PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2233 - Programación Avanzada (I/2016)
Tarea 6

1. Objetivos

- Aplicar nociones de networking para crear un sistema.
- Aplicar conceptos de manejo de bytes para trabajar con los archivos.
- Aplicar conocimientos de interfaces.

2. Introducción

¡¡Lo lograste!! ¡¡*Ayudantados Trouble* es un éxito en ventas y no deja de superar récords!!. Después de un año de una vida perfecta, y habiendo olvidado los lamentables eventos recientes, lograste fundar tu propia empresa de videojuegos: **IziPizi**. Tu éxito es tal que deseas expandirte a otras áreas, y decides probar tu suerte en el rubro de la mensajería. Esto, a través de una aplicación que, si bien es un simple chat, funciona y tiene el mejor nombre[1]: **BlinkChat**. Cuando pensabas que nada podía ir mejor, ocurre lo inevitable: Desde lo más profundo del abismo, el benévolo Bucchi vuelve y solo busca venganza, por lo que ha decidido fundar su propia empresa de mensajería con el objetivo de llevarte a la bancarrota (de nuevo). Para colmo, ha decidido contratar a los mejores hackers del país para romper la seguridad de tu producto.

Ante esta situación, y considerando que ya no hay vuelta atrás (has invertido todos tus ahorros en BlinkChat), no te queda más opción que mejorar tu aplicación, agregando funcionalidades nunca antes vistas por el ser humano[2] y mejorando la encriptación de los mensajes.

3. Blink Chat

La tarea consiste en implementar **BlinkChat**, una aplicación que se centra en el intercambio de fotografías con otros usuarios, pero que además cuenta con funcionalidades propias de una aplicación de mensajería. Las principales funcionalidades de BlinkChat son:

- Manejo avanzado de usuarios.
- Soporte para envío de archivos.
- Múltiples filtros disponible para las imágenes a enviar.
- Historial y encolamiento de mensajes.
- Chats individuales.
- Soporte para Emojis.

[1] Después de 'Bastian'... claro que sí.
[2] Supuestamente.

3.1. Cliente-Servidor

Su implementación de BlinkChat deberá basarse en la arquitectura *cliente-servidor*, en donde todas las interacciones que se quieran realizar con algún usuario deberán pasar por el servidor. Para lograr lo anterior, deberá implementar un protocolo eficiente de comunicación entre el cliente y el servidor usando el modelo **TCP/IP**.

Todas las interacciones que se esperen del usuario, al menos en la parte del cliente, deben realizarse a través de interfaces gráficas. La interfaz debe ser amigable y enfocada en la usabilidad, es decir, comportarse de acuerdo a lo que espera el usuario de forma natural, sin eventos inesperados.

3.1.1. Sistema de autentificación (registro e ingreso de usuarios)

Al iniciar el programa, este deberá ofrecer dos opciones: registrarse si es un usuario nuevo, o ingresar si es que es antiguo. Si es que se escoge la opción de registro, el chat deberá pedirle al usuario un nombre de usuario, contraseña y confirmación de la contraseña. Se debe verificar que el nombre de usuario no esté registrado, es decir, deben ser únicos entre todos los usuarios de BlinkChat.

La base de datos de los usuarios debe persistir. Es decir, los usuarios registrados durante una ejecución del programa deberán estar disponibles durante una ejecución en un momento diferente. Para hacer esto queda a su criterio el mecanismo a implementar.

Existe un requerimiento especial de seguridad del sistema: las contraseñas nunca deben quedar guardadas en el disco duro sin encriptar. Para encriptar las contraseñas se debe usar el protocolo **hash + salt** que se detalla a continuación:

Cuando se escriba un nuevo usuario al disco duro debes:

1. Generar una secuencia de bytes aleatoria. Esta secuencia será nuestra 'sal'. Se recomienda fuertemente que la obtenga a través del método `urandom` del módulo `os`.
2. Concatenar la 'sal' con la codificación en bytes de la contraseña del nuevo usuario. Recuerda que puedes obtener la codificación en bytes a través del método `encode` de un string. La concatenación de ambas secuencias será nuestra secuencia a encriptar.
3. Pasar la secuencia a encriptar por alguna función de hash. Se recomienda fuertemente usar el algoritmo `sha256` a través del módulo `hashlib`. La secuencia de bytes que sea retornada por la función de hash será nuestra contraseña encriptada.

Luego, cuando un usuario intente ingresar a BlinkChat, debes:

1. Obtener desde la base de datos de BlinkChat la 'sal' y contraseña encriptada del usuario que está intentando ingresar.
2. Concatenar la sal con la secuencia de bytes de la codificación de la contraseña no encriptada, es decir, la contraseña que el usuario que está intentando ingresar escribió. Esta será la secuencia a verificar.
3. Pasar la secuencia a verificar por la misma función de hash que se utilizó al momento de registrar al usuario. La secuencia de bytes que retorna de la función de hash corresponderá a la secuencia encriptada a verificar.
4. Hacer la comparación entre la secuencia encriptada a verificar y la contraseña encriptada. Si son iguales, entonces la contraseña que ingresó el usuario es correcta. De lo contrario, la contraseña es incorrecta.

Para que este protocolo funcione, es necesario guardar la 'sal' aleatoria de cada usuario (en caso contrario, es imposible hacer la comparación).

Además, el servidor debe ser capaz de leer archivos con información para poblar inicialmente el sistema. Específicamente, se entregarán los archivos `{usuarios|chats}.json`. La información en el archivo de usuarios vendrá estructurada de la siguiente forma:

```
[
    {
        "lista_amigos": [
            "marcel",
            "chris"
        ],
        "nombre_usuario": "john",
        "password_no_encriptada": "macoy123"
    },
    {
        "lista_amigos": [
            "marcel",
            "john"
        ],
        "nombre_usuario": "chris",
        "password_no_encriptada": "trustno1"
    },
    {
        "lista_amigos": [
            "john",
            "chris"
        ],
        "nombre_usuario": "marcel",
        "password_no_encriptada": "letmein"
    }
]
```

Por otra parte el formato del archivo de chats es el siguiente:

```
[
    {
        "entre": [
            "chris",
            "marcel"
        ],
        "mensajes": [
            { "emisor": "chris", "texto": "wena" },
            { "emisor": "marcel", "texto": "como andamos?" },
            { "emisor": "chris", "texto": "fineli , tu?" },
            { "emisor": "marcel", "texto": "bien igual" }
        ]
    },
    {
        "entre": [
            "john",
            "chris"
        ],
        "mensajes": [
            { "emisor": "john", "texto": "estai?" },
```

```
            { "emisor": "chris", "texto": "see" },
            { "emisor": "john", "texto": "fito?" },
            { "emisor": "chris", "texto": "se sabe" }
        ]
    }
]
```

Para parsear estos dos archivos pueden usar el módulo `json` de la librería estándar de Python[3].

Una vez que un usuario ha ingresado correctamente, si este ha recibido mensajes mientras no estaba conectado, deberá recibir todos los mensajes o Blinks que posee.

3.2. Amigos

Un usuario debe tener la opción de agregar Blink-amigos, no es necesario que el otro usuario los confirme. Cuando un usuario agrega al otro, la amistad es inmediatamente bidireccional (si A agrega a B, cuando B ingrese, le saldrá A como amigo). Para agregar un Blink-amigo, basta con clickear un botón destinado a agregar amigos e ingresar el nombre de usuario de este.

3.3. Blinks

Un Blink es una imagen que puede ser enviada a uno o más usuarios (estén conectados o desconectados). Su funcionamiento será descrito en detalle en las subsecciones siguientes.

3.3.1. Manejo de Bytes de un Blink

Cada Blink debe encontrarse en formato **PNG** (con la extensión `.png`). Este tipo de imágenes, en su estructura de bytes, comienza siempre con los bytes `137 80 78 71 13 10 26 10`. Luego de esto, los bytes se ordenan en bloques o *chunks*.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	...	n	n+1	n+2	n+3	n+4	n+5	n+6	...
89	80	78	71	13	10	26	10	chunk						...	chunk							...

Los bloques tienen una estructura establecida, que es la siguiente:

Estructura de un chunk o bloque			
Largo de información del bloque	Tipo de bloque	Información	CRC
4 bytes	4 bytes	Largo	4 bytes

Donde los primeros 4 bytes indican el largo que posee la sección de información del chunk. Los siguientes 4 bytes pueden ser decodificados para obtener un string de 4 caracteres que indican qué tipo de información posee el bloque. Luego, viene la sección de la información, que posee la cantidad de bytes indicadas en un principio. Esta información varía dependiendo del tipo de bloque en el que se esté trabajando. Por último, los siguientes 4 bytes corresponden a un código para la información del chunk.

[3] Exactamente, ¡ya no deben hacerlo manual! :).

Tipos de bloque

- `IHDR`
  En este bloque se tiene la metadata de la imagen.

IHDR	
Cantidad de bytes	Descripción
4	Ancho en pixeles
4	Alto en pixeles
1	Profundidad de bits
1	Tipo de colores
1	Tipo de compresión
1	Tipo de filtro
1	Tipo de entrelazado

- `IDAT`
  Este bloque posee los bytes de la imagen comprimidos. En una imagen pueden existir varios bloques `IDAT`. Para descomprimir la información de la imagen y poder trabajar con los datos, es necesario concatenar todas las secciones de información de la imagen.

- `IEND`
  Este bloque indica el término del archivo, y en la sección información no posee datos (el largo de la sección es 0).

Como se indica en `IDAT` la información contenida dentro del chunk está comprimida. Por lo tanto, deben utilizar métodos que descompriman los datos para poder trabajar con los pixeles, y que también los compriman para reescribir el archivo y generar el código `CRC` final de cada chunk.

3.3.2. Blinks de un Usuario

Un usuario, al ingresar, deberá recibir todos los Blinks y mensajes que se le han enviado mientras estaba desconectado (como fue mencionado anteriormente). Luego, tendrá la opción de visualizar los Blinks que tiene disponibles (enviados por sus Blink-amigos). Si un usuario posee más de un Blink por Blink-amigo, se deberá representar de forma gráfica mostrando el nombre del usuario que lo envió y el número de Blinks que pertenecen a ese usuario de alguna forma elegante[4]. Al clickear algún usuario, se deberán mostrar de forma cronológica los Blinks de este, y además podrán ser vistos durante un cierto período de tiempo definido por el usuario que envía el Blink antes de ser autodestruido.

3.3.3. Envio de Blinks

Un usuario puede enviarle Blinks a otros usuarios solo si estos se encuentran dentro de su lista de Blink-amigos y/o a su Historia de Blinks. La imagen del Blink puede o no tener un filtro aplicado, y además se debe indicar una cantidad de tiempo (en segundos) por la cual será visualizada por el usuario destinatario. Un Blink debe ser capaz de ser enviado de forma simultánea a múltiples usuarios. Para enviar un Blink, debe aparecer un pop-up dentro de la ventana principal de la interfaz que permita seleccionar una imagen de tu directorio local. **Nota:** Las únicas imágenes que se pueden mandar son **solo las que se encuentran en formato .png**.

[4] Pueden usar de referencia sistemas similares a este... que son solo viles copias.

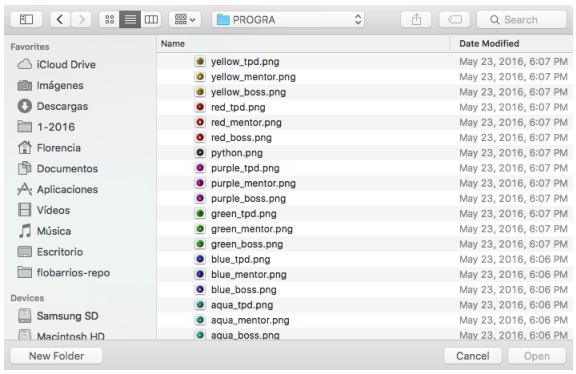


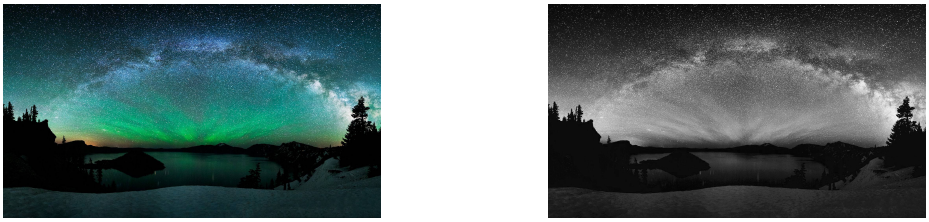
Figura 1: Ejemplo de pop-up

3.3.4. Filtros

Las imágenes pueden ser enviadas con un filtro aplicado. Deberá implementar al menos los siguientes filtros:

- **Escala de grises:**

  Para transformar de formato RGB a escala de grises, el byte de cada pixel en escala de grises debe ser el promedio de los 3 bytes de los pixeles en RGB. Por ejemplo, si RGB = (230, 10,30), deberá ser convertido a (90, 90, 90). **Nota:** Una imagen en blanco y negro tiene un tipo de colores igual a 0, en este formato cada pixel tiene 1 byte.



(a) Imagen Original (b) Imagen en Escala de Grises

Figura 2: Convertir una imagen a Escala de Grises.

- **Sepia:**

  Para aplicar el filtro de sepia sobre el formato RGB, los 3 bytes de los pixeles en RGB deben cambiar a:

```
New_Red = R*.393 + G*.769 + B*.189
New_Green = R*.349 + G*.686 + B*.168
New_Blue = R*.272 + G*.534 + B*.131
```

  Considerar a R, G, B como los valores originales de cada byte. En caso de que alguno de los outputs sobrepase los 255, este simplemente adquiere la cota superior (255).

(a) Imagen Original

(b) Imagen en Sepia

Figura 3: Convertir una imagen a Sepia.

3.3.5. Historia

Cada usuario posee una historia, donde hace de forma pública los Blinks que él desee. Estos funcionan de forma similar a enviar un Blink a un usuario, pero la diferencia radica en que si uno de los Blink-amigos del usuario entra al perfil de otro, este es capaz de ver los Blinks que están en su historia y esta puede ser vista nuevamente una cantidad infinita de veces, a diferencia de un Blink a un usuario que solo lo puede ver una única vez. Se espera que el usuario tenga la opción de agregar un Blink a su historia, directamente o al enviárselo a un/unos amigo/s.

3.3.6. Autodestrucción de Blinks

Luego de que un Blink sea visto por el receptor (esto no aplica para los Blinks vistos desde la historia de alguien), deberá autodestruirse, es decir, no podrán visualizarlos nuevamente después del tiempo fijado por el emisor. Además, toda la información asociada al Blink deberá ser eliminada del cliente del usuario, con el fin de que no pueda volver a ser descargada ni utilizada.

3.4. BSN

No puede existir BlinkChat sin un chat[5], por lo que además de todo lo anterior, deberá implementar un sistema de mensajería para poder mantener conversaciones privadas con sus Blink-amigos. En algún lugar de la interfaz principal debe darse la opción de entrar al chat, donde salen tus amigos y al seleccionar uno puedes ver tu historial de conversación y seguir enviando mensajes.

3.4.1. Decorando el chat

Además de poder escribir, el usuario debe ser capaz de agregarle a su mensaje emojis (al menos 15 emojis diferentes). También debe estar la opción de enviarle zumbidos a tus amigos para llamarles la atención, ya que ellos se han olvidado un poco de ti por tu condición de trabajólico (no es necesario que vibre la pantalla del otro cliente, pero sí que se entere de que alguien quiere llamar su atención).

4. Notas

- Toda la interacción del usuario con Blink Chat debe ser a través de la interfaz. Está prohibido el uso de la consola para tarea.

5. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.4

[5] No chat, no party.

- Esta tarea es estrictamente individual y está regida por el Código de Honor de la Escuela: Clickear para Leer.
- Su código debe seguir la guía de estilos `PEP8`
- Si no se encuentra especificado en el enunciado asuma que el uso de cualquier librería Python está prohibido. Pregunte por foro si se pueden usar librerías específicas.
- El ayudante puede castigar el puntaje[6] de tu tarea si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación de algoritmos.
- Debe adjuntar un archivo `README.md` donde se comenten los alcances y el funcionamiento de su sistema (*i.e.* manual de usuario) de forma *concisa* y *clara*.
- Cree un módulo para cada conjunto de clases. Divídalas por las relaciones y los tipos que poseen en común.
- Cualquier aspecto no especificado queda a su criterio, siempre que no pase por sobre otro.

6. Entrega

- **Fecha/hora:** 17 de Junio - 23:59 hrs
- **Lugar:** GIT - Carpeta: Tareas/T06

Tareas que no cumplan con las restricciones señaladas en este enunciado tendrán la calificación mínima (1.0).

[6] Hasta −5 décimas.