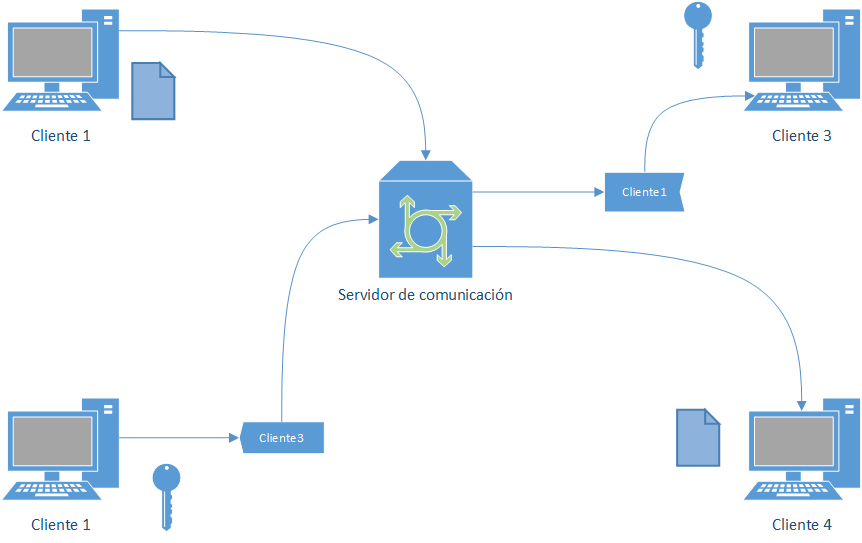
**Introducción**

Para el desarrollo del proyecto se tomaron en cuenta los siguientes requerimientos, ya que describían de manera general el comportamiento del sistema y la arquitectura que se debería usar.

* Cliente
  + El cliente recibirá una lista de participantes conectados, desde el servidor
  + El cliente podrá seleccionar cualquier usuario de la lista para conversar con él.
  + El cliente será completamente asíncrono, de manera que su comunicación será manejada por hilos que controlan el envío y recepción de datos.
  + El cliente será capaz de enviar cualquier tipo de archivos a otros clientes.
  + El cliente podrá usar encriptación simétrica para proteger sus mensajes mediante una contraseña.
* Servidor
  + El servidor se encargará de gestionar las comunicaciones entre los clientes.
  + El hecho de gestionar las comunicaciones implica que todos los mensajes, incluyendo archivos pasan por el servidor.
  + El servidor será capaz de enviar mensajes en modo único, o modo broadcast.

**Proceso.**

* Diseño de la arquitectura.



La arquitectura muestra como el servidor es el encargado de gestionar las comunicaciones entre los usuarios, no se encarga de asegurar los mensajes, solo de reenviarlos.

Por facilidad de implementación, se siguió el modelo de comunicaciones usado por el protocolo TCP/IP, es decir que el servidor envía todos los mensajes que recibe hacia todos los participantes, el mensaje contiene identificadores sobre el destinatario y el remitente, los clientes se encargan de rechazar o desplegar los mensajes específicamente dirigidos hacia ellos.

Para la transferencia de archivos, sin embargo, se realiza lo contrario, el mensaje de archivo solo es enviado al destinatario, con la finalidad de ahorrar espacio, tiempo de procesamiento y envió.

* Creación del servidor

El servidor tiene 3 partes principales

* + Aceptar nuevas conexiones
  + Manejar nuevos clientes
  + Enviar mensajes (Broadcast, Archivos)

El server guarda un diccionario con los sockets y los nombres de los clientes, de esta manera se puede enviar a todos los clientes por separado un mensaje.

Para recibir mensajes el server maneja un hilo que escucha por el socket que ha sido levantado.

Para cada cliente el server asigna un nuevo hilo, de manera que no se interrumpa el envío y recepción de los clientes ya conectados

* Creación del cliente

El cliente por otra parte cuenta con 4 partes principales

* + Enviar mensajes
  + Recibir mensajes
  + Modos de comunicación (Broadcast y Seguro)
  + Interfaz grafica

El cliente puede enviar mensajes a través de su respectivo socket, para ello cada operación esta manejada dentro de su hilo.

Finalmente, el cliente cuenta con la capacidad de encriptar sus mensajes y elegir si es que envía un mensaje de broadcast. Todo ello manejado por la interfaz gráfica, que cuenta con 2 ventanas, una para gestionar los modos de conexión y otra para abrir un chat especifico con otro usuario.

**Explicaciones sobre el código**

El código se divide en varias secciones:

*Para el cliente:*

* Declaraciones de variables de configuración y globales, las cuales generalmente son estados compartidos o configuraciones necesarias para la conexión, como la IP y el puerto.
* Una función para enviar, la cual se encarga de la distribución de los mensajes, dentro de ella se añaden los parámetros de nuestro protocolo de envío personalizado.
* Una función para recibir, la cual se encarga de decodificar los mensajes y controlar el despliegue de los mismos
* Una función para enviar un archivo, ya que esta necesita mantener el socket abierto mientras se envía la información contenida dentro de los archivos.
* Una función para habilitar los modos especiales, como el modo seguro y el modo broadcast.
* Una función para crear la interfaz de chat personalizado con otro usuario.
* La creación de la interfaz gráfica principal.

*Para el servidor*

* Declaraciones de variables de configuración y globales, las cuales generalmente son estados compartidos o configuraciones necesarias para la conexión, como la IP y el puerto.
* Manejo de nuevos clientes
* Envió de mensajes broadcast
* Reenvió de archivos
* Recepción de mensajes

Dentro del código fue necesaria la librería cryptography la cual permite la encriptación de manera simétrica, la funcionalidad fue implementada en un paquete aparte, de manera que no interfiera con el código dedicado a la comunicación por sockets.

El resto de funcionalidad fue implementada usando la librería estándar de Python.

**Comentarios**

Para el desarrollo la parte más complicada fue el envío y recepción de archivos, el principal problema con esta parte era el cierre de la comunicación, no había una manera altamente efectiva de notificar al servidor o al otro cliente que dejara de recibir. Sin embargo, fue posible hacerlo mediante la implementación de un timeout dentro del socket solo para esta parte de la comunicación.

Por lo demás el desarrollo del protocolo y comunicación con el servidor fue bastante sencillo ya que es un ejercicio estándar al momento de utilizar sockets en Python

**Recomendaciones**

* Mejorar el protocolo para no enviar tantos mensajes a través de la red
* Controlar la máxima cantidad de usuarios que puedan establecer una conexión al mismo tiempo
* Mejorar la interfaz grafica
* La aplicación de la seguridad podría ser mejorada bajo la implementación de un algoritmo asimétrico, que no tenga la necesidad de definir una clave previa, tal vez mediante certificados.
* Controlar de mejor manera el envío y recepción de archivos.