Implementación del Proyecto: Mail Client/Server

Para llevar a cabo la realización del proyecto como primer punto se necesitó repartir dentro de los compañeros del grupo quiénes iban a encargarse del servidor y quiénes iban a encargarse del cliente. Seguido a esto, se comenzó con la implementación de las clases Server.java y Client.java.

Para realizar la conexión de las clases por medio de puertos, se necesitó usar las librerías java **ServerSocket** y **Socket**, el propósito de estas es poder hacer conexiones desde el servidor local hacia el cliente que se conecte al puerto predeterminado 1400, además de que el servidor se conecta a otros servidores desde el puerto predeterminado 1500. Y, el DNS estará escuchando desde el puerto 1200. Seguido a esto, se implementaron las librerías de entrada y salida **BufferReader** y **PrintWriter** para poder mandar información entre las clases conectadas con ayuda de la clase Protocol.java.

Se necesitó implementar la librería **Swing** para la creación de la interfaz gráfica de las clases Server.java y Client.java. Donde ambas hacen uso de botones, cuadros de entrada, textos y ventanas. Al correr el Server, la interfaz gráfica mostrará las opciones de: consola del cliente, consola del servidor, consola del DNS, agregar una nueva cuenta y cambiar los puertos de las conexiones. Seguido a esto, se aceptará la conexión a la clase Client, la cual muestra como primera pantalla el ingreso de la cuenta del usuario; si los datos coinciden con alguna cuenta registrada en la base de datos se muestra la siguiente pantalla, de lo contrario, se muestra el error por parte del servidor tanto al usuario como al cliente. Al ingresar a la cuenta se muestran las opciones de: CLIST, GETMAILS, SENDMAIL, NEWCONT y LOGOUT.

- CLIST: Encargada de mandar a llamar la lista de contactos del usuario ingresado
- GETMAILS: Encargada de mandar a llamar la lista de nuevos correos del usuario ingresado
- SENDMAIL: Encargada de mandar correos a distintos contactos.
    - Se revisa que el contacto exista y pertenezca al servidor activo LexUt (servidor local)
    - Si, es de otro servidor, se realiza la conexión a través del puerto 1500 para revisar que el contacto sea parte del otro servidor activo. Esto con ayuda del DNS conectado a través del puerto 1200.
- NEWCONT: Encargada de crear nuevos contactos.
    - Se revisa que el contacto sea parte del servidor activo LexUt (servidor local)
    - Si, es de otro servidor, se revisa que este esté activo o exista
- LOGOUT: Encargada de salir y cerrar la cuenta del usuario

La clase servidor obtiene toda la información de estas operaciones a través de la clase ServerDataBase.java que hereda de la clase DataBase.java y es instanciado por la clase Dao.java. Con ayuda de estas clases, se logra crear la conexión a la base de datos SQL.

Además de las funciones anteriormente listadas, el cliente es el encargado de revisar que el usuario siga activo haciendo uso de la función NOOP, la cual se encarga de que la conexión cliente-servidor se cierre luego de cierto tiempo (40 segundos). Esto se realizó con ayuda de un hilo Runnable que obtiene el tiempo actual del sistema durante todo momento con la función System.currentTimeMillis(), con la cual al pasar 20 segundos manda al servidor la operación NOOP que obtiene como respuesta OK NOOP. Luego de esta respuesta, si vuelven a pasar 20 segundos sin que el usuario realice alguna operación, se cierra la conexión del servidor debido a la inactividad, seguida por el cierre de la conexión del cliente.

Cabe mencionar que ambas interfaces muestran los respectivos errores, para que tanto el cliente como el usuario estén conscientes de que hubo algún fallo en la ejecución de la función solicitada.

Entre las herramientas utilizadas, se encuentran **MySQL Workbench, Github** y **Visual Studio Code.** Las cuales fueron utilizadas para crear la base de datos, mantener actualizado el código y compilar el código respectivamente. Además, para hacer uso de la base de datos, se necesitó descargar **mysql connector-java.jar,** el cual importa la librería **sql** en java. Y, se hicieron uso de las estructuras de datos **ArrayLists** para obtener los correos, los contactos, los servidores y los remitentes de los correos dentro de la base de datos, además de **HashMap** para obtener los IPs y nombres de los servidores. La razón de hacer uso de ArrayLists se debe a la facilidad de obtención de datos, además de que la base de datos está ordenada de manera predeterminada, por lo que no se necesita dar prioridad a ningún dato. Y, el HashMap fue utilizado debido a que tanto el nombre del servidor como la dirección IP son datos únicos, además de estar relacionados mutuamente. Siendo así que, cada nombre del servidor se asignó como llave con su respectivo valor (IP).