UNPSJB

-Lic. en Sistemas OPGCPI-

-Sistemas Distribuidos-

-Laboratorio 3-

-CGI-Ajax-NFS-DNS-

Cátedra
Mg. Ricardo Antonio López
Lic. Cristian Parise

Integrantes:
Aguila Maximiliano
Krmpotic Lucas

November 11, 2018

Contents

1 Desarrolle un experimento (mediante Telnet o analizador de paquetes) que muestre si el servidor de HTTP agrega o quita información a la que genera un programa CGI. Nota: debería programar o utilizar un programa CGI para conocer exactamente lo que genera y debería mostrar la información que llega del lado del cliente, a nivel de HTTP o HTML (preferentemente HTTP).

1.1 Experimento con modulo "requests" de python

La aplicación cgi para este ejercicio se encuentra en *"1/cgi-bin/hola.py"*

```
In [7]: import requests
        import json
        response = requests.post("http://localhost:8080/cgi-bin/hola.py")
        print("------------ cabeceras de la petición ----------------")
        print()
        print(json.dumps(dict(response.request.headers), indent=2))

        print("------------ cabeceras de la respuesta ----------------")
        print()
        print(json.dumps(dict(response.headers), indent=2))

        print("------------ texto de la respuesta ----------------")
        print()
        print(response.text)
```

```
------------ cabeceras de la petición ----------------

{
  "User-Agent": "python-requests/2.20.1",
  "Accept-Encoding": "gzip, deflate",
  "Accept": "*/*",
  "Connection": "keep-alive",
  "Content-Length": "0"
}
------------ cabeceras de la respuesta ----------------

{
  "Date": "Sun, 11 Nov 2018 18:06:31 GMT",
  "Server": "Apache/2.4.35 (Unix)",
```

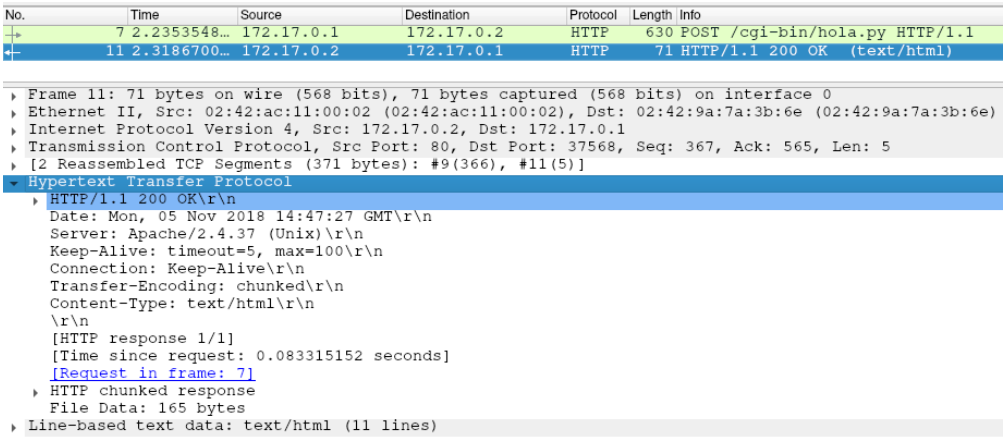
```
  "Keep-Alive": "timeout=5, max=100",
  "Connection": "Keep-Alive",
  "Transfer-Encoding": "chunked",
  "Content-Type": "text/html"
}
------------ texto de la respuesta ----------------


<HTML><head><title>CGI</title></head><BODY>

<font color = blue>

<TITLE>CGI script output</TITLE>
<h1 style="TEXT-ALIGN: center">HOLA MUNDO</h1>

</font>

</HTML>
```

Indicar que cgi también resuelve (además de las cabeceras) el contenido (response.text) html de las respuestas de error del servidor

```
In [11]: response = requests.post("http://localhost:8080/cgi-bin/algoquenoexiste.py")
         print("------------ cabeceras de la respuesta ----------------")
         print()
         print(json.dumps(dict(response.headers), indent=2))

         print("------------ texto de la respuesta ----------------")
         print()
         print(response.text)
```

```
------------ cabeceras de la respuesta ----------------

{
  "Date": "Sun, 11 Nov 2018 18:10:15 GMT",
  "Server": "Apache/2.4.35 (Unix)",
  "Vary": "accept-language,accept-charset",
  "Accept-Ranges": "bytes",
  "Keep-Alive": "timeout=5, max=100",
  "Connection": "Keep-Alive",
  "Transfer-Encoding": "chunked",
  "Content-Type": "text/html; charset=utf-8",
  "Content-Language": "en"
```

```
}
------------ texto de la respuesta ----------------

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
<title>Object not found!</title>
<link rev="made" href="mailto:you@example.com" />
<style type="text/css"><!--/*--><![CDATA[/*><!--*/
    body { color: #000000; background-color: #FFFFFF; }
    a:link { color: #0000CC; }
    p, address {margin-left: 3em;}
    span {font-size: smaller;}
/*]]>*/--></style>
</head>

<body>
<h1>Object not found!</h1>
<p>


    The requested URL was not found on this server.



    If you entered the URL manually please check your
    spelling and try again.



</p>
<p>
If you think this is a server error, please contact
the <a href="mailto:you@example.com">webmaster</a>.

</p>

<h2>Error 404</h2>
<address>
  <a href="/">localhost</a><br />
  <span>Apache/2.4.35 (Unix)</span>
</address>
</body>
```




```
</html>
```

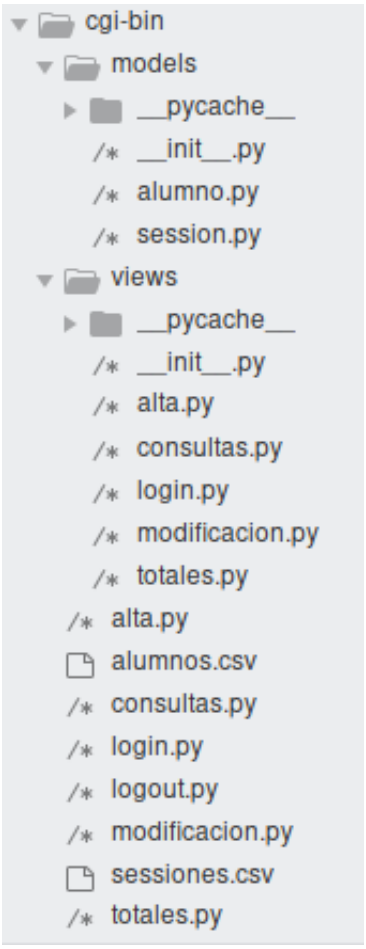
1.2 Experimento desde Wireshark

1.3 Conclusion

Analizando el header, podemos ver que a nivel HTTP se agregan los campos de estado, fecha, servidor y última modificación.

2 Se debe desarrollar una aplicación que presente al usuario una página web con una opción de carga de datos y una opción de modificación de datos. La opción de carga consistirá en mostrar un formulario con un conjunto de datos a completar y que luego almacene estos datos en un archivo de texto plano. Los datos son:

- Nombre y Apellido (hasta 70 caracteres)
- Número de Alumno/Legajo (cantidad de dígitos limitada)
- Sexo (lista desplegable para elegir entre Masculino o Femenino)
- Edad (hasta dos dígitos)
- Contraseña (que no se muestre en pantalla)



Para el caso de "modificación", se deberá requerir Numero de Alumno y la Contraseña, si este par es encontrado en el archivo de texto plano, se muestra lo que ya se tiene para ser modificado como si se estuvieran cargando los datos, pero con los valores iniciales/default cargados del archivo. Utilizar cookies para permitir sucesivas modificaciones sobre los datos sin volver a pedir el legajo y palabra clave.

Se deben desarrollar las páginas estáticas HTML, el o los programas CGI para procesar los datos y la distribución de estos componentes en el sistema de archivos.

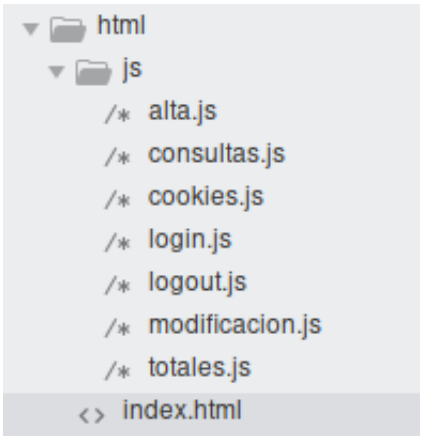
Justifique si este ejercicio puede ser considerado una aplicación web.

2.1 Implementacion

Para la implementacion del **Servidor** utilizamos el modemo MVC (Model, View, Controller) para un manejo claro de la funcionalidad.

Donde los archivos: + login.py + logout.py + alta.py + modificacion.py + consultas.py + totales.py

Son los **Controladores** y los archivos dentro de la carpeta **Models** son el **Modelo** y los archivos dentro de la carpeta **Views** son nuestras **Vistas**.



Luego en la carpet **html** tenemos solamente una plantilla html, **index.html**.

Esto es asi y no tenemos las demas plantillas, porque utilizamos **AJAX** para formar y armar el contenido de la pagina. Según la peticion requerida, se va insertando en el container de la pagina los distintos htmls.

Dentro del directorio **js** tenemos los archivos javascript con las peticiones que realizara el navegador a nuestro servidor.

El controlador a esta accion, verifica si la peticion es un **GET** o un **POST** y segun esto realiza una determinada funcion.

El **Controlador** verifica el **GET** o **POST** de la siguiente manera:

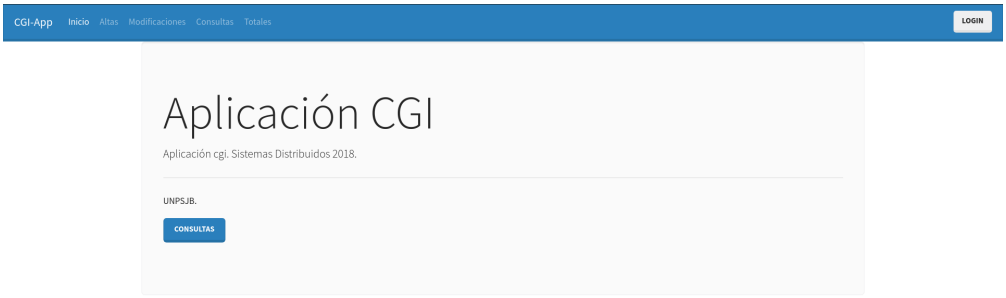
```
if os.environ["REQUEST_METHOD"] == "POST":
    # Accion al POST

if os.environ["REQUEST_METHOD"] == "GET":
    # Accion al GET
```

Otra desicion que se tomo, fue la de utilizar archivos CSV para menejar las sesiones y los alumnos dados de alta.

Para manejar este tipo de archivo utilizamos la libreria de python **pandas** que nos brinda un manejo mas transparente y no tan rudimentario de este tipo de archivos (csv).

Como ventaja de esta libreria, pudimos manipular mas facilmente los archivos csv sin necesitar de bucles para las consultas. Un claro ejemplo de utilizacion en nuestro proyecto es en la clase **alumno.py** del **modelo** con su metodo **get_totales()**.

```
def get_totales(cls):
    alumnos = pd.read_csv(MODEL_FILE)

    result = {
        "mujeres1":len(alumnos[(alumnos['sexo'] == "femenino") & \
                               (alumnos['edad'] < 21)]),
        "mujeres2":len(alumnos[(alumnos['sexo'] == "femenino") & \
```




```
                        ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
    "mujeres3":len(alumnos[(alumnos['sexo'] == "femenino") & \
                        (alumnos['edad'] > 40)]),
    "varones1":len(alumnos[(alumnos['sexo'] == "masculino") & \
                        (alumnos['edad'] < 21)]),
    "varones2":len(alumnos[(alumnos['sexo'] == "masculino") & \
                        ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
    "varones3":len(alumnos[(alumnos['sexo'] == "masculino") & \
                        (alumnos['edad'] > 40)]),
    "otro1":len(alumnos[(alumnos['sexo'] == "otro") & \
                    (alumnos['edad'] < 21)]),
    "otro2":len(alumnos[(alumnos['sexo'] == "otro") & \
                    ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
    "otro3":len(alumnos[(alumnos['sexo'] == "otro") & \
                    (alumnos['edad'] > 40)]),
}

return result
```

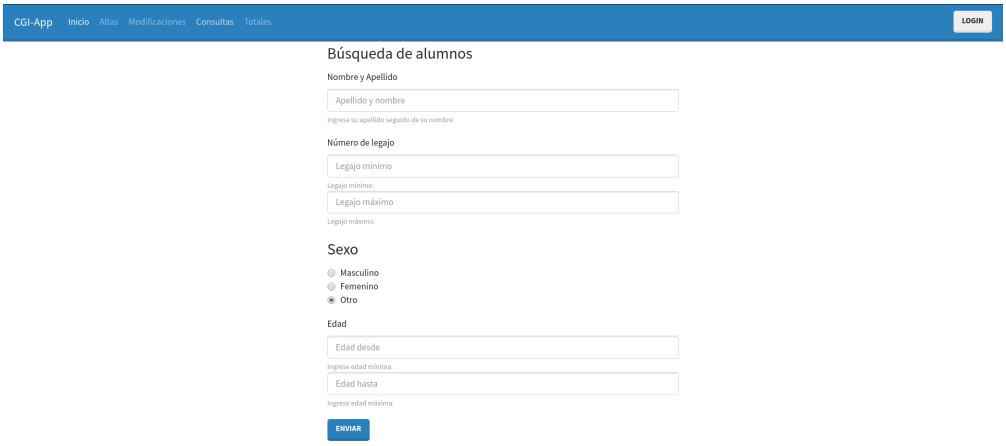
2.2 Vista de la Aplicacion CGI

Utilizamos **bootstrap** y **css** para que la implementacion de la pagina de la aplicacion y los formularios.

2.3 Conclusion

Esta aplicacion que se implemento puede ser considerada una aplicacion web ya que cumple todos sus fundamentos.

El cliente (navegador) para cada llamada a un servicio del servidor realiza una llamada **cgi** hacia el servidor (que son scripts cgi que residen en el servidor)y el mismo responde de acuerdo a la peticion realizada.



3 Agregue a la aplicación del ejercicio 3 cuatro posibilidades de consultas básicas: por nombre y apellido (pueden haber "*"), Número de Alumno/Legajo (puede ser un intervalo), sexo, y edad (puede ser un intervalo). Los resultados deben mostrarse visualmente diferenciados

COMENTARIO: este ejercicio es para que se vea que el programa CGI debería combinar la generación de HMTL con el resultado de lo que se procesa localmente en el servidor. Y que el HTML generado puede ser "largo". El programa CGI puede estar en el lenguaje que se desee.

3.1 Solucion

Para la solucion de este punto, se decidio agregar en la aplicacion un apartado para **consultas** en el cual se despliega un formulario para ingresar el/los filtros para realizar la busqueda.

Como se menciono anteriormente, se utilizo la libreria de python **pandas** para el manejo de los archivos **.csv** para asi poder manupularlos de una manera mas facil y menos rudimentaria.

4 Agregue a la aplicación la consulta de valores totales por rango de edad (0-20, 20-40, mas de 40) y sexo.

COMENTARIO: este ejercicio es para que se vea que la mayor parte del procesamiento en el lado del servidor puede ser solamente para computar valores simples a mostrar, pero que pueden ser resultado de procesamiento relativamente complejo.

4.1 Impleentacion

De la misma forma que se hizo con las consultas del punto anterior, para este caso se agrego un apartado **totales** a nuestra aplicacion en la cual se muestra como resultado una tabla con los totales por rango de edad y sexo.

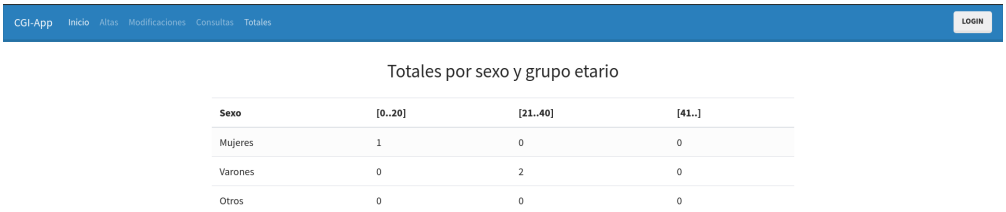
La resolucion tambien fue realizada con la libreria de python **pandas**.

```
def get_totales(cls):
    alumnos = pd.read_csv(MODEL_FILE)

    result = {
        "mujeres1":len(alumnos[(alumnos['sexo'] == "femenino") & \
                               (alumnos['edad'] < 21)]),
        "mujeres2":len(alumnos[(alumnos['sexo'] == "femenino") & \
                               ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
        "mujeres3":len(alumnos[(alumnos['sexo'] == "femenino") & \
                               (alumnos['edad'] > 40)]),
        "varones1":len(alumnos[(alumnos['sexo'] == "masculino") & \
                               (alumnos['edad'] < 21)]),
        "varones2":len(alumnos[(alumnos['sexo'] == "masculino") & \
                               ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
        "varones3":len(alumnos[(alumnos['sexo'] == "masculino") & \
                               (alumnos['edad'] > 40)]),
        "otro1":len(alumnos[(alumnos['sexo'] == "otro") & \
                            (alumnos['edad'] < 21)]),
        "otro2":len(alumnos[(alumnos['sexo'] == "otro") & \
                            ((alumnos['edad'] > 20) & (alumnos['edad'] < 41))]),
        "otro3":len(alumnos[(alumnos['sexo'] == "otro") & \
                            (alumnos['edad'] > 40)]),
    }

    return result
```

4.2 Vista desde la aplicacion

5 Tome el código que se adjunta como Anexo 5 del apunte de AJAX y agregue las siguientes mejora de funcionalidad:

- **a)** Agregar un botón a la ventana principal de chat, a efectos que un usuario ingresante pueda registrarse en el sitio. El usuario ingresará su "Nick" y al pulsar el botón, se registrará en un archivo data/usuarios.txt en el servidor.
- **b)** Producido el registro, se devolverá al usuario una lista total de usuarios registrados



en una pequeña ventana a la derecha de la ventana común de mensajes y también se devolverá toda la conversación entre usuarios registrada hasta ese momento.

- **c)** Intentar establecer la actualización incremental del chat, mediante el almacenamiento en cada cliente de la última línea de chat enviada.

5.1 Diseño e implementación del servidor

5.1.1 Modelos

Se definieron dos modelos que resuelven la lógica de negocio de la aplicación. Por un lado la clase Session (similar a la implementada en el ejercicio anterior) y por otro la clase Message.

La clase Session posee los siguientes atributos:

```
self.id # identificador único autoincremental para manejo del archivo
self.nickname # nickname del usuario
self.cookie # objeto cookie
self.last_msg # identificador del último mensaje entregado al usuario
```

y los siguientes métodos

```
def save(self):
# guarda una sessión en el archivo
def update(self, last_msg):
# actializa el id del último mensaje entregado al usuario
def _new_cookie(self):
# genera una nueva cookie de sesión
def delete_cookie(self):
# elimina una cookie y borra la sesión del registro

@classmethod
def get_users(cls, nickname):
# obtiene la lista de usuarios conectados
@classmethod
```

```
def get_current_session(cls):
# devuelve una instancia de Session que representa la sesión actual
@classmethod
def exists(cls):
# chequea si existe una sesión
```

Los atributos de la clase Message son:

```
self.id # identificador único autoincremental para manejo del archivo
self.user # nickname del usuario que envió el mensaje
self.text # texto del mensaje
```

y tiene los siguientes métodos:

```
def save(self):
# guarda un mensaje en el archivo

@classmethod
def get_messages(cls, last_msg_id=None):
# Obtiene una colección de mensajes, desde last_msg_id en adelante
# o desde 0 si last_msg_id es None

@classmethod
def get_last_msg_id(cls):
# obtiene el id del último mensaje registrado en el archivo
```

5.1.2 Controladores

Los controladores de la aplicación se encuentran definidos en los siguientes archivos:

```
login.py
# maneja peticiones post login
logout.py
# maneja peticiones post logout
messages.py
# maneja peticiones get para obtener mensajes y post para guardarlos
users.py
# maneja peticiones get para obtener los usuarios conectados
main.py
# recibe un get y resuelve mostrar la vista login o chat
# según usuario autenticado
```

5.2 Diseño e implementación del cliente

Similarmente a como se resolvió en el ejercicio anterior, la aplicación tiene una sola página (index.html) y vía ajax se carga el contenido dinámicamente en el div principal (id="frame"):

```
<body>
  <div class="container-fluid">
    <div class="row justify-content-md-center frame">
      <div id="frame">

      </div>
    </div>
  </div>
</body>
```

5.2.1 Controladores del cliente

5.3 Diseño e implementación del servidor

5.3.1 Modelos

Se definieron dos modelos que resuelven la lógica de negocio de la aplicación. Por un lado la clase Session (similar a la implementada en el ejercicio anterior) y por otro la clase Message.

La clase Session posee los siguientes atributos:

```
self.id # identificador único autoincremental para manejo del archivo
self.nickname # nickname del usuario
self.cookie # objeto cookie
self.last_msg # identificador del último mensaje entregado al usuario
```

y los siguientes métodos

```
def save(self):
# guarda una sesión en el archivo
def update(self, last_msg):
# actializa el id del último mensaje entregado al usuario
def _new_cookie(self):
# genera una nueva cookie de sesión
def delete_cookie(self):
# elimina una cookie y borra la sesión del registro

@classmethod
def get_users(cls, nickname):
# obtiene la lista de usuarios conectados
@classmethod
def get_current_session(cls):
# devuelve una instancia de Session que representa la sesión actual
@classmethod
def exists(cls):
# chequea si existe una sesión
```

Los atributos de la clase Message son:

```
self.id # identificador único autoincremental para manejo del archivo
self.user # nickname del usuario que envió el mensaje
self.text # texto del mensaje
```

y tiene los siguientes métodos:

```
def save(self):
# guarda un mensaje en el archivo

@classmethod
def get_messages(cls, last_msg_id=None):
# Obtiene una colección de mensajes, desde last_msg_id en adelante
# o desde 0 si last_msg_id es None

@classmethod
def get_last_msg_id(cls):
# obtiene el id del último mensaje registrado en el archivo
```

5.3.2 Controladores

Los controladores de la aplicación se encuentran definidos en los siguientes archivos:

```
login.py
# maneja peticiones post login
logout.py
# maneja peticiones post logout
messages.py
# maneja peticiones get para obtener mensajes y post para guardarlos
users.py
# maneja peticiones get para obtener los usuarios conectados
main.py
# recibe un get y resuelve mostrar la vista login o chat
# según usuario autenticado
```

5.4 Diseño e implementación del cliente

Similarmente a como se resolvió en el ejercicio anterior, la aplicación tiene una sola página (index.html) y vía ajax se carga el contenido dinámicamente en el div principal (id="frame"):

```
<body>
  <div class="container-fluid">
    <div class="row justify-content-md-center frame">
      <div id="frame">

      </div>
    </div>
  </div>
</body>
```

5.4.1 Controladores del cliente

Las peticiones post a las rutas *cgi-bin/login.py*, *cgi-bin/logout.py* y *cgi-bin/messages.py* son gestionadas por los scripts:

```
login.js
logout.js
send_message.js
```

Su funcioanmiento es similar en los 3 casos.

Al cargarse la página se ejecuta el script definido en *main.js* que lanzará la primera petición get a la ruta *cgi-bin/main.py*:

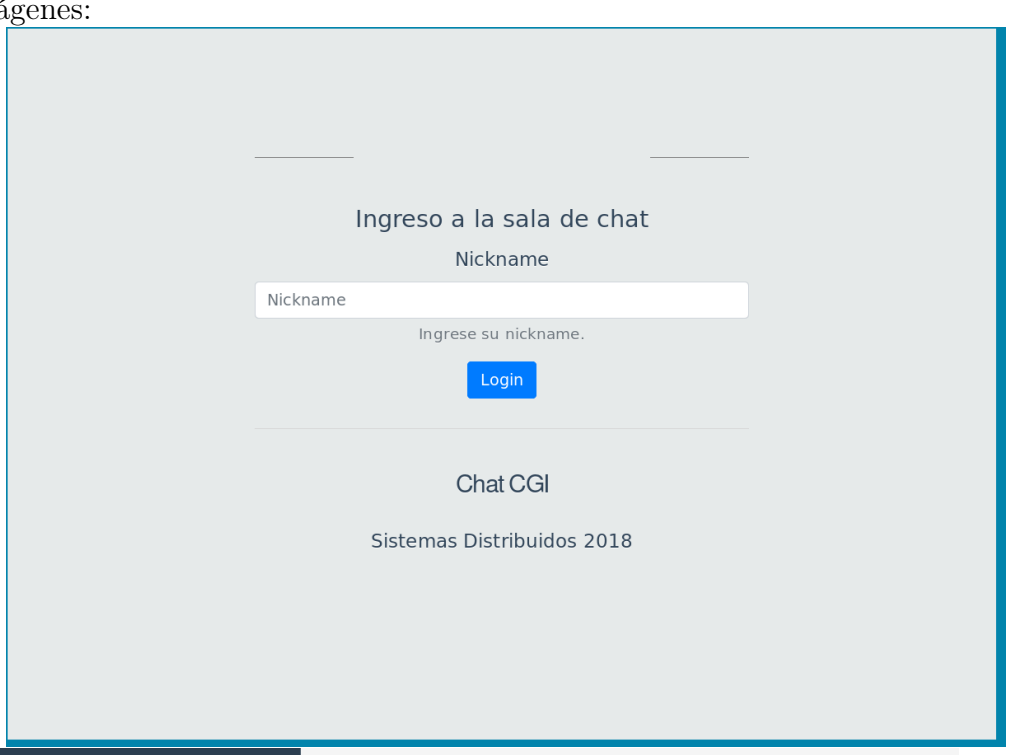
```
$( document ).ready(function(){

    main();
    function main(){
        var container =  ("#frame")
        .ajax({
            method: "GET",
            url: "/cgi-bin/main.py",
        })
        .done(function (res) {
            container.html(res);
        })
        .fail(function (err) {
            container.html(err);
        });
    }
});
```

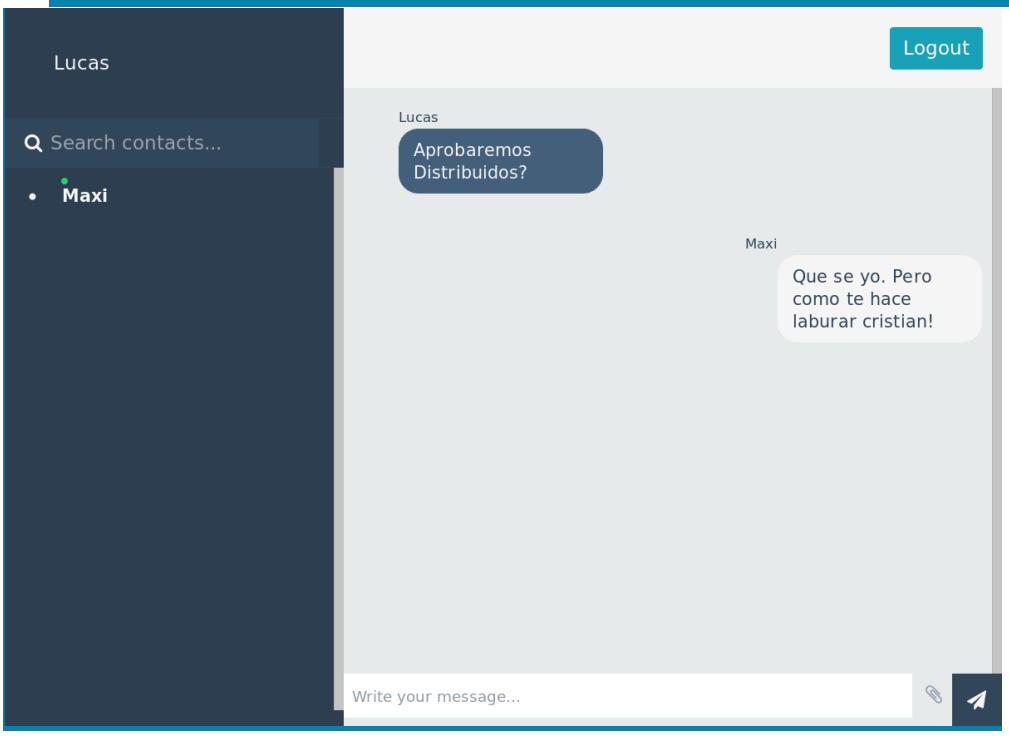
En ese mismo script se verifica si existe una cookie, en caso de existir se intentará obtener del servidor los mensajes y los usuarios conectados (si no está autenticado el usuario, éste devolverá error). Esto último es parte de lo que debe consultarse periódicamente y por eso se define dentro de una función que engloba estas dos tareas y que se ejecuta cada 2 seg.

```
var interval = null;
let cookie = Cookies.get('session');

if (typeof cookie === "undefined"){
    clearInterval(interval);
} else {
    interval = setInterval(updateChat, 2000);
}

function updateChat(){
```

```
    update_messajes();
    update_users();
}
```

El resultado final en términos de vistas de la aplicación puede apreciarse en las siguientes imágenes:





6 DFS:

- **a)** Configurar NFS y efectuar un compartimiento de archivos entre un Servidor y un Cliente Linux. Describir la operatoria en el informe.
- **b)** Configurar Samba y efectuar un compartimiento de archivos entre un Windows y un Cliente Linux. Describir la operatoria en el informe.

6.1 NFS

Un NFS nos permite acceso local a archivos remotos y utiliza una arquitectura estándar cliente-servidor para compartir archivos entre maquinas con SO basados en unix. Aún así, no es necesario que las dos máquinas corran el mismo sistema operativo.

En nuestro caso configuramos NFS para una distribucion ArchLinux.

Como primer paso debemos instalar el paquete **nfs-utils**.

```
sudo pacman -S nfs-utils
```

6.1.1 Configuracion

- Iniciar y abilitar los servicios de NFS

```
systemctl start nfs-server.service
```

```
systemctl enable nfs-server.service
```

Ahora bien, algunos de los archivos más importantes que entran en juego con NFS son (Con sus rutas absolutas en ArchLinux): + /etc/exports: Es el archivo primario de configuración de NFS. Todos los archivos y carpetas exportados que se especifican en este archivo son los que se sirven. + /etc/fstab: Archivo de FS de linux en el que se especifican los puntos de montaje de las diferentes particiones. Se utiliza para definir un mounpoint para el/los directorio/s de NFS (Y que persistan luego de reiniciar el sistema). + /etc/idmapd.conf: Archivo en el que se realiza el mapeo de IDs. Utilizado principalmente para definir el nombre del dominio para que el servicio se configure correctamente.

6.1.2 Mapeo de ID

- Para este paso hay que modificar el archivo de configuracion **/etc/idmapd.conf** y establezca el campo **Domain** con el nombre de dominio.
- Seguidamente, creamos el recurso que vamos a servir. En este caso es una carpeta que ubicaremos en **/srv/nfs4/datos**

```
sudo mkdir -p /srv/nfs4/datos
```

- Luego creamos la carpeta del sistema de archivos del servidor en donde será montado el recurso a compartir:

```
# Directorio NFS
/mnt/datos                              /srv/nfs4/datos none          bind                    0 0
```

```
# /etc/exports - exports(5) - directories exported to NFS clients
#
# Example for NFSv2 and NFSv3:
#  /srv/home        hostname1(rw,sync) hostname2(ro,sync)
# Example for NFSv4:
#  /srv/nfs4        hostname1(rw,sync,fsid=0)
#  /srv/nfs4/home   hostname1(rw,sync,nohide)
# Using Kerberos and integrity checking:
#  /srv/nfs4        *(rw,sync,sec=krb5i,fsid=0)
#  /srv/nfs4/home   *(rw,sync,sec=krb5i,nohide)
#
# Use `exportfs -arv` to reload.

/srv/nfs4/ 192.168.0.1/24(rw,fsid=root,no_subtree_check)
/srv/nfs4/datos 192.168.0.1/24(rw,no_subtree_check,nohide)
```

```
sudo mkdir /mnt/datos
```

Se deben otorgar permisos de lectura/escritura al directorio **datos** para que los clientes puedan escribir en él.

- Luego, como siguiente paso es montar el directorio que desea compartir, en este caso /**mnt**/**datos**, en el directorio compartido de NFS a través de la instrucción de montaje.

```
mount --bind /mnt/datos /srv/nfs4/datos
```

Para hacer que los cambios sean permanentes en cada reinicio del servidor, añada el enlace de montaje a **fstab**

6.1.3 Exportacion

Como siguiente paso, se añaden los directorios que se compartirán y la dirección IP o nombre de servidor de las máquinas cliente a las que les estará permitido montarlas en **exports**:

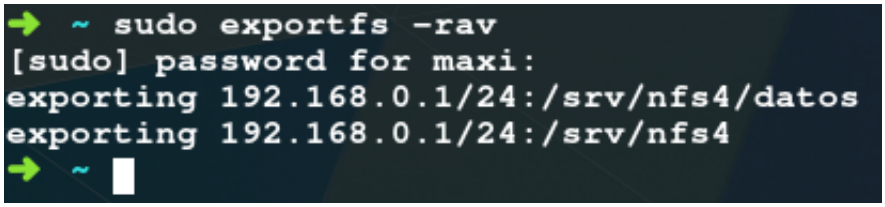
Si se modifica /**etc**/**exports** mientras el servidor está funcionando, debe volver a exportarlos para que los cambios surtan efecto.

```
exportfs -rav
```

Finalmente, para levantar el servidor, debemos arrancar los servicios **rpcbind** y **nfs-client.target**.

```
systemctl start rpcbind nfs-client.target
```

Ahora desde el lado del **Cliente**, tambien necesitan el modulo **nfs-utils** para poder conectarse.



6.1.4 Montaje en Linux

Para mostrar los sistemas de archivos exportados por el servidor:

```
showmount -e servername
```

A continuación se monta omitiendo la raíz de exportación del servidor NFS:

```
mount -t nfs4 servername:/datos /mountpoint/on/client
```

6.2 SAMBA

Como paso inicial debemos instalar el paquete de **samba**.

```
sudo pacman -S samba
```

Samba se configura en el /**etc**/**samba**/**smb.conf**.

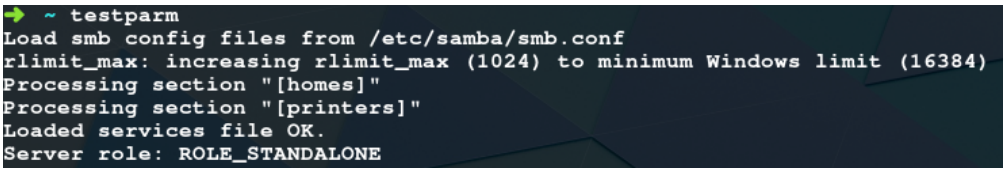
Debido a que el paquete samba no proporciona este archivo, se debe crear antes de iniciar samba.

Utilizamos un ejemplo documentado **smb.conf.default** desde el repositorio de git de **Samba** para la configuración /**etc**/**samba**/**smb.conf**.

Link: https://git.samba.org/samba.git/?p=samba.git;a=blob_plain;f=examples/smb.conf.default;hb=HEAD

- Siempre que modifique el smb.confarchivo, es recomendable ejecutar el comando **testparm** para verificar si hay errores sintácticos.

```
testparm
```



6.2.1 Gestión de usuarios

Samba requiere una cuenta de usuario de Linux. Puede usar una cuenta de usuario existente o crear una nueva. Aunque el nombre de usuario se comparte con el sistema Linux, Samba usa una contraseña separada de la de las cuentas de usuario de Linux. Reemplazar **samba_usercon** la cuenta de usuario Samba elegida:

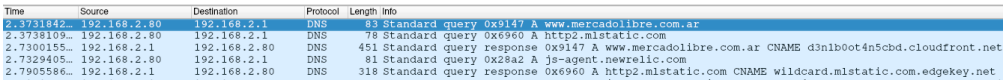
```
smbpasswd -a samba_user
```

Luego, del lado del cliente (Workgroup del equipo): Debemos asegurarnos que los servicios smbd y nmbd estén corriendo (Ya sea con START o ENABLE). Esto inicia la escucha del servidor.

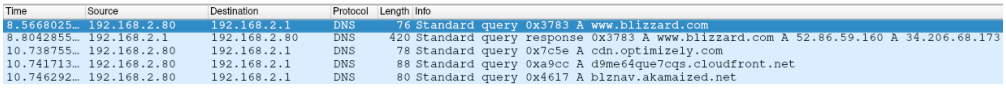
Ahora es necesario crear el recurso compartido. Hacemos una carpeta en **/var/lib/samba** llamada **usershare** (Quedando como: **/var/lib/samba/usershare**).

Ahora vamos a modificar los permisos sobre esa carpeta. Típicamente lo que se hace es crear un grupo de usuarios que se puede llamar **sambashare** y se le asignan permisos de lectura/escritura sobre el recurso compartido. Paso a paso tenemos:

- Creamos el grupo sambashare con el comando **groupadd -r sambashare**
- Cambiamos el grupo dueño de la carpeta compartida con el comando "chown root:sambashare /var/lib/samba/usershare".



Time	Source	Destination	Protocol	Length	Info
2.3731842…	192.168.2.80	192.168.2.1	DNS	83	Standard query 0x9147 A www.mercadolibre.com.ar
2.3738109…	192.168.2.80	192.168.2.1	DNS	78	Standard query 0x6960 A http2.mlstatic.com
2.7300155…	192.168.2.1	192.168.2.80	DNS	451	Standard query response 0x9147 A www.mercadolibre.com.ar CNAME d3n1b0ot4n5cbd.cloudfront.net
2.7329405…	192.168.2.80	192.168.2.1	DNS	81	Standard query 0x28a2 A js-agent.newrelic.com
2.7905586…	192.168.2.1	192.168.2.80	DNS	318	Standard query response 0x6960 A http2.mlstatic.com CNAME wildcard.mlstatic.com.edgekey.net



Time	Source	Destination	Protocol	Length	Info
8.5668025…	192.168.2.80	192.168.2.1	DNS	76	Standard query 0x3783 A www.blizzard.com
8.8042855…	192.168.2.1	192.168.2.80	DNS	420	Standard query response 0x3783 A www.blizzard.com A 52.86.59.160 A 34.206.68.173
10.738755…	192.168.2.80	192.168.2.1	DNS	78	Standard query 0x7c5e A cdn.optimizely.com
10.741713…	192.168.2.80	192.168.2.1	DNS	88	Standard query 0xa9cc A d9me64que7cqs.cloudfront.net
10.746292…	192.168.2.80	192.168.2.1	DNS	80	Standard query 0x4617 A blznav.akamaized.net

- Cambiamos los permisos a nivel de FS sobre la carpeta compartida con el comando "chmod 770 /var/lib/samba/usershare"
- En el archivo de configuración de samba (smb.conf) debemos agregar el recurso compartido. Un posible ejemplo de esto podría ser:

```
[global]
 usershare path = /var/lib/samba/usershare
 usershare max shares = 100
 usershare allow guests = yes
 usershare owner only = yes
```

Finalmente, agregamos el usuario al grupo **sambashare** con el comando **gpasswd sambashare -a usuario**. Luego de esto, es necesario reiniciar la sesión del SO para que se apliquen los cambios.

```
gpasswd sambashare -a usuario
```

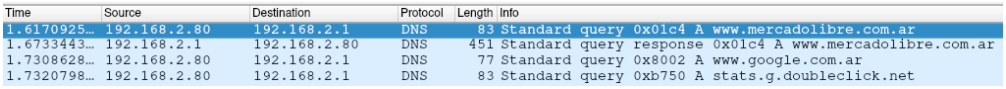
7 DNS:

Objetivo: Observar las llamadas implícitas a DNS y ponderar tiempos de acceso. Efectuar el siguiente experimento, verificando información con la ayuda de un analizador de protocolo:

- **a)** Desde el browser haga una conexión a una página WEB de argentina, que haya utilizado hace mucho tiempo o que no haya utilizado (un diario, un blog, etc.). Obtenga la diferencia temporal entre la consulta que se observa en el analizador y su respuesta.

- **b)** Haga lo mismo con una página no frecuentada por Ud. que se supone que está situada en un servidor europeo o de Asia. (la página oficial del grupo Nightwish, o algún diario galés, etc.). Compare sus conclusiones con las obtenidas en el caso a).

Se puede ver claramente que que el tiempo de respuesta es mucho mayor a la respuesta a un servidor nacional (como el del anterior caso).



Time	Source	Destination	Protocol	Length	Info
1.6170925…	192.168.2.80	192.168.2.1	DNS	83	Standard query 0x01c4 A www.mercadolibre.com.ar
1.6733443…	192.168.2.1	192.168.2.80	DNS	451	Standard query response 0x01c4 A www.mercadolibre.com.ar
1.7308628…	192.168.2.80	192.168.2.1	DNS	77	Standard query 0x8002 A www.google.com.ar
1.7320798…	192.168.2.80	192.168.2.1	DNS	83	Standard query 0xb750 A stats.g.doubleclick.net

- **c)** Vuelva nuevamente a invocar la página seleccionada en a), (que se supone ya reside en su caché). Compare resultados.

Se puede ver la significativa mejora en el tiempo de respuesta de la peticion, esto es porque la pagina reside en nuestra cache. Siempre que una pagina resida en la cache, la respuesta sera mucho mas rapida.

- **d)** Aporte las conclusiones que puede sacar del experimento.

Como pudimos observar en las distintas experiencias, es más rápido resolver una consulta de una página nacional (www.lanacion.com.ar) que una europea (www.fortnite.com). Sin embargo los tiempos de respuesta cambian favorablemente cuando se encuentran cacheadas las páginas, las respuestas son significativamente mas rapidas.