National College of Ireland

Configuration Manual

M.ScResearch Project
M.Sc. in CyberSecurity

Souvick Pradhan
Student ID: 23329882

School of Computing
National College of Ireland

Supervisor: Eugene McLaughlin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Souvick Pradhan
Student ID:	23329882
Programme:	M.Sc. in Cyber Security **Year:** 2026
Module:	M.Sc. Research Project
Lecturer:	Eugene McLaughlin
Submission Due Date:	28.01.2026
Project Title:	A Prescriptive Framework for Securing Third-Party Remote Access to Programmable Logic Controllers (PLCs) on an Automotive Assembly Line Using Four Phased Zero Trust Model

Word Count: 1374 **Page Count:** 14

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other authors' written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Souvick Pradhan

Date: .1.2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Souvick Pradhan
Student ID: 23329882

1 System Overview & Environment Requirements

1.1 Purpose of This Document

This manual explains how the whole system is meant to be set up so the user can get everything running without guessing too much. The setup includes Controller, MongoDB, Relay, Socket Server, Connector, and Client. All of these pieces talk to each other to form a simple Zero Trust-style remote access system. (Rose, 2020).

1.2 Host System Requirements

The deployment was carried out using lightweight Ubuntu 22.04 virtual machines. These VMs were hosted on a Windows machine with the following basic hardware:

- CPU: **AMD Ryzen 5 4600H (6 cores)**
- RAM: **6 GB allocated to Controller VM**, and **2 GB** for each of the smaller services
- Disk: **50 GB** for the Controller VM
- OS: **Ubuntu 22.04 LTS** used across all VMs
- Virtualization: **VirtualBox**

This setup is enough for running MongoDB, the Controller UI, and the three Python services without too much slowdown.

1.3 Virtual Machine Allocation

VM	CPU	RAM	Disk	OS
Controller	2 cores	6 GB	50 GB	Ubuntu 22.04
PLC Machine	2 cores	4 GB	25 GB	Ubuntu 22.04
Relay	1 core	2 GB	10 GB	Ubuntu 22.04
Socket Server	1 core	2 GB	10 GB	Ubuntu 22.04
Connector (OT gateway)	1 core	2 GB	10 GB	Ubuntu 22.04
Client VM	1 core	2 GB	10 GB	Ubuntu 22.04

The Controller VM needs the most resources because it runs Node.js and MongoDB. The remaining machines only handle networking traffic or token verification, so they remain light.

1.4 Required Software & Dependencies

Controller (Node.js)

Packages from package. Json (summarized):

- next, react, react-dom: UI rendering.
- mongoose: talking to MongoDB.
- next-auth: login + JWT/session handling
- socket.io-client: real-time updates
- axios / fetch: API communication
- tailwindcss: UI styling
- typescript + @types: safer typing

Final > Controller > {} package.json > {} dependencies

```
{
  "name": "controller",
  "version": "0.1.0",
  "private": true,
  ▷Debug
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "lint": "next lint"
  },
  "dependencies": {
    "jsonwebtoken": "^9.0.2",
    "mongoose": "^8.0.3",
    "next": "14.0.3",
    "next-auth": "^4.24.5",
    "react": "^18",
    "react-dom": "^18"
  },
  "devDependencies": {
    "@types/jsonwebtoken": "^9.0.6",
    "@types/node": "^20",
    "@types/react": "^18",
    "@types/react-dom": "^18",
    "autoprefixer": "^10.0.1",
    "eslint": "^8",
    "eslint-config-next": "14.0.3",
    "eslint-config-prettier": "^9.0.0",
    "eslint-plugin-prettier": "^5.0.1",
    "eslint-plugin-tailwindcss": "^3.13.0",
    "eslint-plugin-unused-imports": "^3.0.0",
    "postcss": "^8",
    "prettier": "^3.1.0",
    "tailwindcss": "^3.3.0",
    "typescript": "^5"
  }
}
```

Python Runtime (Relay, Socket Server, Client, Connector) Main

Python libraries:

- pymongo: MongoDB queries
- python-socketio: real-time communication

- python-dotenv: reading .env files (when used)
- cryptography / pyjwt: decoding and verifying JWT tokens.
- requests: talking to the Controller.
- built-in socket, asyncio: for UDP tunnelling and async loops.

1.5 Ports Used in the System

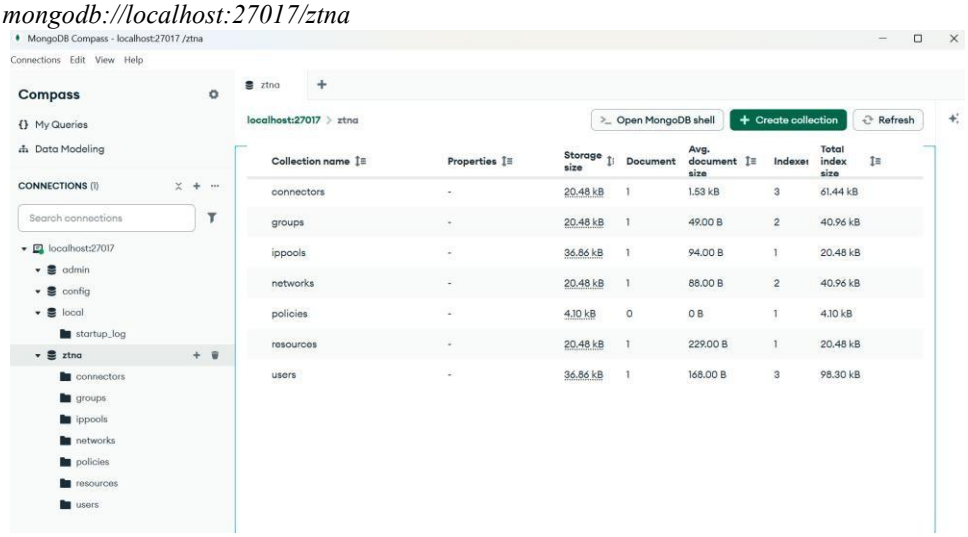
Component	Port	Protocol
Controller	3000	HTTP
MongoDB	27017	TCP
Socket Server	5001	WebSocket/HTTP
Relay	9090	UDP
Connector	(dynamic)	UDP
Client	(dynamic)	UDP

2 Configuration Steps

2.1 MongoDB Setup

MongoDB is installed on the Controller VM. No initial setup is required because the Controller will automatically create collections on first run. (MongoDB, 2024) Connection string:

mongodb://localhost:27017/ztna



2.2 Controller Configuration (Next.js)

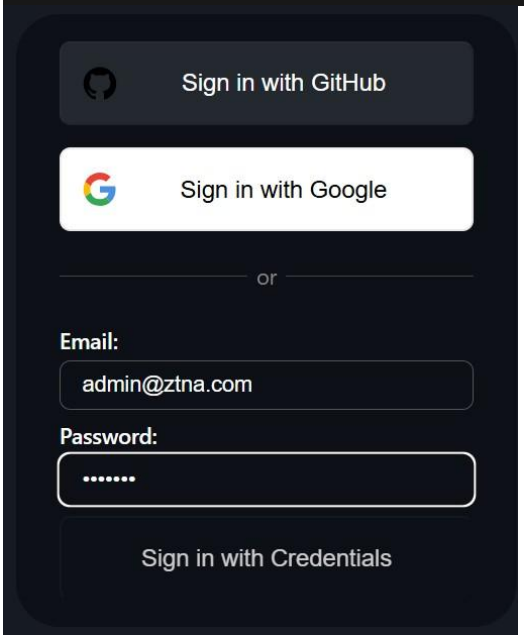
To start with the Controller UI, the following steps are carried out:

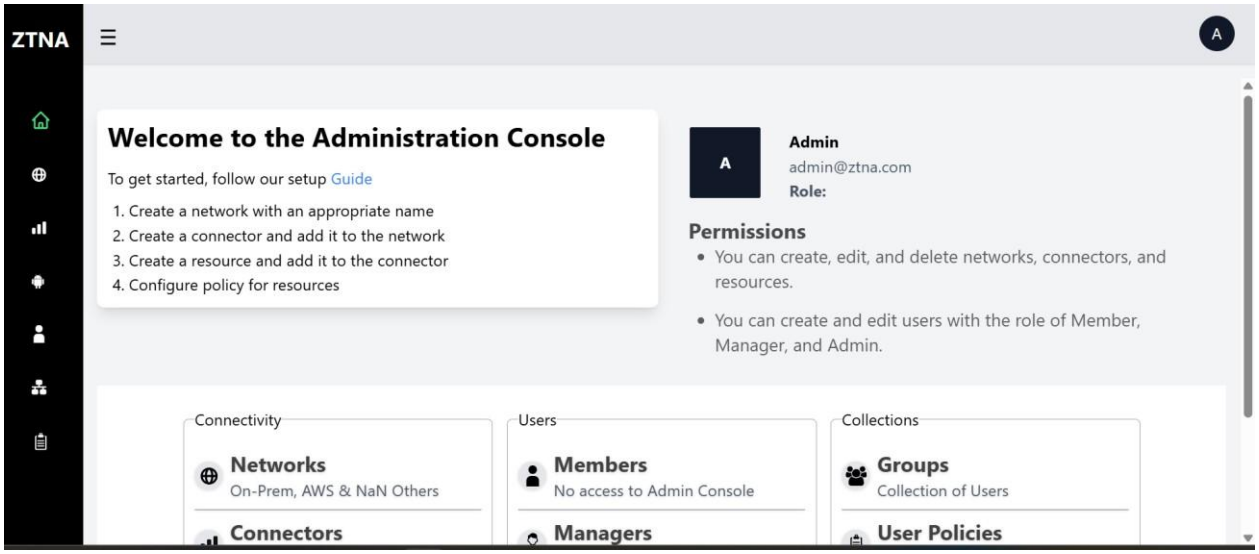
1. Install Node.js dependencies.
2. npm install.
3. Create the. env.local file with keys, secrets and URLs.
4. Start the Controller using:
5. npm run dev.
6. Open the web browser and visit:
7. http://localhost:3000

Once everything loads, the login screen should appear. (Vercel, 2024)

```
PS E:\Final\Final\Controller> npm run dev

> controller@0.1.0 dev
> next dev

   ▲ Next.js 14.0.3
   - Local:        http://localhost:3000
   - Environments: .env.local

Attention: Next.js now collects completely anonymous telemetry regarding usage.
This information is used to shape Next.js' roadmap and prioritize features.
You can learn more, including how to opt-out if you'd not like to participate in this anonymous program, by visiting
the following URL:
https://nextjs.org/telemetry

 ✓ Ready in 3.5s
```

```
 ✓ Ready in 2.3s
 ✓ Compiled /src/middleware in 259ms (163 modules)
 ○ Compiling /api/auth/[...nextauth] ...
 ✓ Compiled /networks in 1835ms (262 modules)
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
 ✓ Compiled in 2.5s (298 modules)
 ○ Compiling /api/network ...
 ✓ Compiled /api/network in 2.4s (637 modules)
 ✓ Compiled (640 modules)
MongoDB Connected: undefined
MongoDB Connected: 127.0.0.1
MongoDB Connected: 127.0.0.1
MongoDB Connected: 127.0.0.1
MongoDB Connected: 127.0.0.1
MongoDB Connected: 127.0.0.1
MongoDB Connected: 127.0.0.1
```



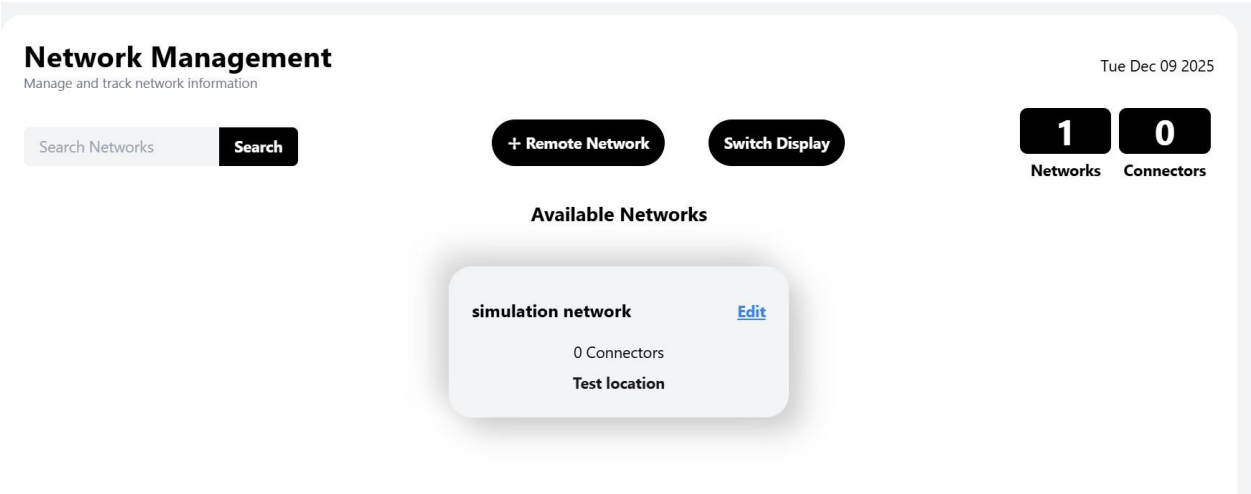


2.3 Creating Networks, Connectors, Resources, Users, and Groups

These steps were performed through the Controller UI, which was connected to MongoDB (verified through both testing docs and SS folder images:).

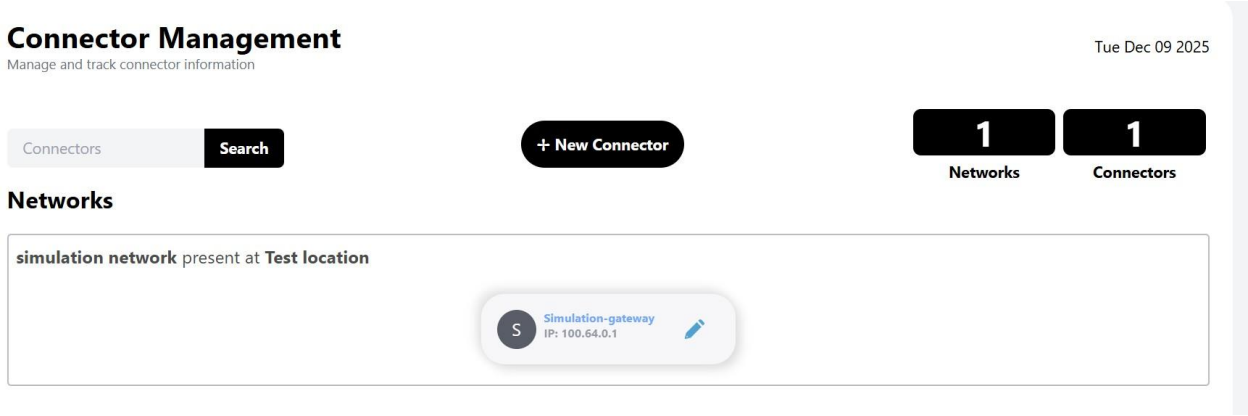
2.3.1 Network Setup

- Open **Networks** then go to **Remote Network**
- Create the network (e.g., *simulation network*)
- Confirm that it appears in the list.



2.3.2 Connector Setup

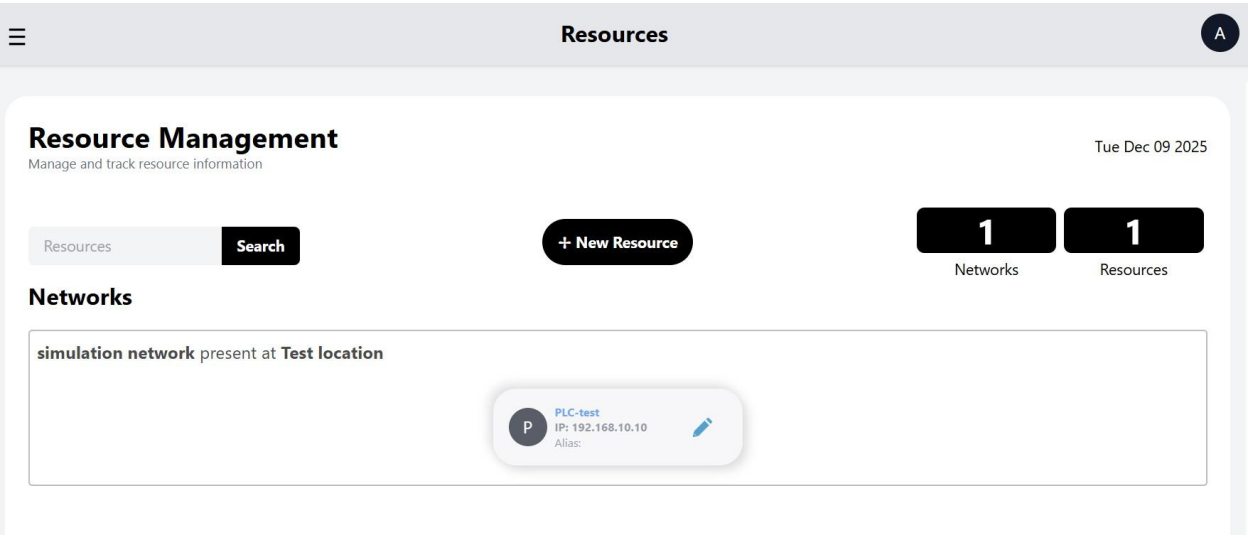
- Go to **Connectors**
- Add a new connector (*Simulation-gateway*)
- Status should be set to **Enabled.**



2.3.3 Resource Setup

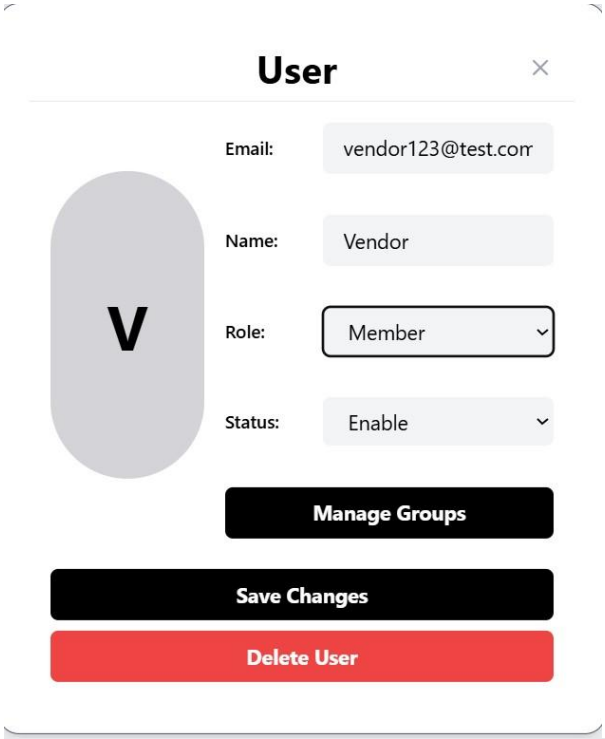
- Go to **Resources**
- Add a PLC or test IP, for example:

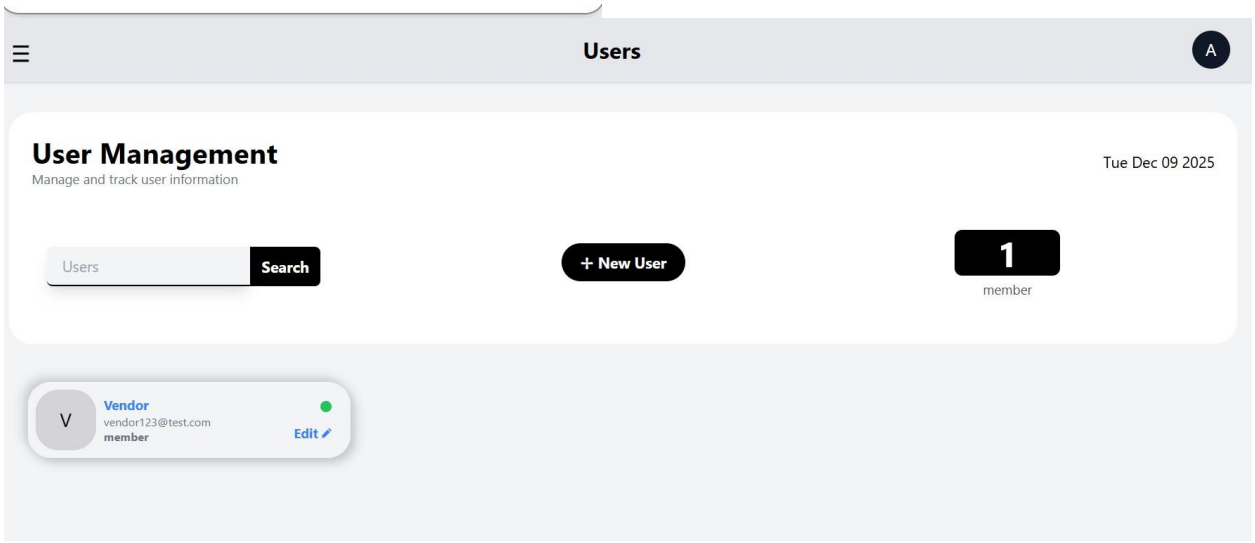
- Name: PLC-1-Test
- IP: 192.168.10.10
- Network: *simulation network*



2.3.4 User Setup

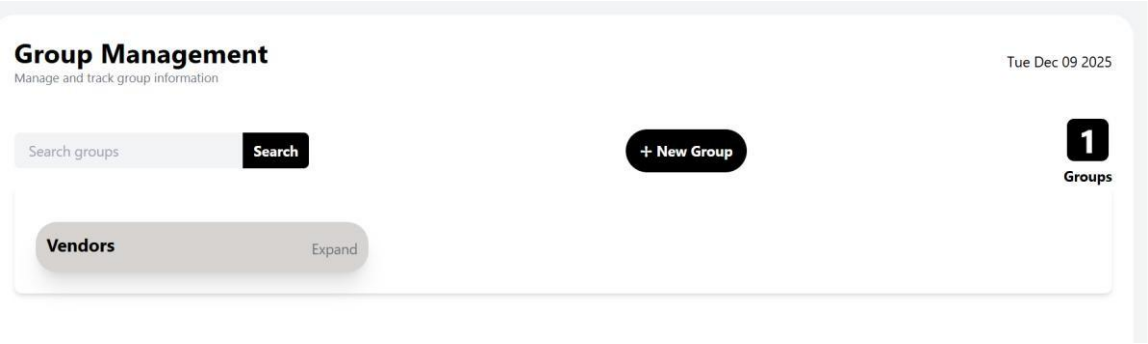
- Create a vendor user (e.g. vendor@test.com)
- The default admin user remains admin@ztna.com





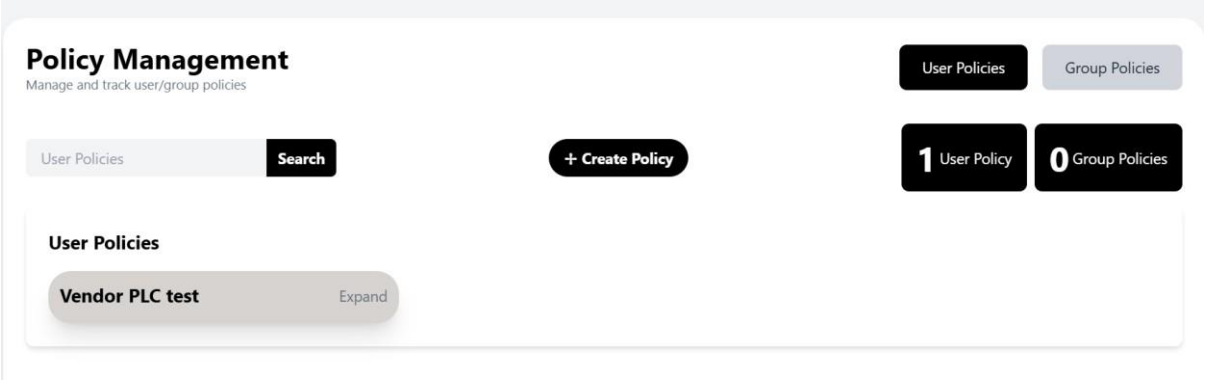
2.3.5 Groups (Optional)

- The Controller UI shows groups.
- A group may be associated with the user if needed.



2.3.6 Creating Policies

- Go to **Policies**
- Create a policy linking **Vendor to PLC-1-Test**
- This determines which resource the client can access.



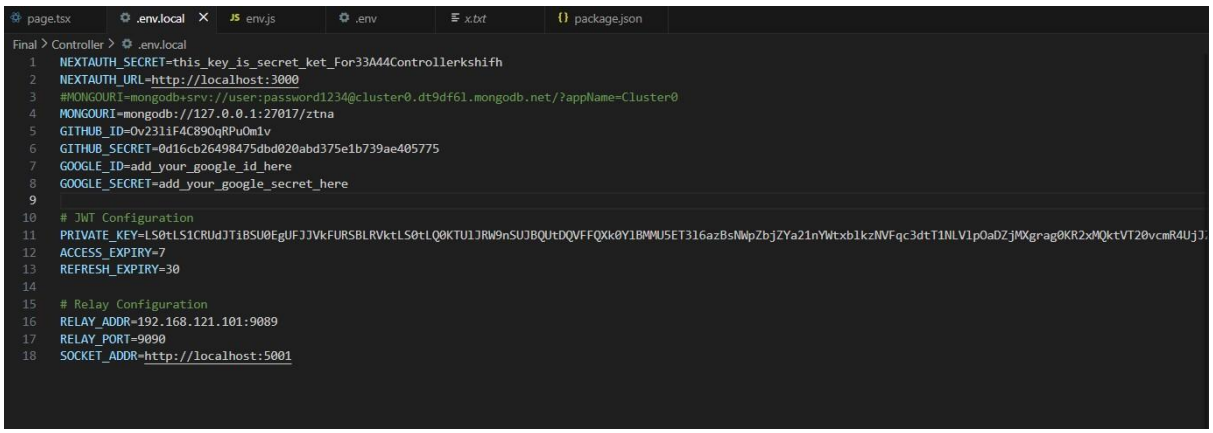
3 Environment Configuration

3.1 Controller (.env.local)

Purpose:

- Stores all secrets used by Next AUTH
- Stores the MongoDB URI
- Contains the Controller's private key (JWT signing key) (Jones, 2015)
- Holds Relay and Socket Server addresses.
- After filling in the .env.local, the Controller can be launched normally with:

  npm run dev



3.2 Relay

Purpose:

- Holds the Controller public key.
- Stores the Controller URL
- Allows the Relay port to be set from an .env file if needed.

How to Run Relay with .env:

- Place .env inside relay/app/
- At the top of server.py, add:

from dotenv import load_dotenv load_dotenv()

- Replace hard-coded values in the Relay code with:

os.getenv("PUBLIC_KEY") os.getenv("CONTROLLER_URL")
int(os.getenv("RELAY_PORT"))

- Start the Relay:

python server.py

Now the Relay will load its values from the .env file at startup.

4 Connector Deployment (Dockerised)

The connector is provided as a Dockerised service. Users do not need to run Python files or install dependencies manually. Instead, the connector container is started directly using Docker. (Docker, 2024)

Before running the command, the Controller UI must be used to generate a Connector Access Token and Refresh Token. Once those are available, the connector can be deployed by running:

Here is the command the user has to run:

```
docker run -d \
  --name ztna-connector \
  -e ACCESS_TOKEN=<paste-your-connector-access-token-here> \
  -e REFRESH_TOKEN=<paste-your-connector-refresh-token-here> \
  -e RELAY_IP=<relay-ip-address> \
  -e RELAY_PORT=9090 \
  -e SOCKET_SERVER_ADDR=http://<socket-server-ip>:5001 \   connector-image: latest
```

After running this command, the container will start in the background. You can check if it's running by typing: docker ps

5 Client Configuration (client.py & GetToken.py)

The Client service uses two Python files for authentication and tunnel setup.

5.1 Token Retrieval (GetToken.py)

GetToken.py opens a tiny local HTTP server to receive authentication tokens from the

Controller using PKCE. (Sakimura, 2015) Key

behaviors:

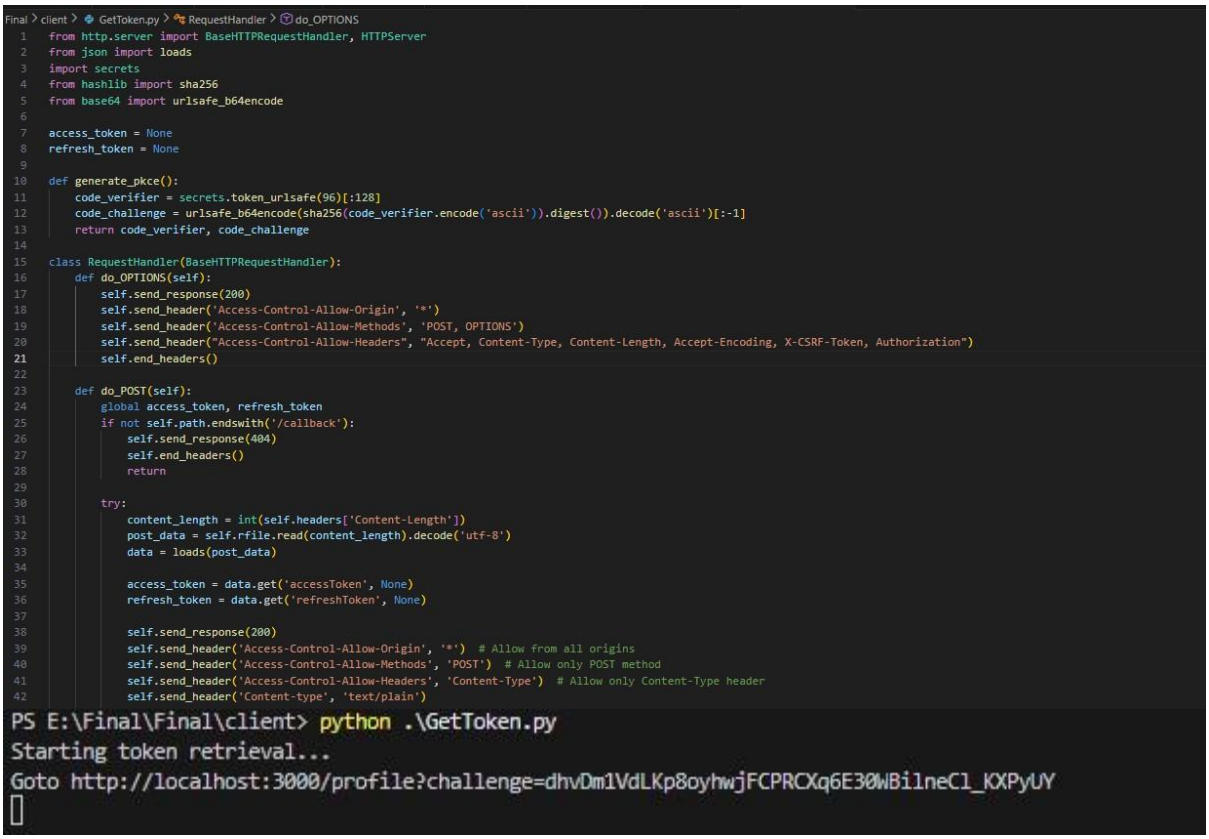
- Listens on **port 8080** for /callback
- Generates PKCE values (code_challenge, code_verifier)
- Prints a login URL such as:
- http://localhost:3000/profile?challenge=<value>
- After the user opens this URL and logs in, the script receives:

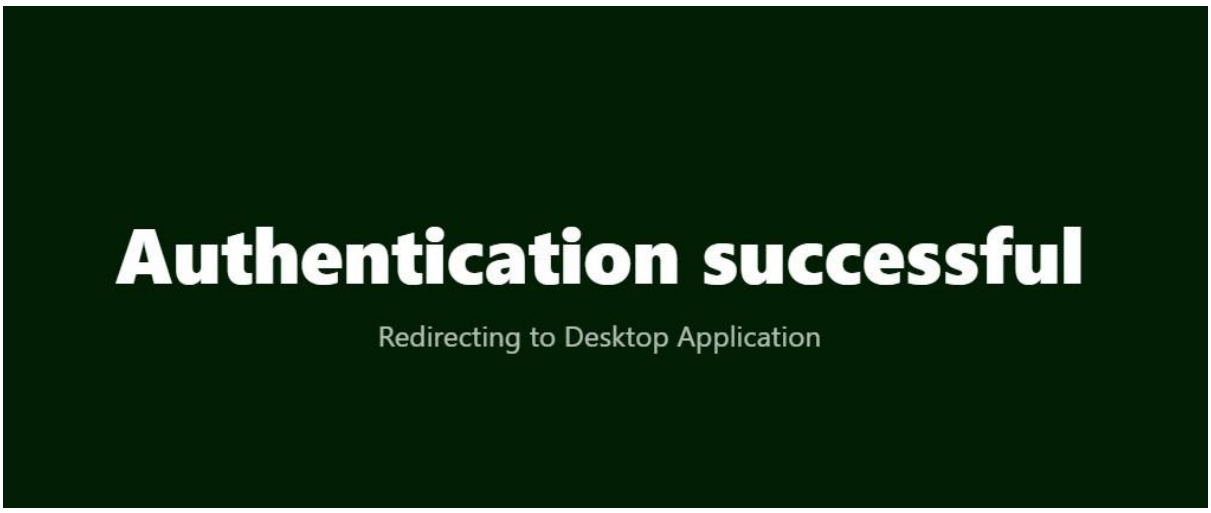
  - Access Token
  - Refresh Token
  - Code Verifier

The script prints them in the terminal for client use (Hardt, 2012).

User Flow:

1. Ensure Controller is running.
2. Run: python GetToken.py
3. Open the printed URL in the browser.
4. Tokens appear in the terminal.





5.2 Client Agent (client.py)

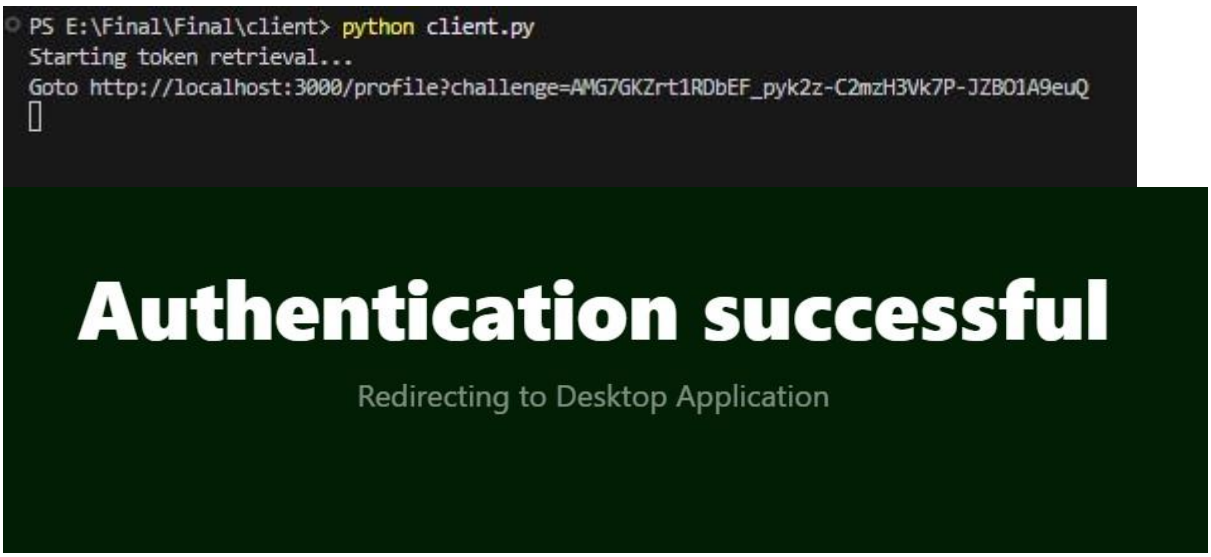
client.py is the component that builds the secure tunnel, creates the TUN interface and forwards packets.

Important internal settings:

- At the bottom, it calls:
- access_token, refresh_token, code_verifier = get_token()
- It decodes the JWT to get the client ID • Key network settings appear near the top:
- relay_address = ('127.0.0.1', 9090)
- socket_server_address = 'http://localhost:5001'
- It creates and removes the virtual interface (tun0)
- It adds and deletes routes to PLC resources.
- It performs a handshake with the Relay using the Relay's public key.
- Finally, it forwards encrypted UDP packets.

The main configuration points the user needs to adjust are:

1. relay_address: the IP and port of the Relay
2. socket_server_address :WebSocket control-plane server
3. The fact that token collection is done automatically by calling get_token()



Final > client > client.py > ...

```python
import asyncio
import socketio
import json
from GetToken import get_token
from jwt import JWT
from threading import Thread
from socket import socket, AF_INET, SOCK_DGRAM
from TunTap import create_tun_interface, delete_tun_interface
from Route import add_route, del_route, check_route
from Utils import generate_session_key, encode_message, decrypt_data, decode_message, encrypt_data
from Constants import BUFFER_SIZE
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from hashlib import sha256
from select import select
import os
from scapy.all import *

jwt = JWT()

sio = socketio.AsyncClient()

access_token = None
refresh_token = None
code_verifier = None
pending_pings = []

my_ip = None
my_id = None

tun = None
ifname = None

relay_address = ('192.168.121.101', 9090)
socket_server_address = 'sockerserver.com'

resources = {} # {resource_id: {address: resource_address, protoStatus, portStatus, connectorIp }}
resourcesAddressToId = {} # {resource_address: resource_id}

@sio.event
async def connect():
    global access_token, code_verifier
```

6 Conclusion

This setup guide describes the manner in which every component of the system should be configured to ensure that the entire Zero Trust workflow can operate without issues. MongoDB, Relay, Socket Server, Connector and Client relied on one another, and therefore, the manual was dedicated to demonstrating where the environment files will reside, how the services should be launched, and what the user is supposed to press or execute. The testing images and installation instructions indicated that the Controller was connected correctly with MongoDB, new networks and connectors were visible accordingly, and the client was able to authenticate with the help of PKCE and then be assigned its permitted routes. Although the system is composed of a number of small elements, the elements are all predictable in nature, and once the environment variables and the tokens are placed in their respective locations, the services function as expected. All in all, this should provide a simple method of enabling someone to replicate the setup and test the Zero Trust behaviour in a small laboratory setup.

References

Docker, I. (2024). *Docker Documentation*. Retrieved from https://docs.docker.com/

Hardt, D. (2012). *The OAuth 2.0 Authorization Framework (RFC 6749)*. IETF. Retrieved from https://datatracker.ietf.org/doc/rfc6749/

Jones, M. B. (2015). *JSON Web Token (JWT) (RFC 7519)*. IETF. Retrieved from https://datatracker.ietf.org/doc/rfc7519/

MongoDB, I. (2024). *MongoDB Manual*. Retrieved from https://www.mongodb.com/docs/

Rose, S. B. (2020). *Zero Trust Architecture (NIST Special Publication 800-207)*. National Institute of Standards and Technology. doi:http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf

Sakimura, N. B. (2015). *Proof Key for Code Exchange by OAuth Public Clients (RFC 7636)*. IETF. Retrieved from https://datatracker.ietf.org/doc/rfc7636/

Vercel. (2024). *Next.js Documentation*. Retrieved from https://nextjs.org/docs