

INFORME DESARROLLO DE CHAT CLIENTE SERVIDOR

HILOS – SOCKETS Y HEROKU

PRESENTADO POR:

JULIAN GIRALDO CARDONA

LUIS FERNANDO ZULUAGA

JUAN DAVID ALVAREZ

SISTEMAS DISTRIBUIDOS

INGENIERIA DE SISTEMAS Y COMPUTACIÓN

UTP CUBA

09/12/2020

INTRODUCCIÓN

La red cliente-servidor es una red de comunicaciones en la cual los clientes están conectados a un servidor, en el que se centralizan los diversos recursos y aplicaciones con que se cuenta; y que los pone a disposición de los clientes cada vez que estos son solicitados. En Python, podemos hacer un cliente-servidor de manera sencilla utilizando sockets para hacer la conexión y también hilos para así recibir peticiones de múltiples clientes.

OBJETIVO PRINCIPAL

Desarrollar un chat cliente-servidor utilizando hilos y sockets en Python 2.7, el servidor debe de recibir una serie de comandos de parte el cliente y responder a estos, el cliente puede crear una cuenta y acceder a ella cuando lo desee.

OBJETIVOS ESPECÍFICOS

- Elaborar un servidor desarrollado en ambiente Python.
- Hacer posible que el servidor pueda recibir 3 clientes simultáneamente.
- Diseñar interfaz gráfica para facilitar el manejo.
- Realizar un menú de comandos para el cliente.
- Montar el servidor en Heroku CLI.

METODOLOGIA

Primero comenzamos creando un simple servidor mediante un socket, se indica la dirección que en este caso será el “localhost” y el puerto 6030. También definimos una clase Client utilizando hilos para que así por cada cliente que esté ingresando el servidor pueda recibirlo y ejecutarlo.

Ahora bien, vamos a establecer una serie de puntos de las tareas que el servidor debe hacer para así garantizar una conexión estable con el cliente y unas respuestas rápidas.

VISTA SERVIDOR:

Guardar datos de cliente:

El servidor tiene una dirección local para hacer un manejo de los usuarios y las contraseñas, entonces el cliente envía los datos por una lista “la lista se debe desarmar para enviarla por s.send()”, el servidor recibe los datos mediante un ciclo **for** para así agrupar los datos en ese instante. Algo importante para mencionar es que el cliente y el servidor constantemente se envían datos, ya sean peticiones y respuestas. Pero, para que el servidor entienda que esos datos que está enviando el cliente son exactamente para registrar un usuario, el cliente debe de enviar un código referencia para así en el momento en el que el servidor reciba los datos pueda entender qué tipo de datos le están enviando. El servidor crea un archivo en su dirección local y guarda el usuario. Antes de esto verifica si el nombre de usuario ya está elegido.

Verificación del usuario:

Al igual que la acción anterior, el cliente envía los datos junto con un código de referencia, por así decirlo una “etiqueta”. El servidor recibe esta etiqueta y consulta con los datos guardados en la dirección local para encontrar el usuario y contraseña y así hacer la verificación. Si la verificación es exitosa el servidor envía una bandera y el cliente la interpreta.

Enviar mensaje a los diferentes usuarios:

El servidor contiene una lista basada con hilos para así llevar un registro de todos los clientes que se han conectado, entonces, en el momento en que el cliente envía el mensaje junto con la etiqueta, el servidor obtiene dicho mensaje y lo reenvía a todos los clientes conectados.

Mostrar usuarios:

En el momento en que un cliente se registra, el servidor guarda el nickname en una lista para así ir agrupando todos los usuarios, entonces, cuando el cliente envía la solicitud de ver esa lista con todos los usuarios, el servidor lo que hace es enviar dicha lista.

Mostrar salas:

Cuando un Cliente crea una nueva sala desde la interfaz de usuario, se envían unos datos al servidor para así comunicar que se ha creado una sala y especificar quien es el dueño de esa sala. También se guardan los nombres de la sala en una lista y la información necesaria.

- ❖ Para cada uno de estos métodos el servidor primero debe de entender qué tipo de datos está recibiendo por parte del cliente, entonces. En este código establecimos un par de etiquetas para así entender cada solicitud.

Etiqueta '1':

Esta etiqueta es la que le indica al servidor que el cliente desea registrarse.

```
['Julian', '123', '1']  
Julian registrandose.  
['Julian', '123', '1']  
Julian se ha registrado.  
.
```

Como se observa en la imagen, así el cliente envía los datos y del mismo modo, el servidor los recibe. Con la etiqueta '1'

Etiqueta '2':

Esta etiqueta es la que indica al servidor que el cliente desea iniciar sesión.

```
['Julian', '123', '2']  
Julian iniciando sesion.  
Inicio correcto
```

Como se observa en la imagen, así el cliente envía los datos y del mismo modo, el servidor los recibe. Con la etiqueta '2'

Etiqueta '3':

Esta etiqueta es la que indica al servidor que el cliente está enviando un mensaje y este debe reenviarse a todos los clientes.

```
['Julian: hola', '3', 'lobby']  
se ha recibido un mensaje
```

El servidor recibe la lista con el mensaje ya insertado con el nombre del usuario, también recibe la etiqueta y la sala desde donde manda el mensaje, la etiqueta de la sala es fundamental ya que esto indica el mensaje hacia donde debe ser enviado.

Etiqueta '4':

Esta etiqueta es la que indica al servidor que desea ver todos los clientes conectados.

Etiqueta '5':

Esta etiqueta es la que indica al servidor que desea ver todas las salas.

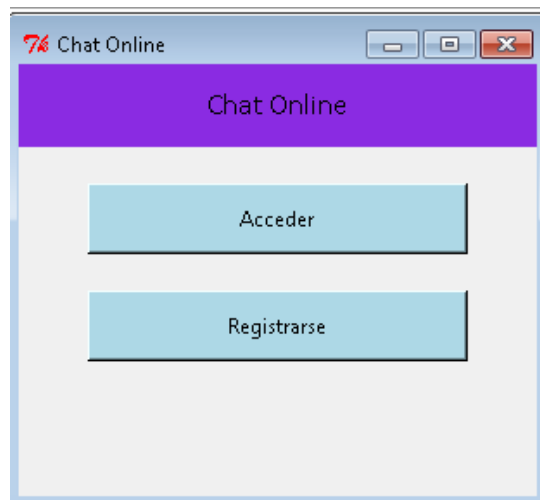
Gracias a estas etiquetas es posible la conexión y comprensión de solicitudes entre los dos códigos. Por parte del cliente también hay una serie de etiquetas o también llamadas banderas. La ventaja del cliente es que, en la mayoría de los casos, cuando envía una petición, en el momento exacto recibe la respuesta así que nos podemos ahorrar unas cuantas etiquetas.

Básicamente, con la serie de imágenes que se mostraron y el código que también viene adjuntado con este informe, podemos observar cómo funciona el servidor mediante el uso de etiquetas, ahora procederemos a observar como es la vista desde el cliente mediante la interfaz.

VISTA CLIENTE

Ventana principal

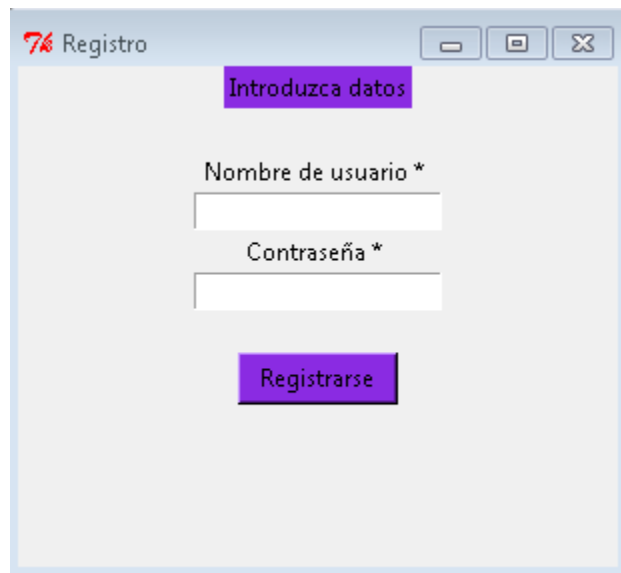
El cliente comienza observando una ventana principal donde tiene dos opciones, registrarse o iniciar sesión.



Para hacer posible el uso de interfaz, utilizamos la librería Tkinter de Python 2.7, Acceder y registrarse son dos botones que cada uno llama a una ventana nueva.

Al presionar registrarse:

Si el cliente presiona el botón Registrarse, se abrirá una nueva ventana donde se pedirán los datos de nombre de usuario y contraseña, estos datos son enviados hacia el servidor para guardar los datos o verificar si ya hay un usuario creado.

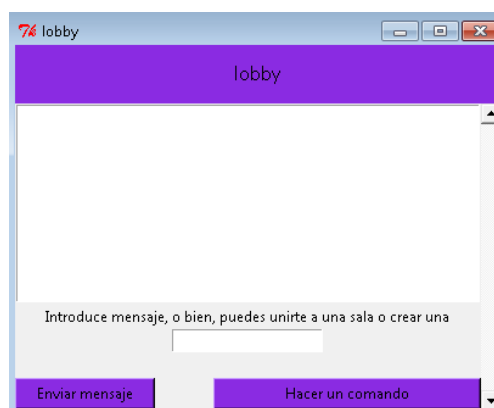


Si el usuario ya está creado entonces no se puede realizar el registro, pero, si es un nuevo usuario sí puede hacer el registro.

Al presionar Acceder:

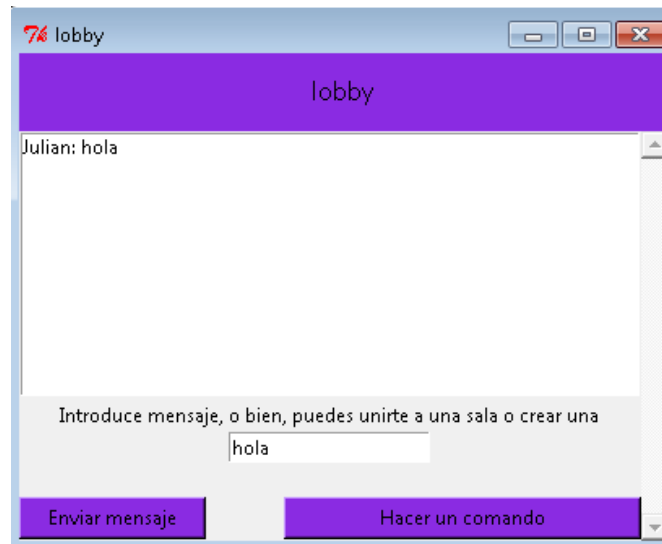
Cuando el cliente presiona el botón Acceder, se despliega una nueva ventana donde se pedirán los datos de usuario y contraseña, si los datos son correctos, entonces, se despliega la ventana donde se encuentra la sala de chat y también la opción de hacer comandos, pero si es incorrecta se despliega un aviso diciendo la contraseña o el usuario incorrecto.

Acceso exitoso:



En la ventana del lobby (sala principal), el usuario puede hacer dos cosas, la primera es enviar un mensaje o la segunda es abrir la ventana de comandos.

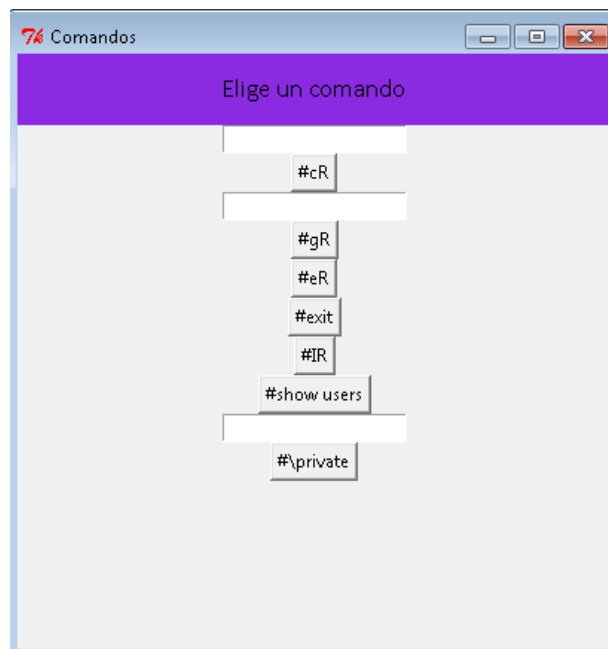
En el momento de enviar mensaje:



El mensaje se escribe en la casilla de texto y luego se oprime el botón enviar mensaje, luego de esto, el cliente envía el mensaje al servidor con la información de la sala y también la etiqueta "3" de mensaje. Cuando el servidor recibe el mensaje hace eco y el cliente lo recibe y lo muestra en pantalla.

En el momento de Hacer un comando:

Cuando elegimos la opción de hacer un comando se despliega la ventana de los diferentes comandos que hay disponibles.



Elegimos la opción de estandarizar los comandos ya que, desarrollar un intérprete en Python nos llevaría más tiempo y la prioridad ahora es manejar sockets e hilos, los comandos son los siguientes:

#cR

Este comando lo que hace es crear una nueva sala con el nombre que uno le quiera asignar y lo que recibe es una cadena de texto que se puede ingresar desde la casilla.

#gR

Este comando nos ingresa a una sala, básicamente hace lo mismo que crear salas ya que el servidor no lleva un registro de todas las salas actuales así que no se puede realizar esta verificación.

#eR

Este comando nos regresa a la sala principal.

#exit

Cierra la interfaz y se desconecta del servidor.

#IR

Muestra las salas desde el servidor.

#show users

Muestra todos los usuarios (no implementado)

#!/private

Envía un mensaje privado a un usuario (no implementado).

DEPLOY EN HEROKU DEL SERVIDOR

Para hacer un deploy completamente exitoso, debemos de instalar una serie de librerías y programas adicionales:

```
$ pip install gunicorn
```

Primero instalamos gunicorn

```
$ pip install virtualenv
```

Luego instalamos virtualenv, este nos servirá para crear un entorno virtual.

```
pip install Flask
```

Después de instalar virtualenv y crear un entorno virtual, debemos de instalar Flask dentro de ese entorno virtual.

Después de instalar las librerías necesarias también es importante tener instalado en nuestras maquinas Heroku, Git y un buen editor de texto, puede ser notepad++.

Heroku: [The Heroku CLI](#) | [Heroku Dev Center](#)

Git: [Git - Downloads \(git-scm.com\)](#)

Notepad++: [Downloads](#) | [Notepad++ \(notepad-plus-plus.org\)](#)

- ❖ Nota: Recomendamos utilizar una powershell diferente a la predeterminada en Windows ya que esta nos puede generar problemas a la hora de seguir los pasos en el montaje del servidor a Heroku.

PowerShell recomendada: [Releases · PowerShell/PowerShell \(github.com\)](#)

Ahora bien, después de tener todos los programas y librerías necesarios para subir el servidor a Heroku, basta con seguir los pasos que se encuentran en la página oficial de Heroku.

En nuestro caso, hicimos todos los pasos necesarios y el servidor quedó en la nube de Heroku de manera satisfactoria.

The screenshot shows the Heroku dashboard for the application 'chat-cuba.herokuapp.com'. The top navigation bar includes links for Overview, Resources, Deploy, Metrics, Activity, Access, and Settings. The main content area is divided into three sections: 'Installed add-ons' (showing no add-ons), 'Dyno formation' (showing the app is using free dynos with a 'web' dyno running 'gunicorn server:app'), and 'Collaborator activity' (showing the user 'julian.giraldo2@utp.edu.co' with 2 deploys). On the right, the 'Latest activity' section lists recent events: deployment of version v4, successful build, deployment of version v3, another successful build, enabling Logplex, and the initial release of version v1.

Link de acceso al servidor: <https://chat-cuba.herokuapp.com/>

Direcciones IP de conexión:

canonical name [chat-cuba.herokuapp.com](#).
aliases
addresses **34.235.104.230**
52.73.228.252
54.204.118.255
34.207.47.24
52.200.37.44
52.72.160.125
52.1.29.127
34.198.35.57

REFERENCIAS

<https://n9.cl/ari8>

<https://n9.cl/ii0r>