

Assignment 1

VCC (CSL7510)

Sireejaa Uppal (M23CSE023)

11th February, 2025



Introduction

This project is part of the VCC-CSL7510: Virtualization and Cloud Computing course and is intended to provide hands-on experience with VirtualBox and virtual machine (VM) setups. The goal of this project is to design and setup many virtual machines, connect them, and deploy a microservice-based application across them.

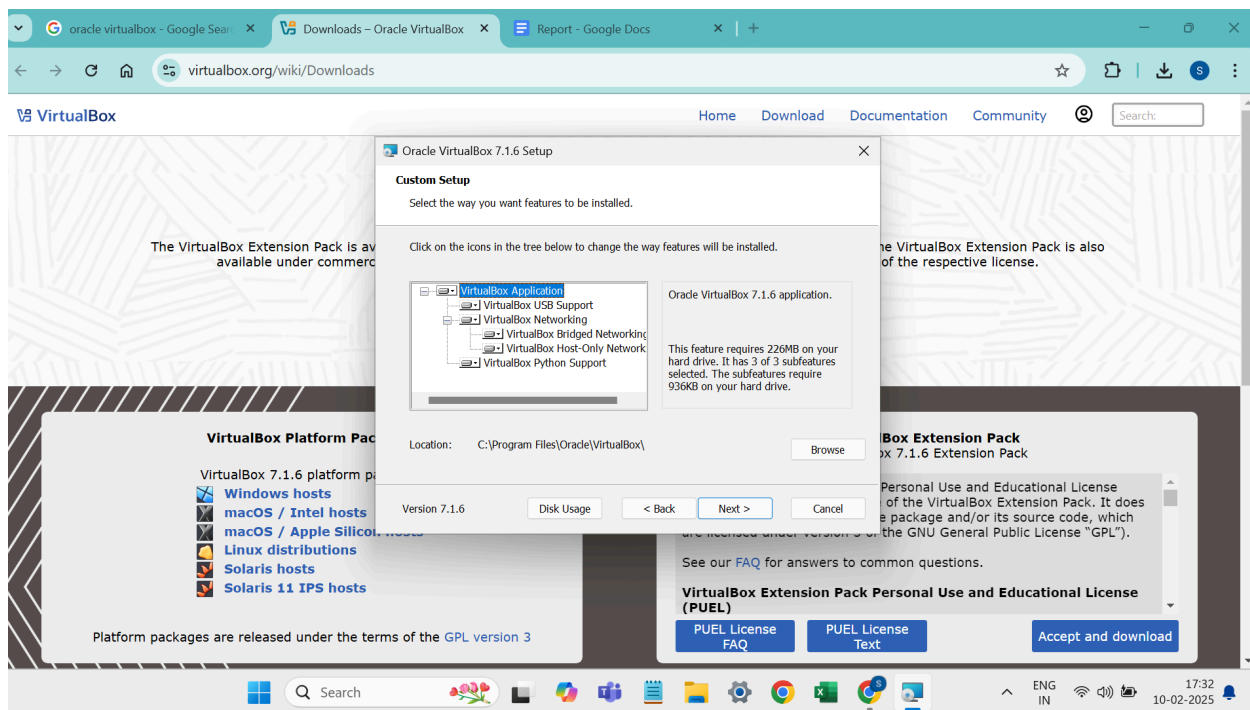
This project's core is a URL Shortener application with a simple interface on VM1 and a backend API on VM2. The frontend, built with HTML, allows users to enter long URLs and generate abbreviated versions, and the backend, built with the Flask framework, handles the logic for URL shortening and retrieval. The project follows a basic microservice architecture, with the frontend communicating with the backend via RESTful API calls.

The goal of this assignment is to not only create a functional web application, but also to investigate virtualization and networking ideas by developing VMs that can communicate with one another in a cloud-like environment. This exercise provides a thorough grasp of deploying apps in a distributed environment.

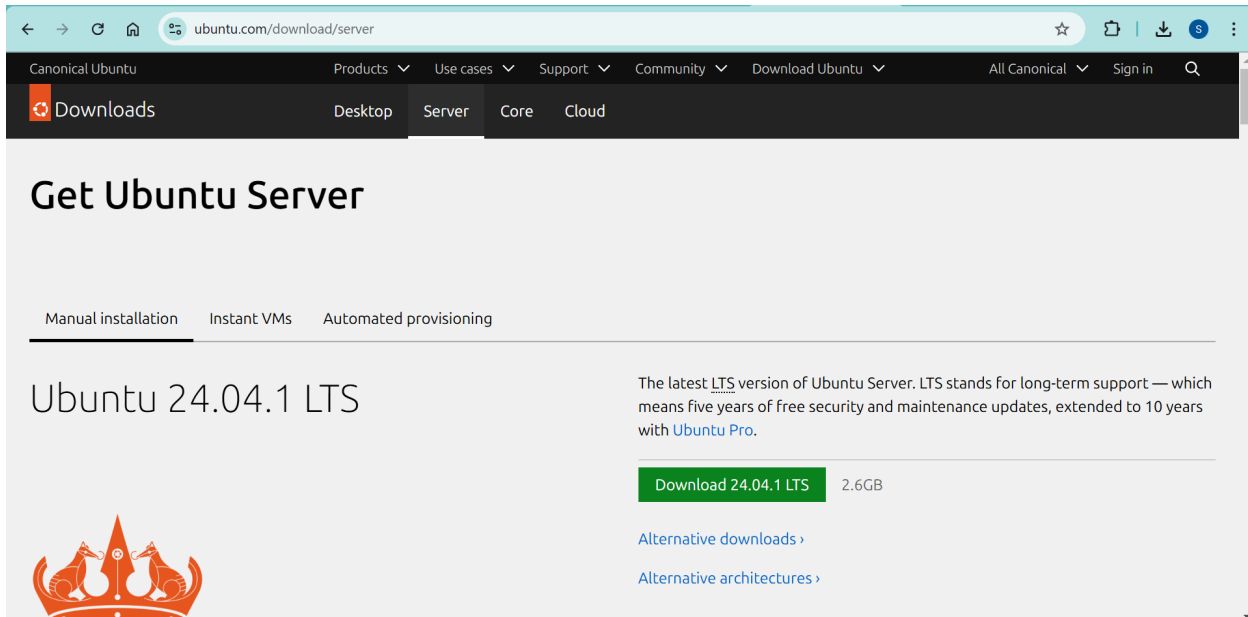
Implementation:

Step 1: Installation of VirtualBox

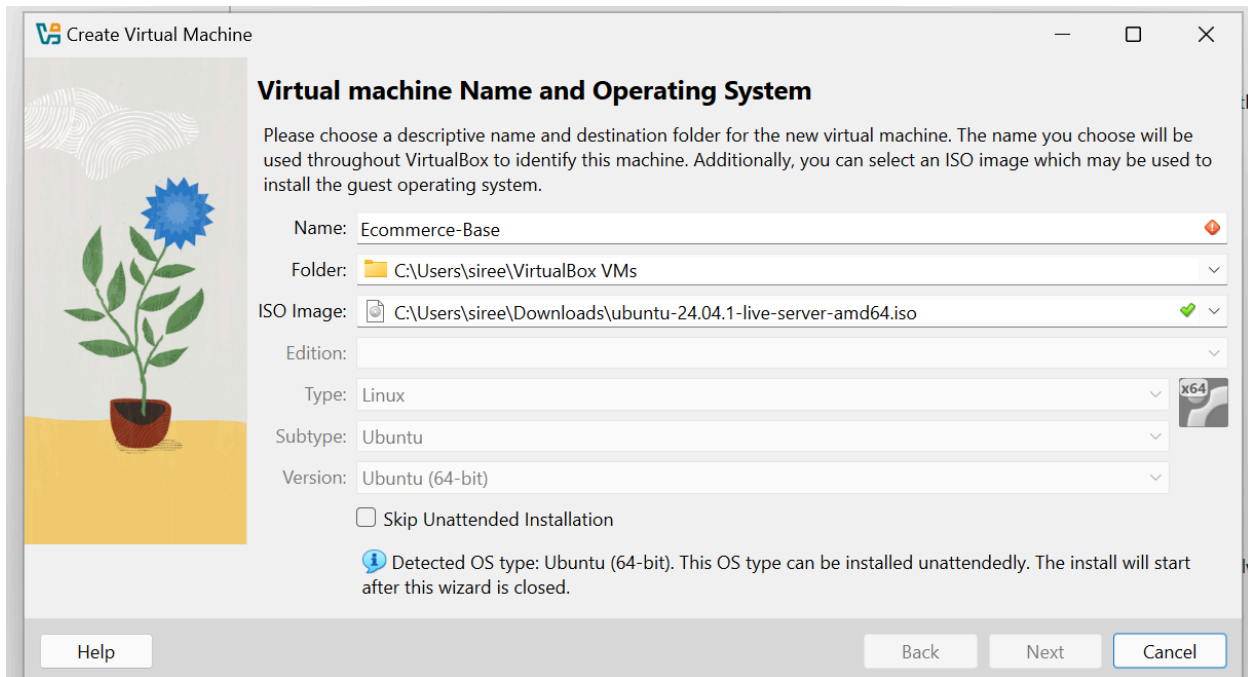
1. Download VirtualBox:
 - Go to the official [VirtualBox website](https://www.virtualbox.org/).
 - Choose Ubuntu Server and Desktop ISO files
 - Download the installer and save it to the system.
2. Install VirtualBox:
 - Run the installer and follow the on-screen instructions to install VirtualBox on the host machine.
3. Verify Installation:
 - Once installed, open VirtualBox and confirm it launches properly.



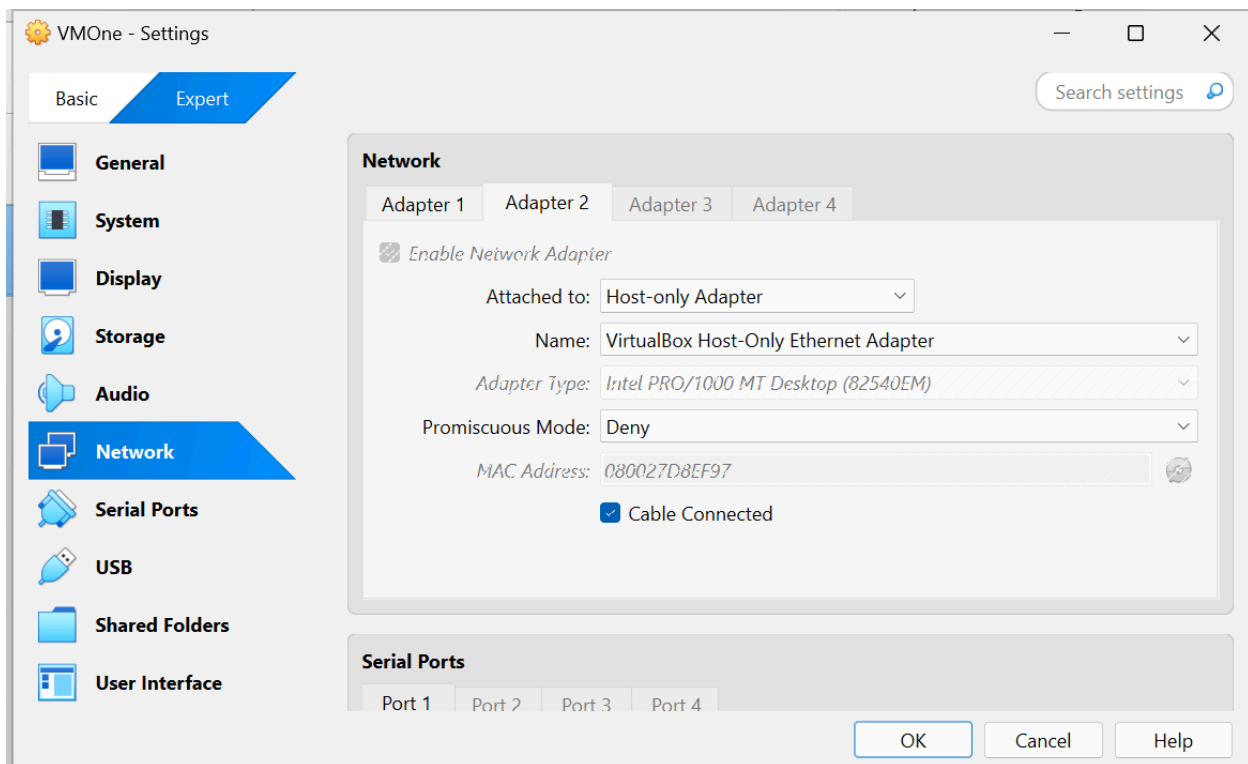
Step 2: Create Virtual Machines (VM1 and VM2)



1. Create VM1 (Frontend):
 - Open VirtualBox and click on New.
 - Name the virtual machine as **VM1**, choose the operating system (e.g., Ubuntu), and proceed.
 - Set the RAM size (2048 MB).
 - Create a new virtual hard disk (20 GB) and choose the storage type (VDI).
 - Start the virtual machine and install the operating system (Ubuntu 24) by selecting the ISO image of Desktop version(5GB ISO file).
2. Create VM2 (Backend):
 - Follow the same steps to create a second virtual machine **VM2** for the backend.
 - Choose the same OS as **VM1** (Ubuntu) and follow the same process for installation except that I choose Ubuntu Server ISO file (2GB).
3. Set Up Operating System:
 - Complete the installation of the operating system (Ubuntu) on both VM1 and VM2.



Step 3: Configure Network Settings



1. Assign Static IP Addresses:
 - In VirtualBox, go to Settings for each VM.

- Navigate to Network and configure the network adapter to Internal Network.
- Set static IP addresses:
 - For VM1, assign **192.168.56.101**.
 - For VM2, assign **192.168.56.102**.
- 2. Check Network Connectivity:
 - Open a terminal on VM1 and check connectivity to VM2:
ping 192.168.56.102
 - Perform the same on VM2 to check connectivity back to VM1:
ping 192.168.56.101
 - Ensure both VMs can ping each other successfully.

Step 4: Install Required Software

- Install Nginx on VM1 (Frontend): Open the terminal on VM1 and run the following commands to install Nginx:
 - **sudo apt update**
 - **sudo apt install nginx**
- Install Python and Flask on VM2 (Backend): On VM2, open the terminal and install
 - **sudo apt update**
 - **sudo apt install python3-pip**
 - **sudo pip3 install flask flask-cors**
- Install Additional Software on Both VMs as and when required:
 - Like CORS, pip and other libraries that were required on the go.

Step 5: Develop the Microservice Application

1. Frontend (index.html on VM1):
 - Create a new file on VM1 called **index.html** in the desired directory.
 - Clone the HTML code provided for the URL Shortener application into **index.html**.
2. Backend (Flask API on VM2):
 - On VM2, create a new file called **app.py**.
 - Clone the Flask backend code into **app.py** (the code provided for URL shortening).

Step 6: Deployment

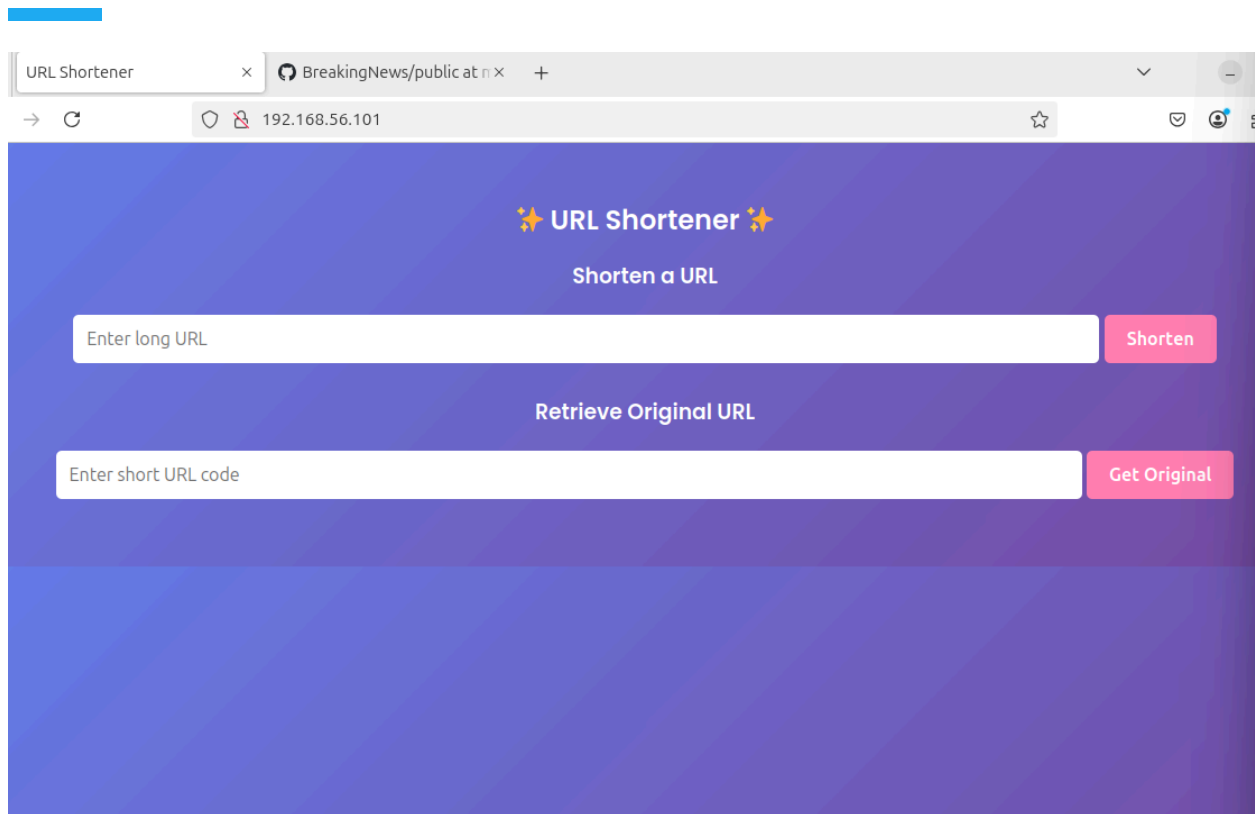
3. Frontend (index.html on VM1):
 - Start nginx server using the command .
 - i. **sudo systemctl -> start nginx**

- ii. Start Nginx `sudo systemctl restart nginx -> Restart Nginx`
 - iii. starts a simple HTTP server using Python on port 80. : `sudo python3 -m http.server 80`
4. Backend (Flask API on VM2):
- On VM2, create a new file called `app.py`.
 - Clone the Flask backend code into `app.py` (the code provided for URL shortening).
 - Run the backend using `python2 app.py` command `python3 app.py`
 - The backend is now be accessible via `http://192.168.56.102:5000`.

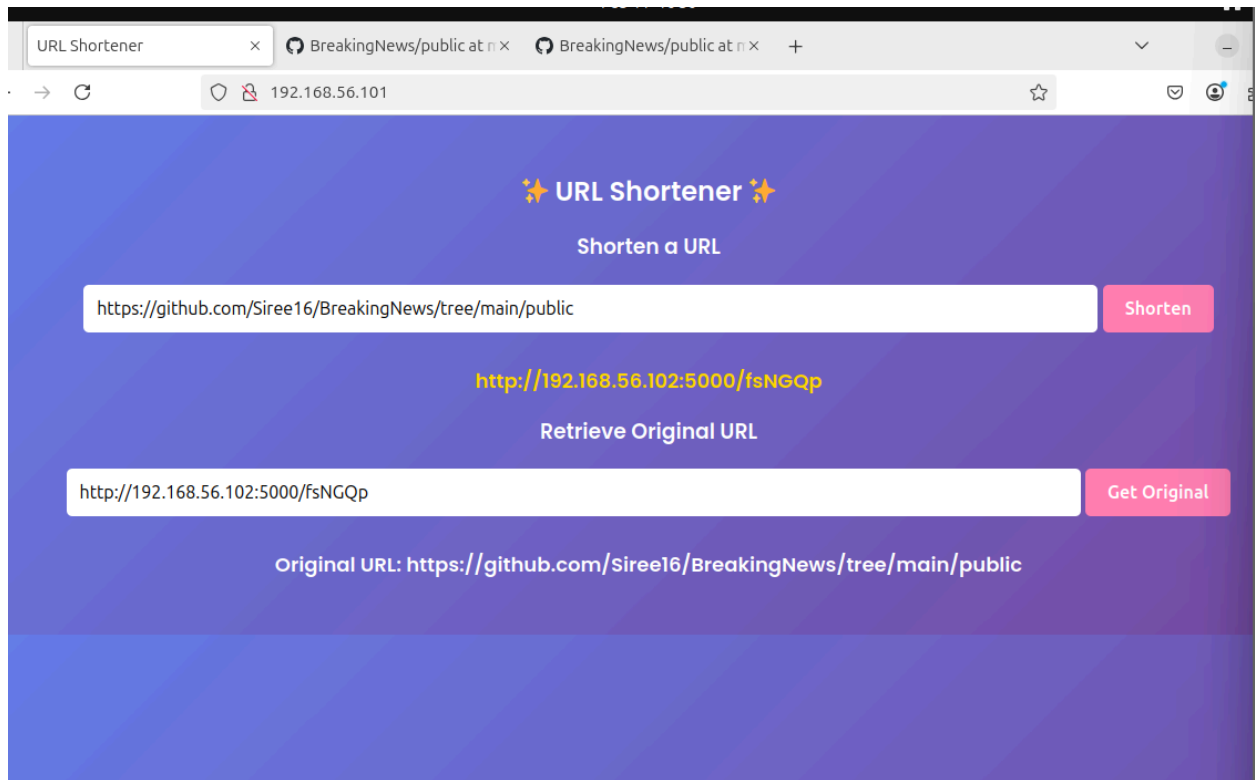
Step 7: Testing the Application

1. Test the Backend API on VM2: In browser on VM1, test the backend by entering the following URL:
`http://192.168.56.102:5000`
 - The Flask application is seen running.
2. Test the Frontend (index.html on VM1):
 - Open `index.html` in the browser on VM1.
 - Start by entering a long URL in the input field and click Shorten.
 - The frontend should makes a **POST** request to the Flask backend (`http://192.168.56.103:5000/shorten`) and receive a shortened URL in response.
 - Similarly, entering a short code to retrieve the original URL using the Get Original feature.
3. Verify Redirection:
 - Test that the URL shortening works as expected by entering the generated short URL in the browser. It redirects to the original URL.

Testing Microservice 1 : URL Shortener



Testing Microservice 2 : Retrieving original URL

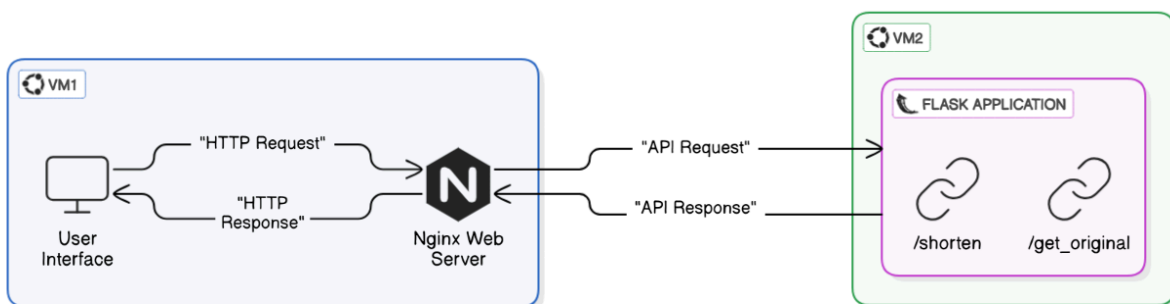


The Architecture

The architecture consists of two virtual machines, VM1 (Frontend) and VM2 (Backend), each serving distinct roles in the microservice-based application. VM1 (running Ubuntu Desktop) hosts Nginx, which serves the frontend `index.html` page and acts as a reverse proxy for API requests. Users interact with the frontend to shorten URLs or retrieve original URLs. VM2 (running Ubuntu Server) hosts a Flask application that handles the backend logic for URL shortening and retrieval. The backend is exposed via two key microservices: the URL Shortening Service (at `/shorten`), which generates a short URL and stores the mapping, and the URL Retrieval Service (at `/get_original`), which retrieves the original URL based on a short code.

Requests flow from the frontend to the backend through Nginx. When a user submits a URL to shorten, the frontend sends a `POST` request to Nginx, which then forwards it to the Flask app on VM2. The Flask app processes the request, generates a short code, and returns the shortened URL, which is sent back through Nginx to the frontend. Similarly, when retrieving a URL, the frontend sends a `POST` request to retrieve the original URL based on the short code, and Nginx forwards this request to the Flask backend, which returns the original URL or an error.

URL Shortener Microservice Architecture






Link to Source Code Repo

<https://github.com/Siree16/VCC>

Link to Recorded Video Demo

 m23cse023.mp4

Plagiarism Clause:: I declare that this project is my own work, and any external resources have been properly credited. I understand that plagiarism will lead to disqualification from the assignment and possible further disciplinary actions.

