_UserName_Input, SignUp_Email_Input, SignUp_Password_Input,
SignUp_MobileNum_Input))
    conn.commit()
conn.close()
    QMessageBox.information(self, "Registration Successful", "Registration successful!")
else:
    QMessageBox.warning(self, "Registration Failed", "Invalid verification code. Registration
failed.")

except sqlite3.Error as e:
print("SQLite error:", e)
    QMessageBox.warning(self, "Registration Failed", "Registration failed. Please try again.")

def authenticate(self):
    Login_UserName_Input = self.ui.LoginUserNameInput.text()
    Login_Password_Input = self.ui.LoginPasswordInput.text()    if
not Login_UserName_Input or not Login_Password_Input:
    QMessageBox.warning(self, "Login Failed", "Please enter both username and password.")

try:
    conn = sqlite3.connect("Account.db")
    cursor = conn.cursor()

    # Check if the user exists and the password matches
    cursor.execute("SELECT UserName, Password FROM UsersInfo WHERE UserName = ?",
(Login_UserName_Input,))
    user_data = cursor.fetchone()
    if
user_data:
        stored_password = user_data[1]    if
Login_Password_Input == stored_password:

```

```

        QMessageBox.information(self, "Login Successful", "Welcome, " + Login_UserName_Input)
self.open_second_window()
        # subprocess.Popen(["python", "D:\\NetAdminPro\\MainInterface.py"])
        # self.close()
        # Add code here to open your main interface window
else:
        QMessageBox.warning(self, "Login Failed", "Invalid password.")
else:
        QMessageBox.warning(self, "Login Failed", "User not found.")

        conn.close()
except sqlite3.Error as e:
print("SQLite error:", e)
        QMessageBox.warning(self, "Login Failed", "Login failed. Please try again.")

def open_second_window(self):
try:
        # Replace 'second_script.py' with the actual name of your second script
subprocess.Popen(['python', 'D:\\NetAdminPro\\main.py'])        self.close()
except Exception as e:
        print("Error running second script:", str(e))
def run_second_script(self):        try:
        # Replace 'second_script.py' with the actual name of your second script
subprocess.run(['python', 'D:\\NetAdminPro\\main.py'], check=True)
except subprocess.CalledProcessError:        print("Error running second
script") if __name__ == "__main__":    app = QApplication(sys.argv)

        window = Login()
window.show()
sys.exit(app.exec_())

```

Main Client.py

```
from ui_ClientInterface import *
from PySide2.QtWidgets import QMessageBox, QMainWindow, QApplication
import sys import socket import threading import subprocess
from PySide2.QtCore import QUrl
from PySide2.QtGui import QDesktopServices

class Client(QMainWindow):
    def __init__(self, parent=None):
        QMainWindow.__init__(self)
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.show()

        # Existing socket for the first code
        self.sock = socket.socket()

        # Connect existing button to the existing function
        self.ui.InfoConnectBTn.clicked.connect(self.connectButtonClicked)
        self.ui.InfoConnectIssueBtn.clicked.connect(self.open_link)

    def open_link(self):
        # Replace 'https://www.example.com' with the actual URL you want to open
        url =
        QUrl('https://docs.google.com/document/d/1ydTi7XODtMvjcA6D1dosMTsCzRaavl
        n0cJwt9zeljc/edit?usp=sharing')
        QDesktopServices.openUrl(url)

    def display_user_info(self):
        try:
            cursor = self.conn.cursor()

            # Fetch user information from the database
            cursor.execute("SELECT UserName, Email, MobileNum FROM UsersInfo")
            user_info = cursor.fetchone()
            if
            user_info:
                username, email, mobile_num = user_info

                # Display username, email, and the last 4 digits of the mobile number in a message box
                info_message = f"Username: {username}\nEmail: {email}\nLast 4 digits of Mobile Number:
                {str(mobile_num)[-4:]}"
                QMessageBox.information(self, 'User Information', info_message)
            else:
                QMessageBox.warning(self, 'Warning', 'No user information found in the database.')

        except sqlite3.Error as e:
            print(f"Error accessing the database: {e}")
            QMessageBox.critical(self, 'Error', 'Error accessing the database.')

    def connectButtonClicked(self):
        global server_ip
        global server_port
        pc_name = self.ui.PcName.text()
        client_ip = self.ui.InfoClientIpAddr.text()
```

```

server_ip = self.ui.InfoServerIpAddr.text()
server_port = self.ui.InfoServerPort.text()

    if not all([pc_name, client_ip, server_ip, server_port]):
        QMessageBox.warning(self, "Warning", "Please fill in all fields.")
return

try:
    # Connect to the server
    self.sock.connect((server_ip, int(server_port)))

    # Send client information to the server
    info_str = f"PC Name: {pc_name}\nClient IP: {client_ip}"
    self.sock.send(info_str.encode())

    # Start receiving commands in a new thread
    receive_command_thread = threading.Thread(target=self.receiveCommand)
    receive_command_thread.daemon = True
    receive_command_thread.start()
except Exception as e:
    QMessageBox.warning(self, "Connection Error", f"Failed to connect: {str(e)}")
return

def receiveCommand(self):
while True:
    data =
self.sock.recv(1024)
data =
data.decode()
    if data:
        # Update your UI or perform any necessary action with the received command
        print("Received command:", data)
        # Execute command received from server (you might need to adapt this based on your needs)
        result = subprocess.Popen(data, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
        stdin=subprocess.PIPE)
        result = result.communicate()
        self.sock.send(result[0])
if __name__ == "__main__":
    app =
    QApplication(sys.argv)
    window =
    Client()
    sys.exit(app.exec_())

```


6. Results

6.1 Test Cases

Sr. No.	Action	Input Data	Expected Result	Result
1.	Login	Username , Password.	Checks whether user exists or not. Validate and redirect to Home Page.	Passed.
2.	Register	User name, Contact No, Email, Password, Confirm Password.	The User details are stored. Redirect to login page.	Passed.

Screen sharing and window

6.2 Test Cases

Sr. No.	Action	Input Data	Expected Result	Result
1.	Remote Access	Server IP address and port number	Screen sharing of a client system screen	Passed.
2.	Connect your pc to the sever	Server IP address and port number client PC name, server IP, and client IP and port number.	To run all window command in the application Start server to send command	Passed.
3	Log file	Server or client IP address and port number	Log file of client showing in the server	Passed

6.2 Screenshoot:-

Sign up & Remote Access

The image displays two screenshots of the NetAdminPro web application interface, showing the 'Sign Up' and 'Remote Access' pages.

Sign Up Page:

- Header: **Sign Up** (with a user icon and a plus sign)
- Sub-header: Enter Your Information Below
- Form Fields:
 - UserName
 - E-Mail
 - Mobile Number With ISD Code
 - Password
 - Confirm Password
- Agree With Our Terms (checkbox)
- Register button (with a right arrow icon)
- Link: [Already Registered ? Login](#)

Remote Access Page:


- Navigation: Previous Page (left arrow) and Next Page (right arrow)
- Header: **Remote Access** (with a monitor icon)
- Sub-header: Enter the Information Below
- Form Fields:
 - Enter IP Address
 - Enter Port Number
- Buttons:
 - Start Server For Screen Sharing
 - Start Server For Send Command

Shared And Network Configuration

NetAdminPro Server

Previous Page

Next Page



Shared File Transfer

Enter The Information Below

Select File


The file You Have Selected

Send File

NetAdminPro Server

Previous Page

Next Page



Network Configuration

Enter the Information Below

● Change I.P. Address Of Client

● Change DNS Of Client

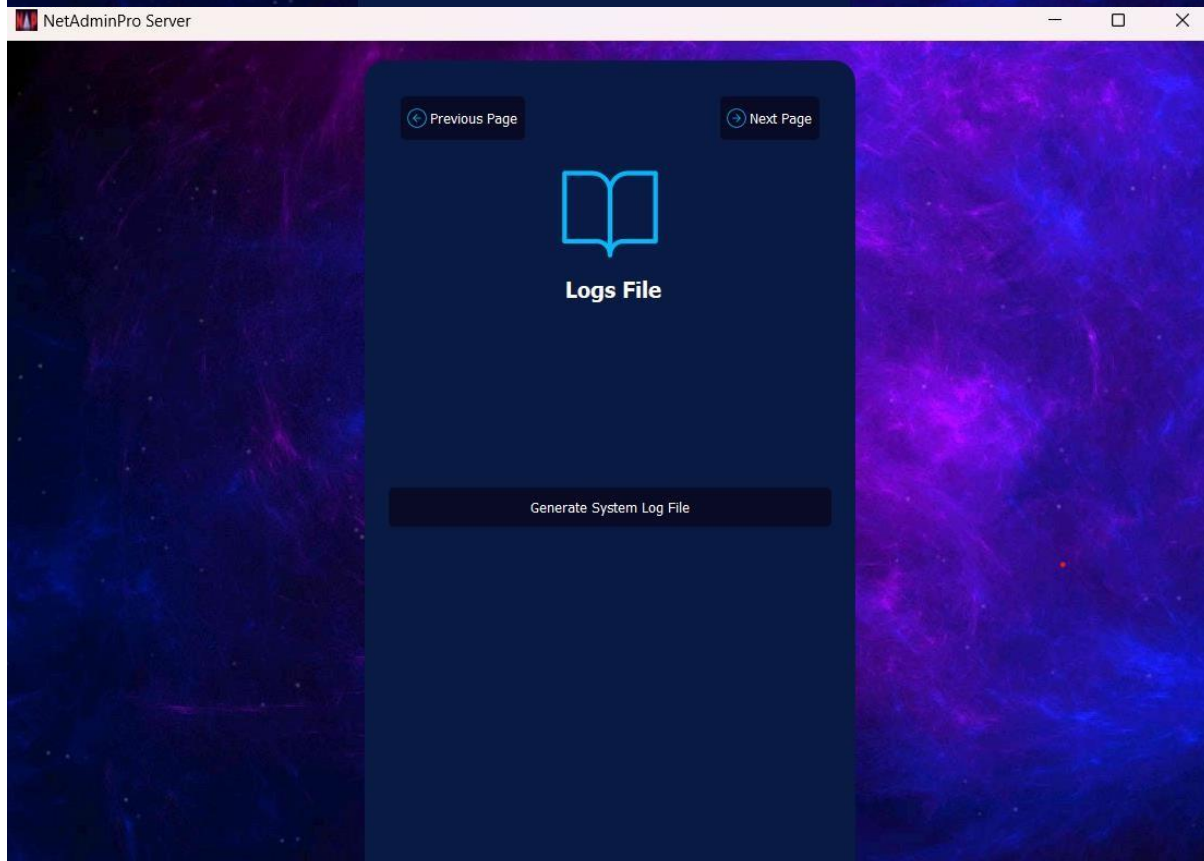
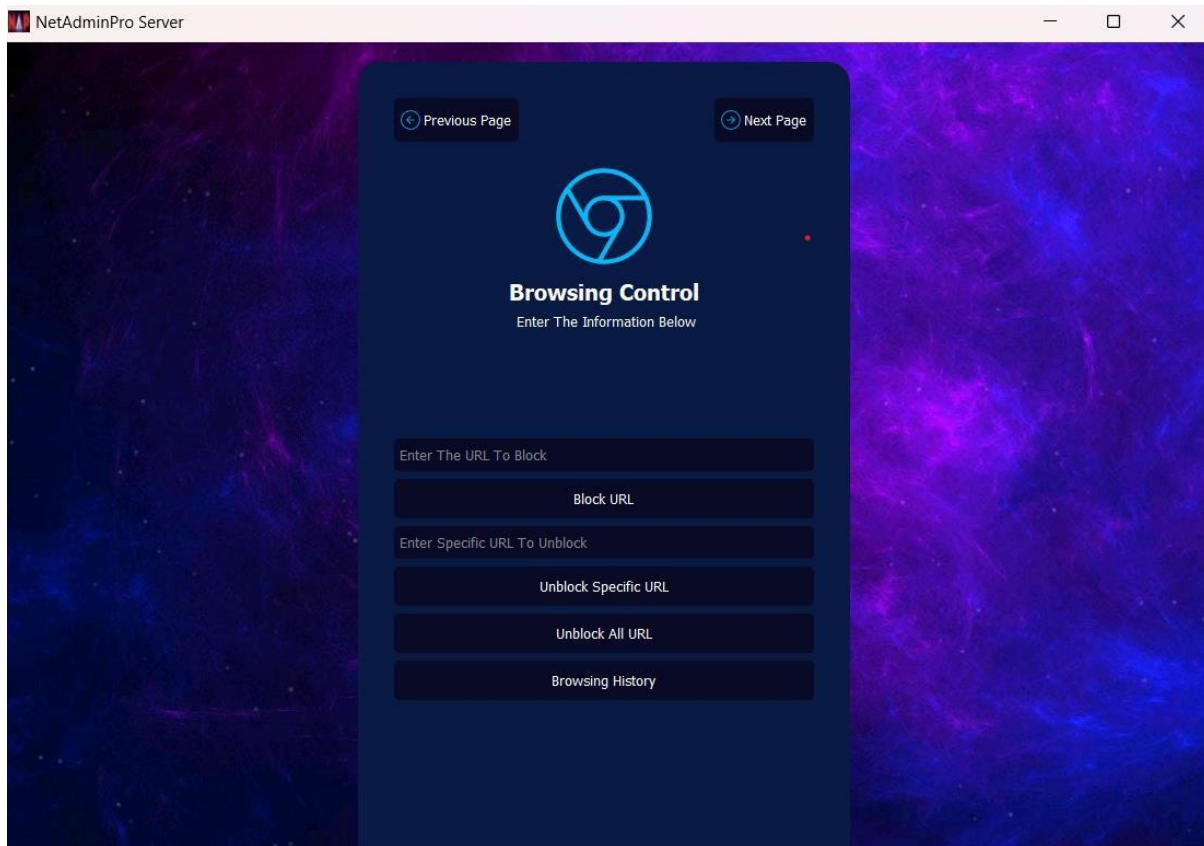
Enter The Information Which You Have Selected

Change

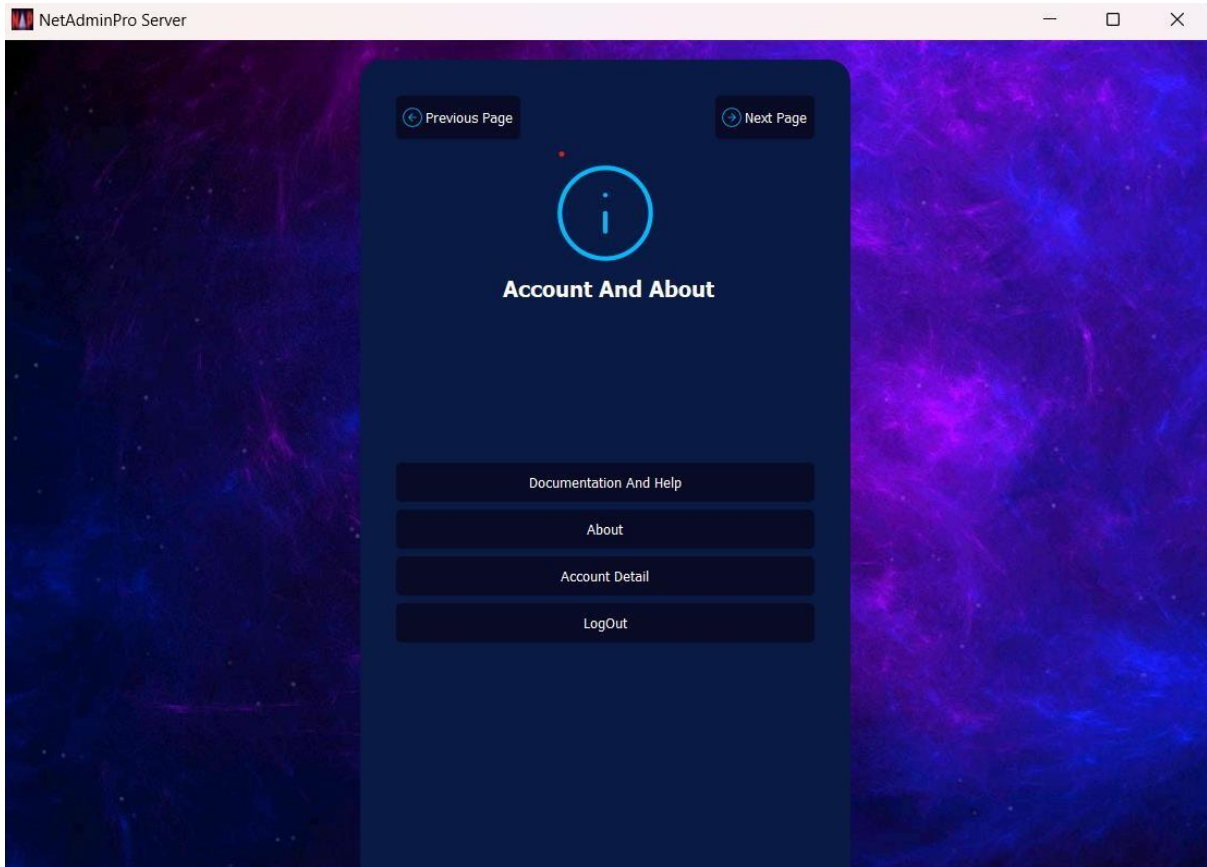
Block Internet

Allow Internet

Browsing control and Logs File



Account and About



7. Conclusion

7.1 Conclusion

In conclusion, the development and implementation of NetAdmin Pro represent a significant stride towards enhancing the efficiency, security, and overall management of network infrastructure within educational institutions. This project aimed to address the unique challenges faced by educational organizations in maintaining a robust and reliable network environment.

1.Enhanced Security:- NetAdmin Pro contributes significantly to strengthening the security posture of educational networks. The implementation of robust authentication and authorization mechanisms mitigates the risk of unauthorized access and potential security breaches.

2.Cost Savings:- By optimizing resource usage and automating routine tasks, NetAdmin Pro contributes to cost savings for educational institutions. Reduced downtime and efficient troubleshooting result in lower operational costs.

3.User Satisfaction:- Students, faculty, and staff benefit from a more reliable and responsive network, leading to increased satisfaction and a positive impact on the overall educational experience.

7.1 Future Enhancement

1. Improved Command Execution:-

- Enhance the command execution mechanism to support a broader range of commands.
- Implement a command history feature.

2. Performance Optimization:-

- Optimize image transmission for better performance.
- Implement compression algorithms to reduce data transfer size.

3. Network Configuration Options:-

- Provide advanced network configuration options, such as proxy support.
- Implement dynamic IP detection and handling.

4. Remote Desktop Recording:-

- Add the ability to record the remote desktop session for later review.
- Implement screenshot capture for specific intervals.

5. Notification System:-

- Implement a notification system to alert users of important events or connection status changes.
- Include sound alerts for critical events.

6. Localization and Internationalization:-

- Support multiple languages to make the application accessible to a broader audience.
- Implement internationalization features for date, time, and numeric form

8. References

1. <https://youtu.be/p3tSLatmGvU?si=kOVDj8I4r-mNUirE>
2. <https://youtu.be/YXPyB4XeYLA?si=UUDX36R3FS6ZUitH>
3. <https://youtu.be/Vde5SH8e1OQ?si=wlujyOKVoUJzW8Tv>
4. https://youtu.be/Z1N9JzNax2k?si=MffH23i2L5rUdx4_
5. <https://www.youtube.com/watch?v=adC48qZ8p5Y>
6. https://youtu.be/Y_iOLhwtfVA?si=bVM3XoVSxAeWXrJW
7. <https://youtu.be/mkBwInKhBsA>
8. <https://youtu.be/jj8L51oiB4c>
9. <https://www.qt.io/>
10. <https://doc.qt.io/>
11. <https://www.python.org/doc/>
12. <https://www.pythonguis.com/pyside2-tutorial/>
13. <https://www.geeksforgeeks.org/python-introduction-to-pyqt5/>
14. https://github.com/KhamisiKibet/QT-PyQt-PySide-CustomWidgets/blob/main/Custom_Widgets/Widgets.py
15. <https://khamisikibet.github.io/Docs-QT-PyQt-PySide-CustomWidgets/>
16. <https://chat.openai.com/share/84de8baf-094f-4640-b587e955db6941b2>