ALUMNO: Cuesta Alario David **DNI:** 20857516-N

Sistemas Web

Hypertext Transfer Protocol HTTP

HTTP es un protocolo de nivel de aplicación originalmente diseñado para la transferencia de recursos de tipo hipertexto entre un cliente y un servidor.

Este protocolo define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores) para comunicarse.

Características:

No guarda información sobre conexiones anteriores.

Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor:

- El cliente realiza una petición enviando al servidor un mensaje con cierto formato.
- El servidor le envía un mensaje de respuesta.

Da soporte directo a aplicaciones para el envío y recepción de datos a través de la World Wide Web **WWW**

Referencia un recurso mediante un identificador único denominado Universal Resource Identifier **URI**

Consta de una base de datos donde se almacena un identificador URL que indica la dirección donde está la información

Flexibilidad para añadir nuevos métodos que aporten nuevas funcionalidades

Objetivos

- **Sencillez** con la finalidad de facilitar la lectura por un humano
- **Extensibilidad** con la finalidad de construir una nueva versión que sea retro compatible

Estructura

Línea inicial

Termina con retorno y un salto de línea.

Contienen la siguiente información:

- Para las peticiones
  - La acción requerida por el servidor
  - La URL del recurso
  - La versión HTTP que soporta el cliente
- Para respuestas encontramos:
  - La versión del HTTP
  - El código de respuesta que indica que ha pasado con la petición
  - La URL del recurso asociada a dicho retorno.

Las cabeceras del mensaje

Contiene los metadatos que se envían en las peticiones y respuesta permitiendo proporcionar información esencial sobre la transacción en curso

Aportan mucha flexibilidad al protocolo permitiendo añadir nuevas funcionalidades sin tener que cambiar la base

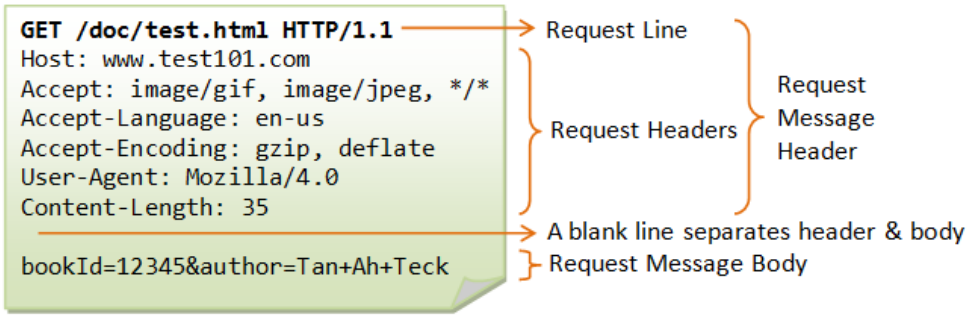
El formato de las cabeceras es el siguiente:

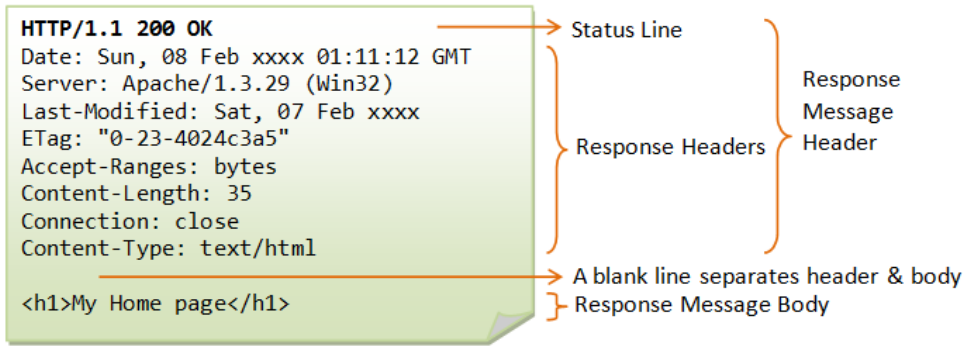
- Cada cabecera es especificada por un nombre de cabecera seguido de dos puntos, un espacio en blanco y el valor de dicha cabecera seguida por un retorno de carro y un salto de línea.
- El bloque de cabeceras termina con una línea en blanco que permite identificar el cuerpo del mensaje (Si no hay cabeceras la línea en blanco debe permanecer)

Cuerpo del mensaje

Su presencia es opcional, depende de la línea anterior del mensaje y del tipo de recurso

Contiene los datos que se intercambian entre el cliente y el servidor.





Sintaxis

Métodos de petición

El protocolo HTTP fine una serie de métodos de petición que pueden utilizarse

Cada método indica la acción que desea que se efectúe sobre el recurso identificado

Lo que este recurso representa depende de la aplicación del servidor

- **GET**: Pide una representación del recurso especificado agregando parámetros a la URL.
- **HEAD**: Pide una representación del recurso especificado, pero en la respuesta no se devuelve el cuerpo.
  - Esto es útil para recuperar los metadatos de los encabezados de respuesta sin transportar el contenido.
- **POST**: Envía los datos para que sean procesados por el recurso identificado.
  - Los datos se incluirán en el cuerpo de la petición.
  - Esto puede resultar en la creación de un nuevo recurso y/o de las actualizaciones de los recursos existentes
- **PUT**: carga un recurso especificado en una conexión socket establecida con el servidor
- **PATCH**: carga un recurso especificado en una conexión socket establecida con el servidor permitiendo escoger parcialmente una o varias partes.
- **DELETE**: Borra el recurso especificado.
- **TRACE**: Este método solicita al servidor que en la respuesta meta todos los datos que reciba en el mensaje de petición.
  - Se utiliza con fines de depuración y diagnóstico ya que el cliente puede ver lo que añaden al mensaje los servidores intermedios
- **OPTIONS**: Devuelve los métodos HTTP que el servidor soporta para un URL específico.
  - Se utiliza para comprobar la funcionalidad de un servidor web mediante petición.
- **CONNECT**: Se utiliza para saber si se tiene acceso a un host
  - se utiliza para saber si un proxy nos da acceso a un host bajo condiciones especiales.

Códigos de respuesta

Se trata de un número que permite identificar fácilmente el tipo de respuesta que ha generado el servidor

El resto del contenido de la respuesta dependerá del valor de este código.

- **1XX** Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- **2XX** Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- **3XX** Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- **4XX** Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición debido a que no era correcta o estaba incompleta.
- **5XX** Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

Cabeceras de los mensajes

- Cabeceras que indican las capacidades aceptadas por el que envía el mensaje:
  - $Acept$: indica el MIME aceptado
  - $Accept - Charset$: indica el código de caracteres aceptado
  - $Accept - Encoding$: indica el método de compresión aceptado
  - $Accept - Language$: indica el idioma aceptado
  - $User - Agent$: para describir al cliente
  - $Server$: indica el tipo de servidor
  - $Allow$: métodos permitidos para el recurso
- Cabeceras que describen el contenido:
  - $Content - Type$: indica el MIME del contenido
  - $Content - Length$: longitud del mensaje
- Cabeceras que hacen referencias a las URIs:
  - $Location$: indica donde está el contenido
  - $Referer$: Indica el origen de la petición
- Cabeceras para control de cookies:
  - $Cookie$: Indica el soporte de cookies
- Cabeceras para describir la comunicación:
  - $Host$: indica máquina destino del mensaje
  - $Connection$: indica qué hacer con la conexión TCP una vez transferida la respuesta (cerrarla, mantenerla)

Funcionamiento

Petición y Respuesta a través de un navegador

Cuando se solicita recurso mediante una URI a través del navegador:

- En primer lugar el navegador analiza cada una de sus partes atendiendo al esquema de la URI
- A continuación el navegador solicita al sistema operativo la resolución del nombre de host del servidor.
- El sistema operativo resuelve esta solicitud a través del servidor DNS y devuelve la dirección IP del servidor al navegador.
- Con este dato, el navegador solicita al sistema operativo el establecimiento de una conexión TCP desde un puerto local al puerto del servidor. Cuando se establece la conexión manda un mensaje de petición con el del protocolo HTTP
- Cuando el servidor web recibe la petición, analiza el método y la URI para saber si el recurso existe y se le puede aplicar la acción solicitada. En caso afirmativo se analizaran las cabeceras de la petición para devolver la versión del recurso que mejor se adapte a las necesidades del cliente.
- Finalmente el servidor mandara un mensaje de respuesta con el formato del protocolo HTTP

```
Librerias
# -*- coding: utf-8 -*-
import httplib
Parametros de la conexión
servidor = 'www.google.com'
conn = httplib.HTTPConnection(servidor, '80')
Realizando la conexión
conn.connect()
print "     Local IP address is " + str(conn.sock.getsockname()[0])
print "     Local TCP port is " + str(conn.sock.getsockname()[1])
Parametros de la petición
metodo = 'GET'
recurso = '/'
cabeceras_peticion = {'Host': servidor,'User-Agent': 'Cliente Python',}
cuerpo_peticion = ''
Realizando la petición
conn.request(metodo, recurso, headers=cabeceras_peticion, body=cuerpo_peticion)
Recibiendo respuesta
response = conn.getresponse()
print "     STATUS: " + str(response.status)
Cerrar la conexión
conn.close();
```

Redirección de una petición

Cuando el servidor web que atiende la petición HTTP y detecta que el recurso solicitado ha cambiado de dirección o existe un servidor web en el que la petición va a poder ser atendida de forma más eficiente puede redirigir al cliente a otra URI mediante los códigos de respuesta 3XX y la cabecera “Location”

Cuando el cliente detecta un código de respuesta 3XX extrae el contenido de la cabecera “Location” y realiza una nueva petición a la URI que se encuentra en dicho campo.

En un navegador, este proceso ocurre de forma transparente para el usuario

```
Librerias
#-*- coding: UTF-8 -*-
import httplib
Parametros de la Primera conexión
servidor = 'www.ehu.es'
conn = httplib.HTTPConnection(servidor)
Realizando la conexión
conn.connect()
print "     Local IP address is " + str(conn.sock.getsockname()[0])
print "     Local TCP port is " + str(conn.sock.getsockname()[1])
Parametros de la petición
metodo = 'GET'
recurso = '/'
cabeceras_peticion = {'Host': servidor,'User-Agent': 'Cliente Python'}
cuerpo_peticion = ''
Realizando la petición
conn.request(metodo, recurso, headers=cabeceras_peticion, body=cuerpo_peticion)
Recibiendo respuesta
respuesta = conn.getresponse()
print "     Status: " + str(respuesta.status)
location = respuesta.getheader("location")
print "     Location: " + location
Obtenemos la nueva dirección
array = location.split('/')
print "     " + str(array)
servidor = array[2]
Cerrar la conexión
conn.close();
```

```
Parametros de la Segunda conexión
conn = httplib.HTTPConnection(servidor)
conn.connect()
Realizando la conexión
conn.connect()
print "     Local IP address is " + str(conn.sock.getsockname()[0])
print "     Local TCP port is " + str(conn.sock.getsockname()[1])
Parametros de la petición
params = {'image': 'websystemsikungfu'}
params_encoded = urllib.urlencode(params)
metodo = 'GET'
recurso = '/downloadImage'
cabeceras_peticion = {'Host': servidor}
cuerpo_peticion = ''
Realizando la petición
conn.request(metodo, recurso, headers=cabeceras_peticion, body=cuerpo_peticion)
Recibiendo respuesta
respuesta = conn.getresponse()
print "STATUS: " + str(respuesta.status)
print "CONTENT: " + respuesta.read()
location = respuesta.getheader("location")
print" location = " + location
Cerrar la conexión
conn.close();
```

Adjuntar datos en una petición

El cliente puede enviar diferentes tipos de datos a un servidor web:

Formato formulario

Permite el envío de datos alfanuméricos en forma de pares nombre-valor.

Cada par debe ir debidamente codificado y formateado tanto en el cuerpo del mensaje como en la URI (en GET y en POST)

- Para realizar la codificación se empleara la función: $Content - Type: application/x - www - form - urlencoded$
- Los caracteres reservados y los no recogidos en el alfabeto US-ASCII debe codificarse en dos pasos antes formar la URI
- El formato de los pares será: $nombre = valor$ utilizando el símbolo "&" como separador entre pares

Método POST

- Se necesitan añadir las siguientes cabeceras para caracterizar el cuerpo del mensaje
  - $Content - Type$: $application/x - www - form - urlencoded$ que indica el formato de la codificación
  - $Content - Length: XXX$ que Indica la longitud del cuerpo del mensaje
- Los pares nombre-valor se añadirán en el cuerpo del mensaje debidamente codificados y formateados.

POST /app/servlet/contactUs HTTP/1.1
Host: www.google.es
Accept: text/html
Accept-Language: en-US,en
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 30

name=XXX&email=YYY&message=ZZZ

Método: GET

- No hace falta indicar las cabeceras $Content - Type$ y $Content - Length$ ya que el cuerpo del mensaje va vacío.

  Los pares nombre-valor se añadirán en la URI debidamente codificados y formateados precedidos el símbolo "?"

GET /app/servlet/contactUs?name=XXX&email=YYY&message=ZZZ HTTP/1.
Host: www.google.es
Accept: text/html
Accept-Language: en-US,en
Connection: keep-alive

Librerias
```
import httplib
import urllib
```
Parametros de la conexión
```
server = 'tic-investigacion-1.appspot.com'
conn = httplib.HTTPConnection(server)
```
Establecer la conexión
```
conn.connect()
```
Parametros de la petición
```
params = {'nan': '12345678'}
params_encoded = urllib.urlencode(params)
uri = '/processForm' + "?" + params_encoded
headers = {'Host': server,}
cuerpo_peticion = ''
```
Establecer la petición
```
conn.request('GET', uri, headers=headers, body=cuerpo_peticion)
```
Procesamos la respuesta
```
respuesta = conn.getresponse()
print "STATUS: " + str(respuesta.status)
print "CONTENT: " + respuesta.read()
```
Cerrar la conexión
```
conn.close();
```

Librerias
```
import httplib
import urllib
```
Parametros de la conexión
```
server = 'tic-investigacion-1.appspot.com'
conn = httplib.HTTPConnection(server)
```
Establecer la conexión
```
conn.connect()
```
Parametros de la petición
```
params = {'nan': '12345678'}
params_encoded = urllib.urlencode(params)
uri = '/processForm'
headers = {'Host': server,
           'Content-Type': 'application/x-www-form-urlencoded
           'Content-Length': str(len(params_encoded))}
cuerpo_peticion = params_encoded
```
Establecer la petición
```
conn.request('POST', uri, headers=headers, body=cuerpo_peticion)
```
Procesamos la respuesta
```
respuesta = conn.getresponse()
print "STATUS: " + str(respuesta.status)
print "CONTENT: " + respuesta.read()
```
Cerrar la conexión
```
conn.close();
```

Formato binario

Formato JSON o XML

Codificación del cuerpo del mensaje

Tipos de datos

Texto Simple $(US-ASCII \quad Latin-1 \quad UTF-8)$

- Por lo general el contenido de una respuesta HTTP es texto: HTML, XML, JSON o CSS.
- Independientemente de su formato, el texto tiene buena capacidad de compresión.
- El texto puede ser visualizado en multitud de programas, pero no todos utilizan la misma configuración

Ficheros Binarios $(.pdf \quad .jpg \quad .docx)$

- El contenido binario sólo tiene sentido para el tipo de aplicación que lo creó. Por lo que si se intenta interpretar con una aplicación distinta se envía a la salida estándar "stdout"

El servidor web puede indicar a través de la cabecera "$Content\text{-}Type$" el tipo de contenido que está devolviendo

$$Content\text{-}Type: text/html;\ charset = ISO\text{-}8859\text{-}1$$

Compresión de datos

El protocolo HTTP soporta varios tipos de algoritmos de compresión para el cuerpo del mensaje.

El algoritmo de compresión se indica en dos cabeceras:

- En la petición el cliente puede indicar soporta compresión **"**$Accept\text{-}Encoding$"
  - En caso negativo el valor de esta cabecera debe ser "$identity$"
- En la respuesta el servidor indica el algoritmo mediante el que ha comprimido los datos **"**$Content\text{-}Encoding$**"**
  - Si el cuerpo del mensaje se envía sin comprimir, esta cabecera no se debe introducir en la respuesta.

La versión comprimida del recurso debe tener un valor de "$ETag$" distinto al de la versión del recurso sin comprimir

Segmentación del cuerpo del mensaje

Un cliente web que está leyendo una respuesta enviada desde un servidor necesita saber dónde termina el cuerpo del mensaje. El protocolo HTTP proporciona la cabecera "$Content\text{-}Length$" que permite al servidor especificar el tamaño del mensaje.

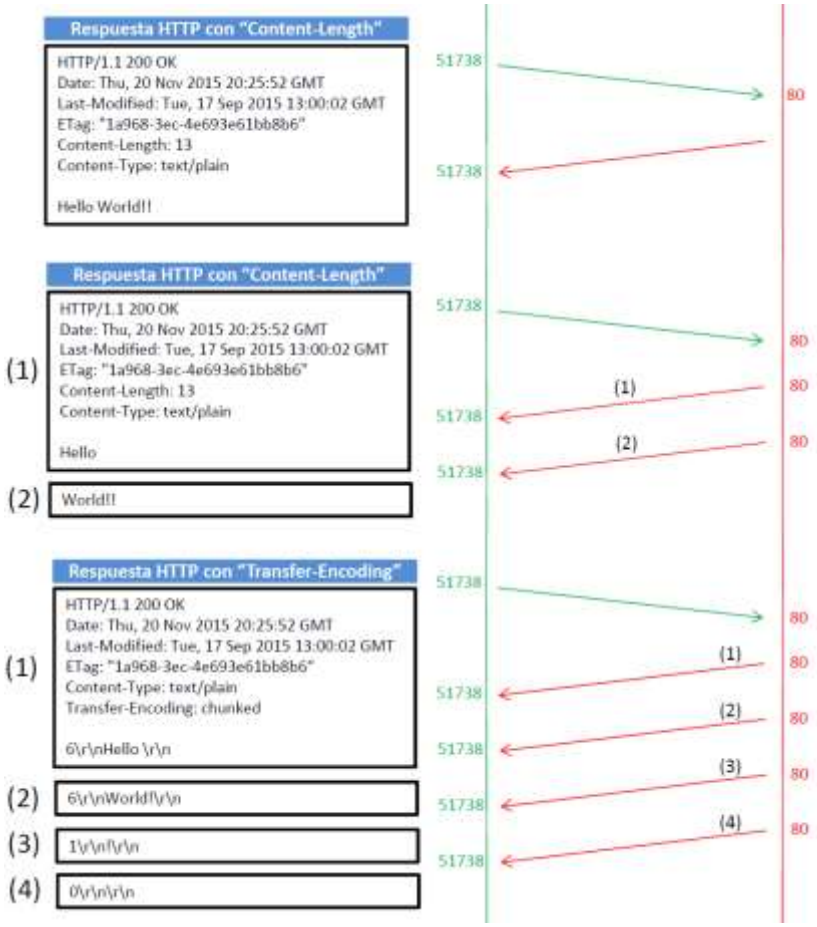
No obstante hay ocasiones en las que se debe devolver una gran cantidad de datos en la respuesta, pero no es posible conocer el tamaño de esos datos hasta que la petición ha sido procesada en su totalidad. El protocolo HTTP proporciona un mecanismo para que el **s**ervidor web pueda enviar partes del contenido solicitado a medida que procesa la respuesta. Mediante la cabecera" "$Transfer\text{-}Encoding$**"** con valor "$chunked$".

"$Content\text{-}Length$**"**

- Los mensajes no se empiezan a enviar hasta que se ha procesado toda la procesado respuesta
- el troceado de los mensajes lo hace la capa TCP

"$Transfer\text{-}Encoding$**"**

- Los mensajes se envían según se van procesando los trozos
- el troceado de los mensajes o hace la capa HTTP.



Caché

Con la finalidad de optimizar el uso del ancho de banda y evitar sobrecargar el servidor web con el procesamiento de peticiones o la generación de respuestas redundantes el protocolo HTTP dispone de mecanismos para evitar la transferencia de recursos cuando éstos no han sido modificados desde la última vez que el cliente los solicitó. Caduque

Cuando un cliente realiza una petición para obtener cierto recurso, el servidor se lo facilita añadiendo cierta meta información mediante la cabecera "***Cache-Control***". Esta cabecera permite establecer un margen temporal durante el cual si el cliente vuelve a solicitar el mismo recurso se accederá al fichero ya existente.

Una vez transcurrido el periodo establecido el recurso en cache caduca. Por lo que si el cliente vuelve a solicitar el recurso tendrá que volverlo a solicitar.

También se pueden utilizar las cabeceras "***If-Modified-Since***" y "***Last – Modified***" para conocer si el recurso se ha modificado desde la última vez que se le solicito.

- Si el contenido ha cambiado se volverá a descargar el recurso con un nuevo valor del "*Cache-Control*"
- Si el contenido no ha cambiado el servidor generará una respuesta con código "*304 Not Modified*" y se actualizara él "*Cache-Control*" del recurso que estaba en la cache

Petición HTTP

```
GET /image.jpg HTTP/1.1
Host: sw2016.com:8080
Accept: image/*
User-Agent: Mozilla Windows Escritorio
```

Respuesta HTTP

```
HTTP/1.1 200 OK
Date: Thu, 20 Nov 2015 20:25:52 GMT
Last-Modified: Tue, 17 Sep 2015 13:00:02 GMT
ETag: "1a968-3ec-4e693e61bb8b6"
Cache-Control: max-age=2592000
Content-Length: 12405
Content-Type: image/jpg

CONTENIDO DE LA IMAGEN
```

Peticion despues de que el recurso caduque

```
GET /image.jpg HTTP/1.1
Host: sw2016.com:8080
Accept: image/*
If-Modified-Since: Tue, 17 Sep 2015 13:00:02 GMT
If-None-Match: "1a968-3ec-4e693e61bb8b6"
User-Agent: Mozilla Windows Escritorio
```

Respuesta despues de que el recurso caduque

```
HTTP/1.1 304 Not Modified
Date: Thu, 20 Jan 2016 20:25:52 GMT
Cache-Control: max-age=2592000
```

En la cabecera "*If-None-Match*" se introduce el valor de la cabecera "*ETag*" devuelto en la respuesta.

Herramientas externas

Universal Resource Identifier URI

Es una cadena de caracteres US-ASCII que permite identificar un recurso en Internet. Su sintaxis es la siguiente:

ESQUEMA : ***PARTE JERARQUICA*** ? ***CONSULTA*****#** ***FRAGMENTO***

URI = scheme ":" *"//" authority ["/" path]* ["?" *query*] ["#" *fragment*]

- **ESQUEMA (SCHEME)**
  - El esquema identifica el **protocolo** que se utiliza.
  - Aunque el protocolo más común es sin duda el **http**, existe una gran variedad de esquemas registrados.
  - **Esquemas registrados**: cid, data, dav, fax, **ftp**, file, gopher, **http**, **https**, imap, ldap, **mailto**, mid, news, nfs, nntp, pop, pres, sip, sips, snmp, tel, telnet, urn, wais, xmpp.

- **PARTE JERÁRQUICA (//authority ["/" path])**
  - Contiene **información del dominio o IP** para **acceder al servidor (authority)** y la **ruta en el servidor (path)** para **acceder al recurso. authority** → egela.ehu.eus ; **path** → /course/view.php
  - Las dos barras inclinadas al principio **(//)** indican que la dirección debe ser pasada al recurso para que éste la interprete.
  - El servidor puede ser un IP o un nombre de dominio (como en el ejemplo) y puede llevar **parámetros** como el **puerto** e información de control de acceso, como en este caso:
    `http://usuario:clave@miembros.sitio.com:80/`
  - Existen caracteres **reservados** que se utilizan para delimitar la dirección y su uso será interpretado como tal. Los caracteres reservados: / : ? # [ ] @ ! $ & ' ( ) * + , ; =
  - Los caracteres **no reservados** se pueden utilizar en la construcción de la ruta son: letras, números y los caracteres **- . _ ~**

- **CONSULTA (query)**
  - La solicitud o consulta indican **variables que se pasan al recurso Web.**
  - Está **separada de la ruta** mediante el signo de interrogación "?" y termina donde empieza el fragmento delimitado por el carácter # si existe.
  - Dentro de la cadena de consulta, el símbolo "+" se reserva como una abreviación del espacio (" "). Por lo tanto, para colocar un signo "+" tal cual debe codificarse

- **FRAGMENTO (*fragment*)**
  - Permite indicar una subdirección dentro del recurso al que apunta la dirección. Esta delimitado por el carácter numeral (#) y se extiende hasta donde se termina el URI.

- Uso de mayúsculas y minúsculas, de acuerdo al estándar STD66, las partes de un URI se comportan de diferente forma:
  - El **esquema y el dominio** son insensibles a mayúsculas y por lo tanto se generalizan como minúsculas.
  - La **ruta** en cambio sí es sensible, al igual que la solicitud y el fragmento.
  - Depende del servidor, puede estar configurado para normalizar todo a minúsculas. Sin embargo, la mayoría de los servidores, en particular Apache, respetan el uso de mayúsculas en las partes apropiadas.

La URI http://myapp.appspot.com/html/form.html devuelve un documento HTML que contiene un formulario cuyo atributo $action$ tiene el valor $/servlet/signin$ por lo que los datos recogidos en el formulario se envían a la URI: http://myapp.appspot.com/servlet/signin

Cookies

El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Pero el protocolo HTML no lo permite

Las cookies son unos paquetes de información enviadas por un sitio web y almacenadas en el navegador del usuario por un tiempo indeterminado, de manera que el sitio web puede consultar la actividad previa del usuario

Sus principales funciones son:

- Conseguir información sobre los hábitos de navegación del usuario e intentos de spyware por parte de agencias de publicidad y otros.
- Llevar el control de usuarios: Cuando un usuario introduce su nombre de usuario y contraseña, el servidor envía una cookie que el cliente puede almacenar para reenviarla en las siguientes peticiones para que no tenga que estar introduciendo el usuario y la contraseña para cada página del servidor.

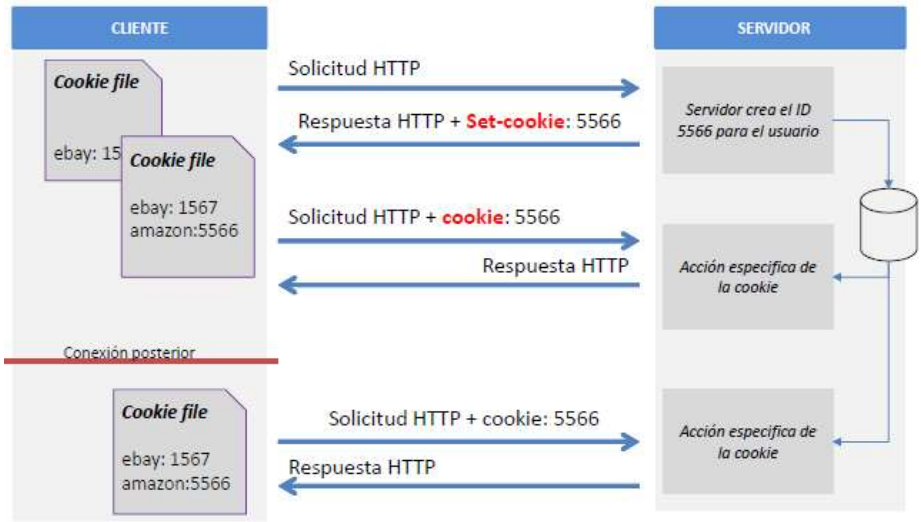
Puede causar problemas de privacidad

Peticion HTTP

```
GET / HTTP/1.1
Host: egela1516.ehu.eus
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20100101 Firefox/33.0
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Respuesta HTTP

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Feb 2016 10:47:36 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: MoodleSessionmdl1516prod=tcfhknala0lrhn3tvos404lpi4;
path=/
Content-Language: es
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-
check=0, no-transform
Pragma: no-cache
Expires: Mon, 20 Aug 1969 09:23:00 GMT
Last-Modified: Fri, 12 Feb 2016 10:47:36 GMT
X-Frame-Options: sameorigin
Content-Encoding: gzip

DATOS
```



Peticion HTTP

```
GET /login/index.php HTTP/1.1
Host: egela1516.ehu.eus
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:33.0)
Gecko/20100101 Firefox/33.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://egela1516.ehu.eus/
Cookie: MoodleSessionmdl1516prod=tcfhknala0lrhn3tvos404lpi4
Connection: keep-alive
```

Respuesta HTTP

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Feb 2016 10:47:45 GMT
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Keep-Alive: timeout=20
Vary: Accept-Encoding
X-Powered-By: PHP/5.3.3
Content-Language: es
Content-Script-Type: text/javascript
content-style-type: text/css
x-ua-compatible: IE=edge
Cache-Control: private, pre-check=0, post-check=0, max-age=0, no-
transform
Pragma: no-cache
X-Frame-Options: sameorigin
Content-Encoding: gzip

DATOS
```

Librerias

```
# -*- coding: utf-8 -*-
import requests
import getpass
```

Conexión a la pagina de registro

```
uri = 'https://egela1819.ehu.eus/login/index.php'

LDAP_usuario = raw_input("Insert your LDAP username: ")
LDAP_password = getpass.getpass("Insert your LDAP password: ")

cuerpo_peticion ={'username': LDAP_usuario,'password': LDAP_password}
response = requests.post('https://egela1819.ehu.eus/login/index.php/', data=cuerpo_peticion)
```

Recibiendo la respuesta

```
print "     STATUS: " + str(response.status_code)

f = open("egela.html", 'w')
f.write(response.content)
f.close
```

Guardamos la Cookie

```
cookie = response.request.headers['cookie']
print cookie
```

Conexión a la pagina de prueba (enviamos la Cookie)

```
uri = 'https://egela1819.ehu.eus/course/view.php'

params = {'id': '17878'}
header = {'Cookie': cookie}
response = requests.get(uri, headers=header, params=params)
```

Recibiendo la respuesta

```
print "     STATUS: " + str(response.status_code)
print response.url
f = open("egelaCursoSW.html", 'w')
f.write(response.content)
f.close()
```

Aplicaciones de servicio

Cuando se habla de Computación en la nube (Cloud Computing) se diferencian tres modelos

Software – as – a – Service SaaS

A este tipo de servicios se acede normalmente a través de un navegador sin atender al software.

El desarrollo, mantenimiento, actualizaciones y las copias de seguridad son responsabilidad del proveedor

Ejemplo: Gmail, Dropbox

Platform – as – a – Service PaaS

Permite desarrollar aplicaciones web y ejecutarlas en la nube sin necesidad de mantener un servidor web
Ejemplo: Google App Engine

Infraestructure – as – a – Service IaaS

Se consigue un mayor control que con PaaS,

Es necesario encargarse de la gestión de la infraestructura.

Ejemplo: Amazon EC2

ThingSpeak

Se trata de una aplicación online que ofrece un servicio de almacenamiento de datos accesible desde HTTP

El API REST de ThingSpeak está constituido por un conjunto de operaciones y servicios que permiten gestionar canales mediante el protocolo HTTP.

La documentación nos indica cómo hay que construir la petición HTTP para el envío de datos:

- https://es.mathworks.com/help/thingspeak
- https://es.mathworks.com/help/thingspeak/rest-api.html

Write Data

URL

`https://api.thingspeak.com/update.<format>`

URL Parameters

Name	Description
`<format>`	(Required) Format for the HTTP response, specified as blank, `json`, or `xml`.

Example `https://api.thingspeak.com/update.json`

Body

Name	Description	Value Type
`api_key`	(Required) Write API Key for this specific channel. You can also send the Write API Key by using a THINGSPEAKAPIKEY HTTP header. The Write API Key is found on the **API Keys** tab of the channel view.	string
`field<X>`	(Optional) Field X data, where X is the field ID	any
`lat`	(Optional) Latitude in degrees	decimal
`long`	(Optional) Longitude in degrees	decimal
`elevation`	(Optional) Elevation in meters	integer
`status`	(Optional) Status update message	string
`twitter`	(Optional) Twitter® username linked to ThingTweet	string
`tweet`	(Optional) Twitter status update	string
`created_at`	(Optional) Date when feed entry was created, in ISO 8601 format, for example: `2014-12-31 23:59:59`. The date you specify must be unique within the channel. Time zones can be specified using the timezone parameter.	datetime

Read Data

URL

`https://api.thingspeak.com/channels/<channel_id>/feeds.<format>`

URL Parameters

Name	Description
`<channel_id>`	(Required) Channel ID for the channel of interest.
`<format>`	(Required) Format for the HTTP response, specified as `json` or `xml`.

Example `https://api.thingspeak.com/channels/266256/feeds.json`

Query String Parameters

Name	Description	Value Type
`api_key`	(Required for private channels). Specify Read API Key for this specific channel. The Read API Key is found on the **API Keys** tab of the channel view.	string
`results`	(Optional) Number of entries to retrieve. The maximum number is 8,000.	integer
`days`	(Optional) Number of 24-hour periods before now to include in response. The default is 1.	integer
`minutes`	(Optional) Number of 60-second periods before now to include in response. The default is 1440.	integer
`start`	(Optional) Start date in format YYYY-MM-DD%20HH:NN:SS.	datetime
`end`	(Optional) End date in format YYYY-MM-DD%20HH:NN:SS.	datetime
`timezone`	(Optional) Identifier from Time Zones Reference for this request.	string
`offset`	(Optional) Timezone offset that results are displayed in. Use the timezone parameter for greater accuracy.	integer
`status`	(Optional) Include status updates in feed by setting "status=true".	`true` or `false`
`metadata`	(Optional) Include metadata for a channel by setting "metadata=true".	`true` or `false`
`location`	(Optional) Include latitude, longitude, and elevation in feed by setting "location=true".	`true` or `false`
`min`	(Optional) Minimum value to include in response.	decimal
`max`	(Optional) Maximum value to include in response.	decimal
`round`	(Optional) Round to this many decimal places.	integer
`timescale`	(Optional) Get first value in this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily".	integer or string
`sum`	(Optional) Get sum of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily".	integer or string
`average`	(Optional) Get average of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily".	integer or string
`median`	(Optional) Get median of this many minutes, valid values: 10, 15, 20, 30, 60, 240, 720, 1440, "daily".	`integer or st`

```
# -*- coding: UTF-8 -*-
import httplib
import urllib
import psutil
import json
def mandardatos(servidor, WRITE_API_KEY):
    conn = httplib.HTTPSConnection(servidor)
    conn.connect()

    cpuPercent = psutil.cpu_percent(interval=15)
    ramPercent = psutil.virtual_memory().percent
    print "      CPU = " + str(cpuPercent)
    print "      RAM = " + str(ramPercent)

    uri = '/update.json'
    parametros = {'api_key': WRITE_API_KEY, 'field1': str(cpuPercent), 'field2': str(ramPercent)}
    params_encoded = urllib.urlencode(parametros)
    headers = {'Host': servidor, 'Content-Type': 'application/x-www-form-urlencoded', 'Content-Length': str(len(params_encoded))}
    conn.request('POST',uri,headers=headers,body=params_encoded)

    respuesta = conn.getresponse()
    print "      STATUS: " + str(respuesta.status)
    conn.close()


def mostrardatos(servidor, CHANNEL_ID, READ_API_KEY ):
    conn = httplib.HTTPSConnection(servidor)
    conn.connect()

    uri = '/channels/' + CHANNEL_ID + '/feeds.json'
    parametros = {'api_key': READ_API_KEY}
    params_encoded = urllib.urlencode(parametros)
    headers = {'Host': servidor, 'Content-Type': 'application/x-www-form-urlencoded', 'Content-Length': str(len(params_encoded))}
    conn.request('GET',uri,headers=headers,body=params_encoded)

    respuesta = conn.getresponse()
    print "      STATUS: " + str(respuesta.status)

    datos = json.loads( respuesta.read())
    for campo in range(0, len( datos['feeds'] ) ):
        cpu = datos['feeds'][campo]['field1']
        ram = datos['feeds'][campo]['field1']
        print "      CAMPO: " + str(campo)
        print "               CPU: " + cpu
        print "               RAM: " + ram

    conn.close()


def borrardatos(servidor, CHANNEL_ID, USER_API_KEY):
    conn = httplib.HTTPSConnection(servidor)
    conn.connect()

    uri = '/channels/' + CHANNEL_ID + '/feeds.json'
    parametros = {'api_key': USER_API_KEY}
    params_encoded = urllib.urlencode(parametros)
    headers = {'Host': servidor, 'Content-Type': 'application/x-www-form-urlencoded'}
    conn.request('DELETE',uri,headers=headers,body=params_encoded)

    respuesta = conn.getresponse()
    print "      STATUS: " + str(respuesta.status)

    conn.close()


CHANNEL_ID = '694428'
WRITE_API_KEY = 'YBP603L2JVW81418'
READ_API_KEY = 'HVCPCW8X4XNW4JY8'
USER_API_KEY = ''
servidor = 'api.thingspeak.com'


mandardatos(servidor, WRITE_API_KEY)

mostrardatos(servidor, CHANNEL_ID, READ_API_KEY )

borrardatos(servidor,CHANNEL_ID, USER_API_KEY)
```

Web Scraping – BeautifulSoap

Es una técnica de recopilación de información que consiste en introducir en la web un programa de software que simule la navegación de un humano.

Utilizando el protocolo HTTP y aprovechando la indexación de la web adoptada por la mayoría de los motores de búsqueda se pueden recopilar datos para almacenarlos en una base de datos central, en una hoja de cálculo o en alguna otra fuente de almacenamiento.

Alguno de los usos del web scraping son:

- La comparación de precios en tiendas,
- La monitorización de datos relacionados
- La detección de cambios en sitios webs
- La integración de datos en sitios webs.

Documentación de BeautifulSoap

- https://www.crummy.com/software/BeautifulSoup/bs4/doc/

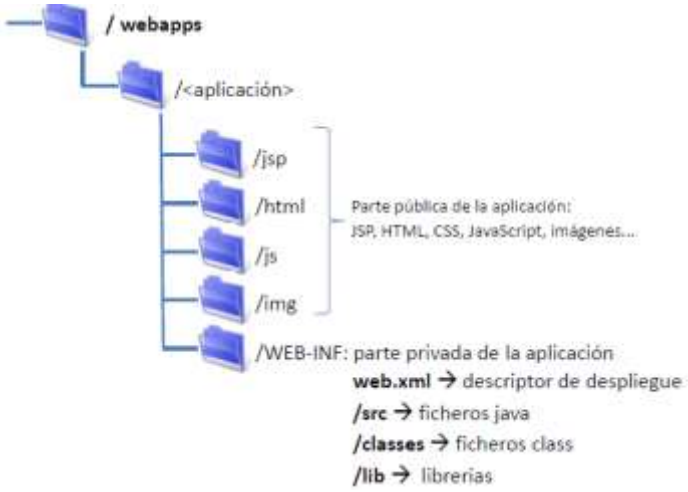
```
# -*- coding: UTF-8 -*-
import requests
import getpass

print" \n======================== egela.loggin ======================\n"
uri = 'https://egela1819.ehu.eus/login/index.php'

LDAP_usuario = raw_input("     Insert your LDAP username: ")
LDAP_password = getpass.getpass("     Insert your LDAP password: ")
cuerpo_peticion ={'username': LDAP_usuario, 'password': LDAP_password}

response = requests.post('https://egela1819.ehu.eus/login/index.php/', data=cuerpo_peticion)

print "     STATUS: " + str(response.status_code)
cookie = response.request.headers['cookie']
print "     COOKIE: " + cookie

f = open("egela.html", 'w')
f.write(response.content)
f.close
from bs4 import BeautifulSoup

print" \n======================== egela.curso ======================\n"
uri = 'https://egela1819.ehu.eus/course/view.php'

params = {'id': '17878', 'lang': 'es'}
header = {'Cookie': cookie}
response = requests.get(uri, headers=header, params=params)

print "     STATUS: " + str(response.status_code)
print "     RESPONSE URI: " + response.url
print "                   "

f = open("egelaCursoSW.html", 'w')
f.write(response.content)
f.close()

respuesta = response.content
if response.status_code == 200:
    print "-------------------- TODOS LOS ENLACES -----------------"
    html = BeautifulSoup(respuesta, "html.parser")

    print "====>> " + str(html.h1.string)
    print "====>> " + str(html.title.string)
    print "====>> usuario: " + str(html.find('span', class_='usertext').string)

    for elemento in html.find_all('li', class_='section main clearfix'):
        print "  --> " + str(elemento.find('div', class_='content').h3.string)
        for link in elemento.find('div', class_='content').find_all('div', class_='activityinstance'):
            print "       *  nombre:  " + link.find('span', class_='instancename').text
            print "       *  enlace:  " + str(link.a.get('href'))

conn.close()
```

```
# -*- coding: UTF-8 -*-
import requests
from bs4 import BeautifulSoup

print" \n====================== GooGle ======================\n"
uri = 'https://www.google.es

response = requests.get(uri)

print "-------------------- RESPUESTA ------------------"

print "     STATUS: " + str(response.status_code)
print "     CABECERAS: " + str(response.headers)

respuesta = response.content
if response.status_code == 200:
    print "-------------------- BeautifulSoup ------------------"
    html = BeautifulSoup(respuesta, "html.parser")
    print "---------------- HEAD ---------------- "
    print str(html.head)
    print "  --> NAME: " + str(html.head.name)
    print "  --> STRING: " + str(html.head.string)
    print "  --> META: " + str(html.head.meta)
    print "  --> META CONTENT: " + str(html.head.meta['content'])
    print "---------------- TITLE ---------------- "
    print str(html.title)
    print "  --> NAME: " + str(html.title.name)
    print "  --> STRING: " + str(html.title.string)
    print "  --> META: " + str(html.title.meta)
    print "---------------- PARRAFO ---------------- "
    print str(html.p)
    print "  --> NAME " + str(html.p.name)
    print "  --> STRING " + str(html.p.string)
    print "  --> META: " + str(html.p.meta)
```

Servidor Web

Apache Tomcat

Es un Software desarrollado con Java que sirve como Servidor Web con soporte de servlets y JSPs capaz de recibir peticiones de páginas web y redireccionarlas a un objeto servlet



Servlet

Es un programa que se ejecuta en el servidor para recibir, procesar y responder las peticiones de un cliente. Aunque pueden responder a cualquier tipo de petición, comúnmente se utilizan en servidores Web que atienden peticiones HTTP

En java un servlet se representa mediante un objeto que extiende de $javax.servlet.http.HttpServlet$

Una aplicación puede estar formada por varios servlets

Los paquetes $javax.Servlet$ y $javax.Servlet.http$ facilitan clases e interfaces para escribir Servlets:

Métodos de HttpServletRequest

Permite realizar peticiones HTTP desde código JavaScript

$String\ getMethod()$

Devuelve el nombre del método con el que se realizó la solicitud

$java.util.Enumeration < String >\ getHeaderNames()$

Devuelve una enumeración que contiene los nombres de todos los encabezados de la petición

Si la solicitud no tiene encabezados, este método devuelve una enumeración vacía

$String\ getHeader(String\ name)$

Recibe como parámetro el nombre de una cabecera y devuelve su valor. Si la cabecera no existe devuelve NULL

$java.util.Enumeration < String >\ getParameterNames()$

Devuelve una enumeración que contiene los nombres de todos los parámetros de la petición

Si la solicitud no tiene parámetros, este método devuelve una enumeración vacía

$String\ getParameter(String\ name)$

Recibe como parámetro el nombre de un parámetro de la petición y devuelve su valor.

Métodos de HttpServletResponse

$void\ setStatus(int\ status)$	Establece el valor del estado de la respuesta
$void\ setContentType(String\ type)$	Establece el tipo de contenido de la respuesta
$void\ setContentLength(String\ type)$	Establece la longitud del cuerpo de la respuesta
$void\ addHeader(String\ name, String\ value)$	Añade una cabecera
$void\ addCookie(Cookie\ cookie)$	Añade una Cookie

$java.io.PrintWriter\ getWriter()$

Devuelve un objeto $PrintWriter$ que puede enviar texto de caracteres al cliente.

$void\ sendRedirect(String\ location)$

Envía una respuesta de redirección temporal para el cliente utilizando la URL de redirección ubicación especificada y borra la memoria intermedia.

$java.lang.String\ getContentType()$

Devuelve el tipo de contenido utilizado para el cuerpo enviado en esta respuesta

Métodos de una sesion

En una sesión se pueden almacenar pares de valores (clave, valor), donde las claves son cadenas de caracteres y los valores pueden ser cualquier objeto Java.

A los atributos de una sesión solo puede acceder el propietario de la sesión

$HttpSession\ session\ =\ request.getSession(\ boolean\ crearNueva\ )$

Devuelve la sesión activa, sino existe crea una nueva o no en función del parámetro recibido

$void\ setAttribute(String\ name, Object\ value)$ Añade a la sesión un objeto.

$Object\ getAttribute\ (String\ name)$ Devuelve un objeto java almacenado en la sesión del usuario, o null

$long\ getCreationTime()$ Devuelve el instante en el que fue creada la sesión.

$String\ getId()$ Devuelve un identificador único para la sesión.

$long\ getLastAccessedTime()$ Devuelve instante en el cual se realizó la última petición asociada con esta sesión;

$int\ getMaxInactiveInterval()$ Devuelve el máximo tiempo de inactividad permitido a la sesión

$void\ invalidate()$ Invalida la sesión.

$void\ removeAttribute(String\ name)$ Elimina de la sesión el atributo asociado al nombre indicado.

Métodos del contexto

En el contexto se pueden almacenar pares de valores (clave, valor) donde las claves son cadenas de caracteres y los valores pueden ser cualquier objeto Java.

Cualquier sesión activa puede acceder a los datos del contexto

$ServletContext\ context = session.getServletContext()$

Devuelve el contexto de una sesión al cual pertenece esta sesión.

$void\ setAttribute\ (String\ name, Object\ object)$

Guarda del objeto que se le pasa como segundo parámetro en el contexto de la aplicación y lo asocia con la cadena de caracteres que se le pasa como primer parámetro

$Object\ getAttribute\ (String\ name)$

Devuelve el atributo asociado al nombre indicado

$void\ removeAttribute(String\ name)$

Elimina el atributo asociado con la cadena de caracteres que se le pasa como argumento.

Despliegue

Una vez terminada la aplicación se realizara el despliegue sobre un servidor de Tomcat. Para ello se copiara en el directorio $\$CATALINA\_HOME/webapps$ el directorio de la aplicación comprimido en un archivo WAR.

- Con eclipse los archivos WAR se generan en : $Project \longrightarrow Export \longrightarrow War\ file$

Es necesario reiniciar el servidor Tomcat

```java
package micarpeta;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class HolaMundo3 extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        // CONEXION
        System.out.println("---> Entering HolaMundo servlet");
        PrintWriter salida = response.getWriter();
        System.out.println("<--- Exiting HolaMundo servlet");

        // IMPRIMIR TODAS LAS CABECERAS DE LA PETICION
        salida.println(" <Br> Las cabeceras de la peticion son: <Br>");
        java.util.Enumeration<String> cabeceras = request.getHeaderNames();
        int i = 0;
        while ( cabeceras.hasMoreElements() )
        {
            String nombre = cabeceras.nextElement();
            String valor = request.getHeader( nombre) ;
            i++;
            salida.println( "    * " + nombre + " : " + valor + "<Br>" );
        }
        salida.println(" <Br> En total "+ i + " Cabeceras <Br>");

        // IMPRIMIR TODOS LOS PARAMETROS DE LA pETICION
        salida.println(" <Br> Los parametros de la peticion son: <Br>");
        java.util.Enumeration<String> parametros = request.getParameterNames();
        int j = 0;
        while ( parametros.hasMoreElements() )
        {
            String nombre = parametros.nextElement();
            String valor = request.getParameter( nombre) ;
            j++;
            salida.println( "    * " + nombre + " : " + valor + "<Br>" );
        }
        salida.println(" <Br> En total "+ j + " Parametros <Br>");
    }
}
```

Web.XML

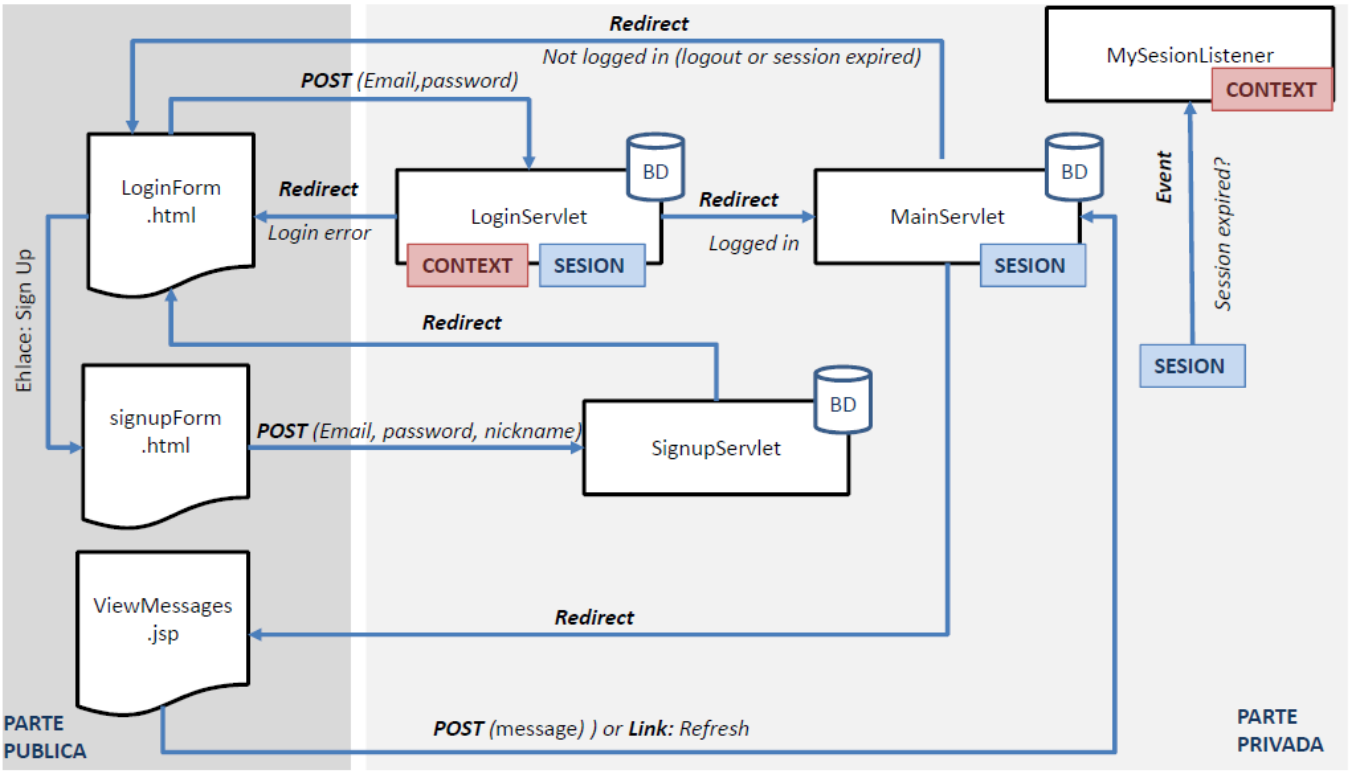
```xml
<servlet>
    <servlet-name>HolaMundo3</servlet-name>
    <servlet-class>micarpeta.HolaMundo3</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HolaMundo3</servlet-name>
    <url-pattern>/servlet/holamundo3</url-pattern>
</servlet-mapping>
```

El fichero ***web.xml*** es el descriptor de despliegue. Su información es utilizada por todas las aplicaciones web para configuran los time-out de sesiones y el acceso a los ficheros principales.

Para llamar a esta aplicación se utilizaría la siguiente URI

http:// Host : PUERTO / Aplicación / SerbletURL

http://localhost:8080/micarpeta /servlet/holamundo3




```java
public class LoginServlet extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    private MySQLdb mySQLdb;

    Importamos la base de datos
    public void init(ServletConfig config) { mySQLdb = new MySQLdb(); }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        Cargamos todos los parámetros de la petición
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        Cargamos el usuario de la base de datos
        String username = mySQLdb.getUsername(email, password);
        Si el usuario no existe redireccionamos al LogingForm y mandamos la respuesta
        if(username == null)
        {
            RequestDispatcher rd = request.getRequestDispatcher("/html/loginForm.html");
            rd.forward(request, response);
        }
        Si el usuario existe creamos una sesión a la que asignamos comoatributo el nombre del usuario
        else
        {
            HttpSession session = request.getSession(true);
            String sessionID = session.getId();
            session.setAttribute("username", username);
            System.out.println("      Sesion de Usuario para " + username + ": " + sessionID);
            Tomamos la lista de usuarios activos del contexto
            ServletContext context = request.getServletContext();
            HashMap<String, String> loggedinUsers = (HashMap) context.getAttribute("loggedin_users");
            Si la lista de usuarios activos esta vacia creamos una nueva
            if(loggedinUsers == null)
            {
                loggedinUsers = new HashMap();
                loggedinUsers.put(username, sessionID);
            }
            Cargamos en el contexto la lista de usuarios activos actualizada
            context.setAttribute("loggedin_users", loggedinUsers);
            Redireccionamos al usuario al MainServlet y mandamos la respuesta
            RequestDispatcher rd = context.getNamedDispatcher("MainServlet");
            rd.forward(request, response);
        }
    }
}
```

```java
public class SignupServlet extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    private MySQLdb mySQLdb;

    importamos la base de datos
    public void init(final ServletConfig config) { mySQLdb = new MySQLdb(); }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        Cargamos todos los atributos del formulario de la peticion
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        String nickname = request.getParameter("nickname");

        Añadimos el nuevo usuario a la base de datos
        mySQLdb.setUserInfo(email, password, nickname);

        Redireccionamos al LoginForm y mandamos la respuesta
        RequestDispatcher rd = request.getRequestDispatcher("/html/loginForm.html");
    }   rd.forward(request, response);
}

public class MainServlet extends HttpServlet
{
    private static final long serialVersionUID = 1L;
    private MySQLdb mySQLdb;

    importamos la base de datos
    public void init(ServletConfig config) { mySQLdb = new MySQLdb(); }

    Las peticiones realizadas mediante Get las redireccionamos al metodo Post
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
    {
        Si la sesion no esta creada redirigimos al LogingForm

        if(request.getSession(false) == null)
        {
            RequestDispatcher rd = request.getRequestDispatcher("/html/loginForm.html");
            rd.forward(request, response);
        }
        else
        {
            Tomamos el parametro "mensage" del formulario desde el que se realizo la petición
            String message = request.getParameter("message");
            if(message!= null)
            {
                Tomamos el usuario de la sesion y actualizamos la base de datos
                HttpSession session = request.getSession();
                String username = (String) session.getAttribute("Username");
                mySQLdb.setMessageInfo(message, username);
            }
            Tomamos todos los mensajes de la base de datos y los cargamos en un atributo de la respuesta
            ArrayList<MessageInfo> messageList = mySQLdb.getAllMessages();
            request.setAttribute("messageList", messageList);
            Redireccionamos al usuario a la pagina ViewMessages y mandamos la respuesta
            System.out.println("     Redireccionando el usuario a viewMessages.jsp");
            RequestDispatcher rd = request.getRequestDispatcher("/jsp/viewMessages.jsp");
            rd.forward(request, response);
        }
    }
}
```

```java
public class MySessionListener implements HttpSessionListener
{
    public void sessionCreated(HttpSessionEvent event)
    {
        System.out.println("Una sesion esta siendo creada");
    }
    public void sessionDestroyed(HttpSessionEvent event)
    {
        Tomamos el ID de la session
        System.out.println("Una sesion esta siendo destruida");
        HttpSession session = event.getSession();
        String sessionID = session.getId();
        Tomamos la lista de usuarios activos del contexto
        ServletContext context = session.getServletContext();
        HashMap<String, String> loggedinUsers = (HashMap) context.getAttribute("loggedin_users");
        Para cada usuario activo
        for(Map.Entry<String, String> entry : loggedinUsers.entrySet())
        {
            si tiene el mismo ID que la sesion
            if(entry.getValue().equals(sessionID))
            {
                Lo eliminamos de la lista de usuarios activos y actualizamos el contexto
                loggedinUsers.remove(entry.getKey());
                context.setAttribute("loggedin_users", loggedinUsers);
                break;
            }
        }
    }
}
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Formulario de Acceso</title>
        <link href="/ShareInfo/css/styleSheet.css" rel="stylesheet" />
    </head>
    <body>
        <header>
            <h1>Web para Compartir Mensajes Cortos</h1>
            <h2>Formulario Login</h2>
        </header>
        <section>
            <form method="POST" action="/ShareInfo/servlet/LoginServlet">
                <table>
                    <tr><td>Email:</td><td><input name="email"/></td></tr>
                    <tr><td>Password:</td><td><input type="password" name="password"/></td></tr>
                </table>
                <button>Enviar</button>
            </form>
        </section>
        <section>
            <a href="/ShareInfo/html/signupForm.html"><font>Registrarse</font></a>
        </section>
    </body>
</html>
```

```html
<!DOCTYPE html>
<html>
    <head>
        <title>SignupForm</title>
        <link href="/ShareInfo/css/styleSheet.css" rel="stylesheet" />
    </head>
    <body>
        <header>
            <h1>Web para Compartir Mensajes Cortos</h1>
            <h3>Formulario de Registro</h3>
        </header>
        <section>
            <form method="POST" action="/ShareInfo/servlet/SignupServlet">
                <table>
                    <tr><td>Email:</td><td><input name="email"/></td></tr>
                    <tr><td>Contraseña:</td><td><input type="password" name="password"/></td></tr>
                    <tr><td>Nickname:</td><td><input name="nickname"></td></tr>
                </table>
                <button>Enviar</button>
            </form>
        </section>
    </body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>
<%@ page import="java.util.*,helper.info.*"%>

<%
    ArrayList<MessageInfo> messageList = (ArrayList) request.getAttribute("messageList");
    ServletContext context = request.getServletContext();
    HashMap<String, String> loggedinUsers = (HashMap) context.getAttribute("loggedin_users");
    HttpSession sesion = request.getSession();
%>

<html>
    <head>
        <title>Visor de Mensajes</title>
        <link href="/ShareInfo/css/styleSheet.css" rel="stylesheet" />
    </head>

    <body>

        <header>
            <h1>Web para Compartir Mensajes Cortos</h1>
            <h3>Vista de Mensajes</h3>
        </header>

        <section>
            <a href="/ShareInfo/servlet/MainServlet">
                <font>Actualizar</font>
            </a>
        </section>

        <section>
            <font>Usuarios Activos: </font>
            <%  for (Map.Entry<String, String> entry : loggedinUsers.entrySet()) { %>
            <%=entry.getKey()%>;
            <% } %>
        </section>

        <section>
            <font> Tu usuario es: </font>
            <% sesion.getAttribute("username") %>;
        </section>

        <section>
            <font> Tu usuario es: </font>
            <% sesion.getAttribute("username") %>;
        </section>

        <section>
            <form method="POST" action="/ShareInfo/servlet/MainServlet">
                <table>
                    <tr>
                        <td>Mensaje:</td>
                        <td><textarea name="message" id="message"></textarea></tc
                    </tr>
                </table>

                <button>Enviar</button>
            </form>                                                              >
        </section>

        <section>
            <table>
                <tr>
                    <th>Usuario</th>
                    <th>Mensaje</th>
                </tr>
                <%
                    for (int i = 0; i < messageList.size(); i++) {
                    MessageInfo messageInfo = messageList.get(i);
                %>
                <tr>
                    <td><%=messageInfo.getUsername()%></td>
                    <td><%=messageInfo.getMessage()%></td>
                </tr>
                <%
                    }
                %>
            </table>
        </section>

    </body>

</html>
```

Páginas web dinámicas

En una página web estática existe un único documento HTML para cada contenido de la página web. Como consecuencia, si se quiere modificar dicho contenido hay que editar el código del documento HTML

Una página web dinámica combina código HTML con otros lenguajes de programación con la finalidad de permitir que el contenido se modifique de forma autónoma en función de un patrón predefinido.

Java Server Pages (JSP)

Permite crear páginas web dinámicas basadas en HTML/XML de manera similar a PHP pero en lenguaje Java.

Se caracteriza porque:

- El código se ejecuta en el servidor y al cliente se le devuelve el documento HTML actualizado en cada instante
- Permite acceder a todos a todos los objetos y clase Java que el servidor tenga instalados
- Permite acceder fácilmente a base de datos usando clases específicas

Java Script (JS)

Se caracteriza porque:

- El código se ejecuta en la máquina del cliente cuando se descarga la página web permitiendo realizar cambios sobre las páginas sin necesidad de recargarlas.
  - Permite dar respuesta inmediata a ciertas operaciones sin necesidad de llamar al servidor
  - Puede causar problemas de privacidad
- Sólo permite utilizar los objetos definidos dentro de los navegadores.
  - No permite utilizar clases java que la máquina cliente tenga instaladas
- No está diseñado para acceder a bases de datos

Listeners

Se trata de un conjunto de interfaces que proporciona el servlet diseñadas para escuchar los diferentes eventos que se producen en el ciclo de vida de la aplicación web y reaccionar ante ellos.

En una aplicación web se producen eventos cuando

- Las aplicaciones, sesiones y peticiones se crean y destruyen.
- Los atributos de la aplicación, sesión y petición se agregan, eliminan o modifican.

Servlet ContextListener

Se encarga de gestionar los eventos de creación y destrucción en el contexto de una aplicación

HttpSession Listener

Se encarga de gestionar los eventos de creación, invalidación y destrucción de sesiones

Servlet RequestListener

Se encarga de gestionar los eventos de creación y destrucción de peticiones

HttpSession AttributeListener

Se encarga de gestionar los eventos que se crean al añadir, eliminar o modificar atributos de la sesión

ServletContext AttributeListener

Se encarga de gestionar los eventos que se crean al añadir, eliminar o modificar atributos del contexto

ServletRequest AttributeListener

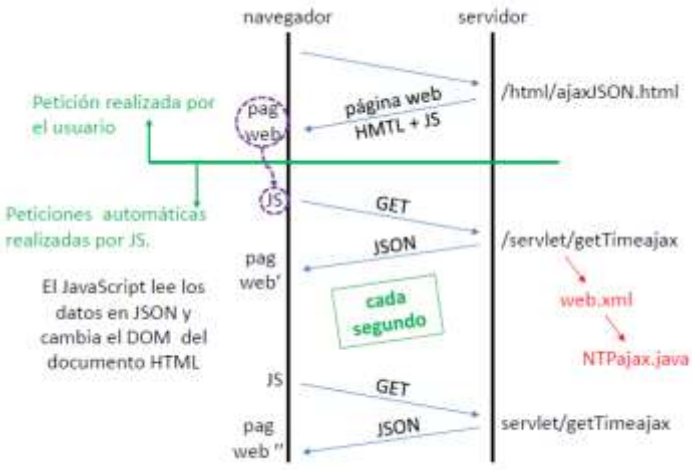
Se encarga de gestionar los eventos que se crean al añadir, eliminar o modificar atributos de las peticiones

NTPajax.java

```
public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {

    Actualizamos la cabecera ContenType de la respuesta
    response.setContentType("application/json");
    PrintWriter out = response.getWriter();
    Cargamos en una estructura HasMap los datos del servidor
    Calendar cal = Calendar.getInstance();
    HashMap<String, Object> hashMap = new HashMap<String, Object>();
    hashMap.put("day", cal.get(Calendar.DAY_OF_MONTH));
    hashMap.put("month", cal.get(Calendar.MONTH));
    hashMap.put("year", cal.get(Calendar.YEAR));
    hashMap.put("hour", cal.get(Calendar.HOUR_OF_DAY));
    hashMap.put("minute", cal.get(Calendar.MINUTE));
    hashMap.put("second", cal.get(Calendar.SECOND));
    Convertimos la estructura HasMap a Json
    Gson gson = new Gson();
    String json = gson.toJson(hashMap);
    response.setContentType("application/json");

    out.println(json);
    out.flush();
    out.close();
}
```

AjaxJson.HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
    <head>
        <title>AJAX example</title>
        <script type="text/javascript">
            Realiza una peticion a la clasegetTimeajax del servlet AJAX
            Toma el Json de la respuesta y obtiene los datos que carga en la etiqueta correspondiente
            function GetTimeIO()
            {
                var request = new XMLHttpRequest();
                request.onreadystatechange = function()
                {
                    if (request.readyState == 4)
                    {
                        if (request.status == 200)
                        {
                            if (request.responseText != null)
                            {
                                var jsonObj = JSON.parse(request.responseText);
                                document.getElementsByClassName("day")[0].innerHTML = jsonObj.day;
                                document.getElementsByClassName("month")[0].innerHTML = jsonObj.month;
                                document.getElementsByClassName("year")[0].innerHTML = jsonObj.year;
                                document.getElementsByClassName("hour")[0].innerHTML = jsonObj.hour;
                                document.getElementsByClassName("minute")[0].innerHTML = jsonObj.minute;
                                document.getElementsByClassName("second")[0].innerHTML = jsonObj.second;
                            }
                        }
                    }
                };

                request.open("POST", "/AJAX/servlet/getTimeajax", true);
                request.send(null);
                setTimeout("GetTimeIO()",1000);
            }
        </script>
    </head>
    Al cargarse la pagina web se ejecuta la funcion GetTimeIO()
    <body onload="GetTimeIO()">
        <p>
            day is <span class="day">0</span><br>
            month is <span class="month">0</span><br>
            year is <span class="year">0</span><br>
            hour is <span class="hour">0</span><br>
            minute is <span class="minute">0</span><br>
            second is <span class="second">0</span><br>
        </p>
    </body>
</html>
```



Web.xml

```
<display-name>AJAX</display-name>
    <servlet>
        <servlet-name>NTPajax</servlet-name>
        <servlet-class>ntpajax.NTPajax</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NTPajax</servlet-name>
        <url-pattern>/servlet/getTimeajax</url-pattern>
    </servlet-mapping>
</web-app>
```

Uso de aplicaciones externas

Open Authorization OAuth

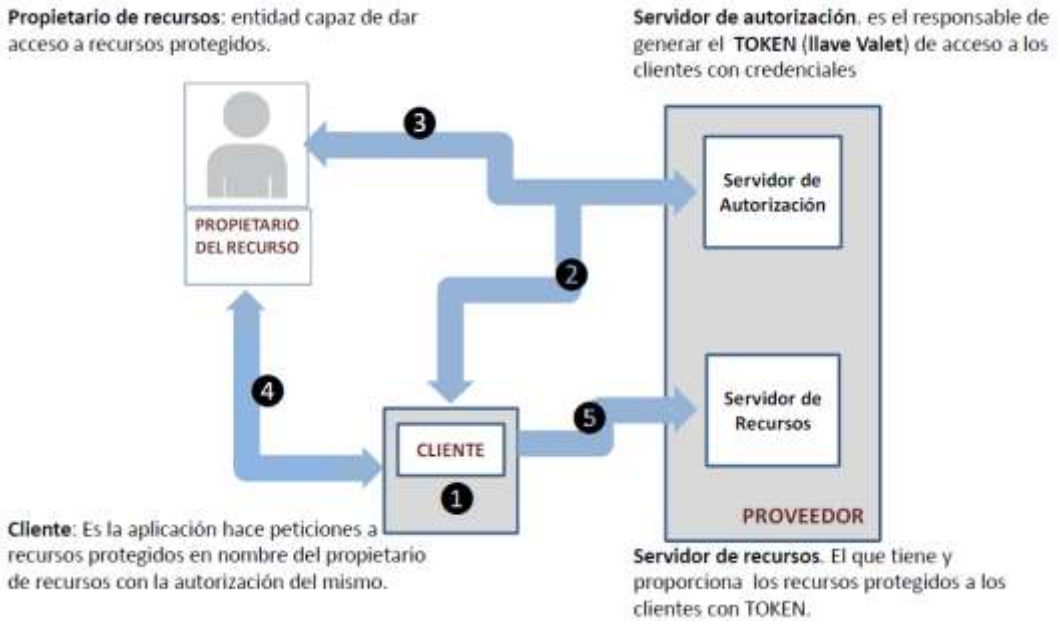
Es un estándar abierto que permite flujos simples de autorización para sitios web o aplicaciones informáticas.

Se trata de un protocolo que permite a un usuario realizar una autorización segura de su información en una aplicación de servicio sin compartir toda su identidad

Este sistema proporciona

- A los desarrolladores un método de interactuar con datos protegidos y publicarlos
- A los usuarios un acceso estándar y simple a sus datos al mismo tiempo que protege las credenciales de su cuenta

Este mecanismo es utilizado por compañías como Google, Facebook, Microsoft, Twitter y Github para permitir a los usuarios compartir información sobre sus cuentas con aplicaciones de terceros o sitios web



Petición del $Auth\_Code$

La aplicación cliente solicita el Auth_Code al servidor utilizando el Client_id

El servidor responde con el Auth_Code en formato Json

Autentificación del permiso

El servidor solicita al usuario la autenticación y el permiso

Petición del TOKEN de acceso.

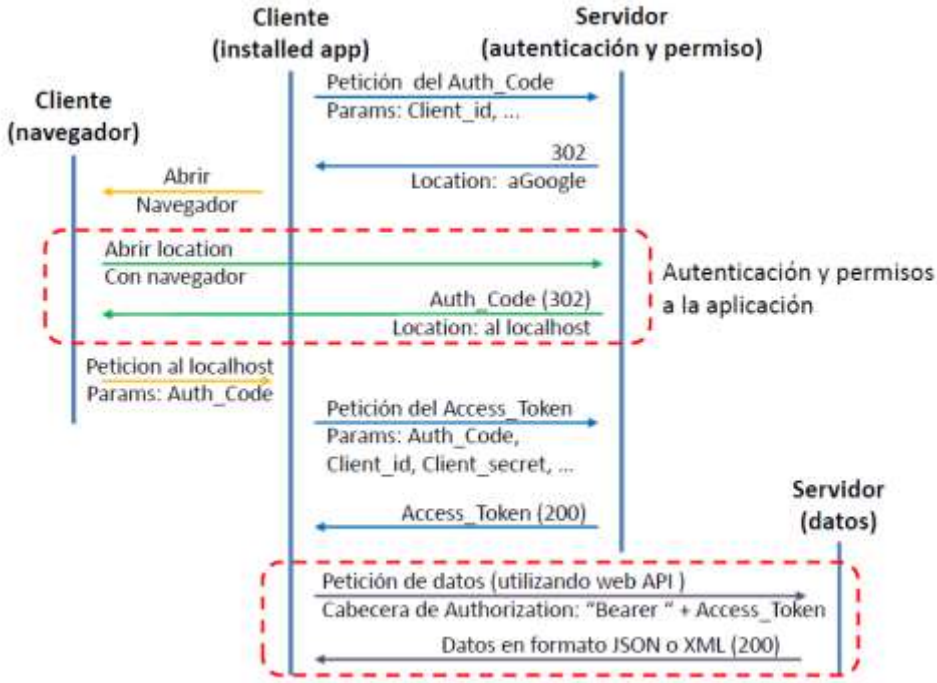
La aplicación cliente solicita el Acces-TOKEN utilizando el Auth_Code

El servidor responde con el Access_Token en formato Json

Acceso a la aplicación

La aplicación cliente solicita la información requerida a la aplicación servidora utilizando el Access_Token

La aplicación servidora responde con la información pertinente



```
# -*- coding: UTF-8 -*-
import httplib
import urllib
import requests
import webbrowser
import socket
import json
```

Tokens de Google

```
ID_de_cliente = '238152268991-hj3mqénodal.apps.googleusercontent.com'
client_secret = 'bNRE7YdjTTT4Masz'
```

Solicitud del Auth_Code

Parametros de la petición

```
redirect_uri = 'http://127.0.0.1'
scope = 'https://www.googleapis.com/auth/calendar'
parametros = { 'response_type': 'code', 'client_id': client_id,'redirect_uri': redirect_uri, 'scope': scope}
cabeceras = {'User_Agent':'Python Client'}
respuesta = requests.get('https://accounts.google.com/o/oauth2/v2/auth', headers=cabeceras, params=parametros, allow_redirects=False)
```

Procesamos la respuesta

```
print respuesta.status_code
location = respuesta.headers['location']
print location
webbrowser.open_new(location)
```

Configuramos las opciones de Sockets y lo vinculamos a la direccion

```
listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
listen_socket.bind(('127.0.0.1', 80))
listen_socket.listen(1)
client_connection,client_address = listen_socket.accept()
print client_connection.getsockname()
print client_address
```

Recibimos los datos del Socket

```
request = client_connection.recv(1024)
print "IP address ( redirect uri): " + request
```

Leemos el Auth_Code

```
aux = request.split('\n')[0].split(' ')[1]
index = aux.find('=')
auth_code = aux[index + 1:]
print "----> Auth Code: " + auth_code
```

Respondemos a la petición

```
http_response = """\
HTTP/1.1 200 OK

<html>
    <head>
        <title>Prueba</title>
    </head>
    <body>
        Se ha autenticado correctamente.
        Close this window.
    </body>
</html>
"""
client_connection.sendall(http_response)
client_connection.close()
```

Solicitud del acces-TOKEN (utilizando el Auth_Code)

Parametros de la petición

```
cabeceras={'User-Agent':'Python Client'}
cuerpo={'code':auth_code,
        'client_id':client_id,
        'client_secret':client_secret,
        'redirect_uri':redirect_uri,
        'grant_type':'authorization_code'
       }
```

Procesamos la respuesta

```
respuesta=requests.post('https://www.googleapis.com/oauth2/v4/token', headers=cabeceras, data=cuerpo)
print respuesta.status_code
json_respuesta = json.loads(respuesta.content)
access_token = json_respuesta['access_token']
print '\nACESS_TOKEN: ' + access_token
```

Utilizamos la aplicacion (utilizando el acces-TOKEN)

Parametros de la petición

```
cabeceras={}
cabeceras['User-Agent'] = 'Python Client'
cabeceras['Authorization'] = 'Bearer ' + access_token
```

Procesamos la respuesta cargandola en un Json

```
respuesta = requests.get('https://www.googleapis.com/calendar/v3/users/me/calendarList', headers=cabeceras)
print respuesta.status_code
print respuesta.content
json_respuesta = json.loads(respuesta.content)
```

Para cad aItem realizamos otra peticion solicitando sus eventos

```
for each in json_respuesta['items']:
    print ' ' + each['summary']
    print 'Id: ' + each['id'] + '\n'

    params={'calendarId':each['id']}

    respuesta = requests.get('https://www.googleapis.com/calendar/v3/calendars/'+each['id']+'/events', headers=cabeceras, params=params)
    json_respuesta2 = json.loads(respuesta.content)
    print json_respuesta2
```

```
# -*- coding: UTF-8 -*-
import httplib
import urllib
import requests
import webbrowser
import socket
import json
import sys
import os

# DOCUMENTACION DE LA API DE DROBOX
# https://www.dropbox.com/developers/documentation/http/documentation
# https://dropbox.github.io/dropbox-api-v2-explorer

def conectDropbox( dropbox_key ):

    # PARAMETROS DE LA PETICION
    servidor = "www.dropbox.com"
    params = { 'response_type' : 'code',
               'client_id': dropbox_key,
               'redirect_uri': 'http://localhost' }
    params_encoded = urllib.urlencode( params )
    recurso = '/1/oauth2/authorize?' + params_encoded
    uri = 'https://' + servidor + recurso
    webbrowser.open( uri )

    # PETICION DE AUTENTICACION
    listen_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    listen_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    listen_socket.bind(('localhost', 80))
    listen_socket.listen(1)
    client_connection, client_address = listen_socket.accept()
    request = client_connection.recv(1024)
    http_response = """\
    HTTP/1.1 200 OK

    <html>
       <head> <title> Prueba </title> </head>
       <body>
          The authentication flow has completed. Close this window.
       </body>
    </html>
    """
    client_connection.sendall(http_response)
    client_connection.close()
    code = request.split('\n')[0].split('?')[1].split('=')[1].split(' ')[0]
    return code

def loginDropbox( code , dropbox_key, dropbox_secret ):

    # PARAMETROS DE LA PETICION
    servidor = 'https://api.dropboxapi.com/1/oauth2/token'
    parametros = {'code': code,
                  'grant_type': 'authorization_code',
                  'client_id': dropbox_key,
                  'client_secret': dropbox_secret,
                  'redirect_uri': 'http://localhost'}
    cabeceras = {'User-Agent': 'Python Client',
                 'Content-Type': 'application/x-www-form-urlencoded'}

    # PETICION DE LOGGING
    respuesta = requests.post( servidor, headers=cabeceras, data=parametros )
    print respuesta.status_code
    json_respuesta = json.loads(respuesta.content)
    access_token = json_respuesta['access_token']
    return access_token

def subir( Access_Token , origen , destino , archivo ):
    ruta_origen = origen + "/" + archivo
    ruta_destino = destino + "/" + archivo

    # PARAMETROS DE LA PETICION
    url = "https://content.dropboxapi.com/2/files/upload"
    headers = {
        "Authorization": "Bearer " + Access_Token,
        "Content-Type": "application/octet-stream",
        "Dropbox-API-Arg": "{\"path\":\"" + ruta_destino + "\",\"autorename\":true,\"mode\":{\".tag\":\"add\"},\"property_groups\":[]}" }
    data = open( ruta_origen , "rb").read()

    # PETICION
    r = requests.post(url, headers=headers, data=data)
    if r.status_code == 200:
        jresp = json.loads(r.content)
        print " "
        print "     El archivo " + jresp['name'] + " se ha cargado correctamente a las " + jresp['server_modified']
        print " "
        print "     " + r.content
        print " "

# Drobox.py subir C:\\Users\\david\\Desktop /tmp/api pruebal.txt
if ( sys.argv[1] == "subir" ):
    origen = sys.argv[2]
    destino = sys.argv[3]
    fichero = sys.argv[4]
    print "SE SUBIRA A " + origen + " EL FICHERO " + fichero + " DE " + destino
    subir(Access_Token, sys.argv[2], sys.argv[3], sys.argv[4])
```

Crear y editar un excel en GoogleDrive

Primera peticion: Creamos el fichero

```
parametros = {'uploadType': 'multipart'}
cabeceras = {'Authorization': 'Bearer ' + access_token,
             'Content-Type': 'multipart/related; boundary=limitador'}
cuerpo = '--limitador\r\n'
cuerpo += 'Content-Type: application/json; charset=UTF-8\r\n\r\n'
diccionario_metadatos = {'name': 'Mi Hoja Calculo',
                         'mimeType': 'application/vnd.google-apps.spreadsheet'}
metadatos_json = json.dumps(diccionario_metadatos)
```

En el cuerpo creamos el fichero. El limitador separa las filas

```
cuerpo += metadatos_json + '\r\n\r\n'
cuerpo += '--limitador\r\n'
cuerpo += 'Content-Type: text/csv\r\n\r\n'
cuerpo += 'CPU,RAM,FECHA\r\n'
cuerpo += '--limitador--'
```

Guardamos el ID del documento

```
respuesta = requests.post('https://www.googleapis.com/upload/drive/v3/files',params=parametros, headers=cabeceras, data=cuerpo)
print "Status: " + str(respuesta.status_code)
print respuesta.content
diccionario_json = json.loads(respuesta.content)
f_id = diccionario_json["id"]
```

Segunda peticion: Solicitamos el fichero partiendo de su ID

```
parametros = {'includeGridData': 'true'}
cabeceras = {'User-Agent': 'Python Client',
             'Authorization': 'Bearer ' + access_token}
recurso = "v4/spreadsheets/" + f_id
respuesta = requests.get('https://sheets.googleapis.com/' + recurso, params=parametros, headers=cabeceras)
print "Status: " + str(respuesta.status_code)
```

Tercera peticion: Introducimos los valores

```
range = 'A1:C1'
recurso = '/v4/spreadsheets/' + f_id + '/values/' + range + ':append'
parametros = {'valueInputOption': 'USER_ENTERED'}
while (True):
    cpuPercent = psutil.cpu_percent(interval=5)
    ramPercent = psutil.virtual_memory().percent
    print " CPU = " + str(cpuPercent) + "    RAM = " + str(ramPercent)

    cuerpo = {"values": [[cpuPercent, ramPercent, time.strftime("%c")]]}
    cuerpo = json.dumps(cuerpo)

    cabeceras = {'User-Agent': 'Python Client',
                 'Authorization': 'Bearer ' + access_token,
                 'Content-Type': 'application/json'}
    print cuerpo
    respuesta = requests.post('https://sheets.googleapis.com' + recurso, params=parametros, headers=cabeceras,
                              data=cuerpo)
    # PROCESAMOS LA RESPUESTA
    print respuesta.request.url
    print "Status: " + str(respuesta.status_code)
```

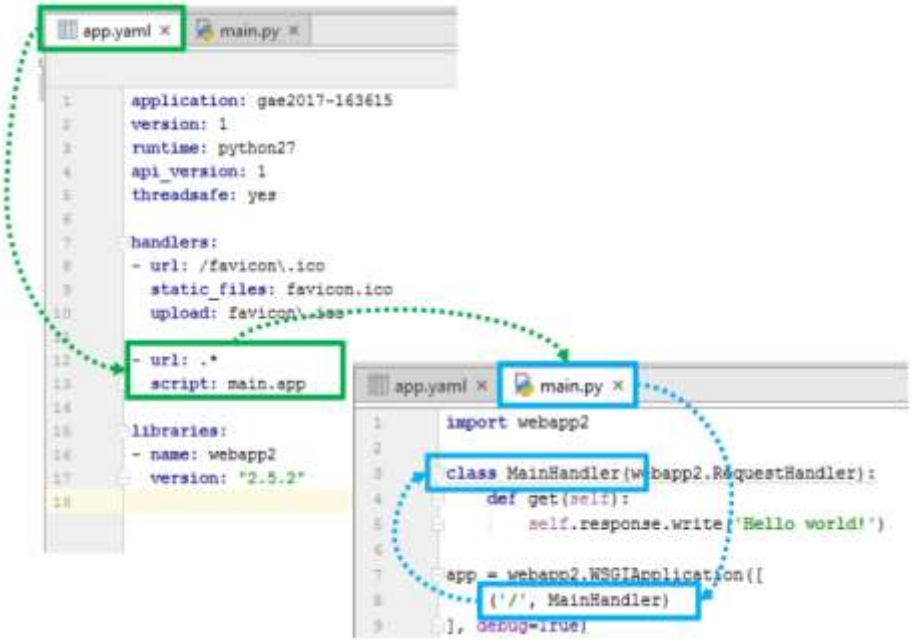
Google App Engine GAE

Las aplicaciones web GAE se caracterizan porque:

- Se escalan de forma automática en función de las necesidades del momento
- Pueden utilizar las cuentas de Google para la gestión de usuarios
- Admiten los siguientes lenguajes de programación: Java, Python y PHP
- Son completamente gratuitas hasta un Gigabyte de almacenamiento

Estructura base de una aplicación web GAE

- **app. yamp** es el descriptor de despliegue.
  - Especifica cómo las rutas URL corresponden a los controladores de solicitud y los archivos estáticos.
  - Contiene información sobre el código de su aplicación (tiempo de ejecución, el último identificador de la versión…)
- Cada servicio en su aplicación tiene su propio archivo app.yaml, que actúa como un descriptor para su implementación. Primero debe crear el archivo app.yaml para el servicio predeterminado antes de poder crear e implementar archivos app.yaml para servicios adicionales dentro de su aplicación.




```
import webapp2

class MainHandler(webapp2.RequestHandler):
    def get(self):
        # por medio de redirect() le decimos al navegador que realice una redireccion
        # es decir,  al navegador le enviamos un respuesta con código del 302
        # y la nueva dirección en la cabecera 'location' con valor 'http://xxx.appspot.com/main'
        self.redirect('/main')

class WelcomePage(webapp2.RequestHandler):
    def get(self):
        self.response.write('<html><head>')
        self.response.write('<title>form</title>')
        self.response.write('</head>')
        self.response.write('<body>')
        self.response.write('<h1>Calculo de la letra de DNI</h1>')
        self.response.write('<form method="GET" action="processForm">')
        self.response.write('Introduce tu numero de DNI sin letra: <input name="dni" type=
        self.response.write('<input type="submit"/>')
        self.response.write('</form></body></html>')

class ProcessForm(webapp2.RequestHandler):
    def get(self):
        dni = self.request.get('dni')

        NIF='TRWAGMYFPDXBNJZSQVHLCKE'
        letra = NIF[int(dni)%23]

        self.response.write(dni+letra)

app = webapp2.WSGIApplication([
    ('/', MainHandler),
    ('/main', WelcomePage),
    ('/processForm', ProcessForm)
], debug=True)
```

```
application: gae2016-1
version: 1
runtime: python27
api_version: 1
threadsafe: yes

handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: .*
  script: main.app

libraries:
- name: webapp2
  version: latest
```

1

Documentación: https://webapp-improved.appspot.com/

App.yaml

```
application: gae2019-236009
version: 1
runtime: python27
api_version: 1
threadsafe: yes

handlers:
- url: /favicon\.ico
  static_files: favicon.ico
  upload: favicon\.ico

- url: /html
  static_dir: html

- url: /js
  static_dir: js

- url: /fechaMain
  script: main.app

- url: /actualizaFecha
  script: main.app

- url: .*
  script: main.app

libraries:
- name: webapp2
  version: "2.5.2"
```

main.py

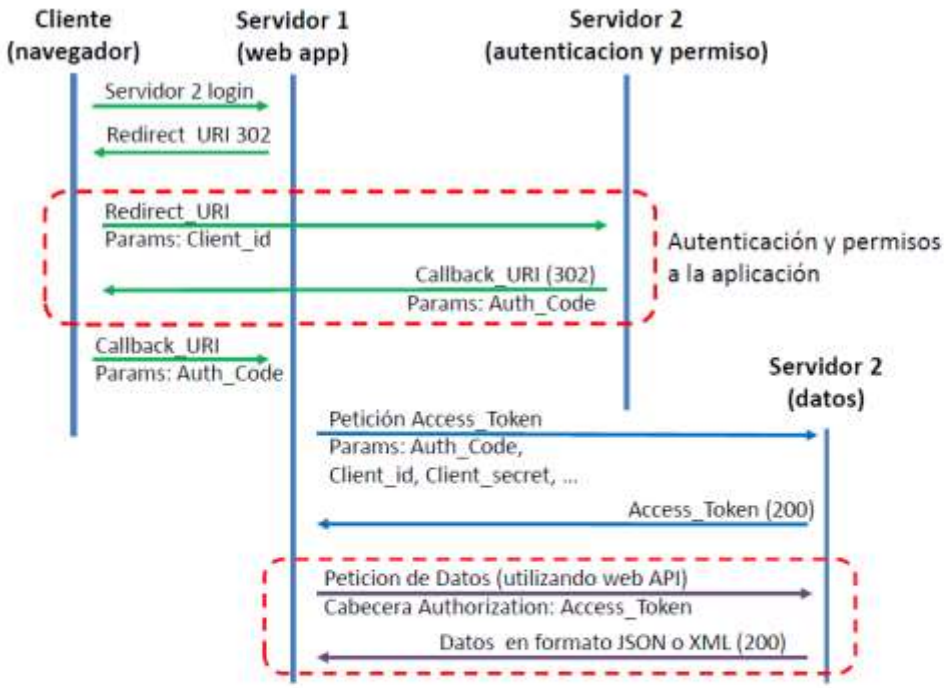
```
import webapp2
from time import gmtime
import json

class MainHandler(webapp2.RequestHandler):
    def get(self):
        self.redirect("/html/fecha.html")

class CurrentTime(webapp2.RequestHandler):
    def get(self):
        data = {}
        data['anno'] = str(gmtime().tm_year)
        data['mes'] = str(gmtime().tm_mon).zfill(2)
        data['dia'] = str(gmtime().tm_mday).zfill(2)
        data['hora'] = str(gmtime().tm_hour).zfill(2)
        data['minuto'] = str(gmtime().tm_min).zfill(2)
        data['segundo'] = str(gmtime().tm_sec).zfill(2)

        json_data = json.dumps(data)
        self.response.headers['Content-Type'] = 'application/json'
        self.response.write(json_data)

app = webapp2.WSGIApplication([
    ('/fechaMain', MainHandler),
    ('/actualizaFecha', CurrentTime),
], debug=True)
```

JavaScript_ajax.js

```
function GetTimeIO() {
    var peticion = new XMLHttpRequest();

    peticion.onreadystatechange = function() {
        if(peticion.readyState == 4) {
            if(peticion.status == 200) {
                if(peticion.responseText != null) {
                    var jsonObj = JSON.parse(peticion.responseText);
                    document.getElementsByClassName("year")[0].innerHTML = jsonObj.anno;
                    document.getElementsByClassName("month")[0].innerHTML = jsonObj.mes;
                    document.getElementsByClassName("day")[0].innerHTML = jsonObj.dia;
                    document.getElementsByClassName("hour")[0].innerHTML = jsonObj.hora;
                    document.getElementsByClassName("minute")[0].innerHTML = jsonObj.minuto;
                    document.getElementsByClassName("second")[0].innerHTML = jsonObj.segundo;
                }
            }
        }
    };

    peticion.open("GET", "/actualizaFecha", true);
    peticion.send(null);
    setTimeout("GetTimeIO()", 5000);
}
```

fecha.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>AJAX example</title>
        <script src="/relojJSP/js/javascript_ajax.js"></script>
    </head>
    <body onload="GetTimeIO()">
        <p>
            <span class="year">0</span> / <span class="month">0</span> / <span class="day">0</span>
            <br/>
            <span class="hour">0</span> : <span class="minute">0</span> : <span class="second">0</span>
        </p>
    </body>
</html>
```




```python
import urllib
import json
from webapp2_extras import sessions
import webapp2
import logging

# Use Requests in GAE
import requests
import requests_toolbelt.adapters.appengine
# Use the App Engine Requests adapter. This makes sure that Requests uses
# URLFetch.
requests_toolbelt.adapters.appengine.monkeypatch()

gae_app_id = 'nombre_de mi_Api_GAE'

gae_callback_url = 'https://' + gae_app_id + '.appspot.com/oauth_callback'

# Api key y Api secret en dropbox
dropbox_app_key = 'xxxxxxxxxxxxx'
dropbox_app_secret = 'xxxxxxxxxxxxx'


class BaseHandler(webapp2.RequestHandler):
    def dispatch(self):
        # Get a session store for this request.
        self.session_store = sessions.get_store(request=self.request)

        try:
            # Dispatch the request.
            webapp2.RequestHandler.dispatch(self)
        finally:
            # Save all sessions.

      self.session_store.save_sessions(self.response)
    @webapp2.cached_property
    def session(self):
        # Returns a session using the default cookie key.
        return self.session_store.get_session()
config = {}
config['webapp2_extras.sessions'] = {'secret_key': 'my-super-secret-key'}
```

Requests no esta en GAE

¡¡¡ Recordar !!!! la URI de callback debe estar definida en **DROPBOX**

Crea la clase BaseHandler para poder trabajar con sesiones con webapp2 en un proyecto GAE

```
class MainHandler(webapp2.RequestHandler):
  def get(self):
    self.response.write('<html><head>')
    self.response.write('<title>LoginAndAuthorize</title>')
    self.response.write('</head>')
    self.response.write('<body>')
    self.response.write('<a href="/LoginAndAuthorize">Login and Authorize with Dropbox</a>')
    self.response.write('</body></html>')
```

```
class LoginAndAuthorize(webapp2.RequestHandler):
  def get(self):
    url = 'https://www.dropbox.com/1/oauth2/authorize'
    parametros = {'response_type': 'code',
                  'client_id': dropbox_app_key,          # App key
                  'redirect_uri': gae_callback_url }
    parametros = urllib.urlencode(parametros)
    self.redirect(url + '?' + parametros)
```

Solicitamos el **code**

Nos la envía a la URI de callback

```
'https://' + gae_app_id + '.appspot.com/oauth_callback'
```

```
class OAuthHandler(BaseHandler):
  def get(self):
    request_url = self.request.url
    code = request_url.split('code=')[1]

    url = 'https://api.dropbox.com/1/oauth2/token'
    parametros = {'code': code,
                  'grant_type': 'authorization_code',
                  'client_id': dropbox_app_key,       # App key
                  'client_secret': dropbox_app_secret,  #App secret
                  'redirect_uri': gae_callback_url}

    cabeceras = {'Content-Type': 'application/x-www-form-urlencoded',}

    respuesta= requests.post(url, headers=cabeceras, data=parametros)

    json_contenido = json.loads(respuesta.content)
    self.session['access_token'] = json_contenido['access_token']
    self.redirect('/CreateFile')
```

Recogemos del **code** de la petición, nos la envía en la URL

Con el code solicitamos el acess-token

Guardamos el acess-token en la sesión.

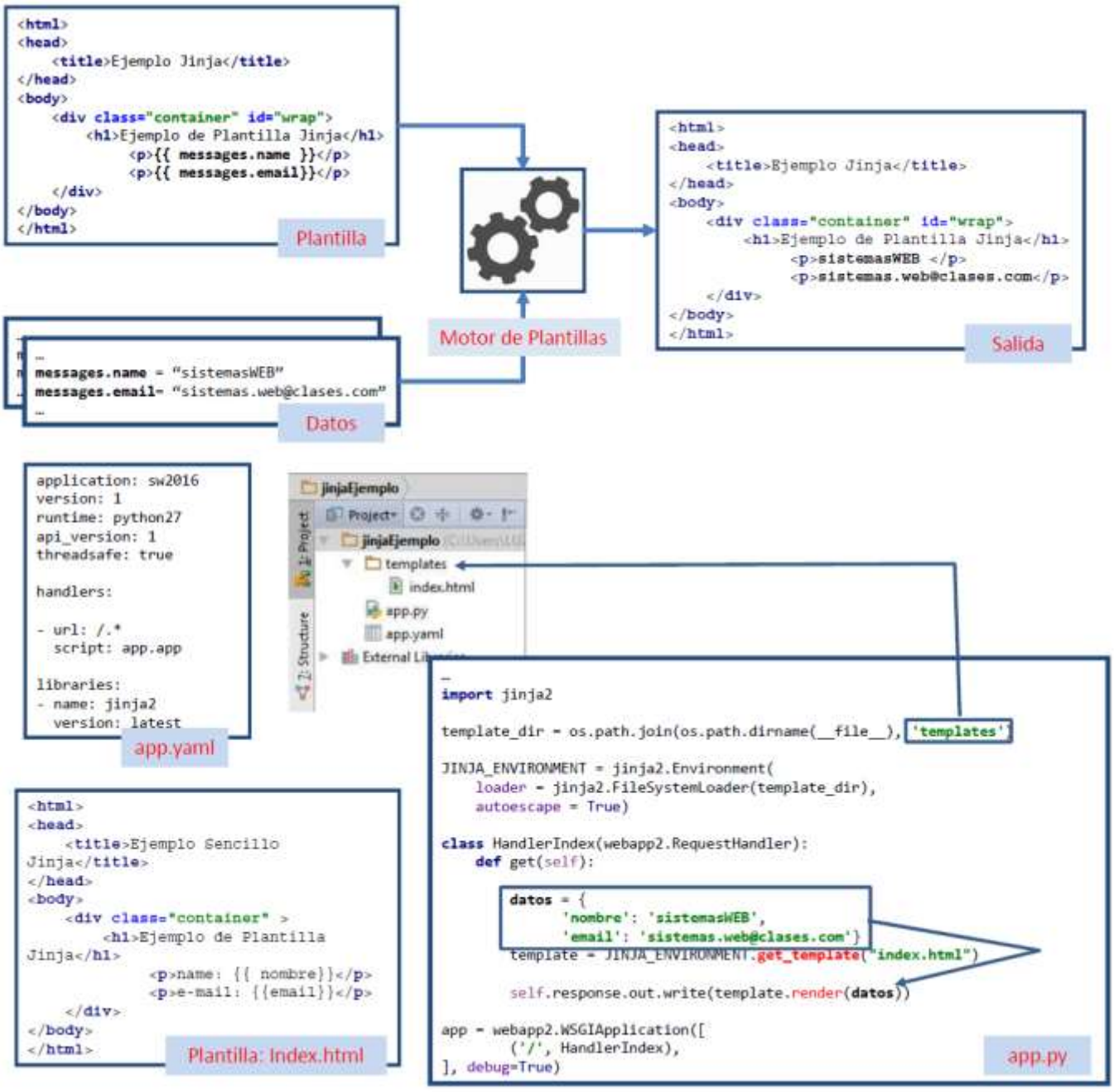
```
class CreateFile(BaseHandler):
  def get(self):

    access_token = self.session['access_token']

    path = '/fichero.txt'
    url = 'https://content.dropboxapi.com/2/files/upload'
    parametros = {'path': path}
    parametros = json.dumps(parametros)

    cuerpo = 'Este es el contenido del fichero.'
    cabeceras = {'Authorization': 'Bearer ' + access_token,
                 'DropBox-API-Arg': parametros,
                 'Content-Type':'application/octet-stream',}

    respuesta = requests.post(url, headers=cabeceras, data=cuerpo)

    self.response.write('<p>Asegurate que se ha generado el fichero en DropBox</p>')
    self.response.write(respuesta.content)

app = webapp2.WSGIApplication([
  ('/', MainHandler),
  ('/LoginAndAuthorize', LoginAndAuthorize),
  ('/oauth_callback', OAuthHandler),
  ('/CreateFile', CreateFile)
], config=config, debug=True)
```

Cogemos el acess-token de la sesión

Jinja2

Es un lenguaje de plantillas escrito en Python que permite insertar datos procesados y texto dentro de un documento de texto. Este sistema permite Separar el código Fuente de la lógica de la presentación

JINJA_ENVIROMENT se utilizara para cargar las plantillas del sistema de archivos en memoria.



OAuth

La autenticación OAuth es el proceso mediante el cual los usuarios otorgan acceso a sus recursos protegidos sin compartir sus credenciales con el consumidor.

OAuth usa tokens generados por el proveedor de servicios en lugar de las credenciales del usuario en las solicitudes de recursos protegidos.

El proceso utiliza dos tipos de Tokens:

- **Request Token**: Utilizado por el consumidor (aplicación) para pedirle al usuario que autorice el acceso a los recursos protegidos.
  El Request Token:
  - Autorizado por el usuario se intercambia por un Access Token.
  - Tiene una duración limitada para garantizar que
    - Se utiliza solo una vez
    - Se utiliza solo para el propósito para el que se autorizó
- **Access Token**: Utilizado por el consumidor para acceder a los recursos protegidos en nombre del usuario.
  Los Access Token:
  - Pueden limitar el acceso a ciertos recursos protegidos
  - Pueden tener una vida útil limitada.
  - Los proveedores de servicios deben permitir que los usuarios revoquen los Access Token.

La autenticación OAuth se realiza en tres pasos:

- El consumidor obtiene un Request Token no autorizado.
- El Usuario autoriza el Request Token.
- El consumidor intercambia el Request Token por un Access Token

OAuth 2.0

En OAuth 2.0 para aplicaciones de escritorio, para retornar el control a la aplicación de origen después de realizar el proceso de autenticación y autorización:

- **El navegador** realiza una petición HTTP a la $redirect\_URI$ (o callback_URI) en la que se encuentra escuchando un servidor asociado a la aplicación.
- **El servidor de autenticación y autorización** devuelve una respuesta 302/303 al navegador con la $redirect\_URI$ (o callback_URI) en la cabecera Location

OAuth 1.0

A diferencia de OAuth 2.0, todas las solicitudes de Token y las solicitudes de recursos protegidos deben ir firmadas por el consumidor y verificadas por el proveedor de servicios.

El propósito de firmar solicitudes es evitar que terceros no autorizados utilicen la Clave del consumidor y los tokens al realizar solicitudes de Tokens o solicitudes de recursos protegidos.

El proceso de firma codifica el Consumer Secret y el Token Secret en un valor verificable que se incluye con la solicitud.

OAuth no exige un método de firma particular, ya que cada implementación puede tener sus propios requisitos únicos. El protocolo define tres métodos de firma:

- **HMAC − SHA1**, **RSA − SHA1 y PLAINTEX**. El consumidor declara un método de firma en el parámetro
- **OAuth_signature_method** genera una firma y la almacena en el parámetro
- **OAuth_signature**

Otros conceptos:

Hipertexto

Texto que contiene elementos a partir de los cuales se puede acceder a otra información.

En el caso de una página web, el lenguaje utilizado para definir el hipertexto es HTML.

Universal Resource Locator URL

Es un URI que además de identificar un recurso, permite localizarlo en Internet porque la propia cadena describe la forma de acceder dicho recurso.

https://egela.ehu.eus/course/view.php?id=3032

En el servidor $egela.ehu.eus$ se encuentra un recurso, accesible mediante protocolo HTTP sobre SSL/TLS, cuyo path completo es $/course/view.php$. Es necesario pasar un parámetro, de nombre id, cuyo valor permite indicar la asignatura concreta a partir de cuyo contenido se debe generar la página web.

Para evitar ambigüedades, en general se recomienda utilizar el término URI.

Dependiendo del esquema utilizado el URI también podrá actuar como dirección del recurso

Proxy

Un proxy es un servidor que hace de intermediario en las peticiones de recursos que realiza un cliente a otro servidor.

Esta situación estratégica de punto intermedio le permite ofrecer diversas funcionalidades: control de acceso, registro del tráfico, restricción a determinados tipos de tráfico, mejora de rendimiento, anonimato de la comunicación, caché web, etc.

Dependiendo del contexto, la intermediación que realiza el proxy puede ser considerada por los usuarios, administradores o proveedores como legítima o delictiva y su uso es frecuentemente discutido

Analizador de protocolos

Se trata de una herramienta que sirve para desarrollar y depurar protocolos y aplicaciones de red. Permite al ordenador capturar diversas tramas de red para analizarlas (en tiempo real o después de haberlas capturado)

Página Web

Una página web es un documento HTML que puede embeber:

– imágenes

– estilos definidos mediante CSS.

– código JavaScript para modificar la interfaz de usuario de forma dinámica.

• Pero también es muy habitual almacenar las imágenes, los CSS y el JavaScript en ficheros que se referencian desde la página web. En este caso, después de recibir el documento HTML, el navegador analiza los enlaces (URIs) que forman la página HTML y realiza sendas peticiones HTTP para descargarlos.

Servlets

Son programas java que se ejecuta en el servidor para procesar la peticiones de un cliente.

- Los servlets reciben peticiones desde un cliente web, las procesan y
- devuelven una respuesta al navegador, normalmente en HTML.
- – Un servlet es un objeto java que pertenece a una clase que extiende de
- javax.servlet.http.HttpServlet

- $\boldsymbol{Server.xml}$ ⟶ Fichero principal de configuración de Tomcat que contiene la configuración de los puertos en que arranca este servicio.
- $\boldsymbol{Tomcat-users.xml}$ ⟶ Se establece la seguridad para el acceso a aplicaciones de administración.
- $\boldsymbol{Context.xml}$ ⟶ Configuración de contexto que es usado en todas las aplicaciones web. Normalmente se usa para configurar donde acceder al fichero de despliegue (web.xml)
- $\boldsymbol{Logging.properties}$ ⟶ configuración por defecto para logging en Tomcat

Preguntas de examen

Las tecnologías en las que se basan los sistemas web son:

- Hypertext Transfer Protocol HTTP
- Hypertext Markup Language HTML
- Uniform Resource Identifier URI

Campo de formulario → Crear un cuadro de texto

Hojas de estilos →Eliminar el subrayado de los textos con enlaces

Etiquetas $< meta >$ → Proporcionar información a los buscadores de una página web

Enlaces → Descargar un archivo de una página web

JavaScript → Crear ventanas de alerta para dar avisos al usuario

La web 2.0 es la utilización de la web de manera interactiva y participativa

En un documento HTML se puede insertar código JavaScript tanto en el $< head >$ como en el $< body >$

JavaScript y java son lenguajes completamente diferentes

En el fichero de despliegue web.xml el valor de $< servlet - name >$ y $< servlet - class >$ puede ser el mismo

-

Referencias

https://aprendiendoarduino.wordpress.com/tag/http/
https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/Identificaci%C3%B3n_recursos_en_la_Web

Manuales del Servlet

https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletRequest.html
https://docs.oracle.com/javaee/5/api/javax/servlet/http/package-summary.html
https://docs.oracle.com/javaee/5/api/javax/servlet/package-summary.html
https://docs.oracle.com/javaee/6/api/javax/servlet/http/HttpServletResponse.html

Listeners

https://docs.oracle.com/javaee/6/api/javax/servlet/ServletContextListener.html