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

IIC2233 - Programación Avanzada (II/2015)
Tarea 6

1. Objetivos

- Aplicar conceptos y nociones de networking para la creación de un servicio en línea.
- Aplicar conceptos del serialización para el manejo de envío de archivos.

2. Introducción

En una universidad no muy lejana se dicta un curso llamado Programación Avanzada, que nada tiene que ver con el curso que usted tiene (para que no se sienta aludido ni ofendido con lo que viene a continuación) en el cual a pesar de las grandes capacidades para programar que tienen sus alumnos, no han logrado poder dominar el misterioso sistema de control de versiones para el desarrollo de software llamado Jit. Después de mucho discutir, los ayudantes de este ramo convinieron que pocos serían capaces de aprender a usarlo de manera correcta. En consecuencia decidieron crear su propio sistema para almacenar datos, haciendo algo con lo que los alumnos se sientan más familiarizados.

Dado que en este cuerpo de ayudantes están todos muy ocupados haciendo `add`, `commit`, `push`, `pull`, `rebase` y `merge`, le han pedido ayuda directamente a ~~los alumnos del ramo~~ usted (que nada tiene que ver con esos alumnos y con ese ramo). Sus amplios conocimientos de networking y de serialización de archivos, debiesen permitirle programar "DrobPox"[1], un programa para subir y compartir archivos en la nube, mediante el uso de una carpeta en su computador. Además el sistema debe permitir mantener una conversación mediante un chat con otras personas que se decida agregar como amigos. De este programa dependen las futuras generaciones de ~~IIC2233~~ aquel ramo de una universidad no muy lejana.

[1] Esta página tiene un comportamiento extrañamente similar.

3. Problema

En esta tarea usted deberá implementar su propia plataforma de subido y compartido de documentos de todo tipo, para uno y muchos usuarios, aplicando conceptos de networking y de serialización. Para esto usted deberá crear dos cosas: un servidor que correrá en su computador y manejará la lógica del sistema y un cliente que podrá correr en cualquier computador que permitirá la interacción de un usuario con el sistema. El programa deberá manejar las siguientes necesidades (las cuales se detallarán con más precisión en las siguientes secciones del enunciado.)

3.1. Manejo de Usuarios

El programa debe ser capaz de manejar una base de distintos usuarios, cada uno con un username que lo identifique únicamente y una clave personal. Su programa debe permitir la creación de una cantidad indefinida de usuarios, los cuales podrán conectarse a su servicio desde cualquier computador. Es importante considerar que todo servicio detallado más adelante requiere que un usuario se encuentre conectado.

3.2. Manejo de Carpetas y Archivos

Cada usuario podrá agregar carpetas y archivos ya existentes en su computador a su "DrobPox". Una vez agregados, estos se deben subir al sistema y el usuario podrá acceder a ellos y descargarlos desde cualquier otro computador en el cual se conecte. Además de esto, después de agregada una carpeta, todo cambio en ella deberá actualizarse en el sistema, guardando información horaria acerca de los cambios. Estas carpetas pueden ser privadas o compartidas entre varios usuarios del sistema. Cualquier cambio realizado en una carpeta compartida, será visualizado por todos los integrantes.

3.3. Chat

El programa debe permitir una interfaz simple de chat, a través de la cual se podrá hablar con otros usuarios del sistema. Este chat permitirá enviar texto plano, algunos Emojis y también cualquier archivo que ya se encuentre en el sistema.

4. Especificaciones de Manejo de Usuarios

Al correr el cliente en algún computador, este debiese mostrar una página de LogIn, en la cual se ofrecerá conectarse o crear un usuario nuevo.

4.1. Creación de usuario

Le deberá preguntar al usuario por un username y una contraseña, las cuales utilizará de ahora en adelante para acceder a este sistema. Es importante considerar que este username debe ser único y no podrá ser usado por otros usuarios.

Esta información debe ser guardada de forma *segura* en el servidor. Esto implica que usted, *bajo ningún motivo* guardará la contraseña del usuario. Para esto se recomienda que observe el protocolo HASH + SALT[2]. Para esto deberá usar la librería `hashlib` que viene incluida en Python.

4.2. Log In

Una vez creado el usuario, este podrá conectarse al sistema si es que ingresa correctamente su username y su contraseña.

[2]Puede leer más acerca de esto aquí

5. Especificaciones de Manejo de Carpetas y Archivos

Su programa debe ser capaz de manejar el subido y bajado de archivos de la siguiente forma.

5.1. Agregar Archivos

El cliente debiese permitirle al usuario seleccionar cualquier archivo de su computador para agregarlo al sistema. Una vez seleccionado, este archivo debiese enviarse al servidor, donde se guardará de alguna forma que usted lo decida.

Cuando ya se encuentre en el sistema, la interfaz debiese mostrarle al usuario cuáles son los archivos que se encuentran online.

5.2. Agregar Carpetas

Cuando algún usuario desee agregar alguna carpeta "DrobPox", este deberá ser capaz de seleccionar una carpeta *ya existente* en su PC. Esta, junto con todos sus contenidos, será subida al sistema. Además de esto, esta carpeta quedará "observada" por el programa (se detallará a continuación).

El programa deberá mostrar visualmente todas las carpetas que el usuario tiene online. Para esto basta que se vea el árbol de las carpetas y sus contenidos, pero no es necesario que se puedan abrir los archivos.

5.3. Observar una Carpeta

Tal como se dijo anteriormente, una carpeta agregada al sistema será observada por su programa. Esto quiere decir que cualquier cambio que se haga a esta carpeta de ahora en adelante será reportado al servidor del programa. Es decir que si un usuario decide modificar el archivo `texto_plano.txt` que se encuentra en una carpeta que ha sido agregada al sistema, ese archivo debiese actualizarse en el servidor. Los cambios a observar dentro de la carpeta son los siguientes:

1. Modificación del contenido de un archivo
2. Creación de un nuevo archivo
3. Eliminación de un archivo
4. Renombrar un archivo
5. Creación de una subcarpeta
6. Eliminar una subcarpeta
7. Renombrar una subcarpeta
8. Reubicación de un archivo o una subcarpeta

En caso de que un archivo sea modificado cuando el programa no esté corriendo, este cambio deberá ser notificado lo antes posible, es decir la próxima vez que el cliente sea ejecutado en ese computador.

NO es necesario que este chequeo se haga constantemente, basta con que usted agregue un botón a su cliente que permita realizar este chequeo. Es decir, cada vez que yo presione ese botón, su programa debiese revisar las carpetas "observadas" para encontrar los cambios realizados y notificarlos al servidor.

5.4. Descargar Archivos

Un usuario podrá descargar sus archivos desde cualquier computador. Para esto, el cliente debe permitir al usuario recorrer las carpetas que tiene online, seleccionar un archivo y descargarlo.

5.5. Descargar Carpeta

Además de descargar archivos por sí solos, un usuario deberá ser capaz de descargar una carpeta completa a cualquier computador. Se le deberá preguntar al usuario dónde desea descargar esta carpeta y el programá procederá a crear las carpetas necesarias ya descargar todos los archivos pertinentes. Una vez descargada la carpeta, esta también quedará "observada" por su programa.

5.6. Actualización de Carpetas

Ya que las carpetas pueden estar en varios computadores, su programa debe ser capaz de actualizar los cambios en todas las carpetas correspondientes para mantenerlas al día con el estado del servidor. Por ejemplo, un usuario agrega una carpeta desde un computador A, luego procede a descargarla en un computador B, donde agrega el archivo `archivo_nuevo.py` y notifica al servidor los cambios. La próxima vez que este usuario corra el cliente de "DrobPox" en el computador A, este debiese actualizar los cambios y descargar el archivo nuevo a la carpeta original.

5.7. Compartir Carpetas

Tal como se mencionó antes, las carpetas pueden ser privadas o compartidas. Al usuario que subió la carpeta se le permitirá compartirla con otros usuarios del sistema. Una vez compartida, estos usuarios debiesen ver esta carpeta dentro de su "bandeja de inicio" en el cliente. Esto les debiese dar la posibilidad de descargar los archivos o la carpeta completa. Estas carpetas se rigen bajo el mismo comportamiento nombrado en el punto anterior, es decir que si dos usuarios tienen acceso a una carpeta compartida, y uno de ellos decide hacer modificaciones en su copia local, estos debiesen ser notificados al sistema y en consecuencia, actualizados en la copia local del otro usuario.

5.8. Historial

Su programa deberá manejar el historial de modificaciones de los archivos y carpetas nombrados en el punto **5.3**. Para esto, si un usuario selecciona un archivo o carpeta dentro de la interfaz del cliente, este deberá darle la opción de ver el historial de modificaciones. Por ejemplo el siguiente es el historial de la carpeta `main_directory`:

```
2015-11-06 10:56:22 - ADDED   'main_directory/asdasd.py' by user1
2015-11-06 10:56:22 - ADDED   'main_directory/sub_dir1' by user1
2015-11-06 10:56:22 - ADDED   'main_directory/inner' by user1
2015-11-06 10:56:22 - ADDED   'main_directory/sub_dir1/sub_dir2' by user1
2015-11-06 11:43:46 - ADDED   'main_directory/inner/hola.txt' by user2
2015-11-06 11:43:46 - ADDED   'main_directory/sub_dir1/sub_dir2/prueba.cs' by user2
2015-11-06 17:26:34 - RENAMED 'main_directory/asdasd.py' to 'wee.py' by user1
2015-11-06 17:26:34 - RENAMED 'main_directory/sub_dir1' to 'waa' by user1
2015-11-06 18:57:09 - REMOVED 'main_directory/waa/sub_dir2' by user2
2015-11-06 18:57:09 - REMOVED 'main_directory/waa/sub_dir2/prueba.cs' by user2
```

5.9. Manejo de Inconsistencias

Puede ocurrir que se realicen dos modificaciones por separado a un mismo archivo, lo que generará problemas de compatibilidad en futuras actualizaciones de la carpeta. Queda a su criterio cómo maneja este problema, siempre y cuando sea algo que tenga sentido y lo especifíque en el ReadMe.

Una manera simple de manejarlo es que cuando ocurran situaciones de este estilo, usted simplemente duplique el archivo y deje las dos versiones, así los usuarios podrán decidir cómo arreglar el problema. Por

ejemplo, si tanto el usuario A como el usuario B modifican el archivo `prueba.py`, el sistema debiese mostar los archivos `prueba.py.VERSION1` y `prueba.py.VERSION2` en la siguiente actualización.

6. Especificaciones del Chat

Se podrán crear chats de dos participantes dentro del sistema. Para esto la interfaz deberá permitir al usuario buscar alguna otra persona del sistema y abrir una ventana de conversación. Esta chat debe permitir las siguientes funcionalidades

6.1. Mensajería instantánea

Los usuarios podrán escribir mensajes en texto plano que se enviarán instantáneamente al otro participante.

6.2. Envío de Archivos

El sistema deberá permitirle al usuario enviar alguno de sus archivos que ya se encuentren online y en el sistema. Es decir, si un usuario ya tiene en "DrobPox" una carpeta con sus tareas de Programación Avanzada, este debiese ser capaz de seleccionar desde la interfaz alguno de esos archivos y mandarlos por la conversación. El otro participante debe ser capaz de aceptar/rechazar este envío y en caso de aceptarlo, seleccionar dónde quiere descargar el archivo.

6.3. Emojis

El chat debe dar soporte de emoticones en la conversación. Al escribir ciertas palabras clave en la conversación y enviarlas el programa deberá reemplazarlos por emoticones. Deben soportar como mínimo las siguientes palabras: `:)`, `:(`, `:o`. Quedan a su criterio los emoticones a utilizar.

6.4. Historial

Todo mensaje enviado por el chat entre dos usuarios será guardado en el servidor. Esto es para que la siguiente vez que uno de estos usuarios abra el chat, sea capaz de leer los mensajes enviados previamente. Este historial no tiene porqué manejar los archivos enviados, pues estos podrían haber sido modificados en el intertanto.

7. Funcionamiento

Su sistema deberá ser capaz de funcionar entre computadores que estén conectados a una misma red local. Para esto es necesario que maneje de forma correcta los puertos y el host asignado a sus sockets. Para que esto funcione correctamente, y para que los ayudantes después sean capaces de corregir la tarea, **deberá seguir los siguientes pasos**.

7.1. Estructura de Archivos

El cliente y el servidor deben ser dos programas de Python que funcionen de forma totalmente independiente. Esto no implica que no pueda tener módulos en común que importe desde ambos programas. Lo importante es que el Cliente esté construido de tal forma que se pueda ejecutar en otro computador sin la necesidad de tener además el programa del servidor. Esto permitirá que se ejecute su programa desde varios computadores a la vez. Es de suma importancia que especifique en el ReadMe y deje bien claro qué archivos son los que se deben tener en un computador para que el Cliente corra bien.

7.2. Servidor

Este programa no debiese necesitar una interfaz gráfica, pues este solo manejará la lógica del sistema y se correrá en un único computador siempre. En este computador es donde usted guardará los archivos "subidos" al sistema, la base de datos de los usuarios y todo lo que usted encuentre pertinente. También es de suma importancia que en este archivo usted escriba las siguientes variables en las primeras líneas: `HOST` y `PORT` de tal forma que cuando el ayudante corrija su tarea, pueda escribir su propia IP y el puerto que desee utilizar como servidor. Estas son las variables que usted debiese usar en el método `bind` del socket del servidor.

Para poder realizar una conexión por red local usted solo debe hacer dos cosas

1. Estar conectado a una red.
2. Setear la variable `HOST` de su servidor como la IP local de su computador. Distintos sistemas operativos tienen distintas formas de obetenerla, pero en todos es un proceso muy simple.

7.3. Cliente

Este programa debiese incluir la interfaz gráfica de "DrobPox", pues este será el programa que ejecutarán los usuarios para poder utilizar su sistema. Es importante notar que este programa no servirá de nada si es que su servidor no se está ejecutando paralelamente. Este programa debiese ser autosuficiente, de tal forma que si se lo envía a un amigo que está conectado a la misma red que su computador, y él lo ejecuta, podrá probar su sistema.

Este archivo también debe tener las variables `HOST` y `PORT` en un lugar de fácil acceso. Estas debiesen tomar el mismo valor que las del servidor nombrado en el punto anterior. Estas debiesen ser usadas en el método `connect` del socket del cliente. La idea es que el ayudante pueda modificar estas de forma expedita al momento de corregir su tarea.

8. Bonus!

Esta tarea poseerá bonus porcentuales con respecto a la nota obtenida. Es importante notar que el máximo bonus posible es de 15 %. Usted podrá elegir la combinación que usted quiera, tomando en cuenta que el porcentaje adicional será acotado.

8.1. Manejo de Versiones - 8 %

El sistema deberá permitir volver atrás hasta 5 versiones en los cambios de cualquier tipo. Es decir, dado un historial de una carpeta o archivo, el usuario podrá seleccionar cualquiera de los últimos 5 cambios y volver a esa versión.

8.2. Chat Grupal - 8 %

La funcionalidad de Chat del sistema ahora debiese permitirle a los usuarios crear chats con una cantidad indefinida de usuarios. Este debiese cumplir con todas las funcionalidades nombradas en la sección **6**.

8.3. Cliente OP - 15 %

Para obtener este bonus, se le deberá permitir al usuario realizar modificaciones a la estructura de carpetas y archivos que tenga online directamente desde la interfaz. Es decir, el usuario deberá ser capaz de seleccionar un archivo o carpeta en el cliente y borrarlo/renombrarlo/moverlo, llevando a cabo las mismas modificaciones que ocurrirían si es que usted hiciese eso en el explorador de su computador. Esto incluye ser capaz de mover subcarpetas y archivos entre distintas carpetas de "DrobPox".

Además, se le deberá permitir al usuario crear carpetas nuevas en la interfaz, ya sea dentro de otras carpetas ya existentes o como carpetas "DrobPox" nuevas.

9. Restricciones y alcances

- Tu programa debe ser desarrollado en Python 3.4
- 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[3] de tu tarea, si le parece adecuado. Se recomienda ordenar el código y ser lo más claro y eficiente posible en la creación algoritmos.
- Debe adjuntar un archivo `README.md` donde comente sus 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 encima de ningún otro.

10. Entrega

- **Fecha/hora:** 28 de Noviembre - 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).

[3] Hasta −5 décimas.