Workshop_2: Sockets and Services

Members:

Andrés Felipe Salazar Malagón – 20202020043

Teacher:

Carlos Andrés Sierra Virguez

Universidad Distrital Francisco José de Caldas

Faculty of Engineering

Computer-Networking

Bogotá, 2025

1. Introduction:

In this workshop, a local network for the Universidad Distrital Francisco José de Caldas was designed and configured using Packet Tracer. The main objective was to implement a functional network infrastructure that allows users to access a local server with the university's home page, as well as backend services developed in Python and JavaScript. Key decisions were made regarding IP address assignment, configuration of services such as DNS, DHCP, and HTTP, and connectivity testing from client devices.

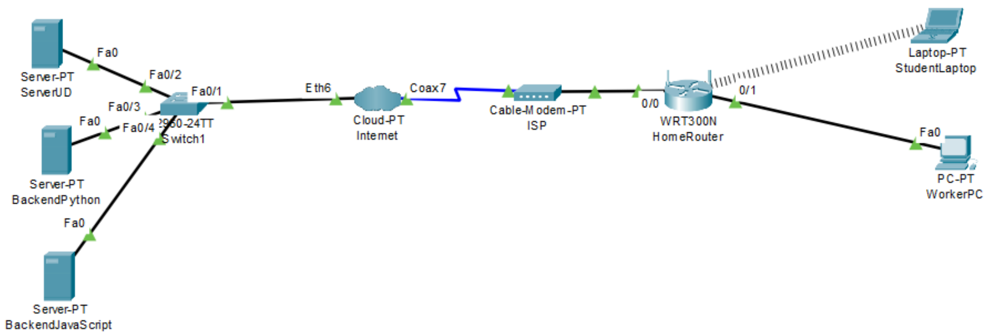
2. Network Design:

2.1. Network Topology:

The network is designed to connect various devices and provide access to the university website. The main components of the network are:

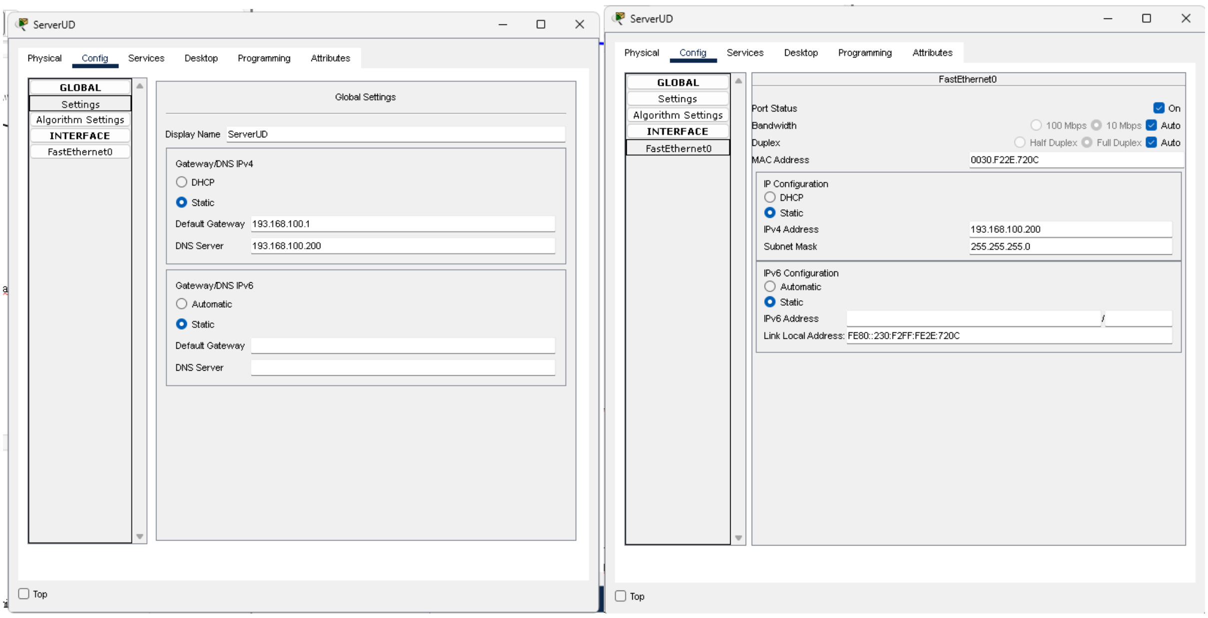
- Local server to host the university website.
- Client devices, such as a PC and a laptop, connected via DHCP.
- Wireless router that allows Wi-Fi access.
- Simulated Internet connection through a Cloud-PT.
- three servers—ServerUD, BackendPython, and BackendJavaScript—are interconnected through a switch.
- This switch acts as a central point to facilitate communication among the servers and to connect them to the rest of the network.

2.2. Network Diagram:



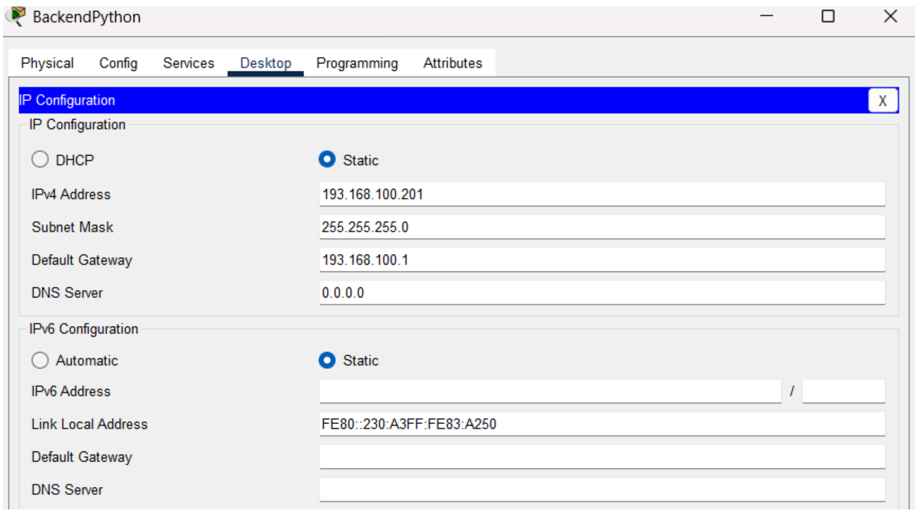
3. Technical decisions:

3.1. ServerUD:



- **Static IP:** A static IP is used for the server to ensure that the address is always the same and accessible from the internal network..
  - **IPv4 Address:** 193.168.100.200
  - **Default Gateway:** 193.168.100.1
  - **Subnet Mask:** 255.255.255.0
  - **DNS Server:** 193.168.100.200

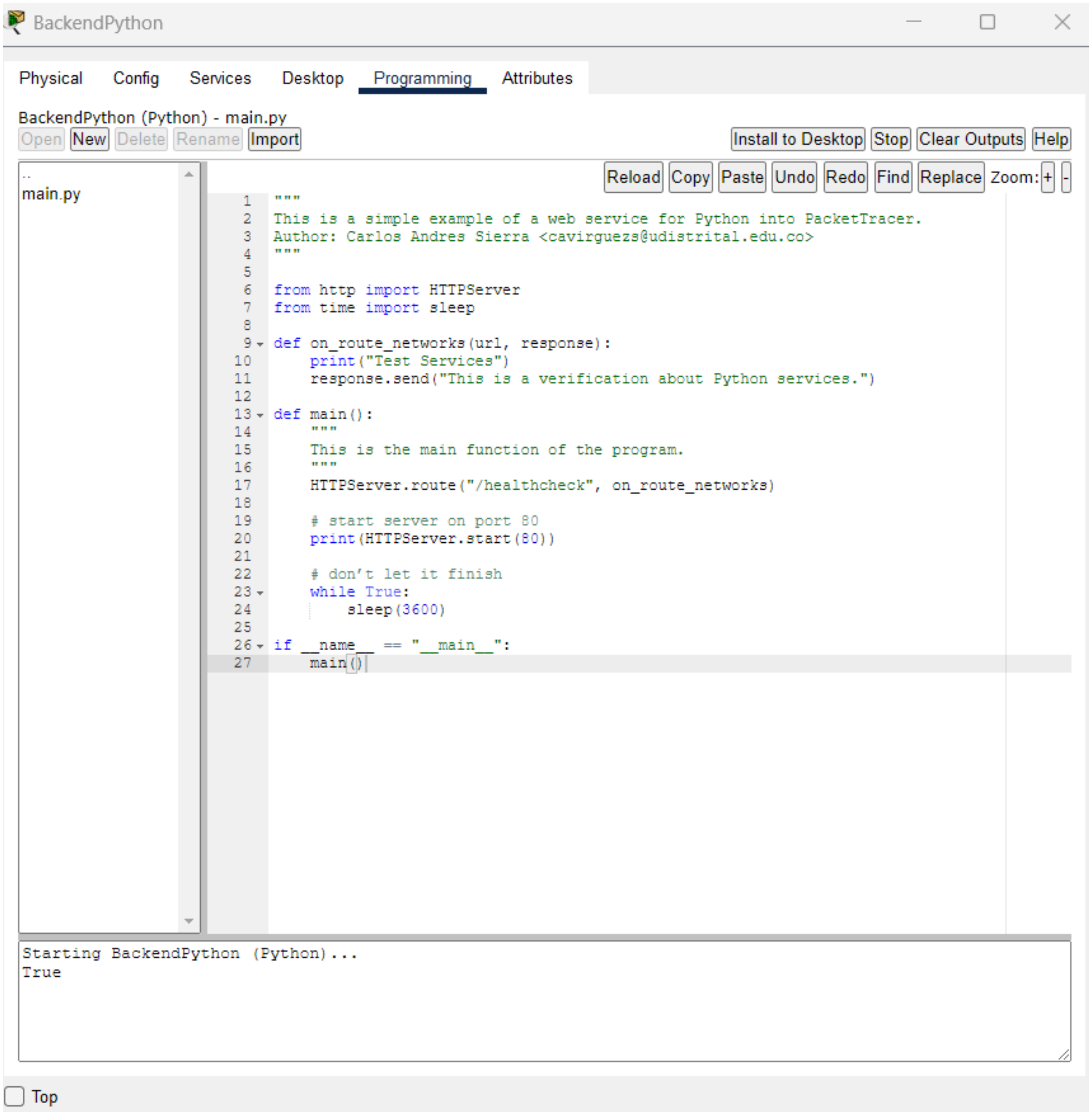
3.2 BackendPython:



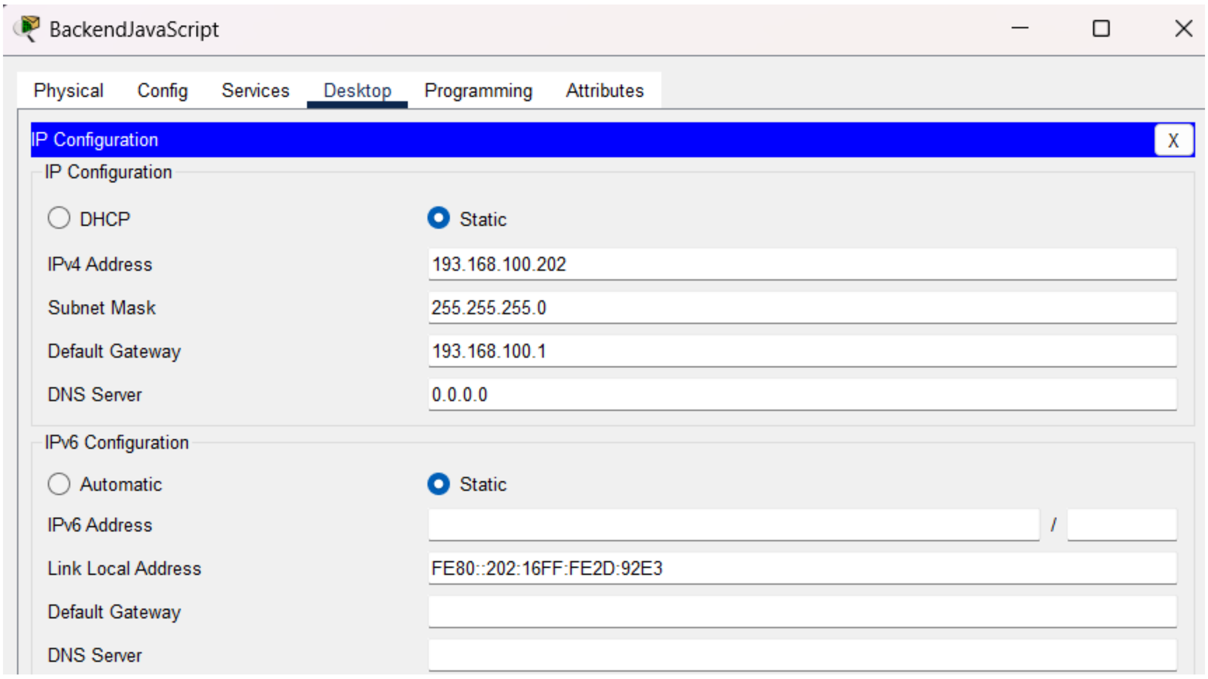
- **Static IP:**
  - **IPv4 Address:** 193.168.100.201
  - **Default Gateway:** 193.168.100.1

- **Subnet Mask:** 255.255.255.0

- **Programming**: This server is configured to run a basic HTTP service using Python's HTTPServer module. The on_route_networks function handles HTTP requests on the /healthcheck route, providing a plain text response confirming that Python services are up. The decision to use Python for this server was due to its versatility and ease of rapid development, as well as being suitable for connectivity testing and health monitoring within the network.



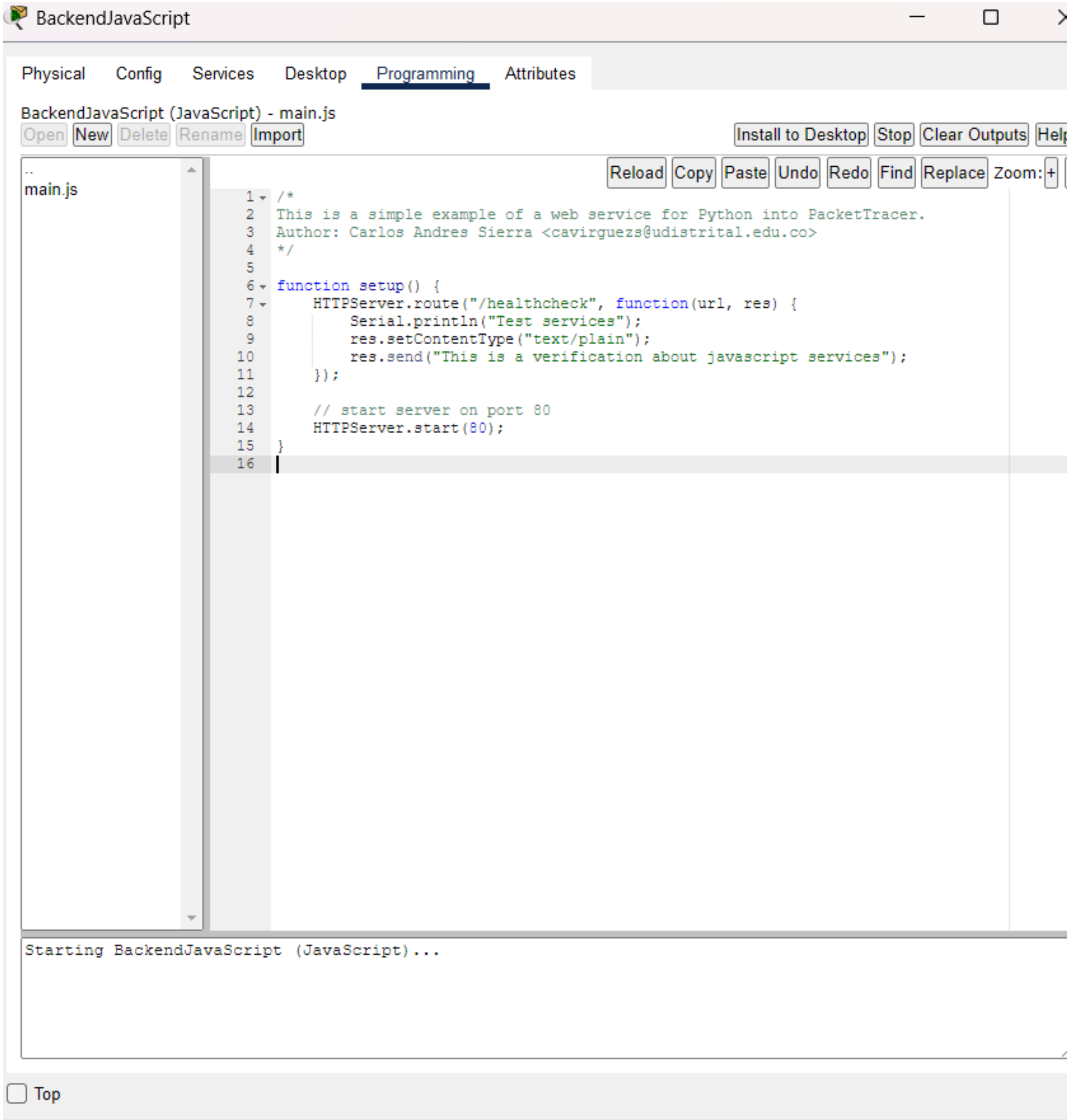
3.3 BackendJavaScript:

- 

  Static IP:

  - **IPv4 Address:** 193.168.100.202
  - **Default Gateway:** 193.168.100.1
  - **Subnet Mask:** 255.255.255.0

- **Programming:** The JavaScript server is implemented using a similar approach, setting up a /healthcheck route to respond with a plain text message indicating that JavaScript services are operational. Using JavaScript for this server has been a decision based on familiarity with the Node.js environment and its high ability to handle asynchronous and concurrent operations. Being JavaScript-based, it could also facilitate future integration with frontend applications developed with technologies such as React.



3.4 Internet Connection: Implementing a simulated Internet connection using a Cloud-PT to provide external connectivity to servers and devices.

3.5 Cable Selection:

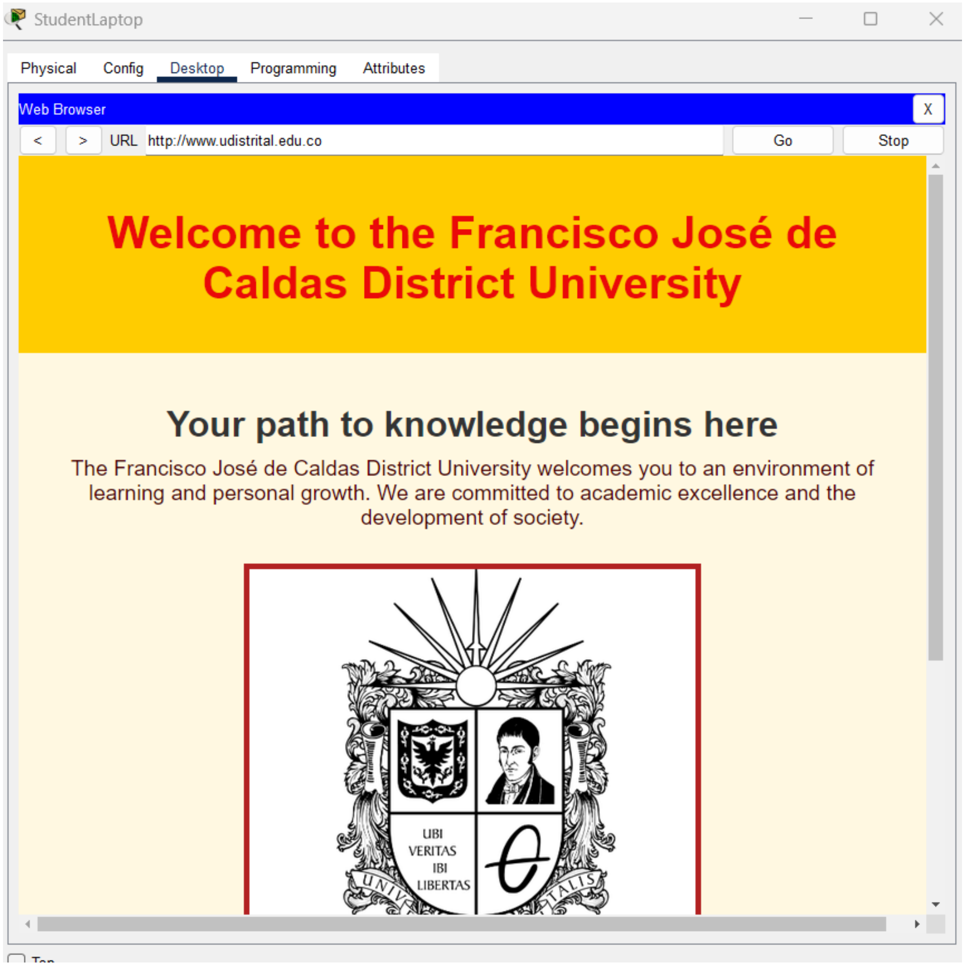
- **Copper Straight-Through Cable:**
  This type of cable was used to connect different devices, such as the server and the cloud. In this case, the server (FastEthernet0/0) is connected to the cloud port (Ethernet6), on the other hand it was used to connect the Cable-Modem-PT (port 1) to the HomeRouter (Ethernet0) and finally it was used to connect the HomeRouter (Ethernet1) to the WorkerPC (Ethernetport)
- **Coaxial Cable (Coaxial7):**

This was used to connect the Cloud-PT (simulating the Internet) to the Cable Modem (Cable-Modem-PT). Coaxial cable is suitable for broadband connections, such as those found in cable Internet services.

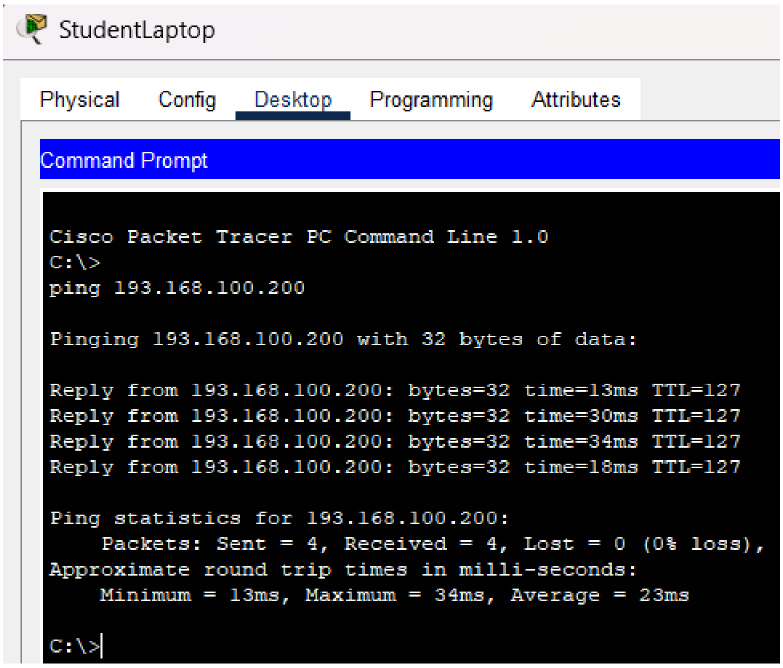
4. Test Results:

4.1. Connection Test from Student Laptop:

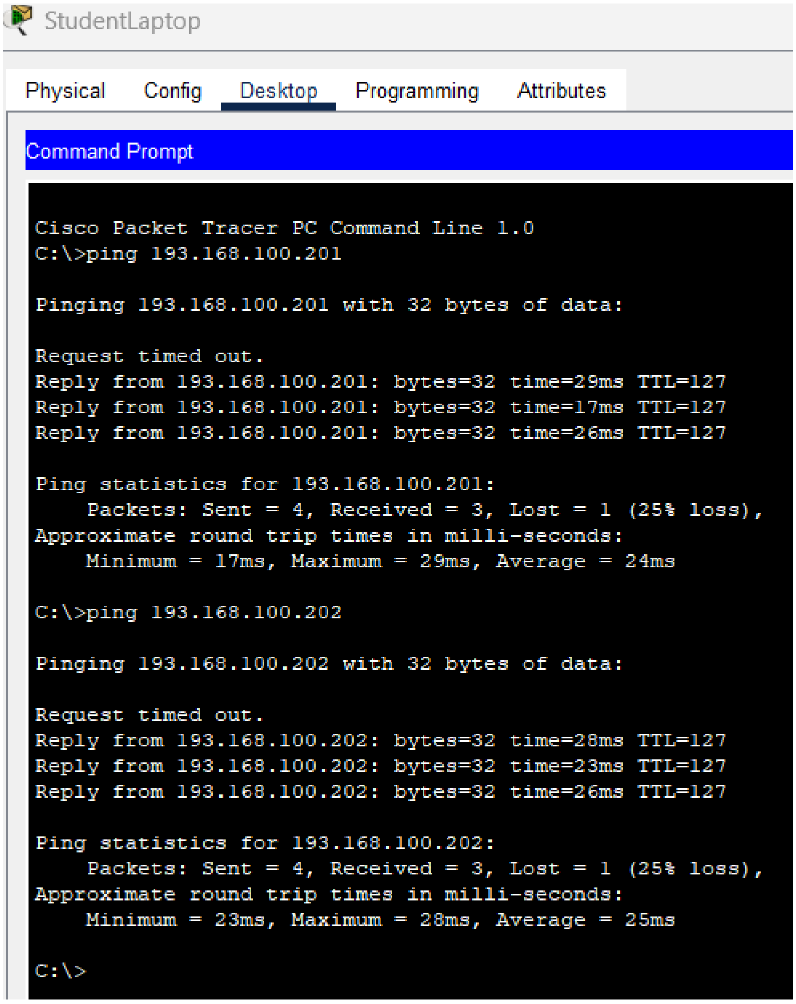
- **Device**: Laptop-PT
- **Connection**: Wi-Fi (SSID: UD_Invitados)
- **IP Configuration**: Assigned by DHCP
- **Result**: Access to the URL www.udistrital.edu.co from the browser was successful, displaying the welcome page.



- Ping ServerUD(193.168.100.200): successful.



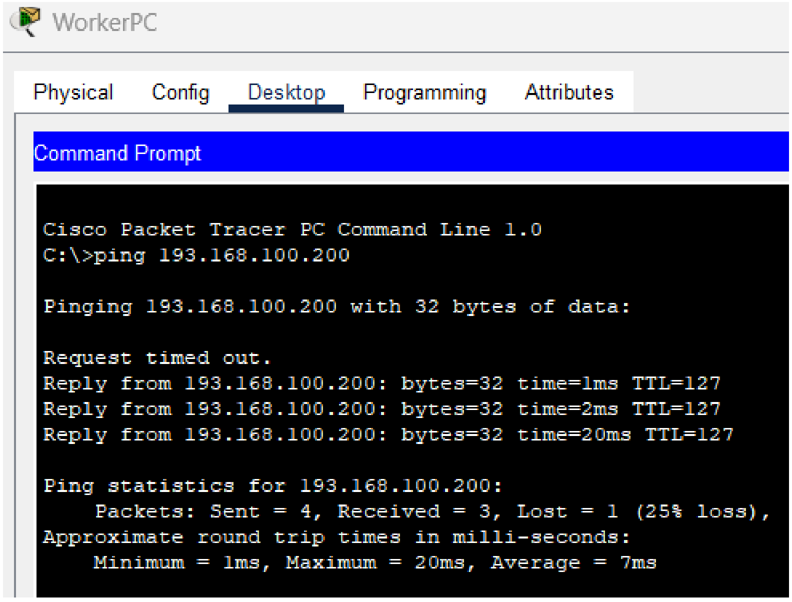
- Ping BackendPython (193.168.100.201) and ping BackendJavaScript(193.168.100.202): successful.



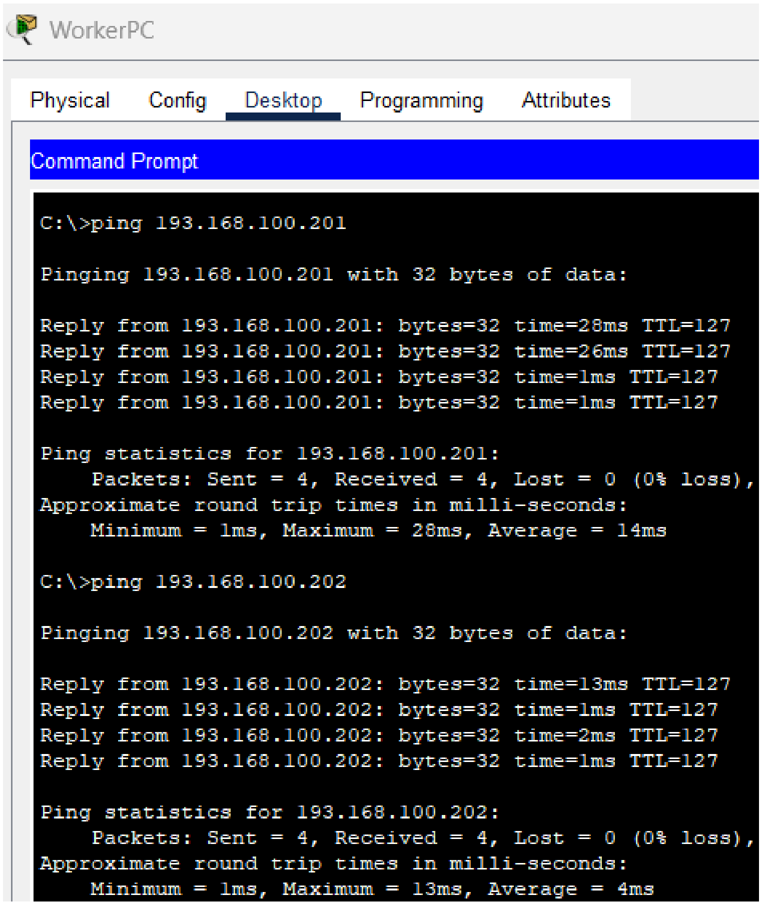
4.2. Test Connection from Work PC:

- **Device**: PC-PT
- **Connection**: Ethernet
- **IP Configuration**: Assigned by DHCP
- **Result**: Access to the URL www.udistrital.edu.co was successful from the web browser.

- Ping ServerUD(193.168.100.200): successful.



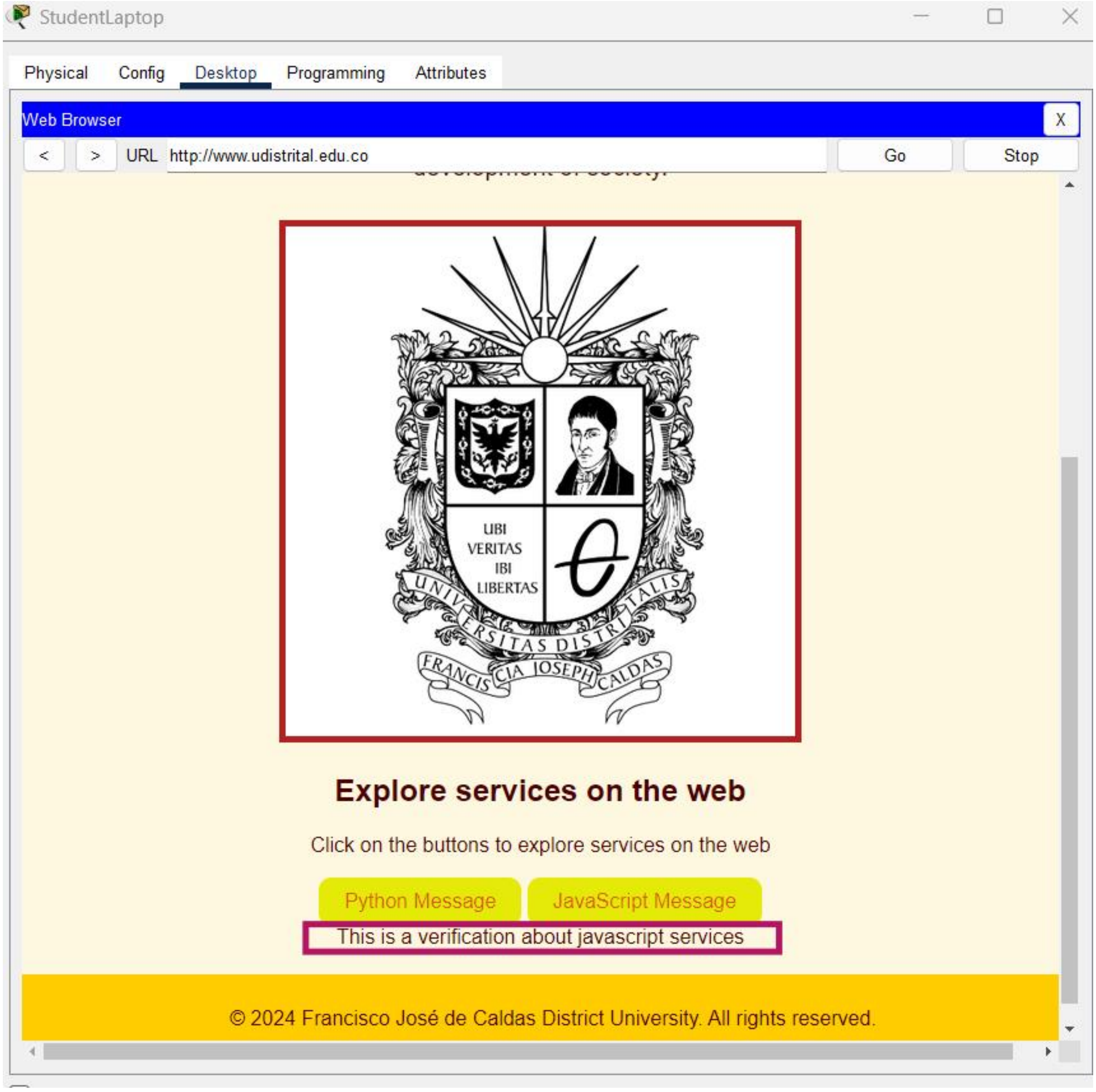
- Ping BackendPython (193.168.100.201) and ping BackendJavaScript(193.168.100.202): successful.



4.3. Using Buttons for Backend Services Laptop:

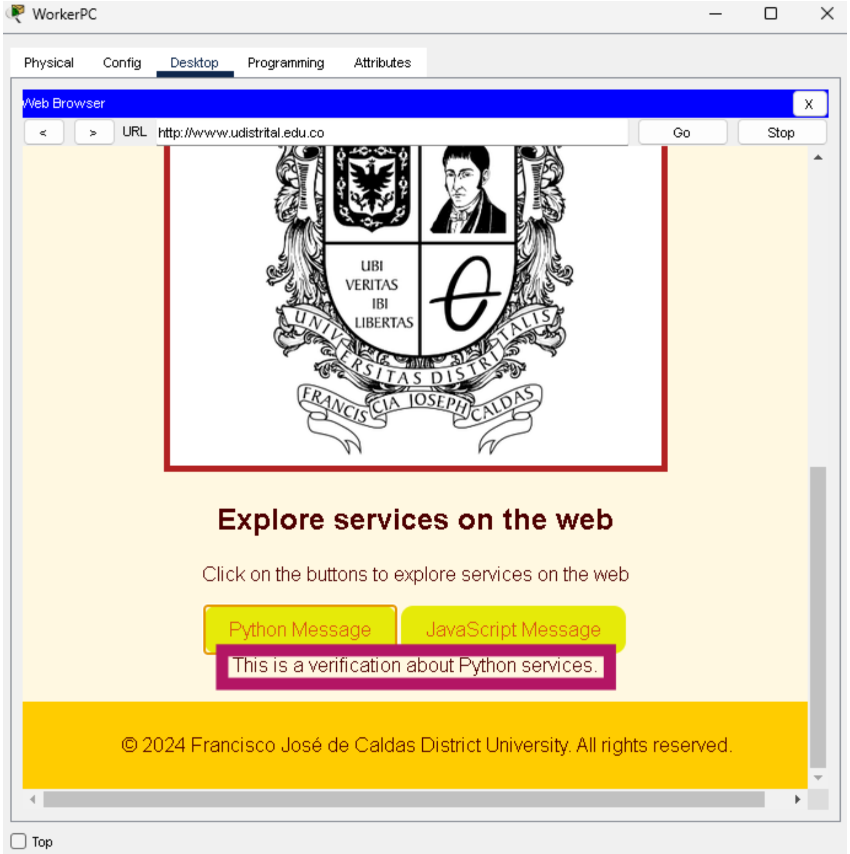
- **BackendPython**: When the user clicks the "Python message" button, the frontend makes an HTTP request to the BackendPython server, specifically to the /healthcheck route. This request reaches the server, which has this route configured to execute the on_route_networks function. This function responds with the message "This is a verification about Python services.", which is sent back to the frontend as a response to the HTTP request.

- **BackendJavaScript**: When the user clicks the "JavaScript message" button, the frontend makes an HTTP request to the BackendJavaScript server, specifically to the /healthcheck route. This request is handled by the server, which has this route configured to execute the JavaScript response function, which prints a verification message ("Test services") and then responds with the text "This is a verification about javascript services.".

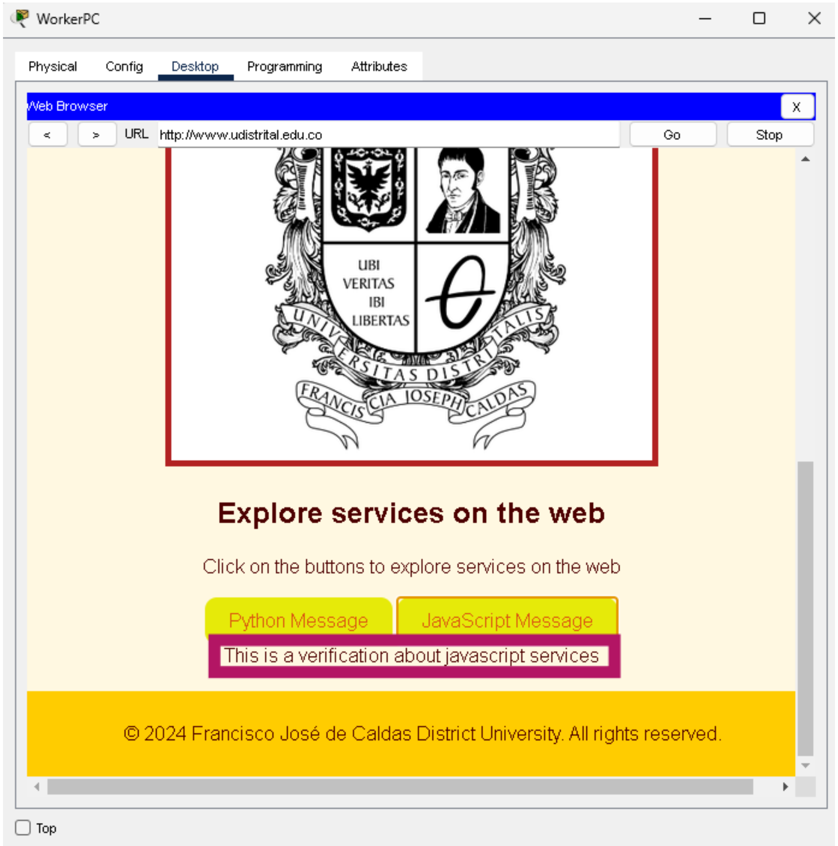


4.4. Using Buttons for Backend Services WorkerPC:

- **BackendPython WorkerPC:**



- **BackendJavaScript WorkerPC:**



Simulation in Packet Tracer to validate how the network works through the OSI Model.

- Simulation from laptop to ServerUD:

Vis.	Time(sec)	Last Device
	0.000	--
	0.001	StudentLaptop
	0.002	HomeRouter
	0.003	ISP
	0.003	--
	0.004	HomeRouter
	0.004	Internet
	0.005	Switch1
	0.006	ServerUD
	0.007	Switch1
	0.008	Internet
	0.009	ISP
	0.010	HomeRouter
	1.987	--

- Simulation from laptop to backend servers:

Simulation Panel

Event List

Vis.	Time(sec)	Last Device
	0.000	--
	0.001	StudentLaptop
	0.002	HomeRouter
	0.003	ISP
	0.003	--
	0.004	HomeRouter
	0.004	Internet
	0.005	Switch1
	0.005	--
	0.006	HomeRouter
	0.006	BackendPython
	0.007	Switch1
	0.008	Internet
	0.009	ISP
	0.010	HomeRouter
	1.995	--

- Simulation from WorkerPC to ServerUD:

Simulation Panel

Event List

Vis.	Time(sec)	Last Device
	0.000	--
	0.001	WorkerPC
	0.002	ISP
	0.002	HomeRouter
	0.002	HomeRouter
	0.002	--
	0.003	WorkerPC
	0.004	HomeRouter
	0.005	ISP
	0.005	--
	0.006	HomeRouter
	0.006	Internet
	0.007	Switch1
	0.008	ServerUD
	0.009	Switch1
	0.010	Internet
	0.011	ISP
	0.012	HomeRouter
	1.997	--

- Simulation from WorkerPC to backend servers:

Simulation Panel

Event List

Vis.	Time(sec)	Last Device
	0.000	--
	0.001	WorkerPC
	0.002	HomeRouter
	0.003	ISP
	0.004	Internet
	0.005	Switch1
	0.006	BackendJavaScript
	0.007	Switch1
	0.008	Internet
	0.009	ISP
	0.010	HomeRouter
	1.987	--

5. Conclusions:

- The implemented network design proved to be efficient and functional, allowing fluid communication between all connected devices, including the servers

(ServerUD, BackendPython, and BackendJavaScript), the laptop, the client PC, and simulated Internet access through Cloud-PT.

- The configuration of the servers allowed access to both the website hosted on ServerUD and the backend APIs in Flask and Node.js. This validates the correct configuration of the services and guarantees interoperability between the different technologies used.
- Tests carried out from different points of the network, including the assignment of dynamic IPs using DHCP and access to web services, confirmed the stability and reliability of the configured infrastructure, making it suitable for its purpose in a university environment.