

CORPORACION UNIVERSITARIA IBEROAMERICANA

FACULTAD DE INGENIERIA

INGENIERIA DE SOFTWARE

ACTIVIDAD 4 - DESARROLLO DE UNA ARQUITECTURA BASADA-CLIENTE

AUTOR

DUBAN LEONARNO GUZMAN MEDINA

ID: 100126620

DOCENTE

JOAQUIN SANCHEZ

MAJAGUAL(SUCRE) – COLOMBIA

2024

**Introducción:**

En la era digital actual, la comunicación entre diferentes sistemas y dispositivos es esencial para el funcionamiento de diversas aplicaciones y servicios. La arquitectura cliente-servidor es reconocida como uno de los modelos más eficaces para estructurar esta interacción. Este modelo se basa en la división de tareas entre proveedores de recursos o servicios, llamados servidores, y consumidores de estos recursos, conocidos como clientes.

El objetivo de este proyecto es implementar una arquitectura básica cliente-servidor utilizando Python, un lenguaje de programación de propósito general ampliamente utilizado en el desarrollo de software. Esta implementación servirá como base para comprender los principios de comunicación en red, así como la interacción entre los distintos componentes del sistema.

En esta aplicación, el servidor se encargará de escuchar y procesar las conexiones de múltiples clientes, enviando respuestas a sus solicitudes. Por otro lado, los clientes se conectarán al servidor para recibir mensajes y datos. Con esta arquitectura, puede aprender sobre sockets, administración de conexiones y comunicación de red, lo que proporciona un punto de partida para proyectos más complejos en el futuro.

La simplicidad de esta implementación permitirá a los desarrolladores y estudiantes obtener un conocimiento práctico de los conceptos de programación de redes, así como la importancia de la segregación de funciones al desarrollar aplicaciones. Como resultado del proyecto, los participantes estarán mejor preparados para resolver problemas más complejos en el campo de la arquitectura de software y la programación distribuida.

**Descripción del producto.**

**Instrucciones:**

1. Primero, ejecute el servidor en una terminal.
2. Luego ejecute el cliente en otra terminal o computadora. El cliente se conecta al servidor, recibe el mensaje y luego cierra la conexión.

**Explicación del código:**

**Servidor:**

* Escuche en el puerto 65432.Espere las conexiones del cliente.
* Cuando un cliente se conecta, el servidor envía un mensaje y luego cierra la conexión.

**Cliente:**

* Se conecta al servidor en localhost en el puerto 65432.
* Recibe el mensaje del servidor y lo imprime.

**Código servidor:**

import socket

def iniciar\_servidor():

# Crear un socket para el servidor

servidor\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Definir la dirección y el puerto del servidor

servidor\_direccion = ('localhost', 65432)

# Enlazar el socket del servidor a la dirección y puerto

servidor\_socket.bind(servidor\_direccion)

# Escuchar conexiones entrantes (máximo 5 conexiones en cola)

servidor\_socket.listen(5)

print(f"Servidor iniciado en {servidor\_direccion}")

while True:

# Esperar una conexión

cliente\_socket, cliente\_direccion = servidor\_socket.accept()

print(f"Conexión aceptada de {cliente\_direccion}")

# Enviar un mensaje al cliente

mensaje = "¡Hola, cliente! Gracias por conectarte."

cliente\_socket.sendall(mensaje.encode('utf-8'))

# Cerrar la conexión con el cliente

cliente\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

iniciar\_servidor()

**Código cliente:**

import socket

def iniciar\_cliente():

# Crear un socket para el cliente

cliente\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Definir la dirección del servidor al que nos conectaremos

servidor\_direccion = ('localhost', 65432)

# Conectar al servidor

cliente\_socket.connect(servidor\_direccion)

print(f"Conectado al servidor en {servidor\_direccion}")

# Recibir el mensaje del servidor

mensaje = cliente\_socket.recv(1024).decode('utf-8')

print(f"Mensaje del servidor: {mensaje}")

# Cerrar la conexión

cliente\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

iniciar\_cliente()

**Enlace GitHub:**  https://github.com/DubAngk/Cliente---Servidor.git

**Montaje de la Arquitectura Basada en Cliente en Python.**

**Las herramientas utilizadas:**

Para implementar la arquitectura cliente-servidor en Python se utilizaron las siguientes herramientas y tecnologías:

**Python:** Lenguaje de programación utilizado para el desarrollo de servidores y clientes. Python es conocido por su sintaxis clara y su extensa biblioteca estándar, que incluye módulos de red.

**Módulo de sockets:** Parte de la biblioteca estándar de Python. Este módulo le permite crear conexiones de red utilizando sockets. Proporciona funciones para establecer conexiones, enviar y recibir datos.

**Editor de código:** Se utiliza un editor de código (como Visual Studio Code,) para escribir y editar código. Esto facilita la escritura y administración de archivos Python.

**Terminal o consola:** Ejecutar scripts de Python mediante una terminal o consola de comandos, que le permite ejecutar el servidor y el cliente por separado.

**Aplicación:**

La implementación de una arquitectura cliente-servidor con Python tiene muchas aplicaciones en el desarrollo de software moderno, cubriendo una variedad de escenarios donde la comunicación entre un servidor y múltiples clientes es crítica. Estas son algunas de las aplicaciones más relevantes:

**Servicios web:** La arquitectura cliente-servidor es la base de muchos servicios web que permiten a los usuarios acceder a información y recursos a través de Internet. Aplicaciones como las API RESTful y los microservicios se basan en este modelo, lo que permite la interacción entre diferentes aplicaciones y sistemas.

**Aplicación móvil:** Muchas aplicaciones móviles dependen de un servidor central para gestionar la autenticación de usuarios, almacenar datos y proporcionar actualizaciones en tiempo real. La arquitectura cliente-servidor facilita la sincronización de datos entre aplicaciones móviles y servidores.

**Juegos en línea:** Los juegos multijugador en línea utilizan una arquitectura cliente-servidor para gestionar las interacciones entre jugadores. El servidor es responsable de mantener el estado del juego, administrar las conexiones de los jugadores y facilitar la comunicación en tiempo real.

**Sistema de mensajería:** Las aplicaciones de mensajería, como WhatsApp y Slack, utilizan esta arquitectura para permitir a los usuarios enviar y recibir mensajes al instante. El servidor actúa como intermediario que gestiona las conexiones y transmite mensajes entre clientes.

**Aplicaciones comerciales:** En entornos empresariales, las aplicaciones cliente-servidor se utilizan para gestionar bases de datos, sistemas de gestión de relaciones con los clientes (CRM) y otros servicios que requieren acceso compartido a datos y recursos.

IoT (Internet de todo): En un entorno de IoT, los dispositivos (clientes) pueden comunicarse con un servidor central para enviar y recibir datos. Esto es importante para monitorear y controlar dispositivos en tiempo real, así como para analizar los datos recopilados por esos dispositivos.

**Objetivo del ejercicio:**

El objetivo principal de este ejercicio es implementar una arquitectura cliente-servidor básica utilizando Python para proporcionar una comprensión básica de cómo se establece la comunicación entre un cliente y un servidor en un entorno de red. Con esta actividad pretendemos:

**Familiarizarse con el Uso de Sockets**: Aprenda a utilizar el módulo Python Socket para crear y administrar conexiones de red y cómo funciona el protocolo TCP/IP para comunicarse entre diferentes sistemas.

**Comprender la comunicación cliente-servidor:** Explore el flujo de datos en una arquitectura cliente-servidor y comprenda cómo un cliente envía solicitudes al servidor y cómo el servidor responde a esas solicitudes.

**Desarrollar habilidades prácticas:** Desarrolle habilidades prácticas de programación de redes, incluida la creación de un servidor que pueda manejar múltiples conexiones de clientes y la implementación de un cliente que interactúe con ese servidor.

**Cree una base para proyectos futuros:** Proporcione una base sólida para proyectos más avanzados que impliquen la creación de aplicaciones distribuidas, servicios web o sistemas en tiempo real que requieran una comunicación efectiva entre varias partes.

**Analizar aspectos de seguridad y escalabilidad:** Introducir conceptos básicos de seguridad en las comunicaciones de red y discutir la escalabilidad del modelo cliente-servidor en aplicaciones del mundo real.

**Conclusión:**

Implementar una arquitectura cliente-servidor en Python resultó ser un ejercicio fundamental para comprender los fundamentos de las redes. En este proyecto, exploramos cómo un servidor puede manejar múltiples conexiones de clientes, brindando respuestas a sus solicitudes en tiempo real. Esta experiencia no solo fortaleció nuestra comprensión de los conceptos teóricos asociados con la programación de redes, sino que también nos permitió desarrollar las habilidades prácticas necesarias para desarrollar aplicaciones distribuidas. La arquitectura cliente-servidor es fundamental para muchas aplicaciones modernas, desde servicios web y aplicaciones móviles hasta sistemas de mensajería y plataformas de juegos en línea. Al comprender cómo funcionan estos sistemas, los desarrolladores podrán crear soluciones eficientes y escalables que satisfagan las necesidades de los usuarios. Además, este proyecto proporcionó una base sólida para crear aplicaciones más complejas que integren aspectos de seguridad, gestión de datos y rendimiento. A medida que avanzamos hacia entornos de desarrollo más complejos, el conocimiento adquirido en este ejercicio servirá como un recurso valioso para resolver problemas de diseño e implementación de sistemas distribuidos.

La arquitectura cliente-servidor no es sólo un componente importante del desarrollo de software, sino que también representa una metodología eficaz para estructurar la comunicación entre diferentes sistemas en un mundo cada vez más interconectado. Este ejercicio fue un paso importante en el desarrollo de habilidades técnicas que son relevantes en el entorno tecnológico actual.

**Bibliografía:**

Beazley, D. M. (2013). *Python Essential Reference* (4th ed.). Addison-Wesley.

McKinney, W. (2018). *Python for Data Analysis: Data Wrangling with pandas, NumPy, and IPython* (2nd ed.). O'Reilly Media.

Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media.

Python Software Foundation. (n.d.). *Python's official documentation*. <https://docs.python.org/3/>