INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

REDES DE COMPUTADORAS

PROF. AXEL ERNESTO MORENO CERVANTES

GRUPO: 3CM7

PRÁCTICA 1 – GOOGLE DRIVE

ALEJANDRO DE JESÚS ZEPEDA FLORES

28 de abril de 2020

Prácticas: https://drive.google.com/drive/folders/12lUUJImgU72oXJfukVcbrB8OjSsMSkb2?usp=sharing

Práctica 1: https://drive.google.com/drive/folders/172ZM70uVCF6UIiZG5ZOsf2pEX3ms0F-s?usp=sharing

OBJETIVOS

Implementar una aplicación Cliente – Servidor utilizando Sockets de Java estilo Google Drive / Dropbox que le permita al usuario realizar las siguientes operaciones:

* Visualizar el contenido de su carpeta (Archivos / subcarpetas).
* Subir archivos y carpetas al servidor o cliente.
* Descargar archivos y carpetas del servidor o cliente.
* Crear y borrar archivos en el servidor o cliente.

INTRODUCCIÓN

Ambas formas de comunicación (UDP y TCP) utilizan la abstracción de sockets, que proporciona los puntos extremos de la comunicación entre procesos. Los sockets (conectores) están presentes en la mayoría de las versiones de UNIX, incluido Linux y también Windows.

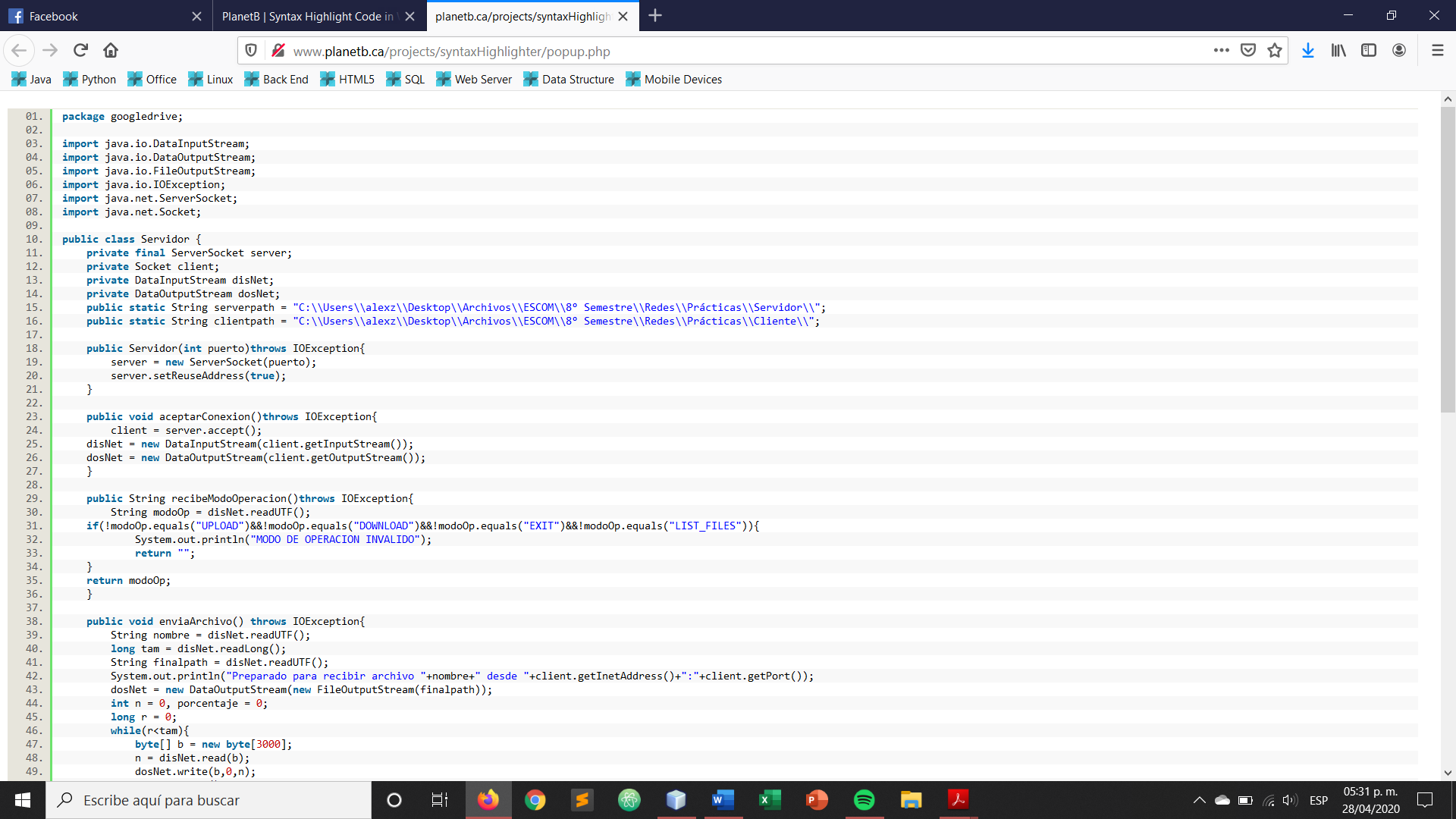
Los sockets permiten conectar dos programas en red para que se puedan intercambiar datos. Los sockets están basados en una arquitectura cliente/servidor. En esta arquitectura uno de los programas debe estar siempre arrancado y pendiente de que alguien establezca conexión con él. Este programa se denomina servidor. El otro programa lo arranca el usuario cuando lo necesita y es el programa que da el primer paso en el establecimiento de la comunicación. Este programa se llama cliente.

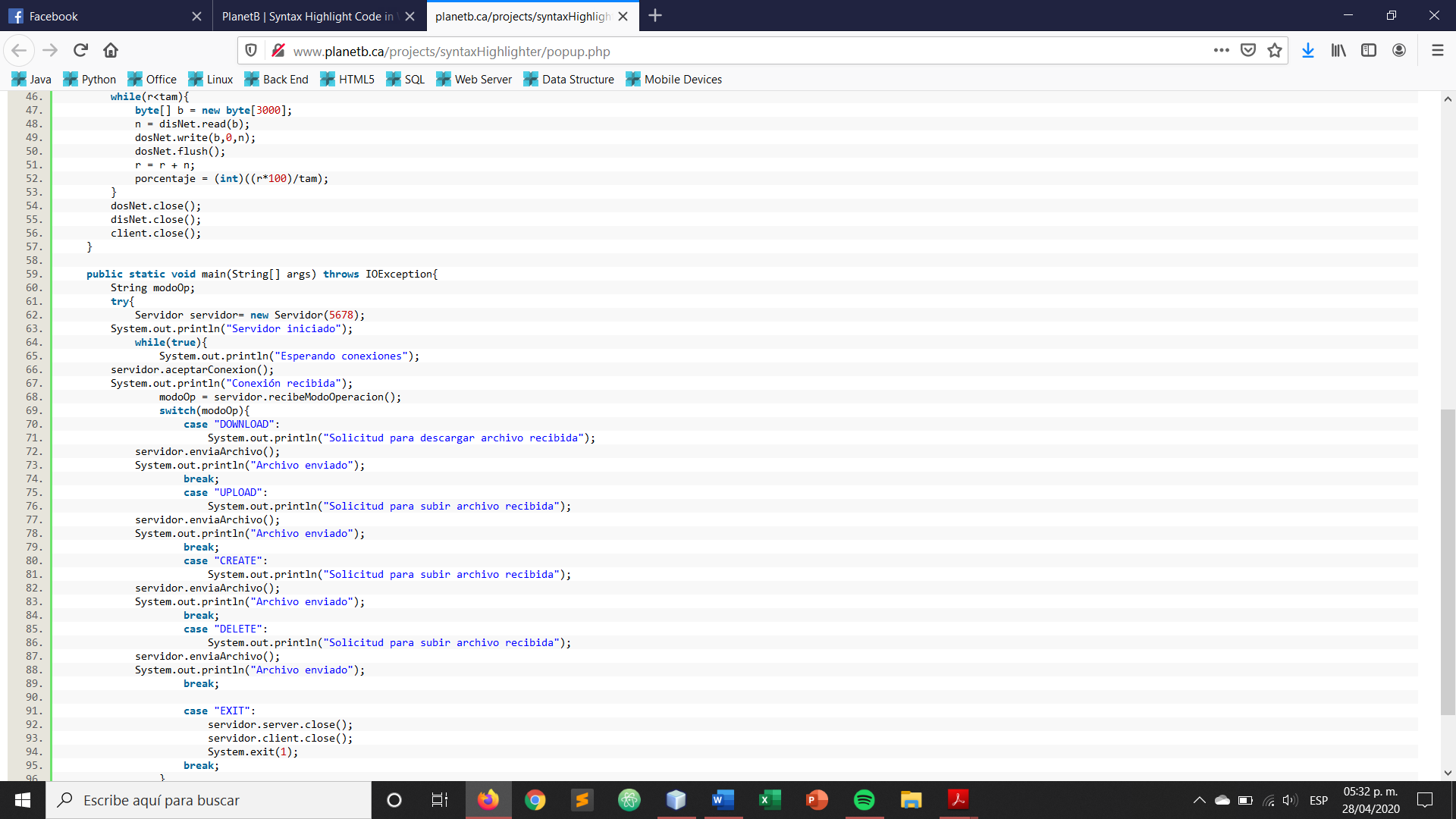
DESARROLLO

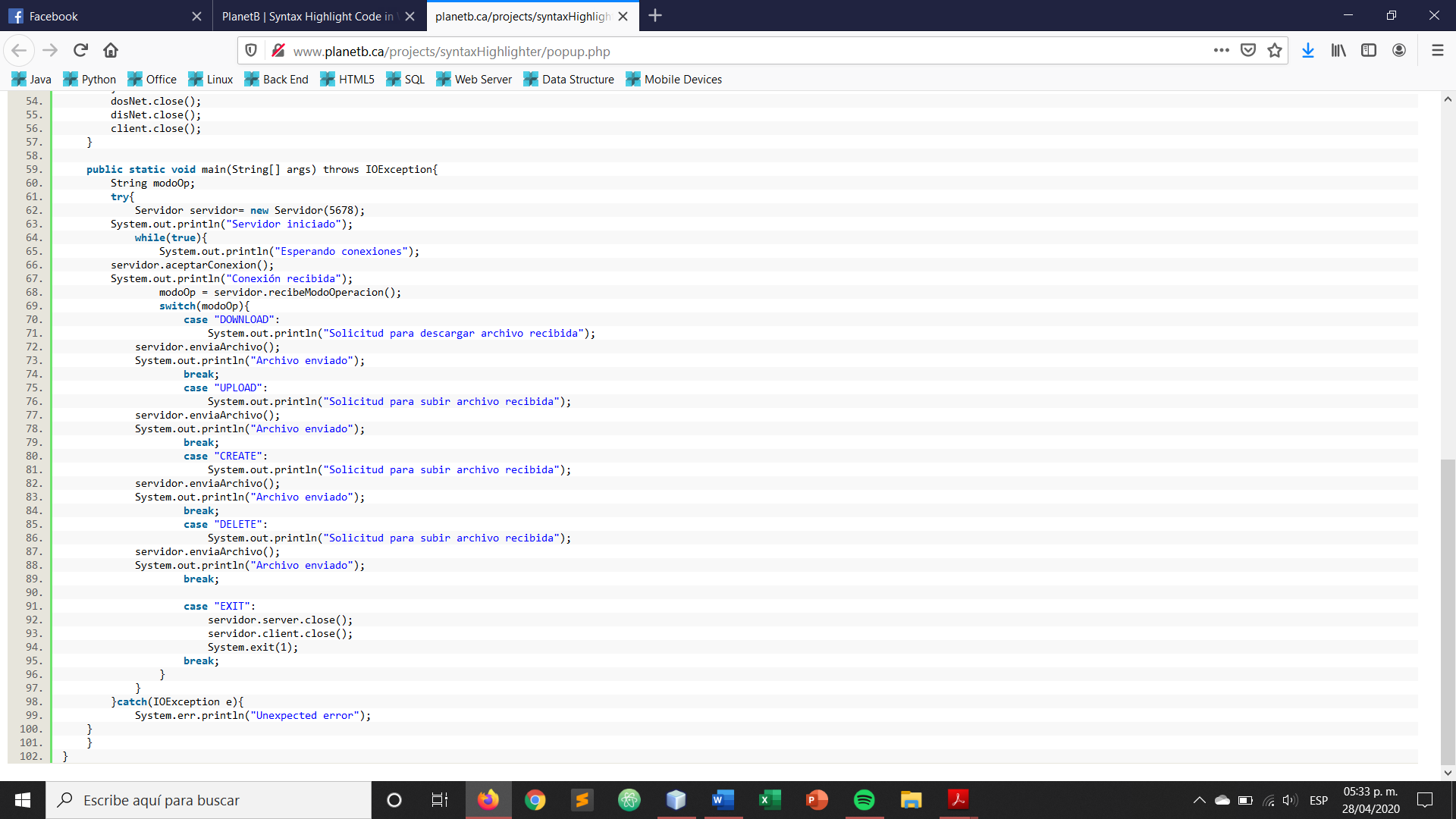
Iniciamos con la programación del Servidor, ya que este es el que estará enviando y recibiendo los datos con los clientes. Realmente las únicas funciones que se pueden observar no realizan un funcionamiento complicado.

* **Servidor(int puerto)**
  + El método **ServerSocket(puerto)** se crea un socket local al que se enlaza el puerto especificado en el parámetro puerto, si se especifica un 0 en dicho parámetro creará el socket en cualquier puerto disponible.
  + La clase **setReuseAddress()** implementa sockets de servidor. Un socket del servidor espera que entren solicitudes a través de la red. Realiza algunas operaciones en función de esa solicitud y, posiblemente, devuelve un resultado al solicitante.
* **aceptarConexión()**
  + En el método **server.accept()**se puede realizar una espera de conexión por parte del cliente mediante el método accept(). Hay que decir, que este método es de bloqueo, el proceso espera a que se realice una conexión por parte del cliente para seguir su ejecución. Una vez que se ha establecido una conexión por el cliente, este método devolverá un objeto tipo Socket, a través del cual se establecerá la comunicación con el cliente.
  + El método **getInputStream()** de la clase Java Socket devuelve una secuencia de entrada para el socket dado. Si cierra el InputStream devuelto, cerrará el socket vinculado.
  + El método **getOutputStream()** de la clase Java Socket devuelve una secuencia de salida para el socket dado. Si cierra el OutputStream devuelto, cerrará el socket vinculado.

A continuación, se muestra el código implementado en el Servidor.







Ahora, vamos a explicar la parte del cliente.

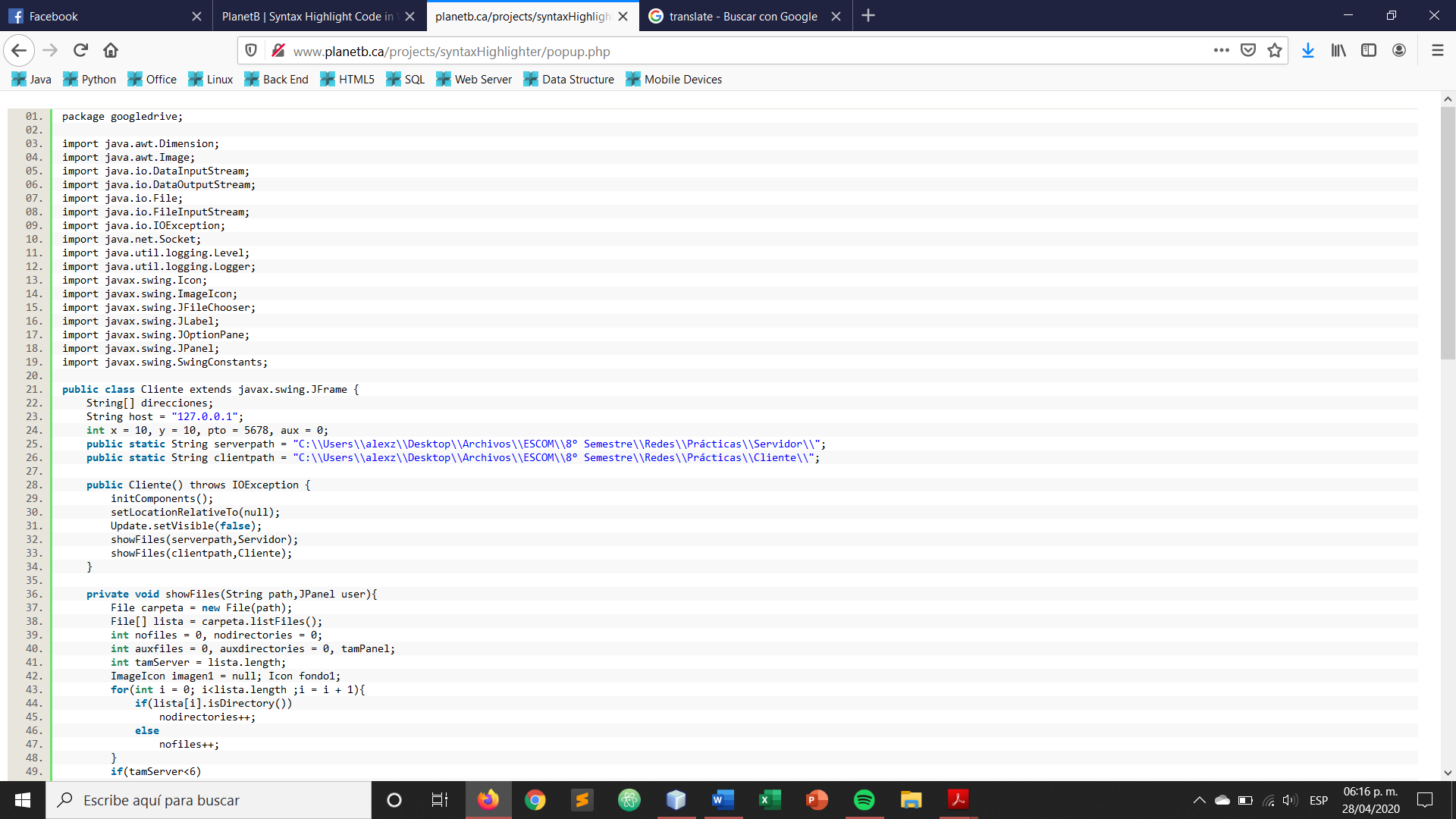
* **showFiles(String path, JPanel user):** con la intención de realizar la interfaz lo más dinámica posible, se creo la función showFiles; la cual recibe dos parámetros, la ruta de la carpeta del cliente o servidor y el panel para mostrarlo.

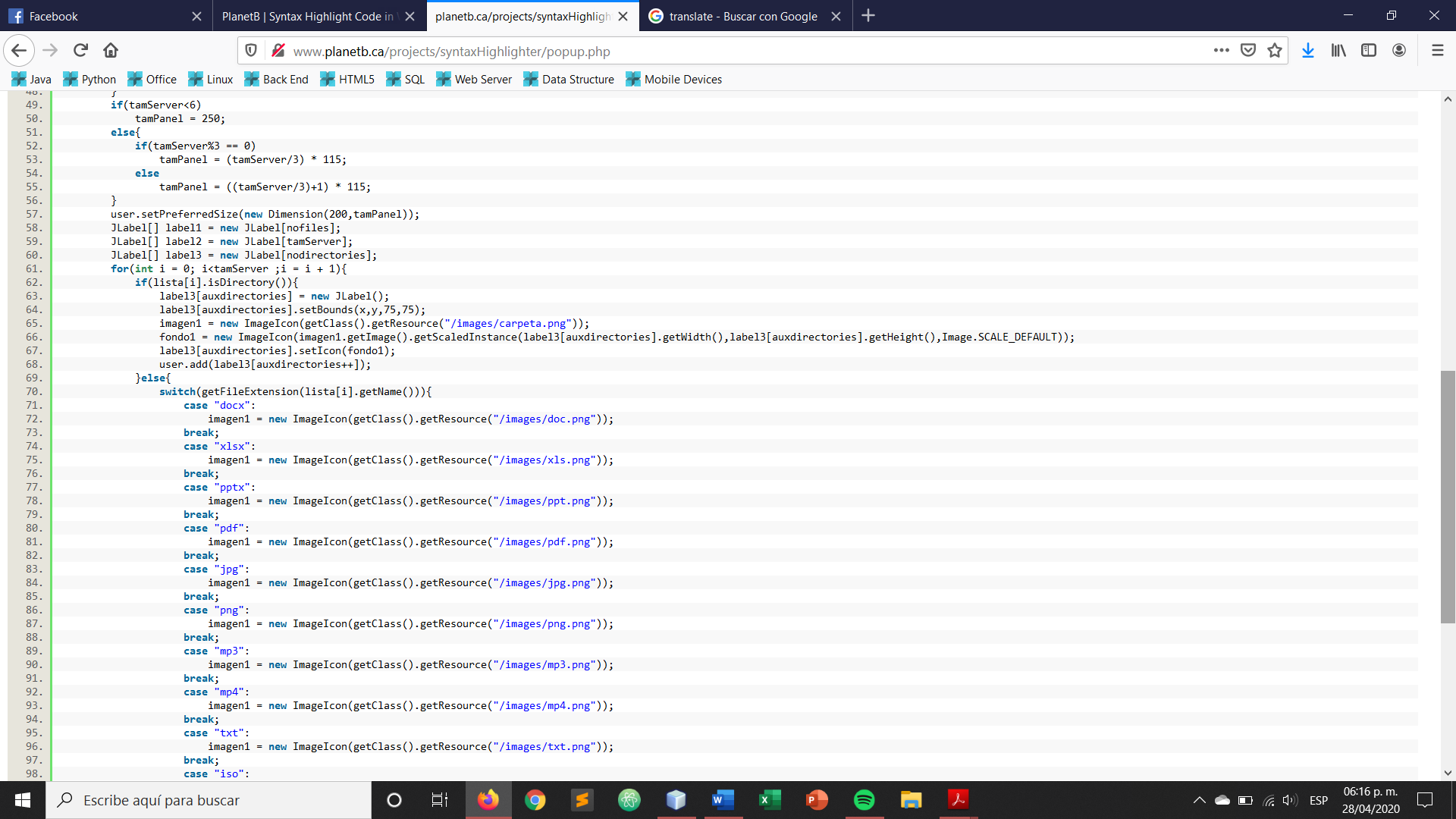
Posteriormente, almacenamos los archivos que se encuentran en dicha carpeta y hacemos un conteo, separando los archivos de las carpetas. Con ese dato, podemos conocer el espacio necesario en el panel para mostrarlos.

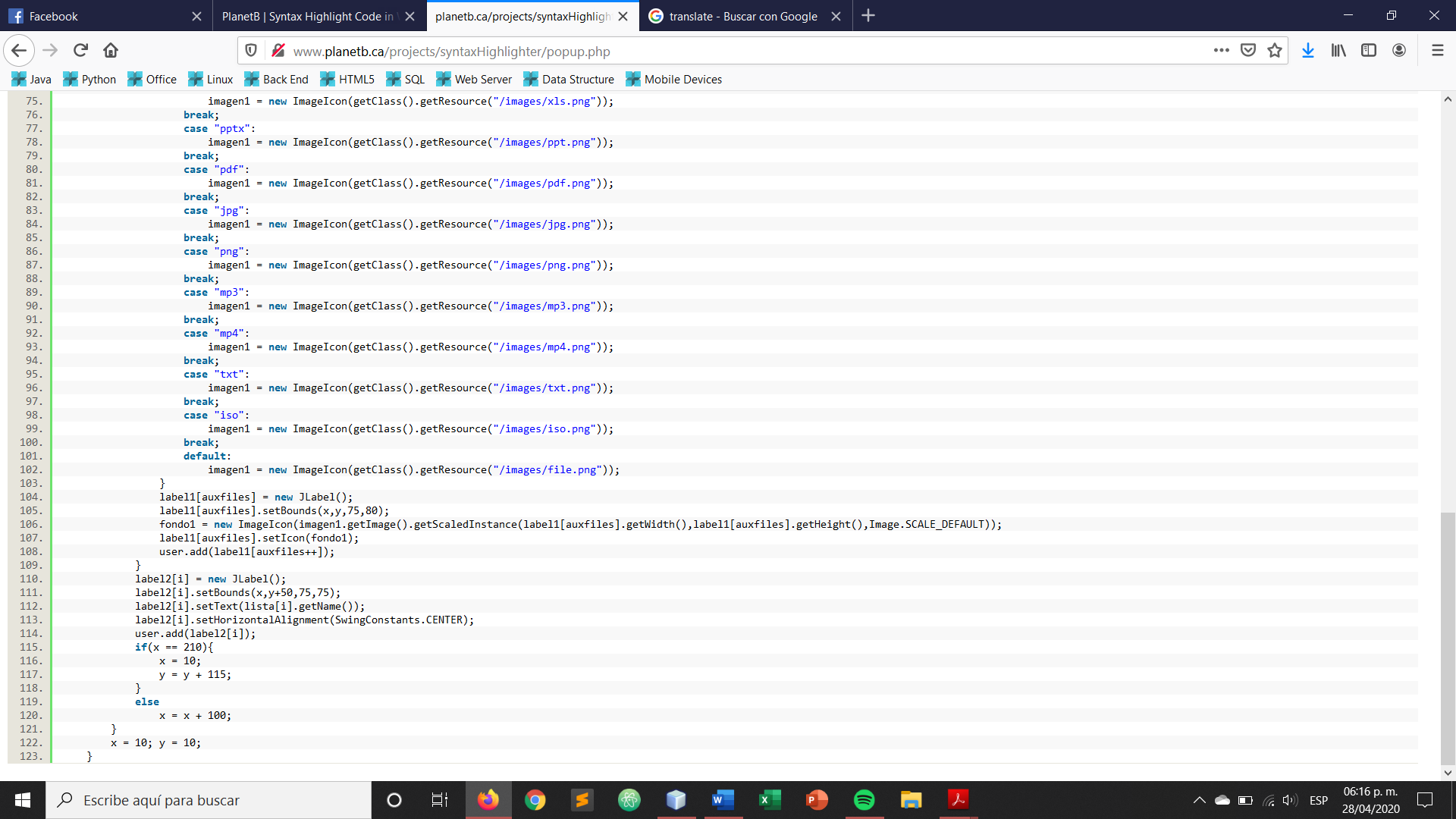
Por último, creamos arreglos de etiquetas para almacenar las imágenes y los nombres de estas, como se puede notar, añadimos el ícono correspondiente a la etiqueta clasificándola por la extensión del archivo en turno.

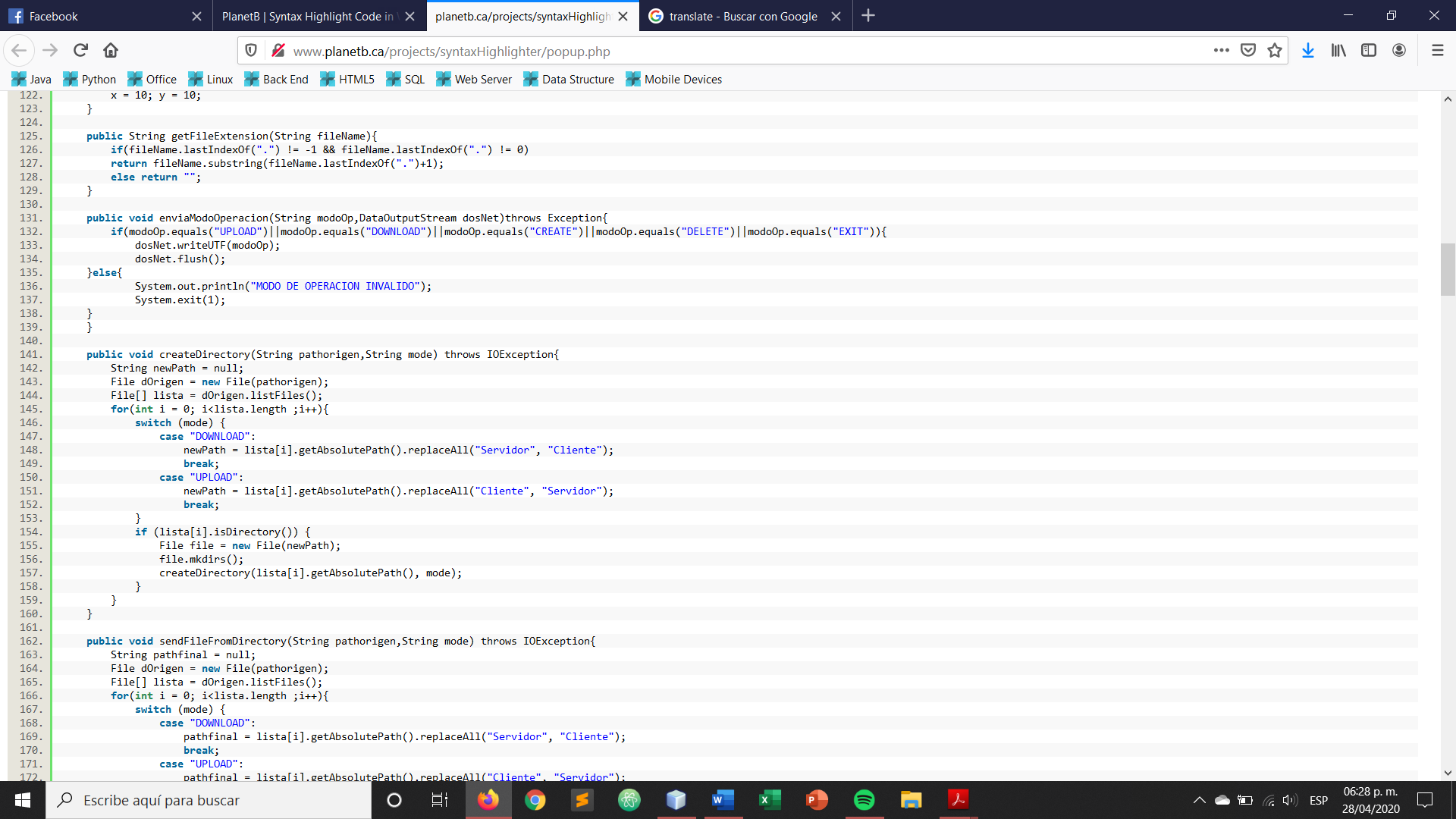
* **createDirectory(String pathorigen, String mode):** esta función tiene mucha importancia, debido a que, en el funcionamiento general de la práctica, nos permite ampliar el rango del objetivo y enviar, no solamente archivos, ahora también carpetas y en caso de que estas carpetas tuvieran subcarpetas, enviarlas del mismo modo.
* **sendFile(File file, String finalpath, String mode):** con la intención de hacerlo general y enviar n cantidad de archivos, por cada archivo se va a crear un socket cliente para conectarse con el servidor y un flujo de salida de datos. Posteriormente, partimos el archivo en bloques de bytes y empezamos a enviarlos hasta cumplir con el tamaño. Al mismo tiempo, en la ruta de destino, se están recibiendo los bloques de bytes y se está volviendo a crear.

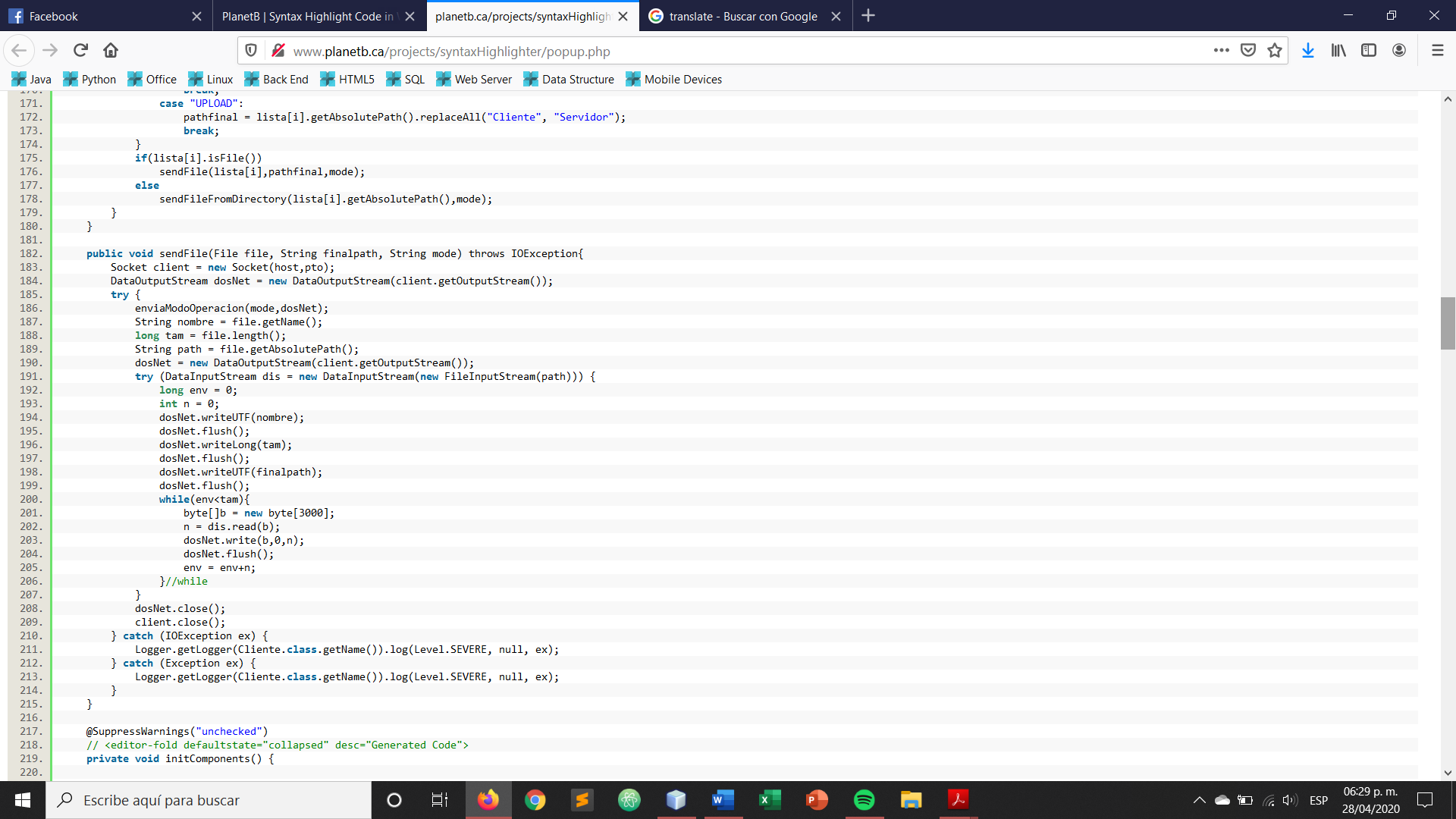
A continuación, se muestra el código del Cliente.

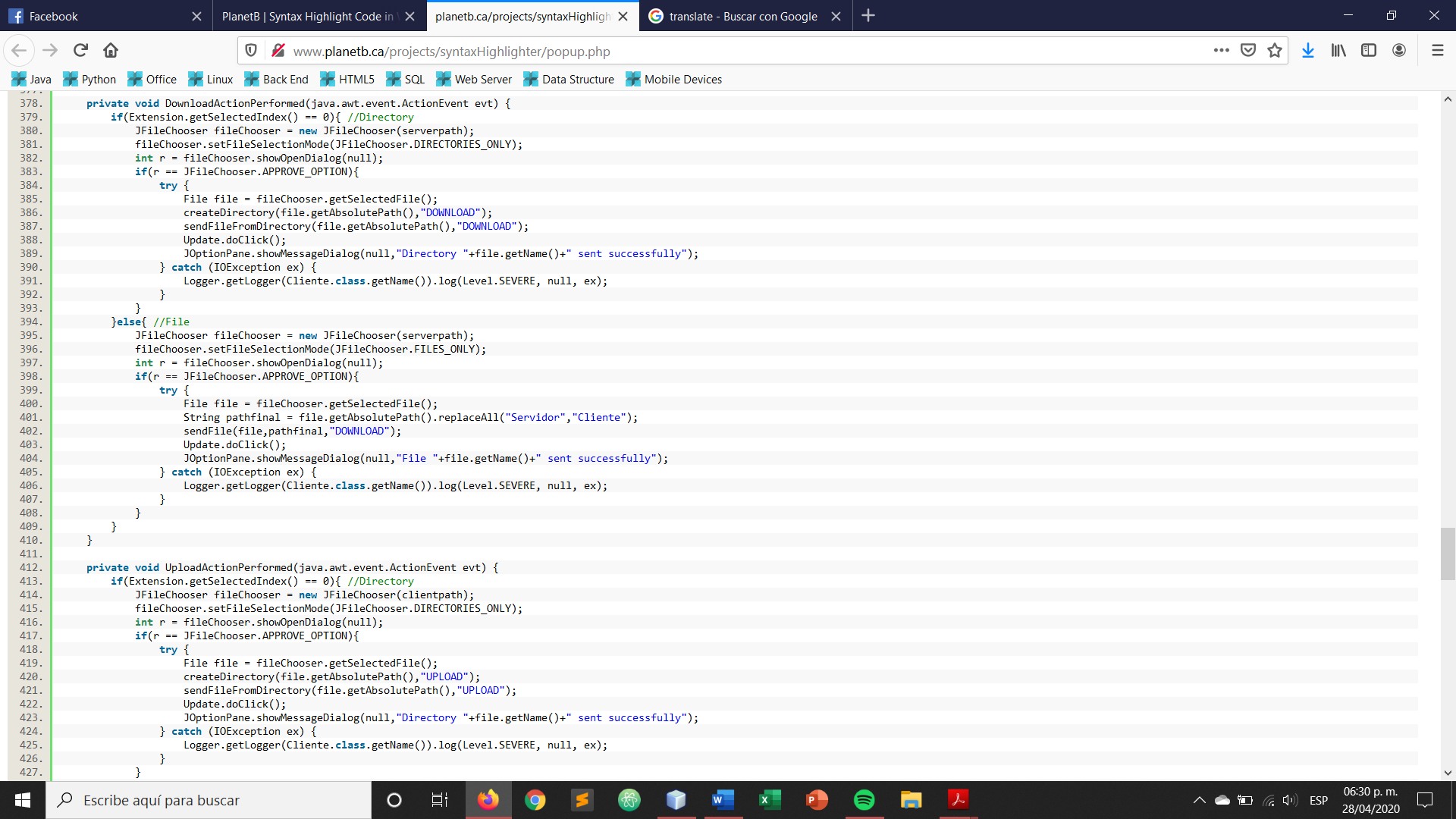


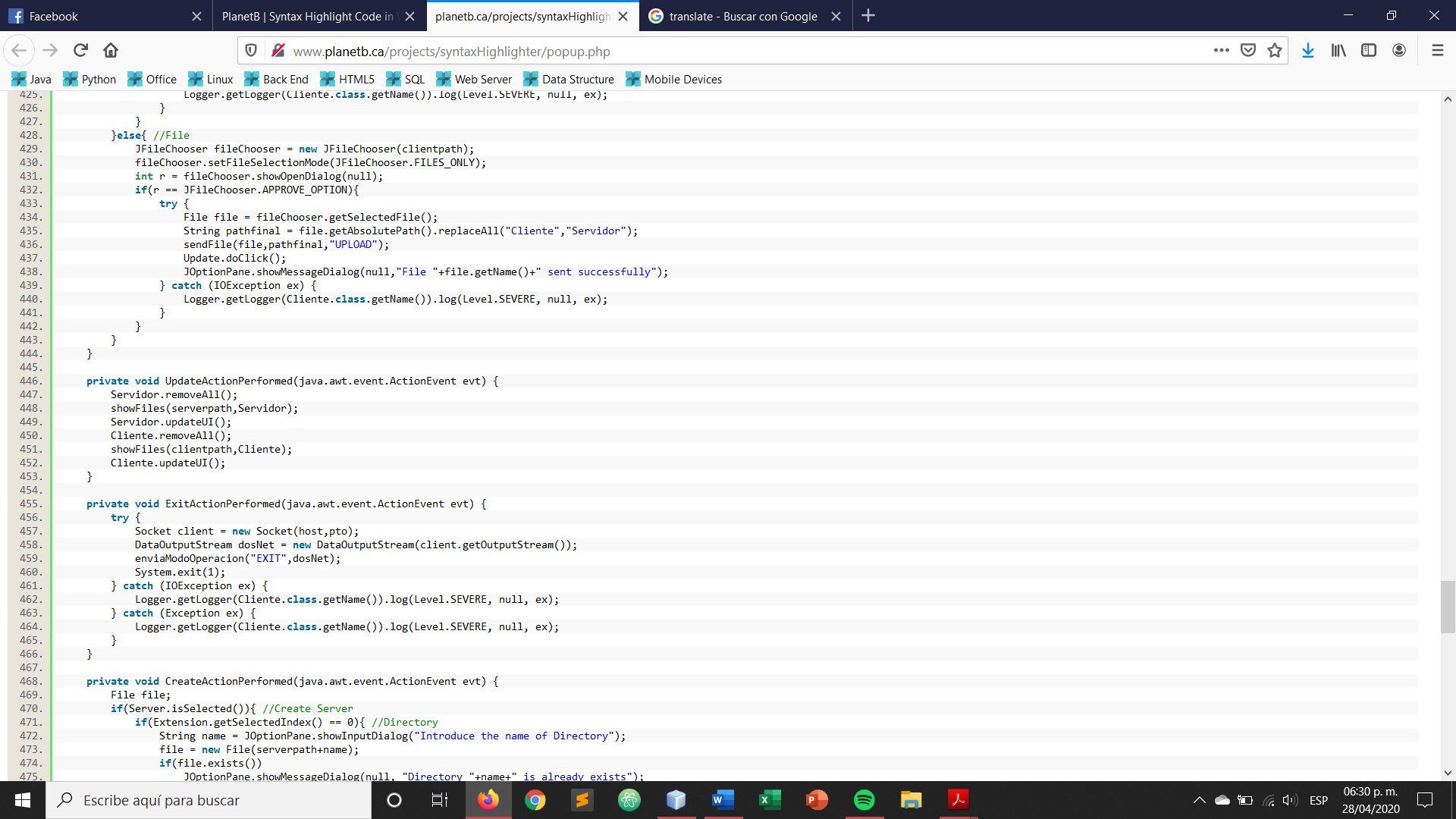


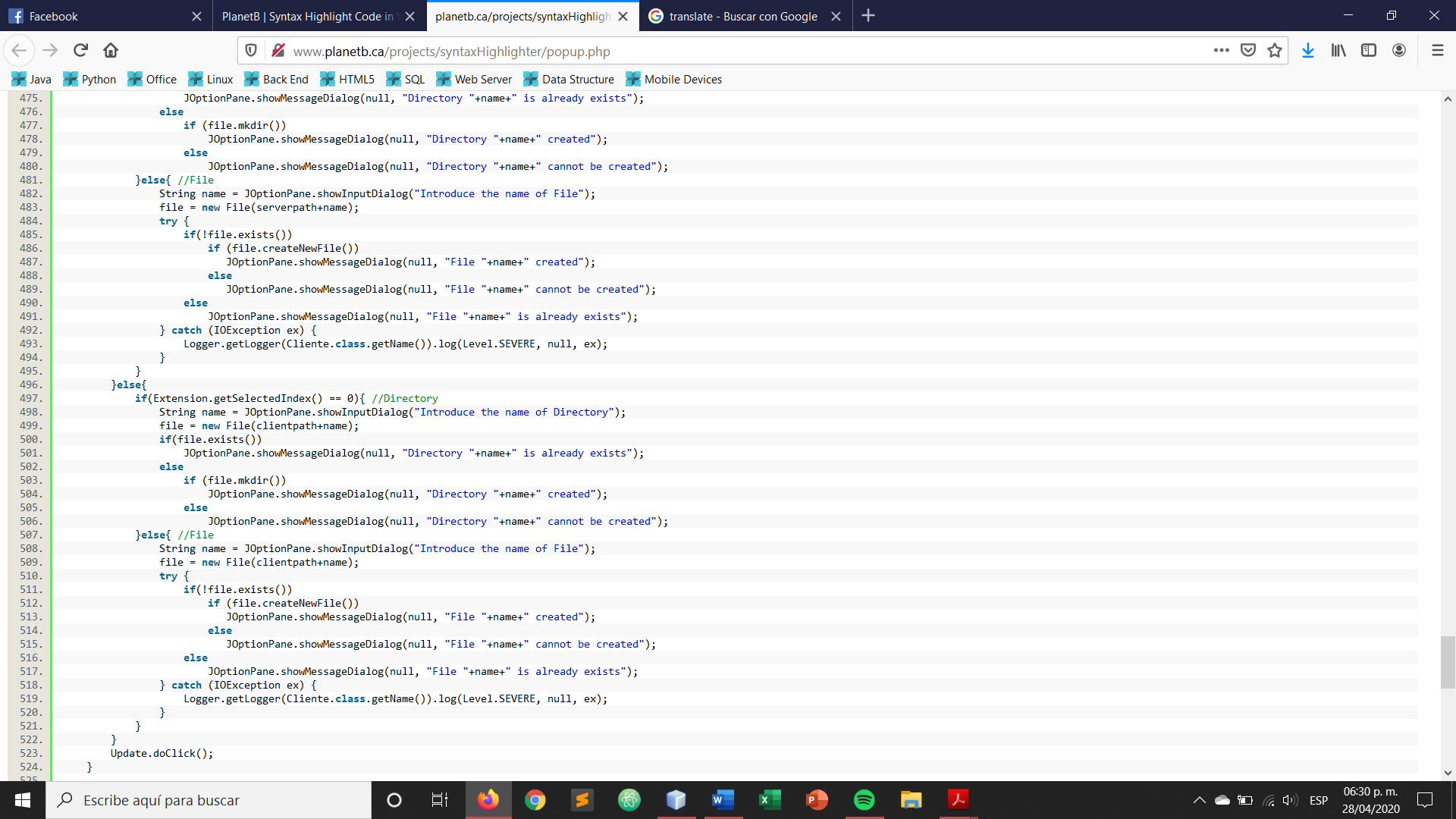


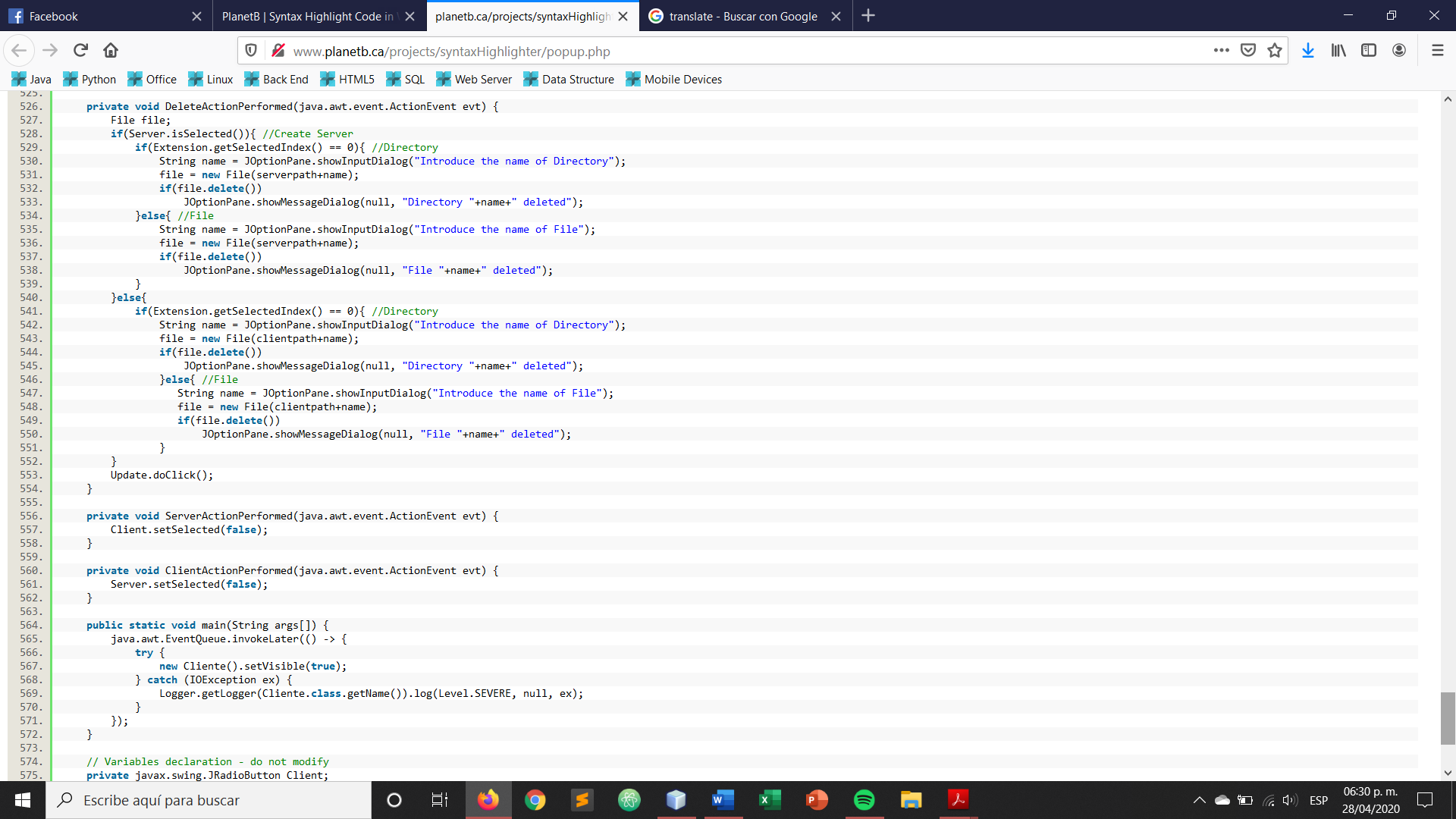












FUNCIONAMIENTO DE LA APLICACIÓN

A continuación, mostraremos un breve ejemplo de cómo funciona nuestra aplicación.

La imagen 1 muestra la interfaz principal, donde muestra dos paneles diferentes, el primero corresponde a la carpeta Servidor y el segundo a la carpeta Cliente. Se pueden observar tanto archivos como carpetas y en la parte inferior se muestran 5 botones y 1 combo box que describiremos a continuación:

1. **Combo Box:** este sirve para seleccionar si lo que se va a utilizar es un archivo o una carpeta, esto se hace con la finalidad de conocer previamente a la acción el proceso que se llevará a cabo para completarla.
2. **Download:** este botón sirve para descargar archivos que se encuentran en el servidor.
3. **Upload:** este botón sirve para subir archivos que se encuentran en el cliente.
4. **Create:** la funcionalidad de este botón es crear archivos o carpetas en ambos usuarios.
5. **Delete:** la funcionalidad de este botón es borrar archivos o carpetas en ambos usuarios.

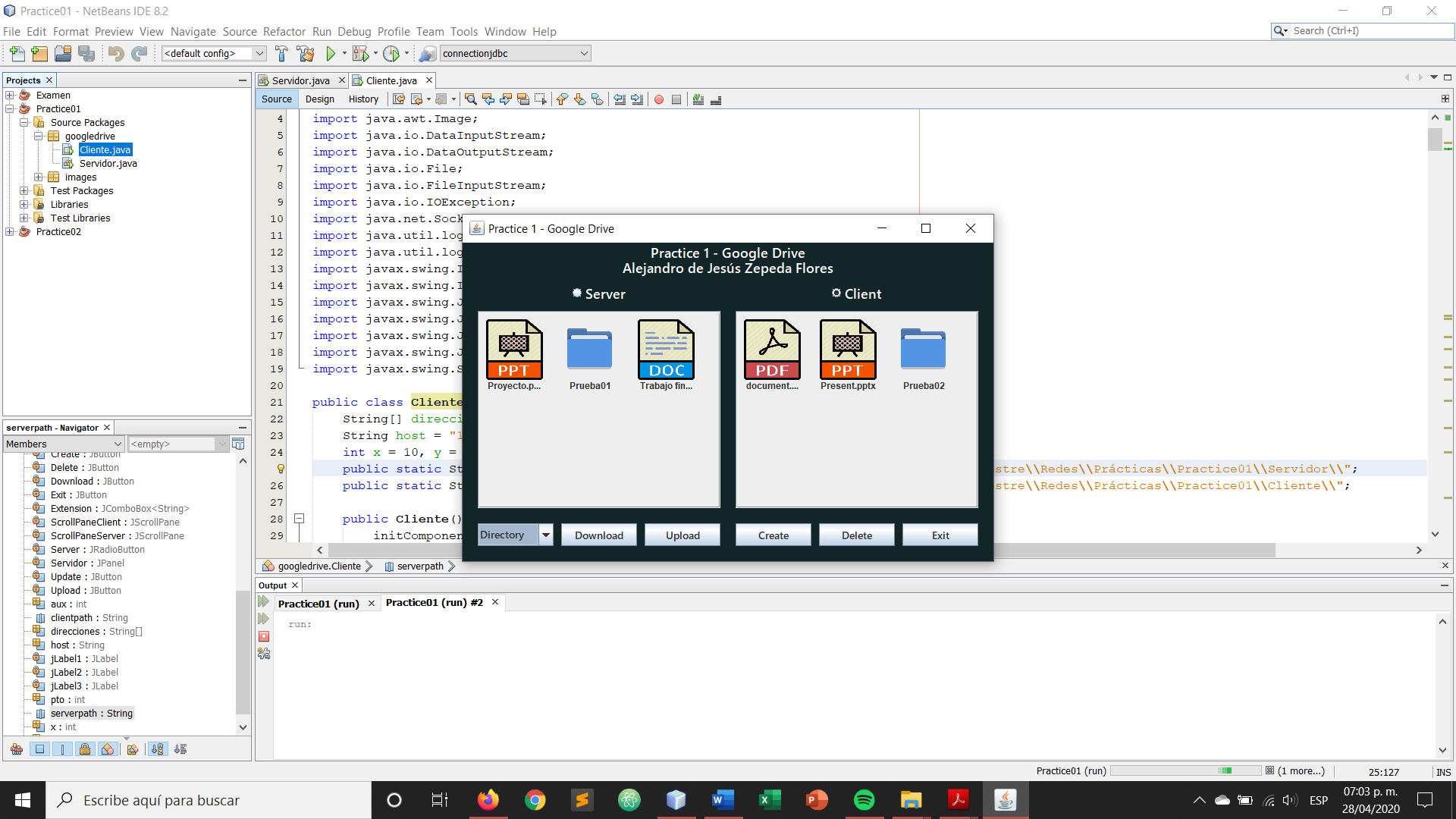


Imagen 1. Interfaz principal.

Ahora vamos a descargar un archivo que se encuentra en el servidor, como se muestra en la imagen 2, procedemos a seleccionar en el combo box la opción de “File” y se nos abre otra ventana donde nos muestra los archivos disponibles y para terminar seleccionamos “Trabajo final.docx”.

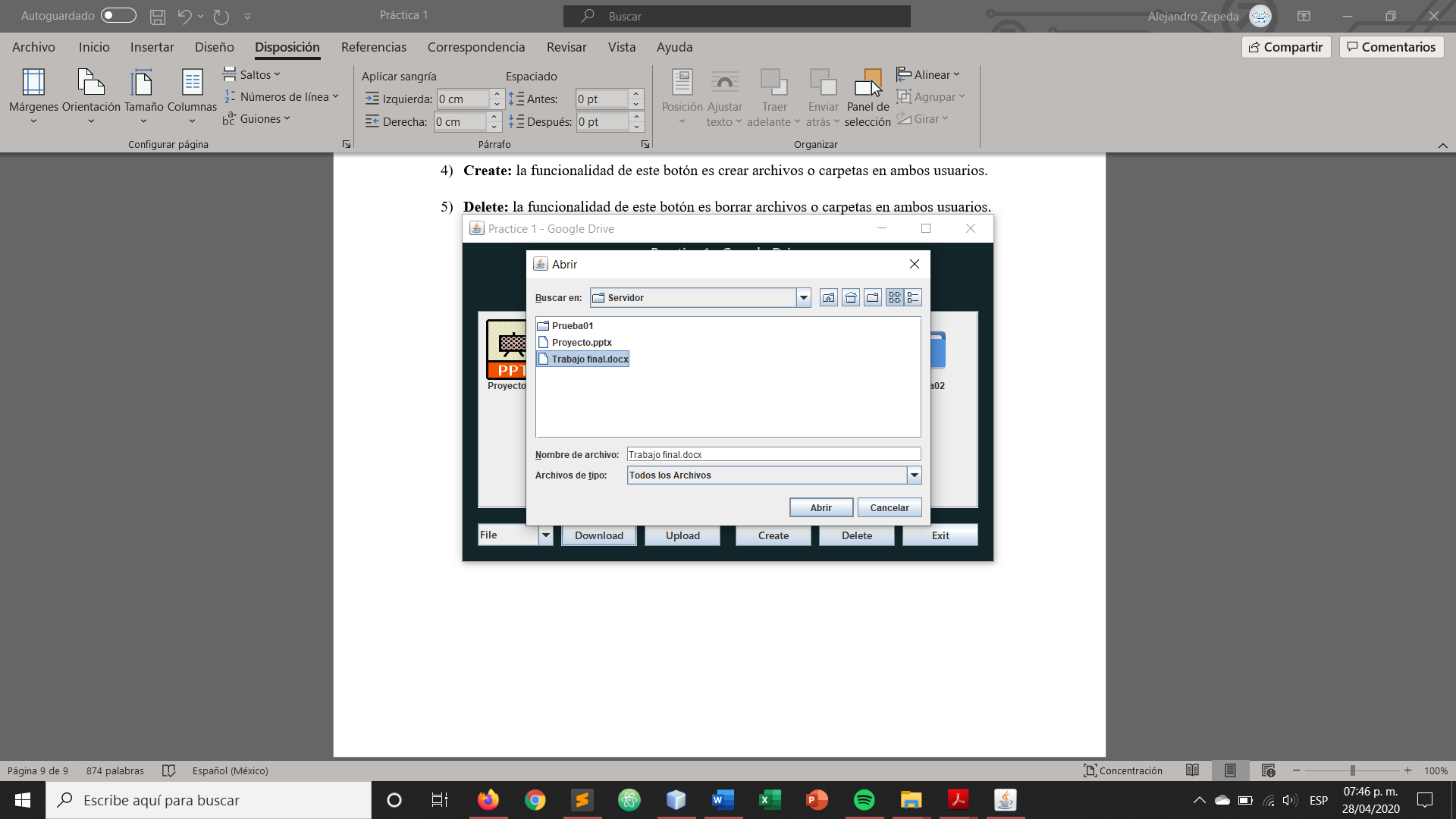


Imagen 2. Descarga de archivos.

Ahora, podemos observar que la aplicación nos muestra una ventana de confirmación de descarga de archivo; además, en la imagen 3, se ve el panel cliente donde podemos ver que ya aparece el archivo que habíamos seleccionado en el servidor

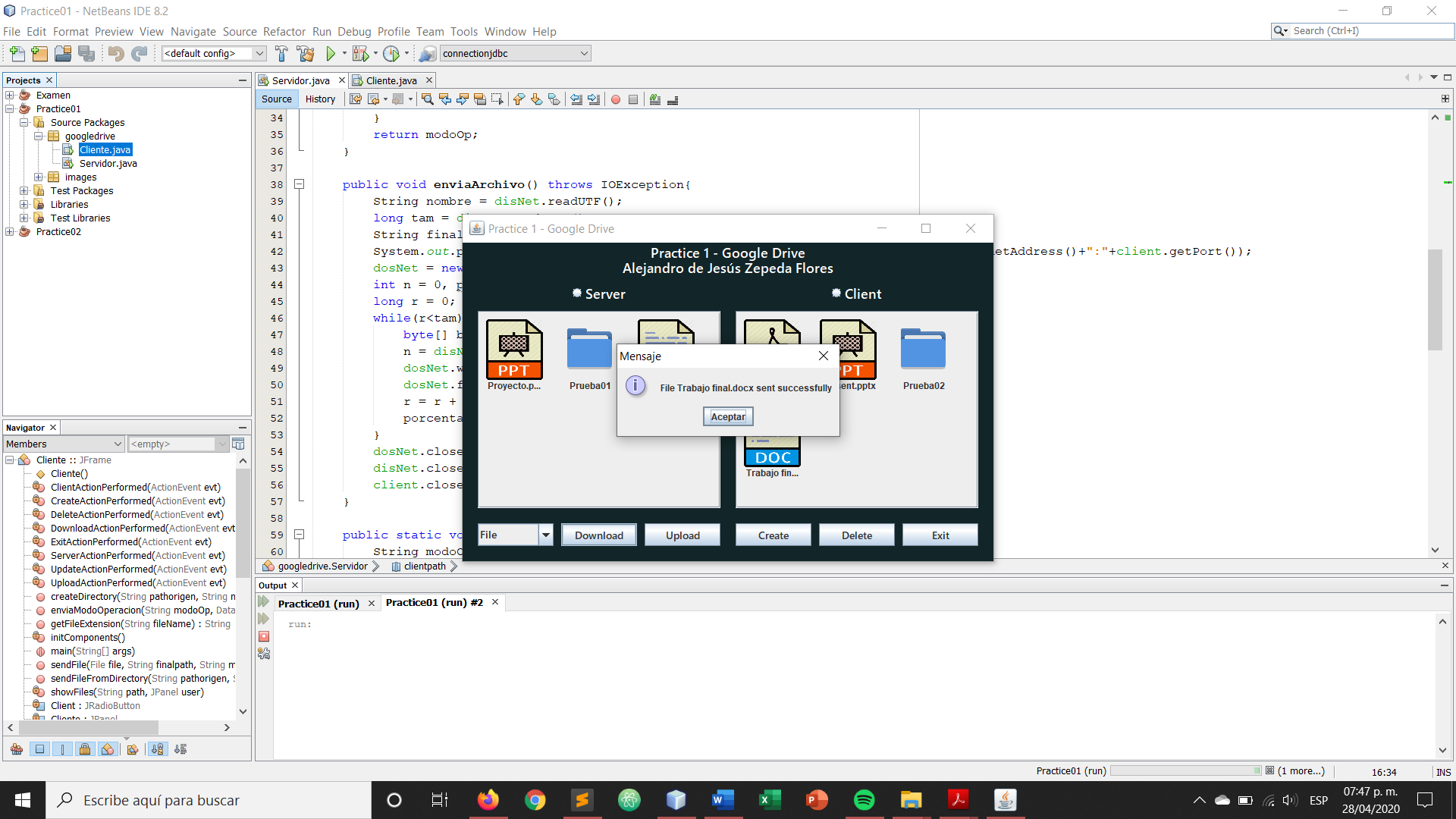


Imagen 3. Mensaje de comprobación.

La siguiente prueba, será subir la carpeta “Prueba02” que se encuentra en el cliente al servidor, por lo que ahora, en el combo box seleccionamos “Directory” y el botón “Upload”. Como se observar en la imagen 4, se abre una nueva ventana donde únicamente muestra las carpetas disponibles; en este caso solo una. La seleccionamos y continuamos con la ejecución

Imagen que contiene captura de pantalla

Descripción generada automáticamente

Imagen 4. Muestra de carpetas en el cliente.

Como sucedió con los archivos, en la imagen 5, se muestra el mensaje de confirmación del envío de la carpeta y vemos en el panel del cliente, que ya aparece la misma carpeta que enviamos desde el cliente.

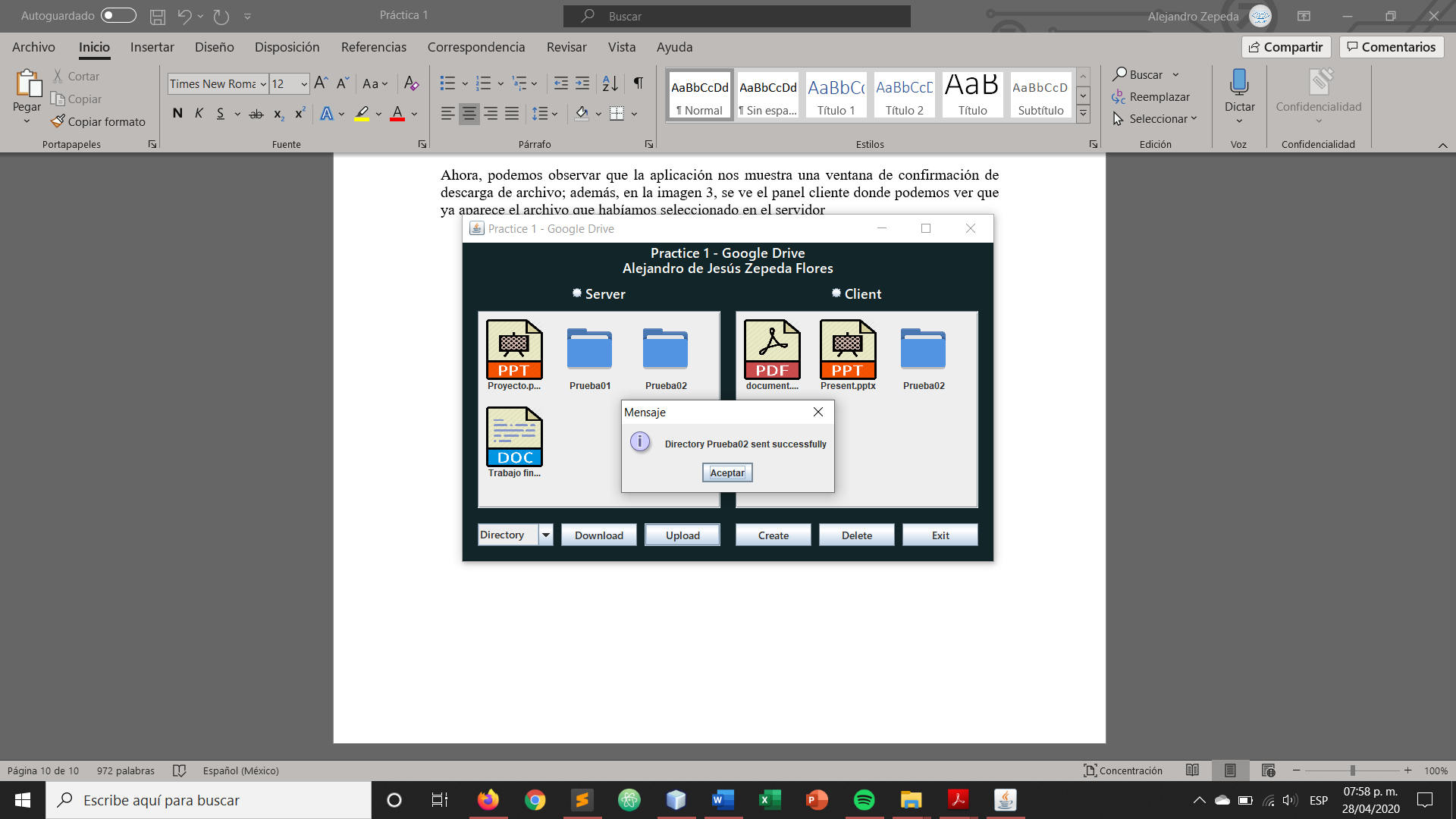


Imagen 5. Mensaje de comprobación.

CONCLUSIÓN

Esta práctica solo es una pequeña muestra de la utilidad que tienen lo sockets en el desarrollo de aplicaciones. En este caso, el envío y almacenamiento de archivos es un caso práctico que muchos servicios de internet suelen usar por lo que el conocer, grosso modo, una implementación de este tipo nos brinda experiencia que nos puede servir en un futuro. Además, el hecho de usar sockets de flujo nos facilita el trabajo de envío de archivos ya que es fácil evitar problemas como la pérdida de información o el hecho de que los datos lleguen

en desorden como ocurriría si se utilizaran sockets de datagrama.