Informe proyecto 2 de Introducción a la Programación: Mensajería Encriptada

El segundo proyecto no resultó ser tan desafiante como el primero. Esto es porque, en esta ocasión, yo sí sabía lo que era un mail. Evidentemente, estaba familiarizada con el tema desde antes. El único obstáculo que pudo haber presentado este proyecto en cuanto a conocimiento en un inicio era que no sabía lo que era un número binario, cómo construirlo y menos como pasar de binario a decimal y viceversa. Este problema lo solucioné rápidamente preguntándole a la profesora, quien me aclaró lo que era un número binario, lo que significaba que trabajáramos con 8 bits, y en rasgos muy generales como, a partir de un número binario, se pasaba a número decimal. El resto fue simplemente construir las funciones pedidas en el enunciado.

En esta oportunidad, mi proyecto presentó algunos problemas específicos. El primero, es que, cuando comencé a probar el servicio de mensajería con mis compañeros, no podía leer sus mensajes. Buscando en el foro, encontré una publicación que aclaraba que el enunciado estaba incorrecto, y que, a la hora de encriptar o desencriptar, no había que usar los últimos caracteres de la clave, sino los primeros (prefijo, no sufijo como salía en el enunciado). Una vez que me percaté de esto y arreglé la función OTP, todo funcionó. Sin embargo, todos los mensajes que antes se habían mandado mal quedaron en el servidor, y no pude deshacerme de ellos. **Esa es la razón por la que aparecen símbolos extraños en la parte de mensajes.**

El segundo problema significativo que tuve fue, que una vez que enviaba un mensaje, el mensaje enviado se duplicaba en la lista de mensajes enviados entregada por el servidor. En realidad, el error era bastante simple, y era que tenía copiado el comando de enviar mensajes (de mensajería) dos veces en el código del programa principal.

Finalmente, el tercer y último (pero gran problema) que tuve en este proyecto fue el ordenar las listas de mensajes enviados y recibidos, todos mezclados, según la fecha. Quize hacerlo así porque considero que es un método de ordenamiento realista y eficiente. En un comienzo, creé un código extremadamente extenso, muy detallado, pero no sirvió. Ocurría un error todo el rato, que no era un error común. Intenté cambiar el nombre de la función, de las variables, pero nada funcionó. De esta forma, mi mente me envió a sentarme frente a una mesa y tomar una hoja de papel y un lápiz. Volví a crear un segundo código, el cual funcionaba para listas pequeñas, de aproximadamente tres elementos. Cuando lo intenté con las listas del servidor el programa colapsó, y, es más: al intentar combinar las dos listas “ordenadas” (enviados y recibidos), la función retornaba una lista vacía. ¿Por qué ocurrió? Hasta este día, honestamente no lo sé.

Miré la hora del reloj y decidí volver a intentar el día siguiente. Llegué a la universidad y en la ventana me senté en una mesa en la sala de estudios, pensando que debía existir una forma más simple de ordenar las listas. Por eso, inventé una función que ordena la lista de enviados y de recibidos por separado. Además, cada mensaje (que era una lista en sí), usando la función, se podía convertir en un string de inmediato, con “De:”, “Para:” y “-“, para poder presentar los mensajes en la pantalla como en el enunciado. Luego, hice un ciclo while para poder juntar ambas listas (porque comandos como append o extend con for retornaban listas vacías), y usé el comando list.sort() para ordenar la lista. Como la lista final fue ordenada en el sentido contrario al que yo quería, usé el comando list.reverse() y el problema se solucionó.

Sección opcional

Hubo un par de cosas que quise incluir o trabajar de más en el proyecto, para que quedara lo más parecido posible a un servicio de mensajería auténtico. Un aspecto importante que modifiqué fue la búsqueda. Yo no sólo quería que al ingresar un string en la barrita de búsqueda se mostraran los mensajes con ese string, sino que lo que a mí me importaba era la "información". Es decir, si yo escribía "cookie," yo quería que mi pantalla de mensajes mostrara cosas como "Cookie," "COOKIE," o "cookie," entre otras variantes de la palabra. Uso este ejemplo porque tengo mensajes con esta palabra y probé las modificaciones que hice con "cookie" y con "hola". Similarmente, quería que la búsqueda filtrara las direcciones de correo, porque quería que mi programa fuera lo más realista posible. Pensando en el usuario, hay veces que este no recuerda lo que otra persona le envió, por ejemplo, necesita ver ese mensaje.

También, el problema anterior se mezcló con que yo no quería que, si por ejemplo ponía "para" en la búsqueda, me mostrara todos los mensajes. Esto lo digo por la forma en que estaba trabajando con la lista final, que había creado, con todos los mensajes como strings de la forma "De: mail Para: mail – mensaje". Este formato lo escogí porque quería que se pareciera al del enunciado, y porque considero que es lo más preciso y conciso posible. Muestra la información esencial requerida a la hora de tratar mensajes recibidos o enviados.

Para lograr solucionar todo lo anterior, lo que hice fue crear una variable que fuera una copia de mi string-mensaje (pensando que ya estaba trabajando con la lista en donde tenía los mensajes recibidos y enviados como string, no como lista). Así, fui cortando esta variable string de modo de ir guardando en otra variable, llamada copia, los aspectos esenciales: ambos mails y el mensaje, todo junto y concatenado. Finalmente, creé otra variable, que era la versión en minúsculas de la variable anterior, que guardaba las características que yo quería. Así, fui buscando el string ingresado en la barrita de búsqueda, también en minúsculas, en el string con los mails y el mensaje concatenados en minúsculas.

Finalmente, hubo unas últimas características que agregué a mi proyecto en mi afán de querer usar todos los métodos del enunciado, de ambas librerías. Entonces, para poder usar el método gui.salir() y gui.preguntar(msg), hice que, si el usuario no era capaz de conectarse al servidor, se le preguntara si quería intentarlo de nuevo o no. Si la respuesta era afirmativa, podía seguir intentando cuantas veces quisiera, hasta que la respuesta fuera negativa, en cuyo caso se abre una alerta, se ejecuta gui.salir() y se cierra el servicio de mensajería, además de desplegarse una especie de "despedida" que dice:

"Adiós!! Recuerda seguirnos en Facebook y Twitter :p"

Quizá lo anterior no sea formal, pero una sonrisa le hace bien a las personas ☺

Además de esto, incluí alertas para cuando un mensaje tuviera más de 45 caracteres y para cuando el usuario alcanzara su límite de mensajes enviados por día. De esta última forma fue como incorporé mensajería.mensajes_enviados_hoy(), método que retornaba una variable int que guardé bajo el nombre "total". Así, cuando el usuario alcanza su límite del día, se abre una ventana de alerta indicando esta situación.