INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

REDES DE COMPUTADORAS

PROF. AXEL ERNESTO MORENO CERVANTES

GRUPO: 3CM7

PRÁCTICA 4 – SERVIDOR HTTP

ALEJANDRO DE JESÚS ZEPEDA FLORES

26 de mayo de 2020

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

Práctica 4: https://drive.google.com/drive/folders/116wwoWUMpaRAfP4NqdOK2bnjP0CuII5L

OBJETIVOS

Desarrollar un servidor FTP basándose en sockets de flujo, hilos y principios de funcionamiento del protocolo HTTP. Dicho servidor, deberá tener implementados los métodos GET, POST, HEAD y DELETE, así como un pool de conexiones.

INTRODUCCIÓN

Con Servidor web podemos referirnos a hardware o software, o a ambos trabajando juntos.

* En cuanto a hardware, un servidor web es una computadora que almacena los archivos que componen un sitio web (ej. documentos HTML , imágenes, hojas de estilos CSS y archivo JavaScript) y los entrega al dispositivo del usuario final. Está conectado a internet y es accesible a través de un nombre de dominio como mozilla.org.
* En cuanto a software, un servidor web tiene muchas partes encargadas del control sobre como tienen acceso los usuarios a los archivos, por lo menos un servidor HTTP. UN servidor HTTP es una pieza de software que comprende URLs (direcciones web) y HTTP (el protocolo que tu navegador usa para ver las páginas web).

Al nivel más básico, siempre que un navegador necesite un archivo almacenado en un servidor web, el navegador lo solicita vía HTTP. Cuando la petición llega al servidor web correcto (hardware), el servidor HTTP (software) envía el archivo antes solicitado, también a través de HTTP.

* Un servidor web estático, o pila, consiste en una computadora (hardware) con un servidor HTTP (software). Llamamos a este “estático” debido a que el servidor envía los archivos almacenados “tal cual” a tu navegador.
* Un servidor web dinámico consiste en un servidor web estático con un software extra , lo más común es que sea una aplicación servidor y una base de datos. Llamamos a esto “dinámico” por qué la aplicacion servidor actualiza los archivos almacenados antes de enviarlos mediante el servidor HTTP.

**Protocolo HTTP**

HTTP es la abreviación de Protocolo de Transferencia de Hipertexto y es un protocolo de aplicación para permitir la comunicación entre sistemas físicamente dispersos. Fue originalmente ideado por Sir Tim Berners Lee en 1989. Ahora está coordinado por el W3C.

En su formato más básico, es lo que establece cómo las páginas web se comunican desde el servidor web al navegador del usuario.

**¿Cómo funciona HTTP?**

HTTP es un protocolo de aplicación para transferir recursos a través de Internet. HTTP usa el Puerto 80, que es el puerto desde el que el servidor web acepta peticiones. La mayoría de los recursos son archivos (imágenes, etc.) pero puede incluir otro tipo de datos como scripts.

Las sesiones HTTP se abren por un cliente HTTP (es decir, el navegador del usuario) a través de un agente de usuario y se envía un mensaje de petición de conexión al servidor HTTP (es decir, el servidor web). El mensaje de petición se conoce también como ‘petición del cliente’ y consiste en las siguientes líneas:

* Línea de petición
* Encabezados
* Línea vacía
* Un cuerpo del mensaje opcional

Una vez que la respuesta ha sido entregada el servidor web cierra la conexión. Este tipo de conexión es conocido como Stateless. La conexión solo existe por el periodo que dure el intercambio de datos. En función de la disponibilidad o no del recurso, HTTP proporciona un código de estatus apropiado (también conocido como ‘respuesta del servidor’), determinado por el protocolo. Son los siguientes:

* 1xx: un simple mensaje informativo
* 2xx: éxito de algún tipo.
* 3xx: el cliente a otra URL.
* 4xx: un error en el lado del cliente.
* 5xx: un error en el lado del servidor.

**Métodos HTTP**

Los métodos más importantes de HTTP (especialmente para hacer aplicaciones REST) son POST, GET,PUT, DELETE y HEAD.

* **GET:** El método GET se emplea para leer una representación de un resource. En caso de respuesta positiva (200 OK), GET devuelve la representación en un formato concreto: HTML, XML, JSON o imágenes, JavaScript, CSS. En caso de respuesta negativa devuelve 404 (not found) o 400 (bad request). Por ejemplo, en la carga de una página web, primero se carga la url solicitada: *GET php .net / docs HTTP /1.1*
* **POST:** Aunque se puedan enviar datos a través del método GET, en muchos casos se utiliza POST por las limitaciones de GET. En caso de respuesta positiva devuelve 201 (created). Los POST requests se envían normalmente con formularios:

*<form action =" formget .php " method ="get ">*

*Nombre : <input type =" text " name =" nombre " ><br >*

*Email : <input type =" text " name =" email " ><br >*

*<input type =" submit " value =" Enviar ">*

*</form >*

Rellenar el formulario anterior crea un HTTP request con la request line:

*POST / formpost .php HTTP /1.1*

El contenido va en el body del request, no aparece nada en la URL, aunque se envía en el mismo formato que con el método GET. Si se quiere enviar texto largo o cualquier tipo de archivo este es el método apropiado.

* **DELETE:** Simplemente elimina un resource identificado en la URI. Si se elimina correctamente devuelve 200 junto con un body response, o 204 sin body. DELETE, al igual que PUT y GET, también es idempotente.

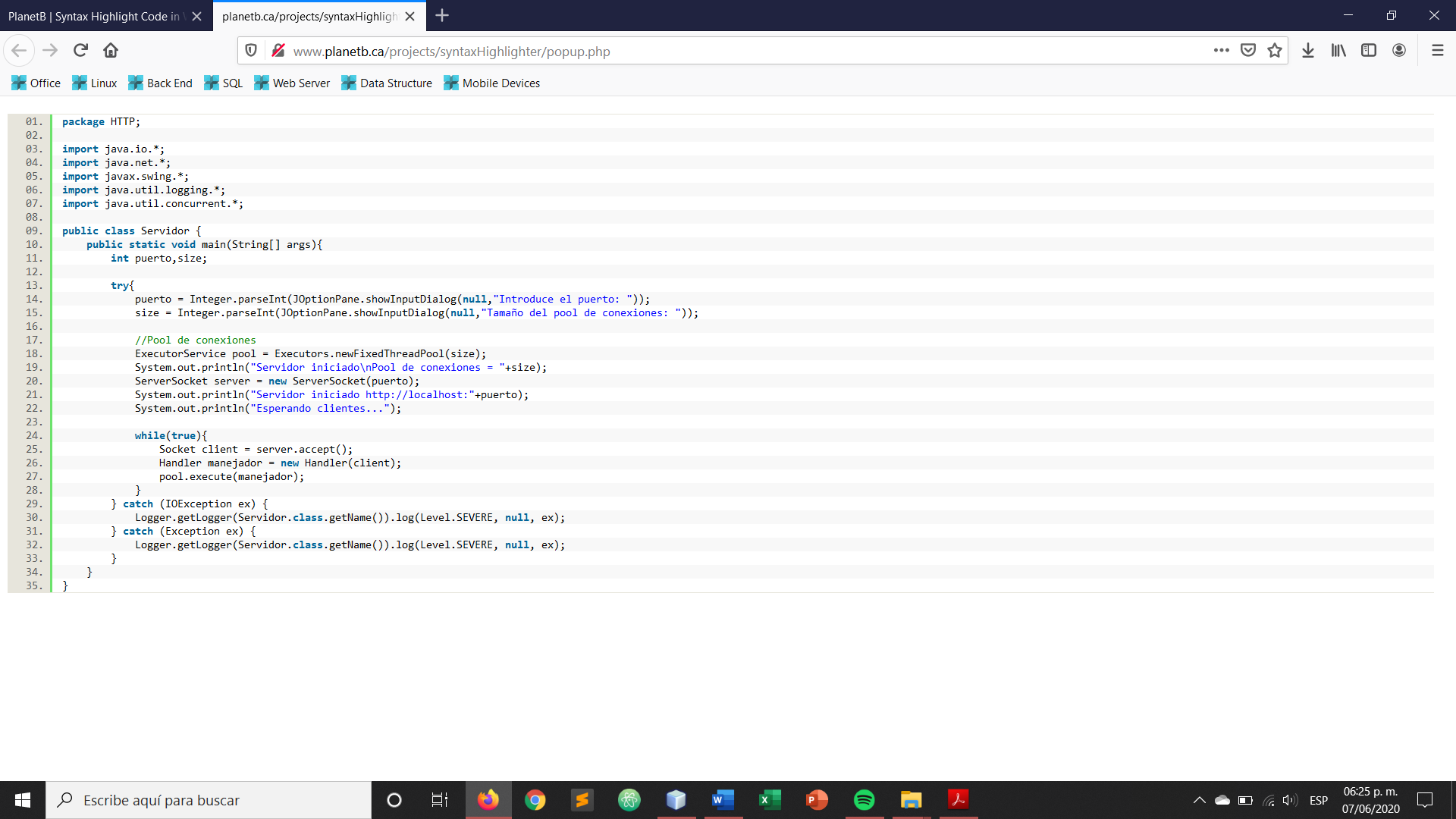
*DELETE ejemplo .com / usuario / peter HTTP /1.1*

* **HEAD:** Es idéntico a GET, pero el servidor no devuelve el contenido en el HTTP response. Cuando se envía un HEAD request, significa que sólo se está interesado en el código de respuesta y los headers HTTP, no en el propio documento. Con este método el navegador puede comprobar si un documento se ha modificado, por razones de caching. Puede comprobar también directamente si el archivo existe. Por ejemplo, si tienes muchos enlaces en tu sitio web, puedes enviar un HEAD request a todos los enlaces para comprobar los que estén rotos. Es bastante más rápido que hacerlo con GET*.*

DESARROLLO

Iniciamos con la programación del Servidor. En esta clase, es donde el usuario introducirá el puerto donde desea correr la aplicación y el tamaño para el pool de conexiones. Posteriormente, creamos el ServerSocket y el pool con ExecutorService.

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

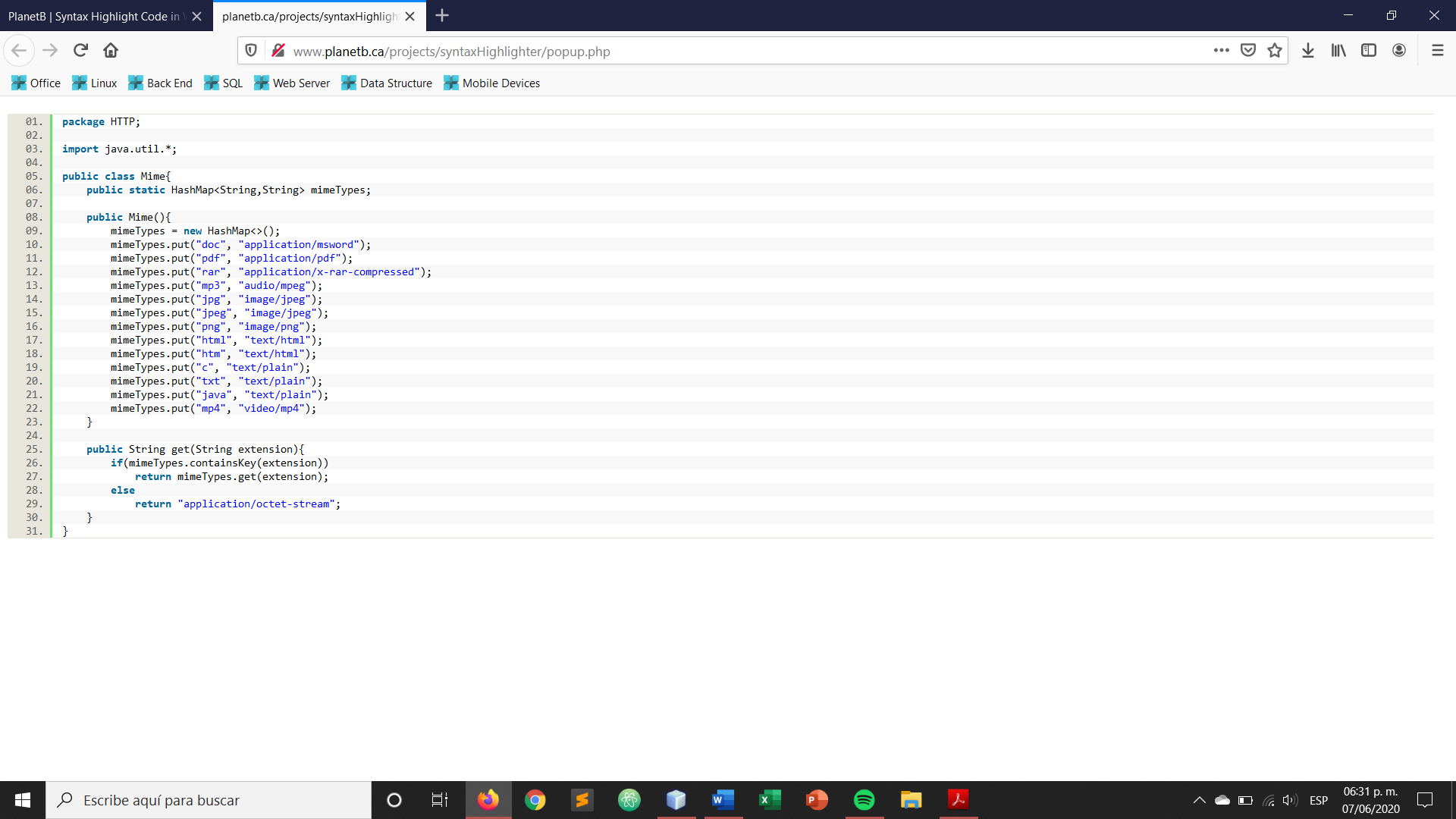


La parte del ciclo while infinito, es para poder estar creando cliente e instanciando la clase Handler, para posteriormente agregarla al pool de conexiones.

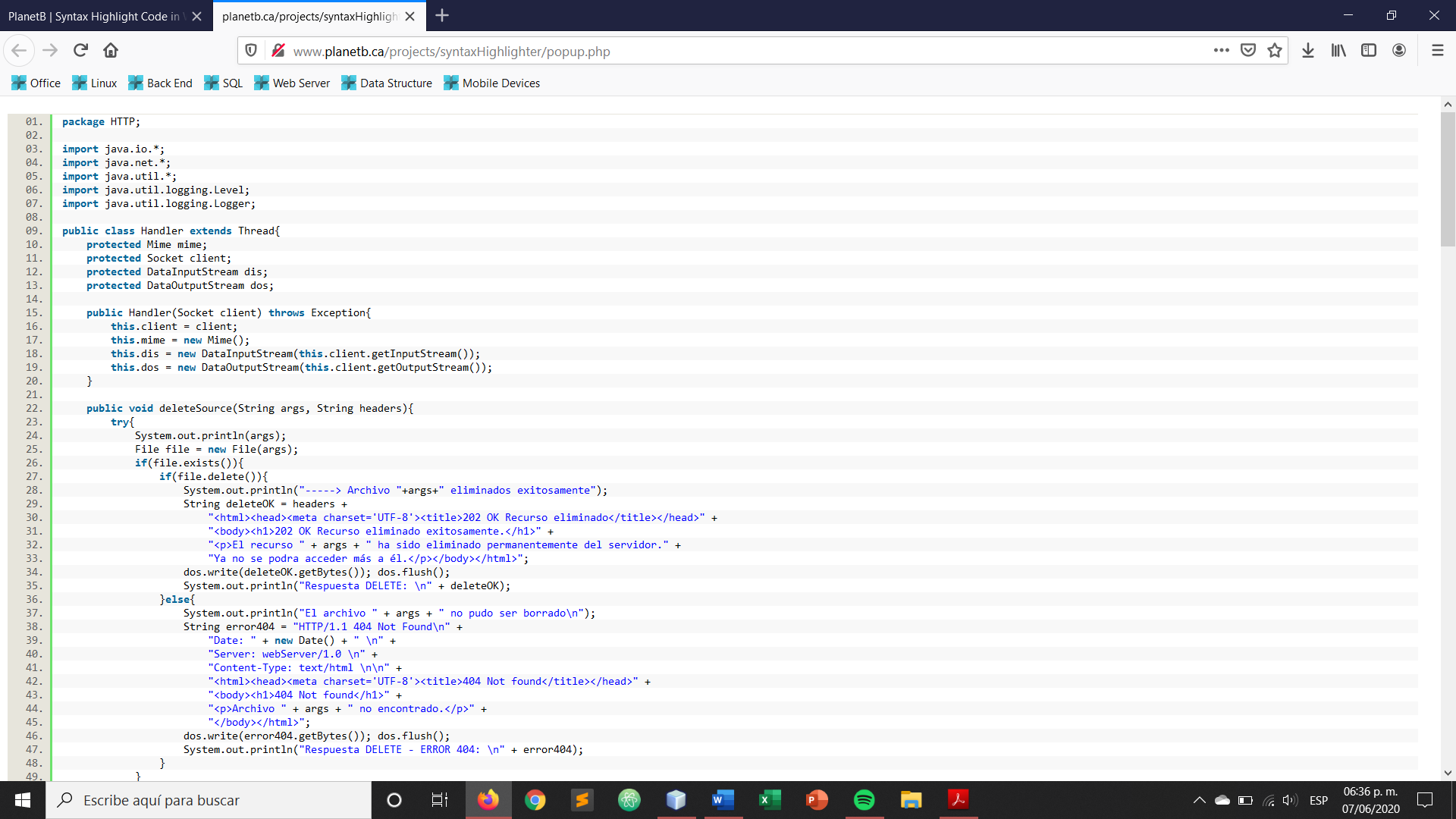
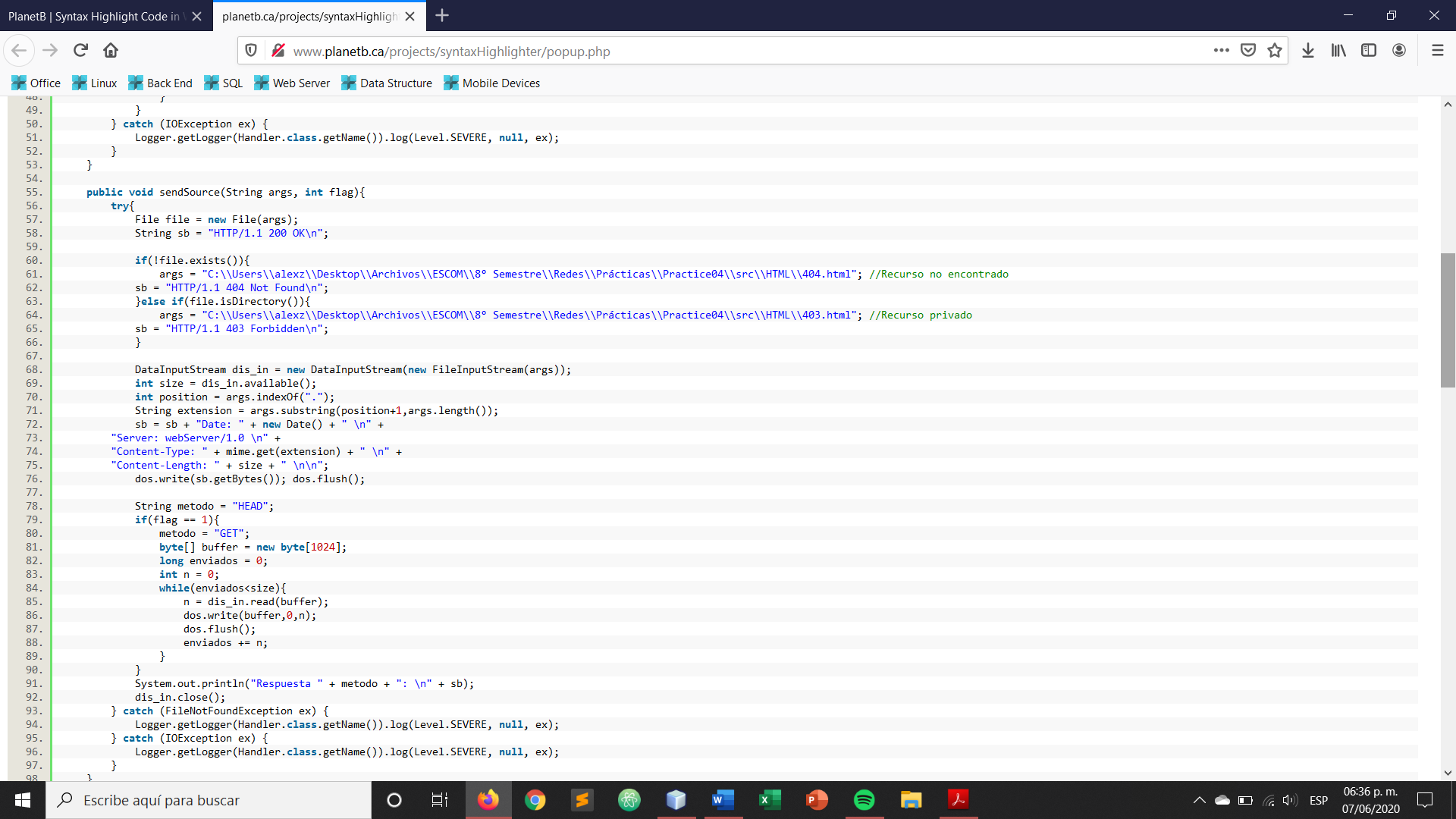
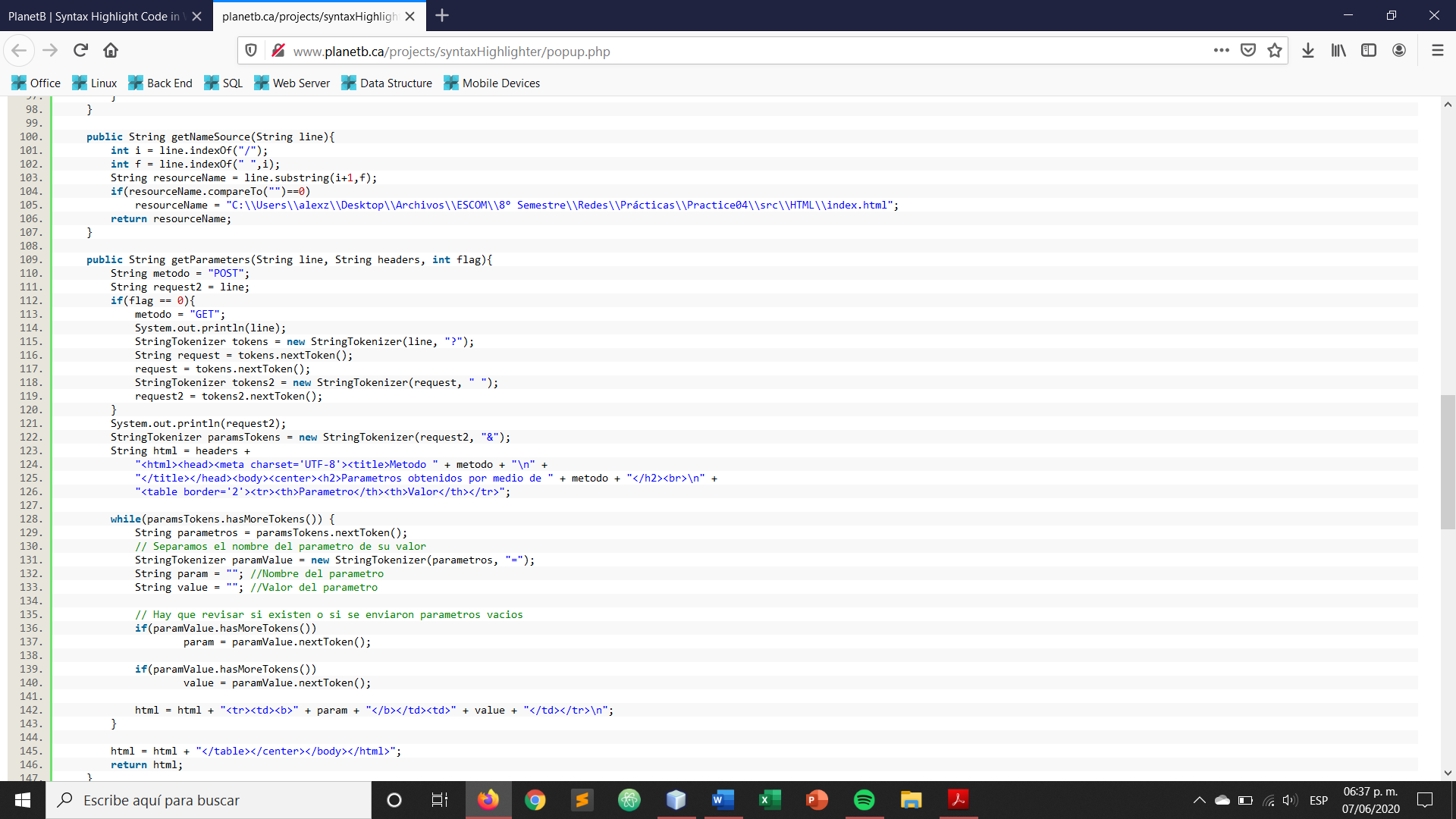
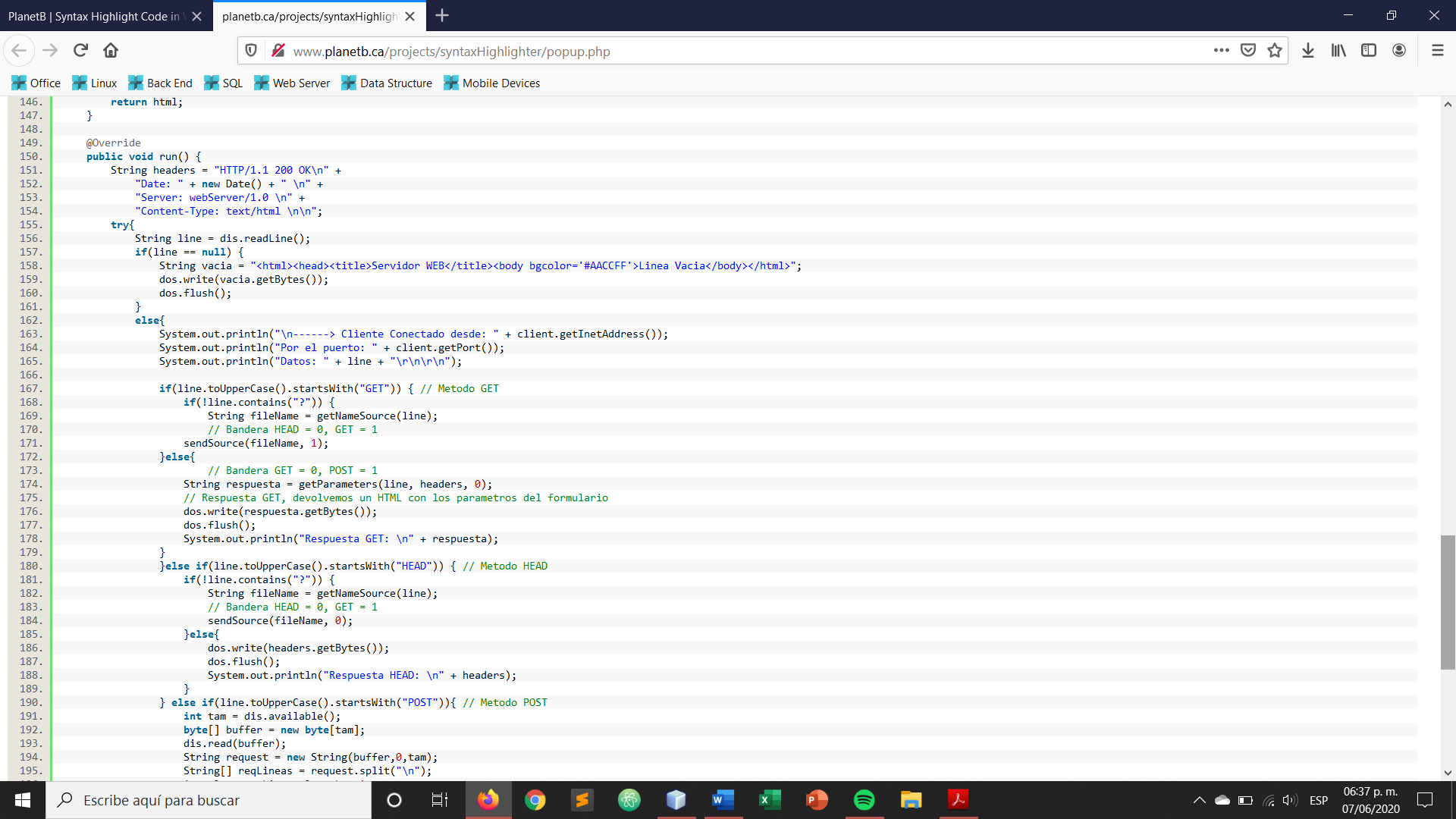
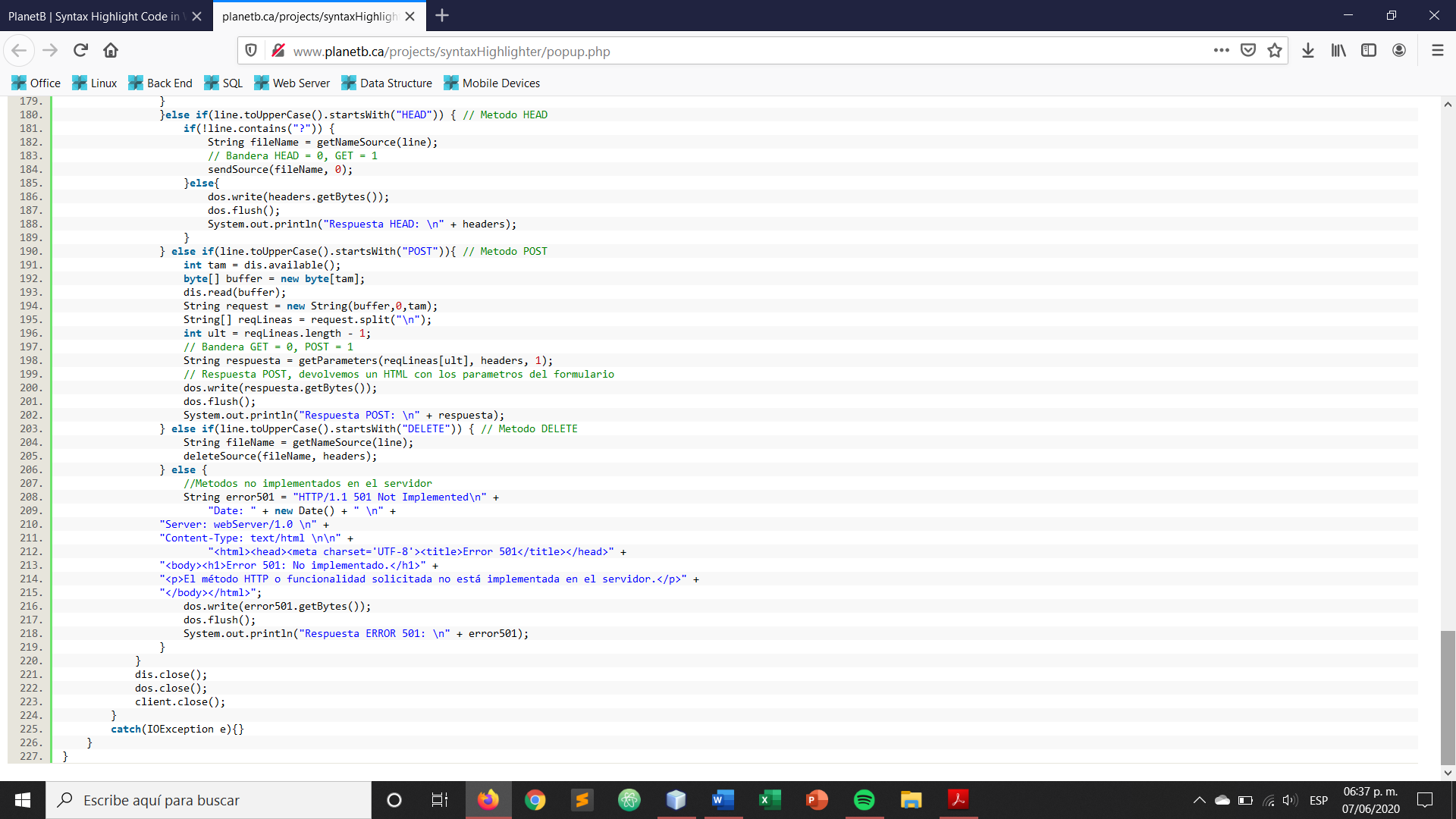
Ahora, vamos a explicar la clase Handler.

Realmente, está clase fue creada para manejar los tipos MIME de los archivos y poder agruparlas dentro de una misma clase; ya que en la clase Handler, requerimos mandar y recibir archivos, este tipo nos facilitará bastante el proceso.

A continuación, se muestra el código de la clase Mime.java.



Por último, vamos a explicar la clase Handler.java

En esta práctica implementaremos un servidor HTTP 1.1 sencillo corriendo sobre algun puerto (indicado por el usuario al ejecutar el servidor), que soporte los métodos GET, POST, HEAD y DELETE.

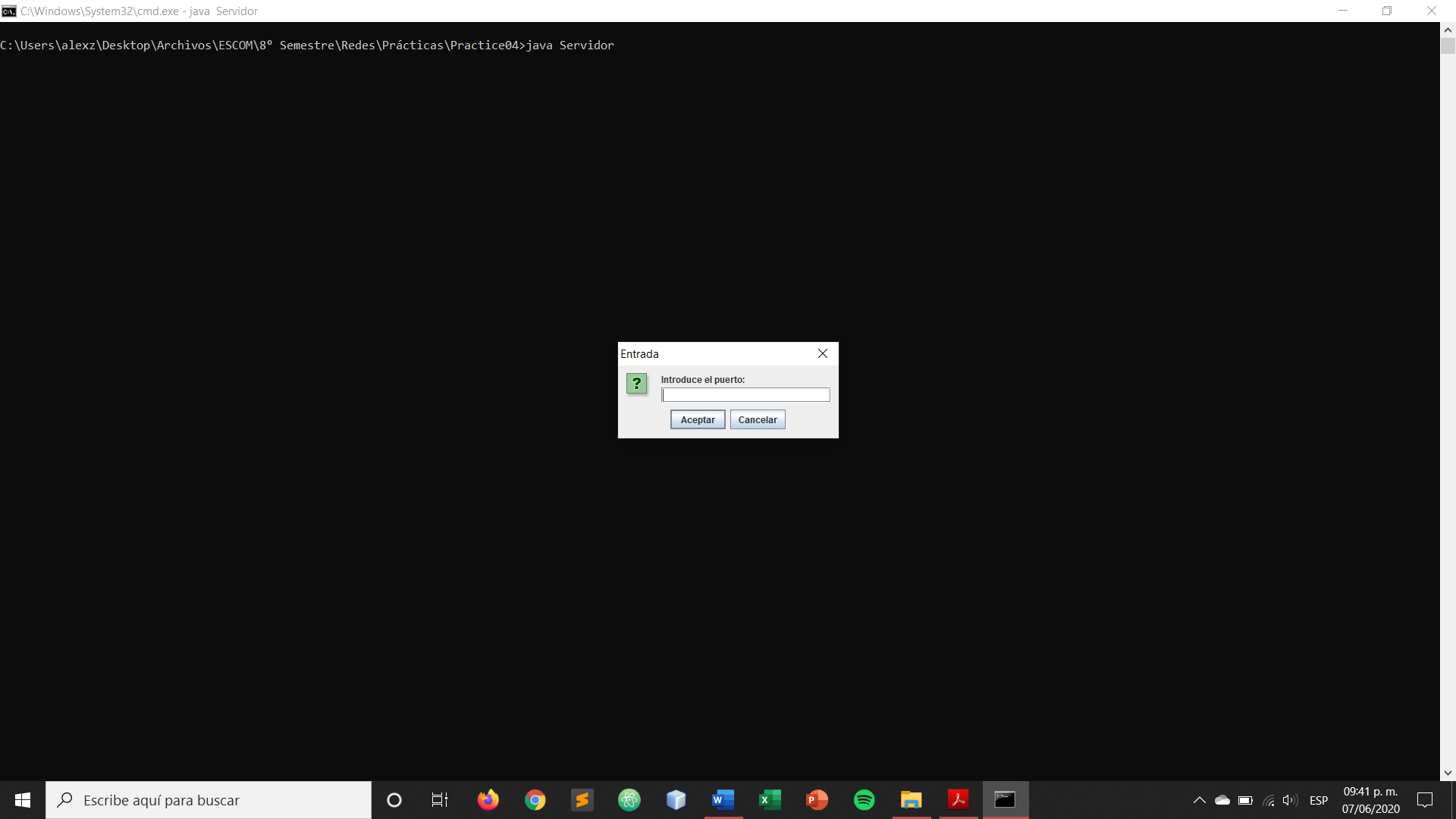
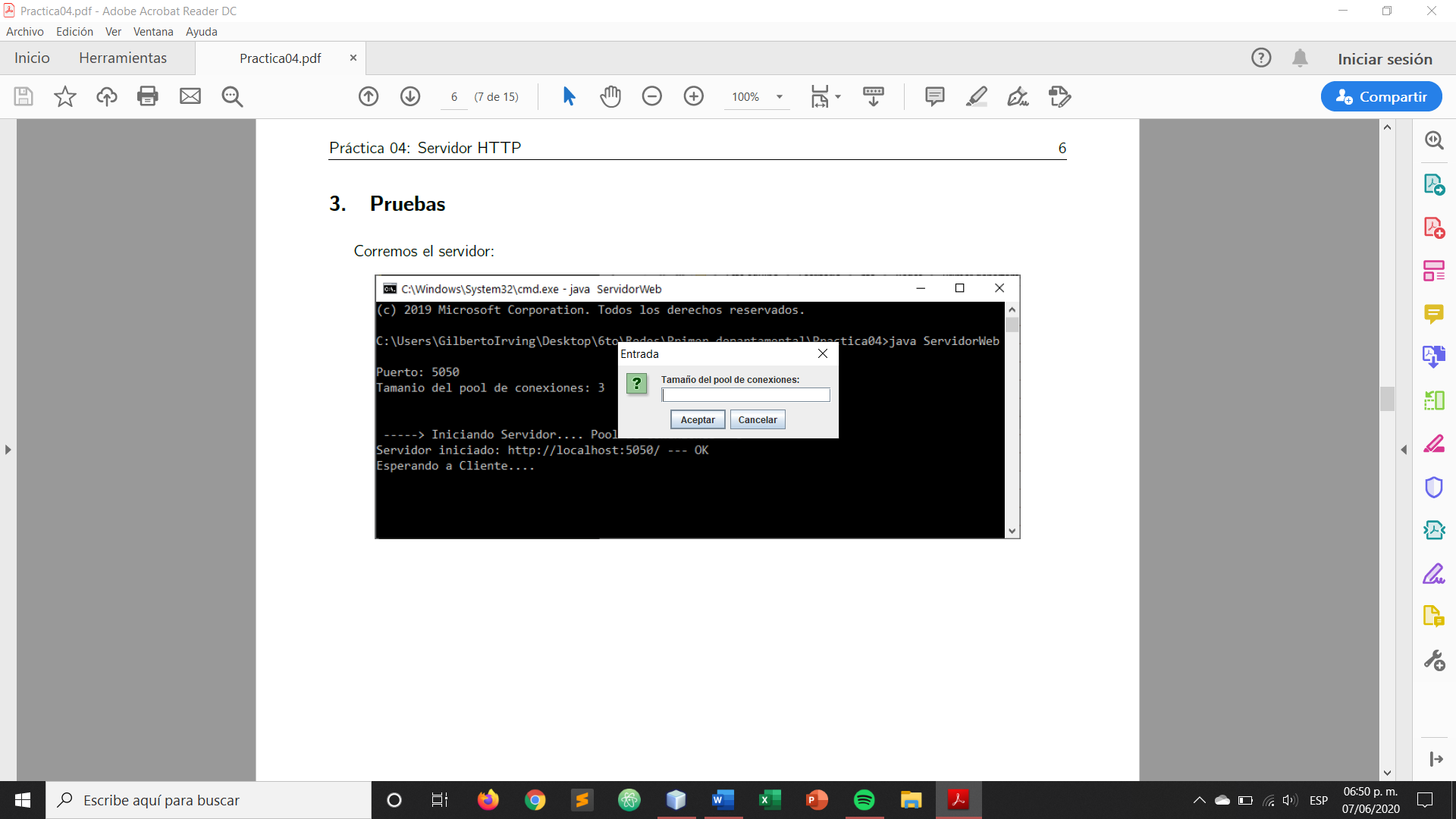
Soportará los encabezados más sencillos, tales como Content-Length, Content-Type y Date. Tendrá como códigos de error los más comunes: 404 (Not Found), 403 (Forbidden), 200 (OK) y 501 (Not Implemented).

Soportará el examen de directorios con previa autenticación, así como el borrado y subida de archivos (no se permite borrado de directorios). Además, para solicitudes GET y POST, las páginas .html podrán contener al estilo PHP las expresiones <?\_\_GET["key"]?> y <?\_\_POST["key"]?>, las cuales el servidor se encargará de sustituir con los valores en la solicitud del mensaje.

El documento predeterminado para carpeta será index.html.

FUNCIONAMIENTO DE LA APLICACIÓN

El primer paso es correr el servidor, donde el programa solicitará al usuario el puerto que desea utilizar y el tamaño del pool de conexiones. Posteriormente, se imprimirá en pantalla la dirección y un mensaje de aprobación para saber que el servidor está activo.



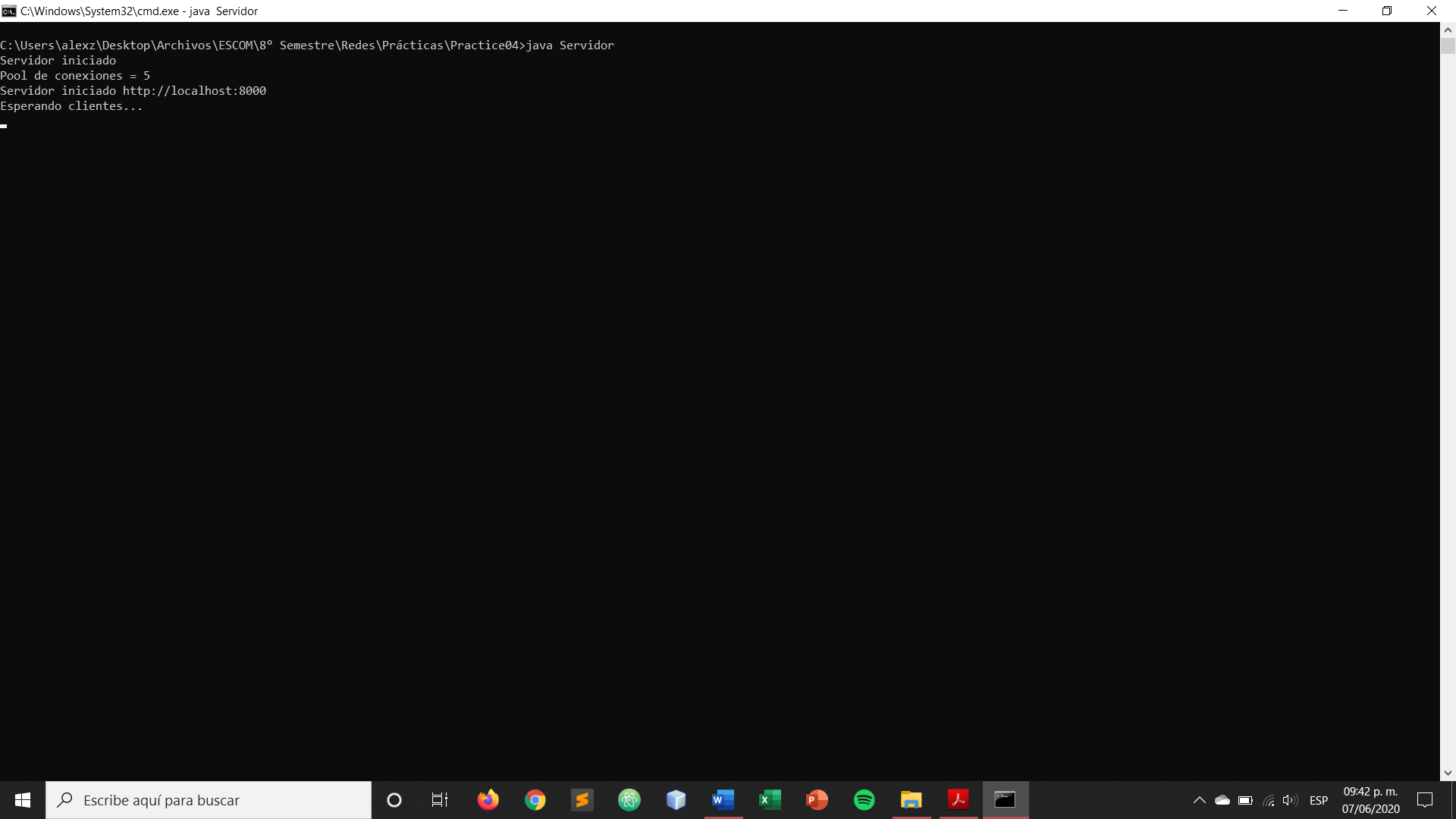


Imagen 1. Servidor iniciado.

Después, desde nuestro navegador web preferido, ingresamos a la dirección *localhost/puerto* donde el puerto será el que especificamos al inicio del programa cuando corrimos el servidor. La primera pantalla que aparecerá es la de los formularios GET y POST.

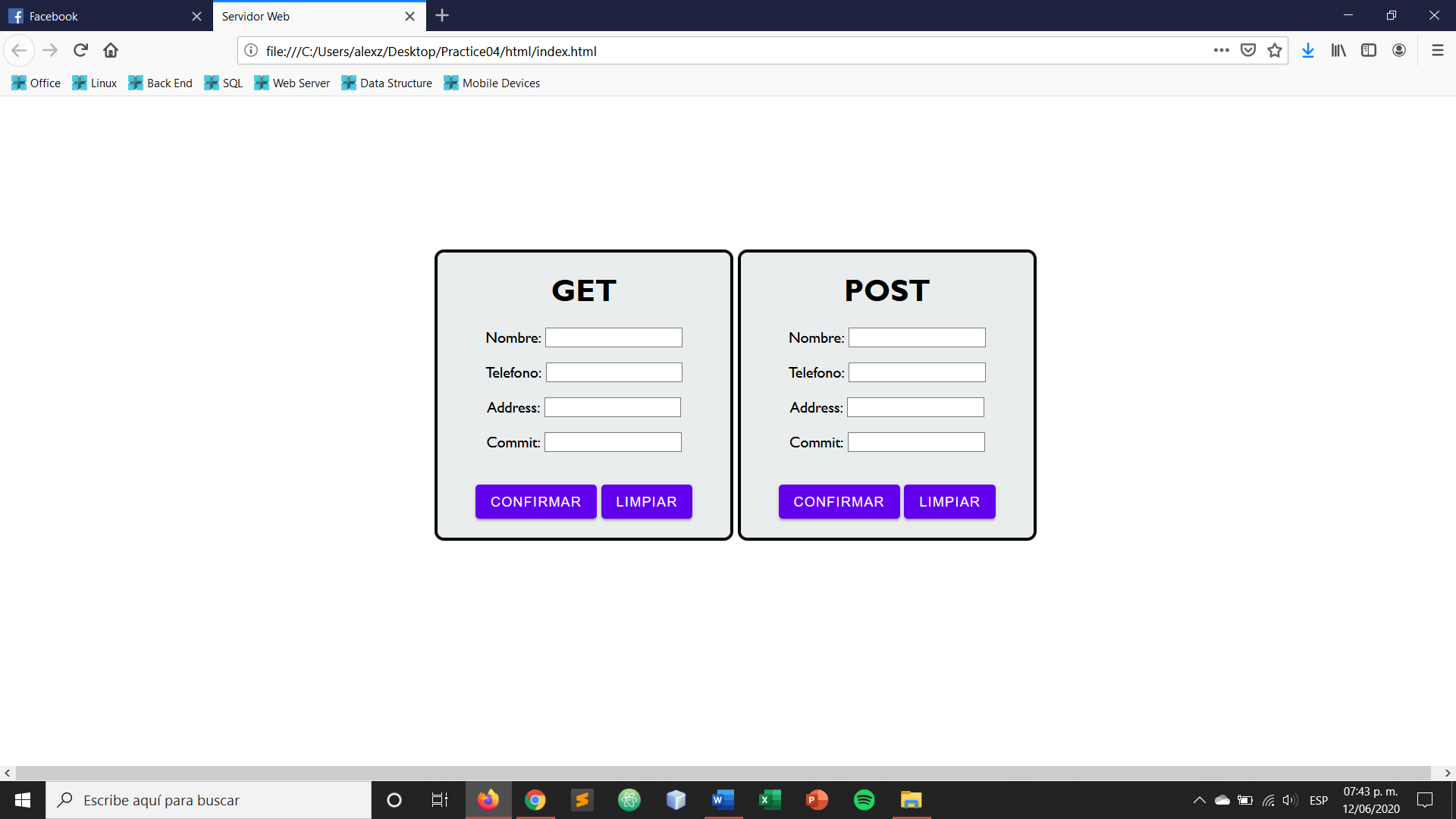


Imagen 2. Formularios GET y POST.

Una vez dentro del index.html si accedemos a “Recursos disponibles” nos redireccionara a la nueva página con documentos que previamente actualizamos en la carpeta y podremos visualizar cada uno de ellos.

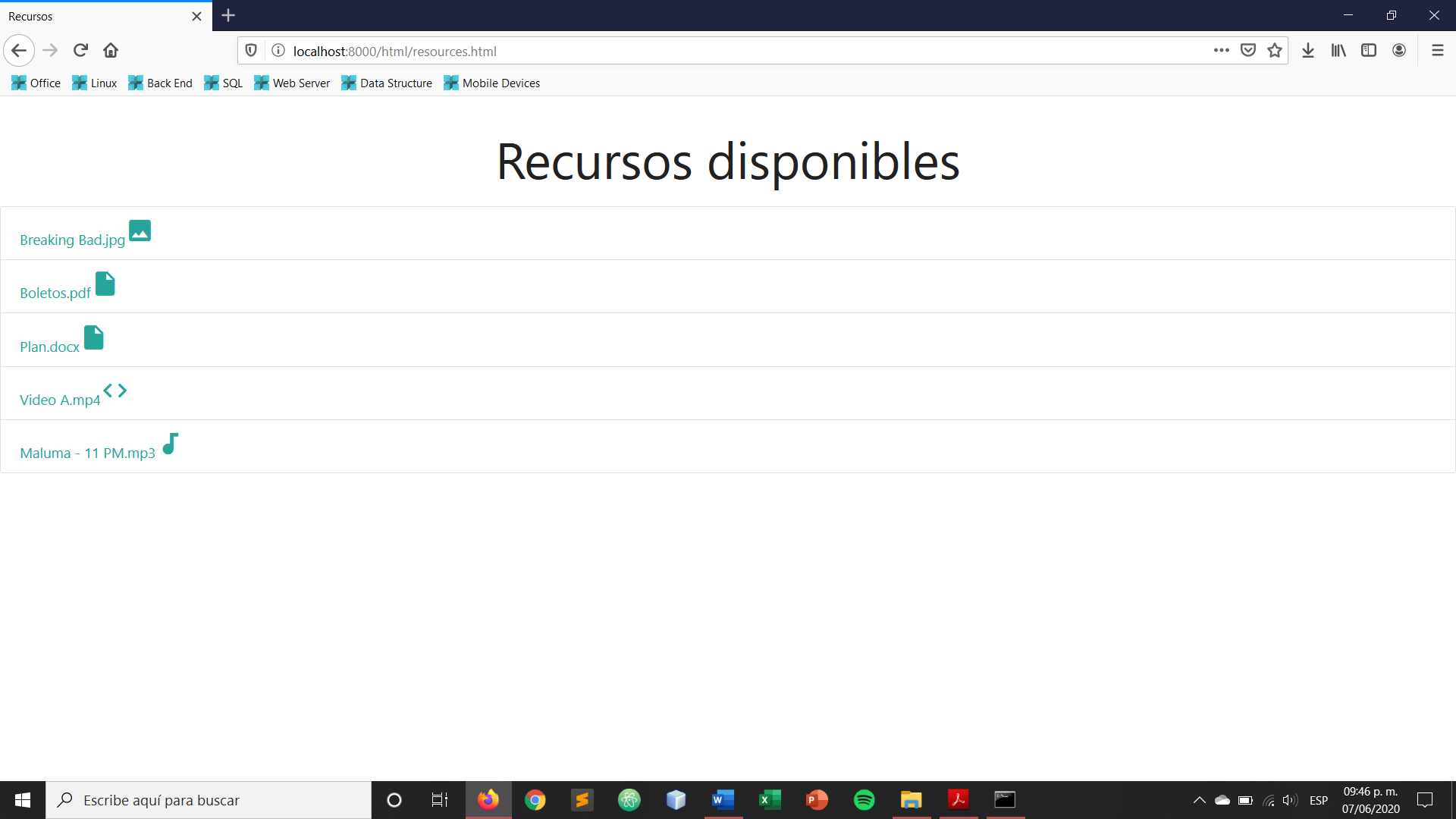


Imagen 3. Recursos disponibles.

Una prueba de ello, es la imagen que almacenamos en el primer elementos de la lista.

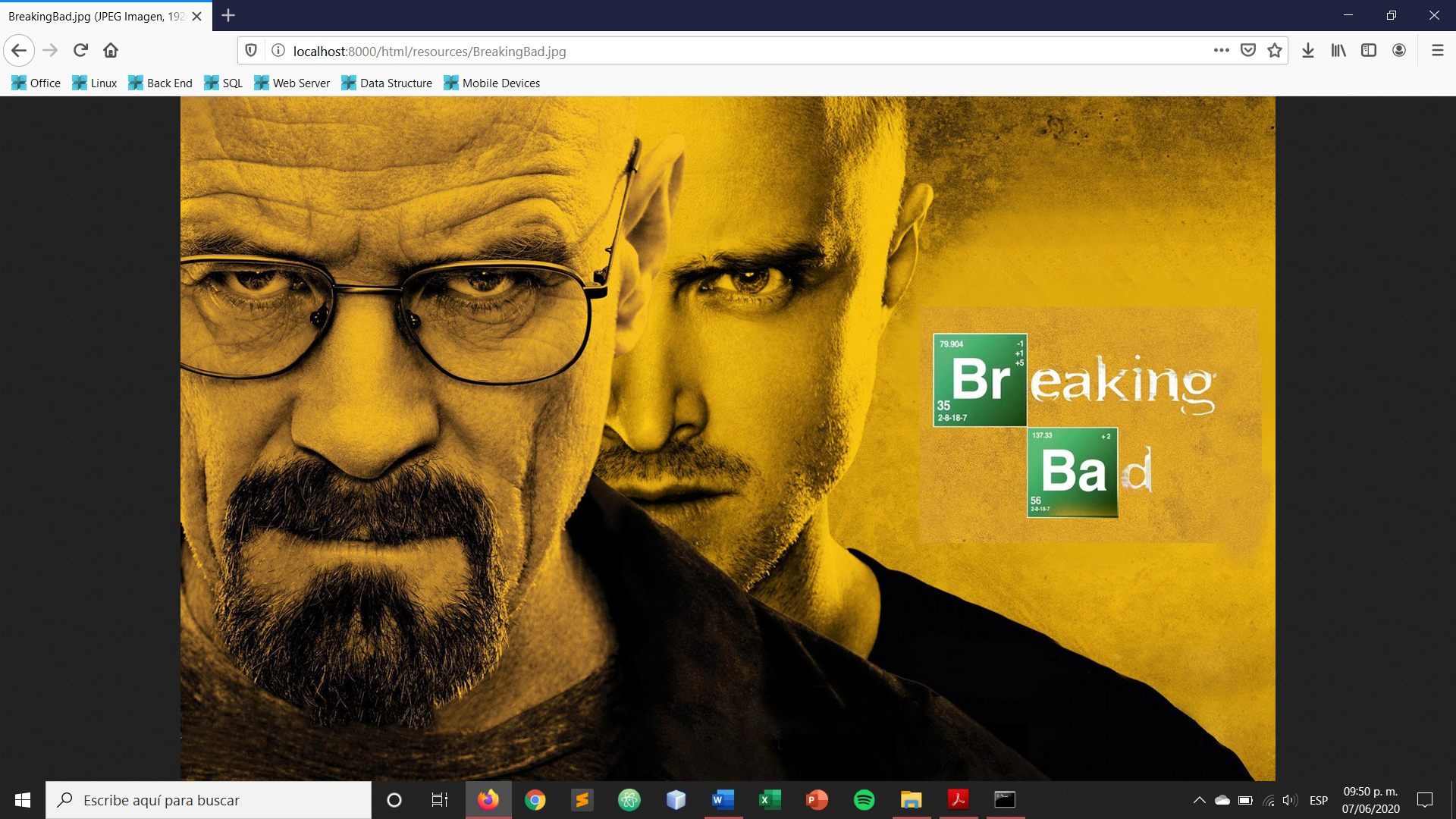


Imagen 4. Breaking Bad.

Ahora, vamos a probar los formularios GET y POST, introduciendo nuestros datos y redireccionando la página para ver que el envío de estos fue exitoso y que se utilizó el método correcto.

Esta prueba es muy sencilla, lo que realmente nos importa es la barra de direcciones, donde se muestra que método es el que se utilizó y como son enviados los datos.

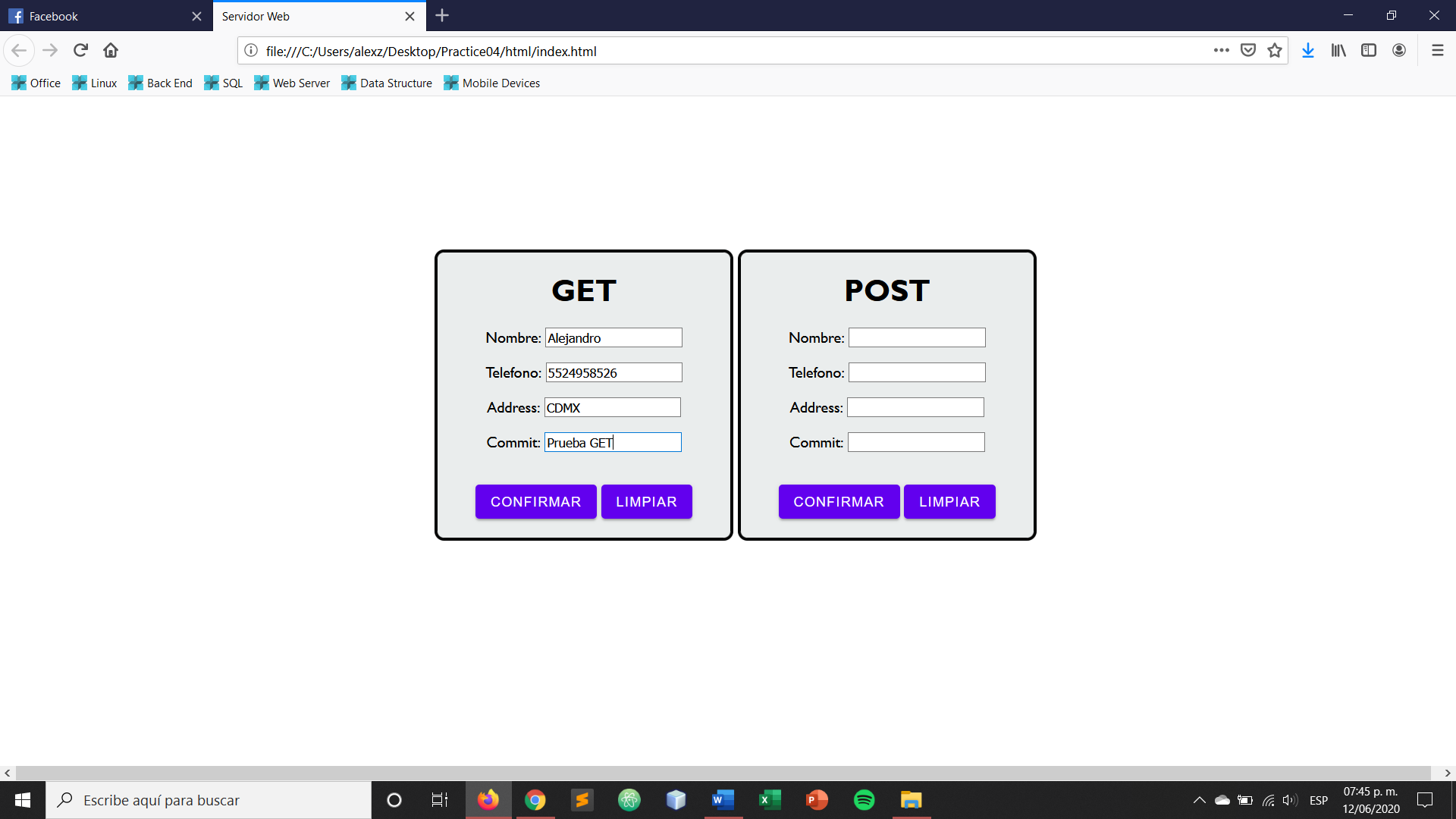




Imagen 5. Prueba método GET.

Para la prueba del método POST, es la misma que la del método GET. Visualmente, no se alcanzan a visualizar cambios y un usuario promedio pensaría que es lo mismo, sin embargo, el cambio está realmente en el envío y tratamiento de datos.

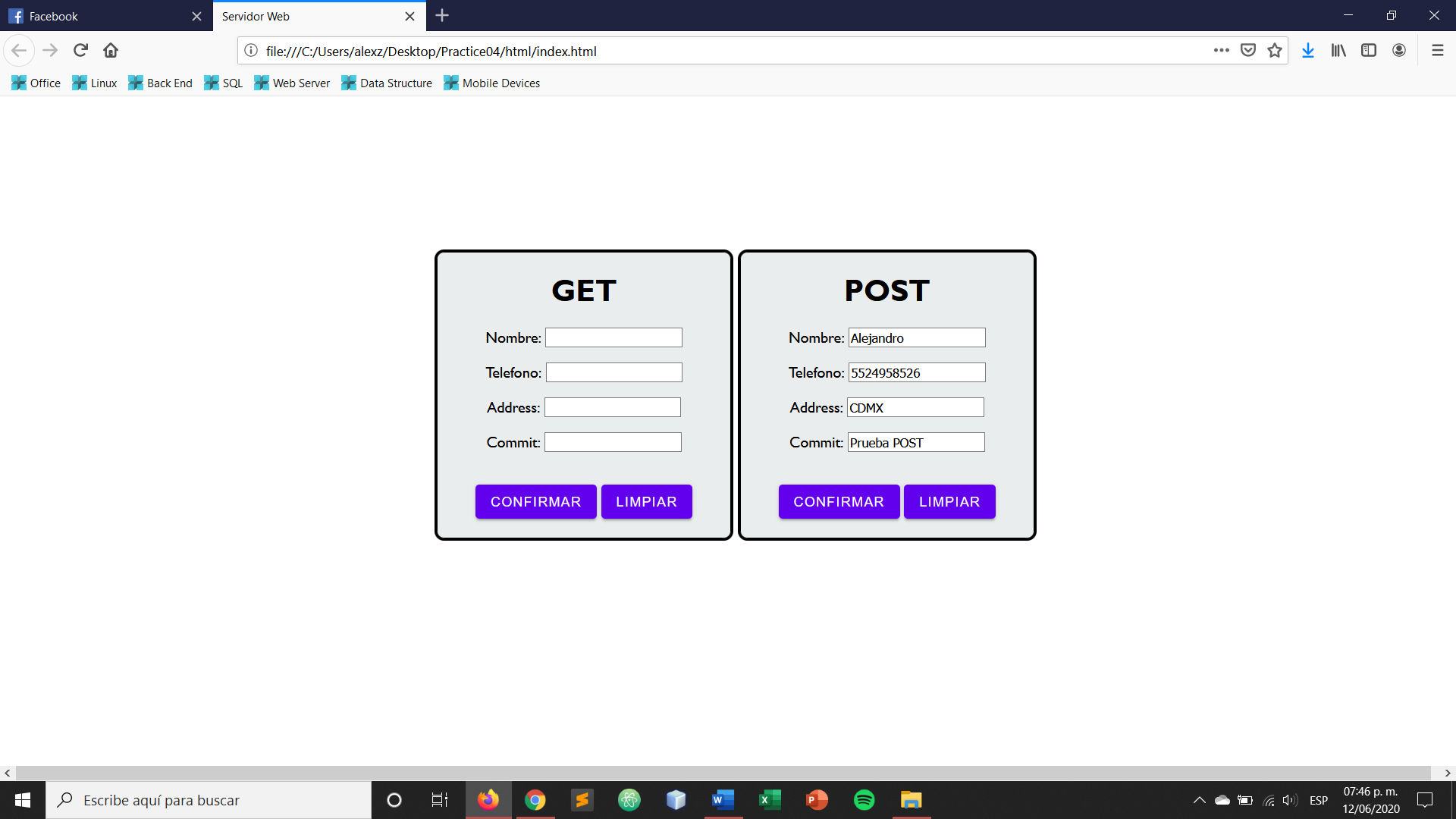


Imagen 6. Prueba método POST.

La última prueba es la del método DELETE. Para esta, requerimos de la aplicación POSTMAN para poder ejecutar la consulta. Donde únicamente tenemos que ingresarla y seleccionar el método. Una vez ejecutado, mostrará el mensaje del recurso eliminado.

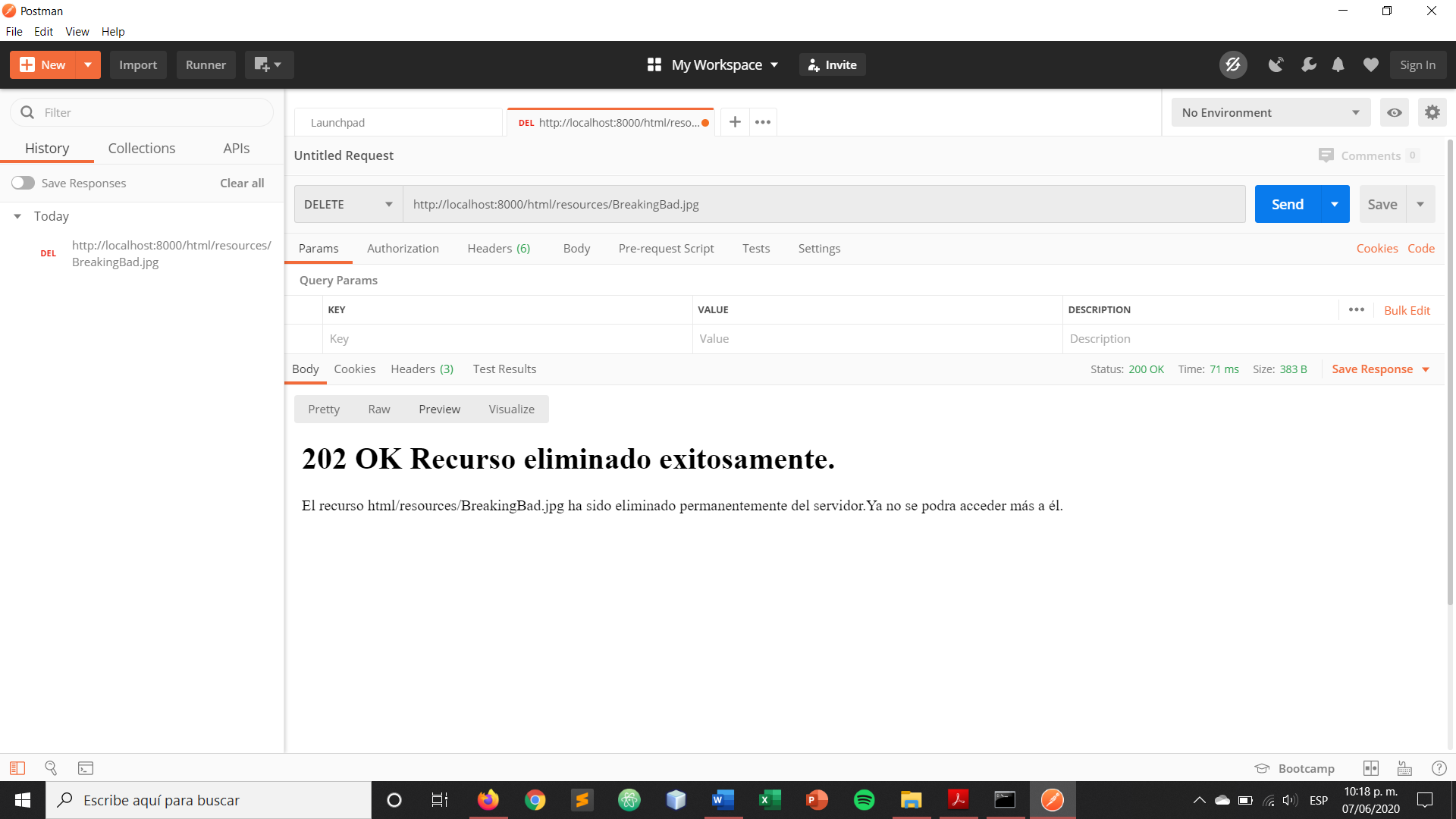


Imagen 7. Prueba método DELETE.

CONCLUSIÓN

En esta práctica implementamos una versión muy sencilla del protocolo HTTP 1.1. Se tuvo que hacer uso de los streams en java y de escribir correctamente los headers y el cuerpo a la salida, tal y como lo dice el protocolo HTTP; y por cada método que íbamos recibiendo, ejecutar las acciones correspondientes, validando todos los posibles códigos de estado de acuerdo a lo que el cliente diga. Usamos expresiones regulares para extraer de forma mucho más sencilla los datos que recibimos de los clientes.

A pesar de ser una versión muy reducida, cumple muy bien con el objetivo de demostrar la implementación del protocolo HTTP desde cero y comprobar que funciona en cualquier navegador, e incluso soporta sustituciones muy básicas de los campos almacenados en el formulario, a través de GET y POST; así como el soporte de un muy básico inicio de sesión mediante el navegador.

Por último, haberlo implementado sobre un pool de conexiones nos da mayor eficiencia, pues no se crea un hilo por cada cliente que va llegando, sino que, si se supera el máximo número de clientes simultáneos, tendrán que esperar hasta que uno se desocupe. Pero como HTTP es stateless, esto pasa muy rápido y, a menos que un cliente esté descargando un archivo muy grande, la asignación de los recursos entre clientes se dará de una forma transparente.