

# WIREGUARD VPN PROJECT – Kaarthee

## 1. Introduction

This report documents the complete design, configuration, and deployment of a manually configured WireGuard VPN. The objective of the project was to understand VPN technology at a low-level by configuring a Linux-based VPN server, a Windows client, routing rules, firewall controls, and verifying secure encrypted communication.

The project intentionally avoids automated tools or scripts. Every component keys, configuration files, networking, routing, firewall rules, and testing was manually implemented to demonstrate a deep understanding of VPN architecture.

## 2. Project Overview

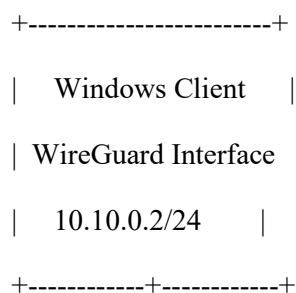
The project implements a secure peer-to-peer VPN between:

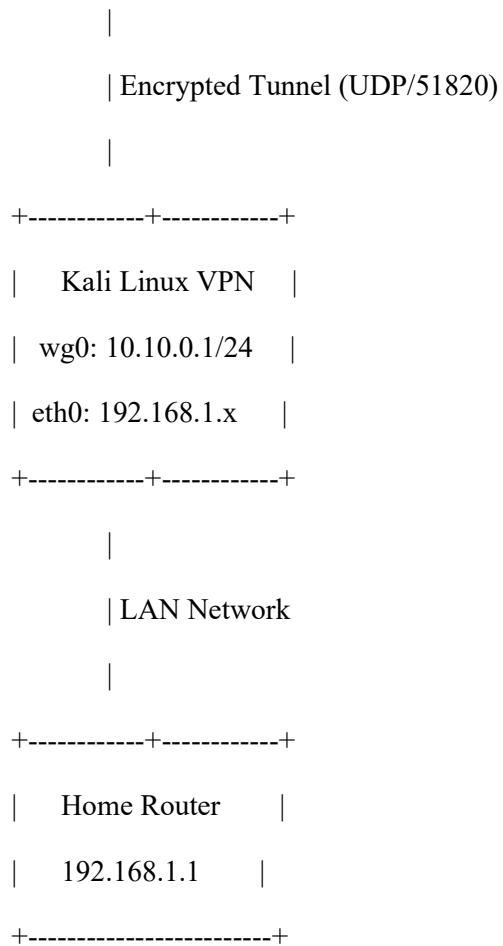
- **Server:** Kali Linux (running in VirtualBox)
- **Client:** Windows host PC
- **WireGuard:** Used for secure encrypted tunnels
- **VirtualBox networking:** Configured using Bridged mode for direct LAN access
- **Routing & Firewall:** Configured with Linux iptables and IP forwarding

This setup enables the Windows system to securely communicate with the Kali server via an encrypted WireGuard tunnel.

## 3. Network Topology

### 3.1 Logical Diagram





### 3.2 VirtualBox Networking

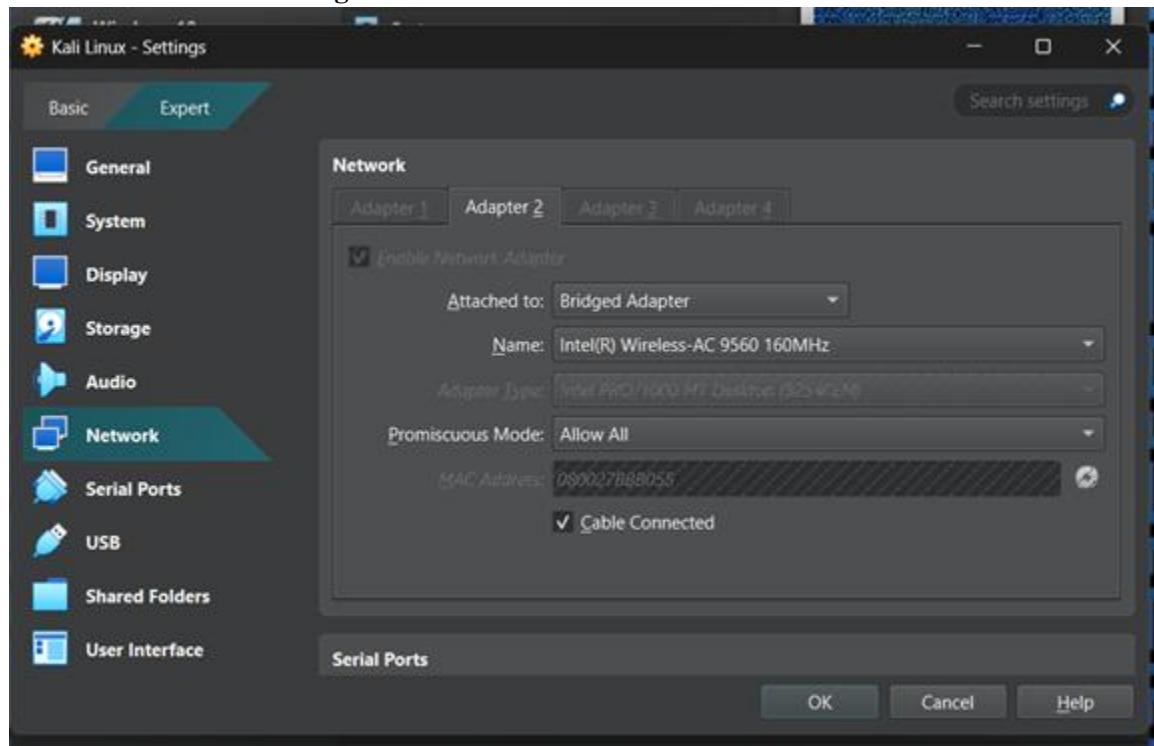


Figure 1 - VirtualBox Adapter 1 (Bridged Mode) configuration for Kali server

Kali's network adapter was set to:

Adapter 1 (eth0): Bridged Mode

Adapter 2: Disabled

## 4. Server Configuration (Kali Linux)

### 4.1 Install WireGuard

```
sudo apt update
```

```
sudo apt install wireguard -y
```

### 4.2 Generate Keys

```
wg genkey | tee server_private.key | wg pubkey > server_public.key
```

### 4.3 Server Configuration File (wg0.conf)

A screenshot of a terminal window titled "Session Actions View Help". The window shows the output of the command "ifconfig" or "ip link show". The output lists several network interfaces:

- lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
- inet 127.0.0.1/8 scope host lo  
 valid\_lft forever preferred\_lft forever
- inet6 ::1/128 scope host noprefixroute  
 valid\_lft forever preferred\_lft forever
- eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc fq\_codel state UP group default qlen 1000
- link/ether 08:00:27:bb:b0:55 brd ff:ff:ff:ff:ff:ff
- inet 192.168.1.37/24 brd 192.168.1.255 scope global dynamic noprefixroute
- eth0  
 valid\_lft 86194sec preferred\_lft 86194sec
- inet6 fe80::f18a:e59c:fc66:93bd/64 scope link noprefixroute  
 valid\_lft forever preferred\_lft forever
- wg0: <POINTOPOINT,NOARP,UP,LOWER\_UP> mtu 1420 qdisc noqueue state UNKNOWN group default qlen 1000
- link/none
- inet 10.10.0.1/24 scope global wg0  
 valid\_lft forever preferred\_lft forever
- docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
- link/ether 02:42:96:d7:00:bc brd ff:ff:ff:ff:ff:ff
- inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0  
 valid\_lft forever preferred\_lft forever

The terminal prompt is "(kali㉿kali)-[~]" and the command entered was "\$".

Figure 2 - Kali network interfaces before/after WireGuard setup

A screenshot of a terminal window titled "(kali㉿kali)-[~]". The command "sudo cat /etc/wireguard/wg0.conf" is run, displaying the configuration file content:

```
[Interface]
Address = 10.10.0.1/24
SaveConfig = true
PostUp = sysctl -w net.ipv4.ip_forward=1; iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
PostDown = iptables -t nat -D POSTROUTING -o eth0 -j MASQUERADE; sysctl -w net.ipv4.ip_forward=0
ListenPort = 51820
PrivateKey = CDT+6NfrjpqlxKWWjY6eaI2phNXaTYYazXzWHoBb4FQ=
```

```
[Peer]
PublicKey = UPzlWrlb3mLtRrwMDEs5nKIGeFoQA90VRpYclan8tAc=
AllowedIPs = 10.10.0.2/32
Endpoint = 192.168.56.1:54533
```

Figure 3 - WireGuard wg0 status on Kali (server side)

Location:

/etc/wireguard/wg0.conf

Content:

[Interface]

Address = 10.10.0.1/24

PrivateKey = <SERVER\_PRIVATE\_KEY>

ListenPort = 51820

[Peer]

PublicKey = <CLIENT\_PUBLIC\_KEY>

AllowedIPs = 10.10.0.2/32

#### 4.4 Enable IP Forwarding

sudo sysctl -w net.ipv4.ip\_forward=1

Make permanent:

net.ipv4.ip\_forward=1

in /etc/sysctl.conf.

#### 4.5 Configure NAT (iptables)

```
(kali㉿kali)-[~]
└─$ sudo iptables -t nat -L -n -v
Chain PREROUTING (policy ACCEPT 356 packets, 74605 bytes)
pkts bytes target    prot opt in     out    source          destination
      3   314 DOCKER    all  --  *      *      0.0.0.0/0        0.0.0.0/0
      ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target    prot opt in     out    source          destination

Chain OUTPUT (policy ACCEPT 8 packets, 2145 bytes)
pkts bytes target    prot opt in     out    source          destination
      0     0 DOCKER    all  --  *      *      0.0.0.0/0        !127.0.0.0/8
      ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 6 packets, 1470 bytes)
pkts bytes target    prot opt in     out    source          destination
      0     0 MASQUERADE all  --  *      !docker0  172.17.0.0/16    0.0.0.0/0
  210 59559 MASQUERADE all  --  *      eth0    0.0.0.0/0        0.0.0.0/0

Chain DOCKER (2 references)
pkts bytes target    prot opt in     out    source          destination
      0     0 RETURN     all  --  docker0 *      0.0.0.0/0        0.0.0.0/0
```

Figure 4 - Server configuration file (wg0.conf)

To allow VPN traffic to access the internet or the LAN:

```
sudo iptables -t nat -A POSTROUTING -s 10.10.0.0/24 -o eth0 -j MASQUERADE
```

Allow forwarding:

```
sudo iptables -A FORWARD -i wg0 -o eth0 -j ACCEPT
```

```
sudo iptables -A FORWARD -i eth0 -o wg0 -m state --state RELATED,ESTABLISHED -j ACCEPT
```

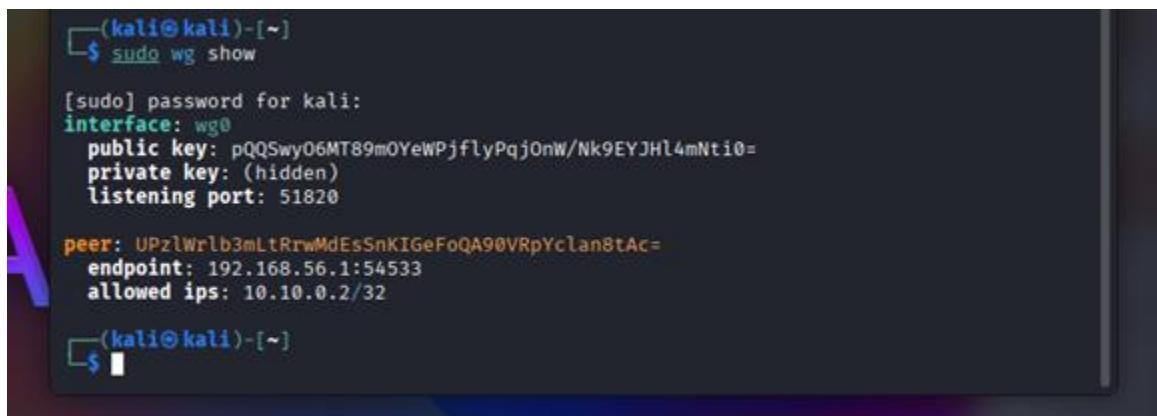
## 4.6 Start WireGuard

```
sudo systemctl start wg-quick@wg0
```

```
sudo systemctl enable wg-quick@wg0
```

## 4.7 Verify

```
sudo wg show
```



```
(kali㉿kali)-[~]
└─$ sudo wg show

[sudo] password for kali:
interface: wg0
  public key: pQQSwy06MT89m0YeWPjflyPqjOnW/Nk9EYJHL4mNti0=
  private key: (hidden)
  listening port: 51820

peer: UPzlWrlb3mLtRrwMdEsSnKIGeFoQA90VRpYclan8tAc=
  endpoint: 192.168.56.1:54533
  allowed ips: 10.10.0.2/32

(kali㉿kali)-[~]
└─$
```

Figure 5 - Linux iptables NAT and forwarding rules for VPN routing

## 5. Client Configuration (Windows)

### 5.1 Generate Keys

Using WireGuard for Windows → “Add Tunnel → Add Empty Tunnel”.

### 5.2 Client Configuration

The screenshot shows a Windows application window titled "client1.conf". The menu bar includes "File", "Edit", and "View". The main content area displays a configuration file with two sections: [Interface] and [Peer].

```
[Interface]
PrivateKey = oMvKc+luUPCE05A9HAl9kn8USV6iKgbf3j8bVZ+fnmo=
Address = 10.10.0.2/24
DNS = 1.1.1.1

[Peer]
PublicKey = pQQSwyO6MT89mOYeWPjflyPqjOnW/Nk9EYJHl4mNti0=
Endpoint = 192.168.1.37:51820
AllowedIPs = 0.0.0.0/0, ::/0
PersistentKeepalive = 25
```

Figure 6 — Windows WireGuard Client (client1) active connection status

[Interface]

PrivateKey = <CLIENT\_PRIVATE\_KEY>

Address = 10.10.0.2/24

DNS = 1.1.1.1

[Peer]

PublicKey = <SERVER\_PUBLIC\_KEY>

Endpoint = <KALI\_IP>:51820

AllowedIPs = 10.10.0.0/24

PersistentKeepalive = 25

### 5.3 Activation

Click **Activate** → tunnel established.

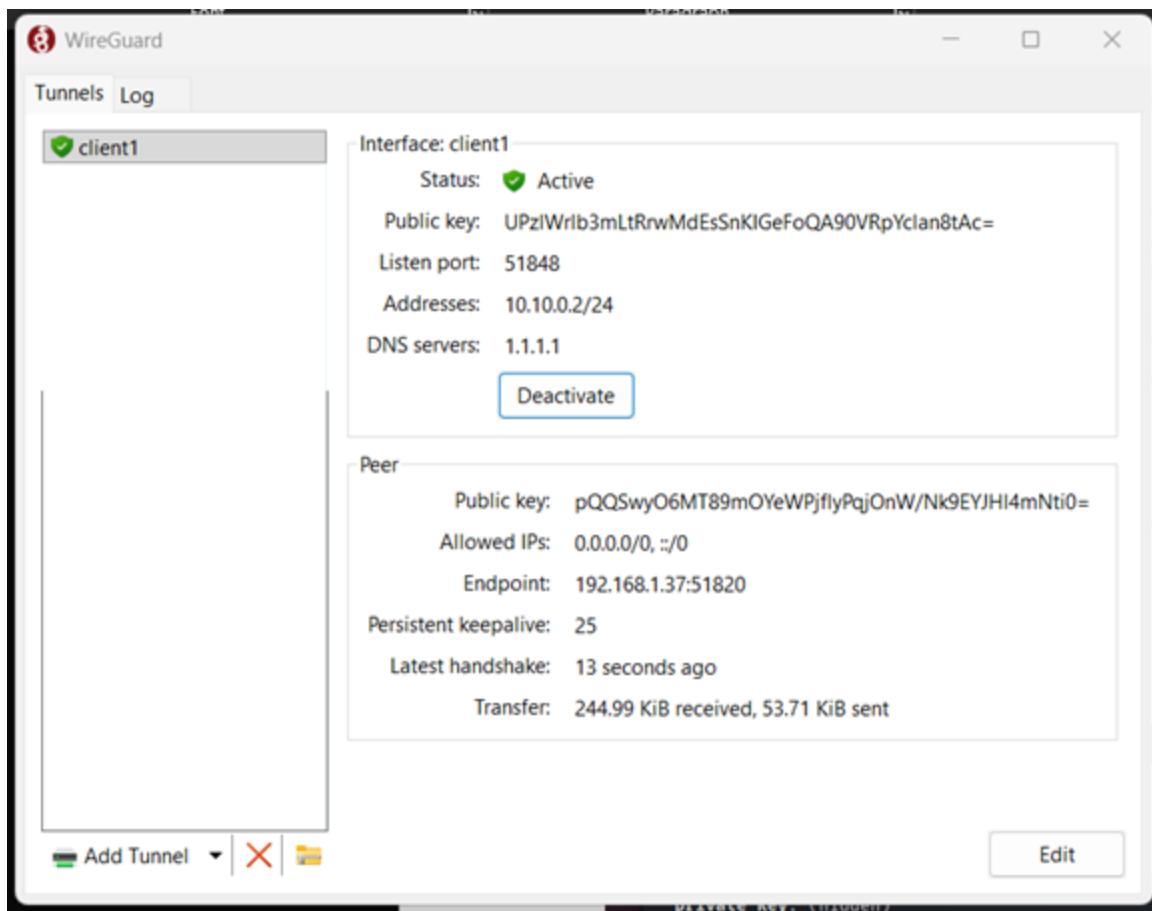


Figure 7 - Windows client configuration (client1.conf)

## 6. Connectivity Testing

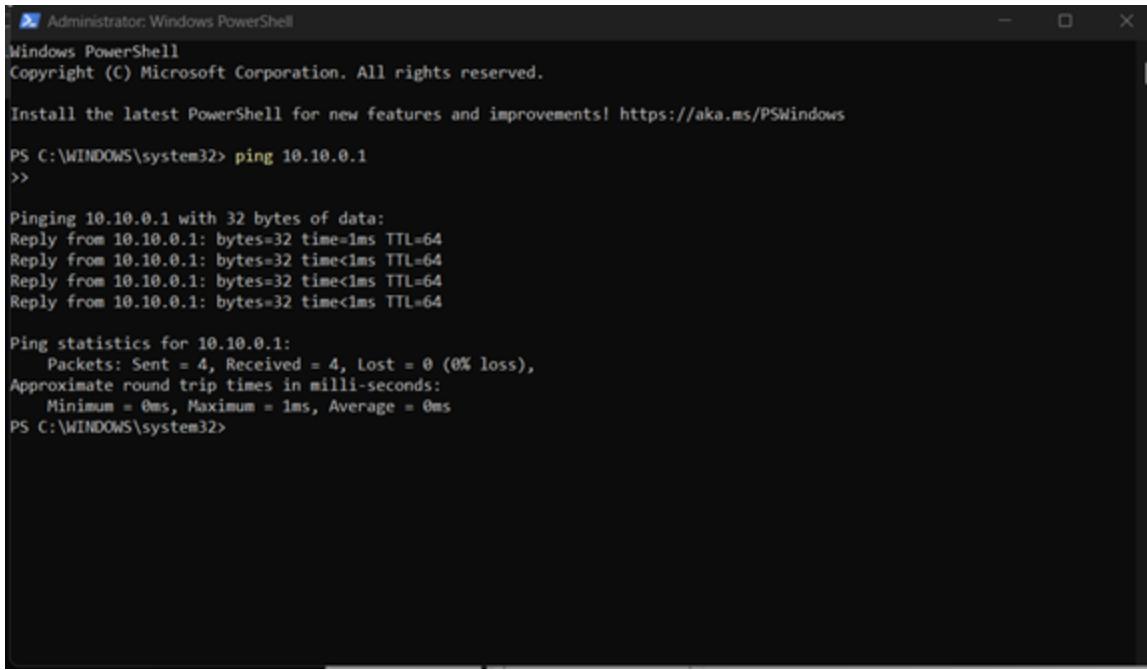
### 6.1 Ping Test

From Windows:

```
ping 10.10.0.1
```

Result:

```
Reply from 10.10.0.1: bytes=32 time<1ms TTL=64
```



```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\WINDOWS\system32> ping 10.10.0.1
>>

Pinging 10.10.0.1 with 32 bytes of data:
Reply from 10.10.0.1: bytes=32 time=1ms TTL=64
Reply from 10.10.0.1: bytes=32 time<1ms TTL=64
Reply from 10.10.0.1: bytes=32 time<1ms TTL=64
Reply from 10.10.0.1: bytes=32 time<1ms TTL=64

Ping statistics for 10.10.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms
PS C:\WINDOWS\system32>
```

Figure 8 - Successful ICMP ping from Windows client to VPN server (10.10.0.1)

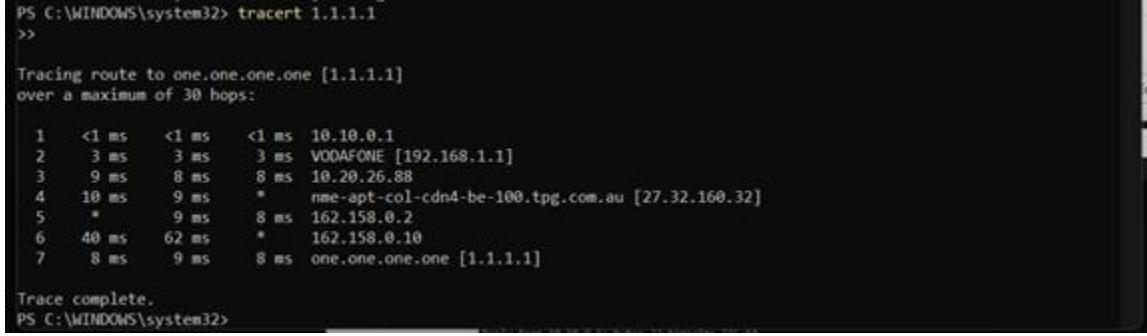
## 6.2 Route Trace

tracert 1.1.1.1

Output (tunnel is being used):

```
1 <1ms 10.10.0.1 (Kali VPN)
2 5ms 192.168.1.1 (Home Router)
```

...



```
PS C:\WINDOWS\system32> tracert 1.1.1.1
>>

Tracing route to one.one.one.one [1.1.1.1]
over a maximum of 30 hops:

 1  <1 ms   <1 ms   <1 ms  10.10.0.1
 2  3 ms    3 ms    3 ms  VODAFONE [192.168.1.1]
 3  9 ms    8 ms    8 ms  10.20.26.88
 4  10 ms   9 ms   *     rme-apt-col-cdn4-be-100.tpg.com.au [27.32.160.32]
 5  *       9 ms    8 ms  162.158.0.2
 6  40 ms   62 ms   *     162.158.0.10
 7  8 ms    9 ms    8 ms  one.one.one.one [1.1.1.1]

Trace complete.
PS C:\WINDOWS\system32>
```

Figure 9 - Traceroute showing traffic entering VPN via 10.10.0.1

## 6.3 Public IP Check

```
PS C:\WINDOWS\system32> (Invoke-RestMethod -Uri "https://ifconfig.co/ip").Trim()
>>
60.241.37.137
PS C:\WINDOWS\system32>
```

Figure 10 - Confirming traffic routing through Kali (public IP unchanged)

(Invoke-RestMethod -Uri "https://ifconfig.co/ip").Trim()

Result:

60.241.37.137 (normal ISP IP, no hiding – routing only)

This confirms:

- Traffic routed through Kali
- VPN working
- Public IP unchanged (as expected, since not set for full internet proxy)

## 7. Project Repository Structure

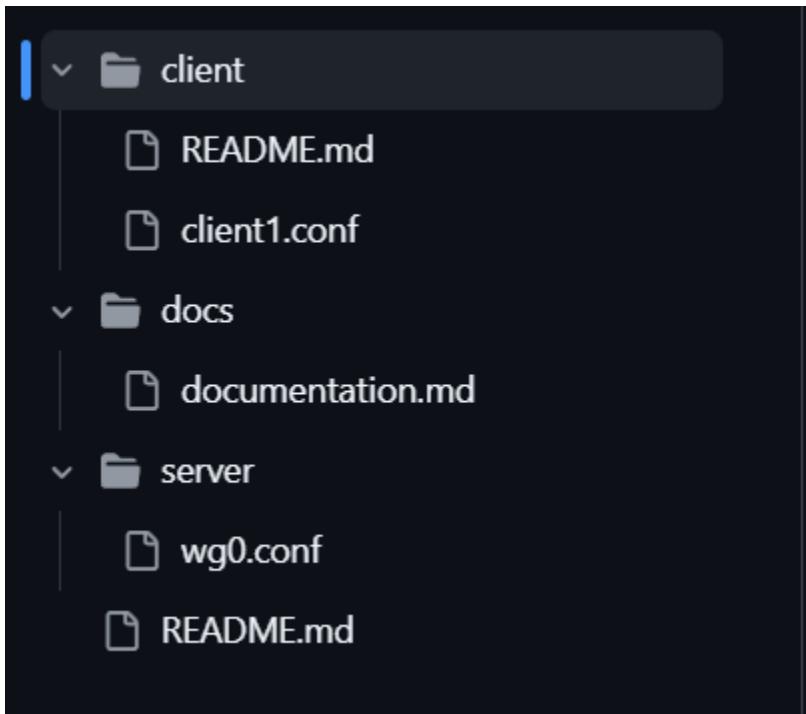


Figure 11 - GitHub repository containing server, client, and documentation folders

Wireguard-VPN/

|

```
|── server/
|   └── wg0.conf
|
|── client/
|   └── client1.conf
|
|── docs/
|   ├── documentation.md
|   └── diagrams (optional)
|
└── README.md
```

## 8. Future Enhancements

Potential improvements:

### 8.1 Full Internet Proxying

Tunnel *all* Windows traffic through Kali.

### 8.2 DNS-level Ad Blocking

Integrate:

- Pi-hole
- Unbound

### 8.3 Multi-client Support

Add multiple peers:

AllowedIPs = 10.10.0.3/32

### 8.4 Hardening

- Disable IPv6
- Add ufw rules

- Restrict AllowedIPs for micro-tunneling

## 8.5 Monitoring

Use:

- Grafana
- Prometheus
- WireGuard exporter

## 9. Conclusion

This project successfully implemented a fully functional WireGuard VPN using Kali Linux as the server and Windows as the client. The setup demonstrates low-level control over tunneling, key management, routing, NAT, and firewall configuration. Through manual implementation, the project provided deep insight into VPN architecture, encryption, and network communication flows.

The foundation built here enables future upgrades such as full-tunnel mode, multi-client support, security hardening, and integration with additional network monitoring tools.