UT 4 – Explotación y generación de Servicios en Red

Programación de Servicios y Procesos

Curso 2024-25

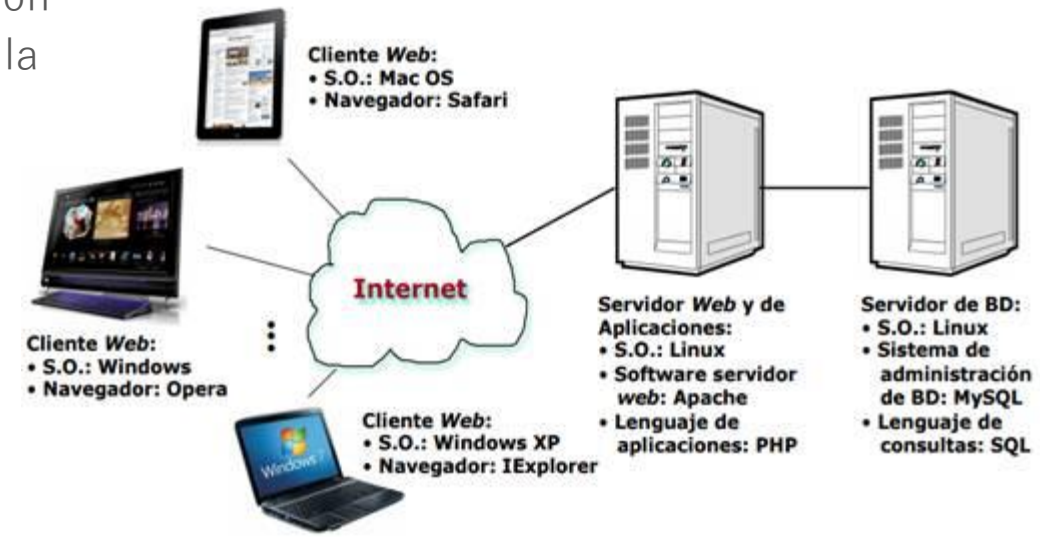
Profesor: Agustín González-Quel

Servicios

¿Qué es un servicio?

- Es un software que ofrece una funcionalidad dando respuesta a peticiones de otros sistemas.
- Los servicios se ejecutan en un equipo y, normalmente, está en permanente escucha ante peticiones que le puedan llegar
- El modelo más extendido es el cliente-servidor.

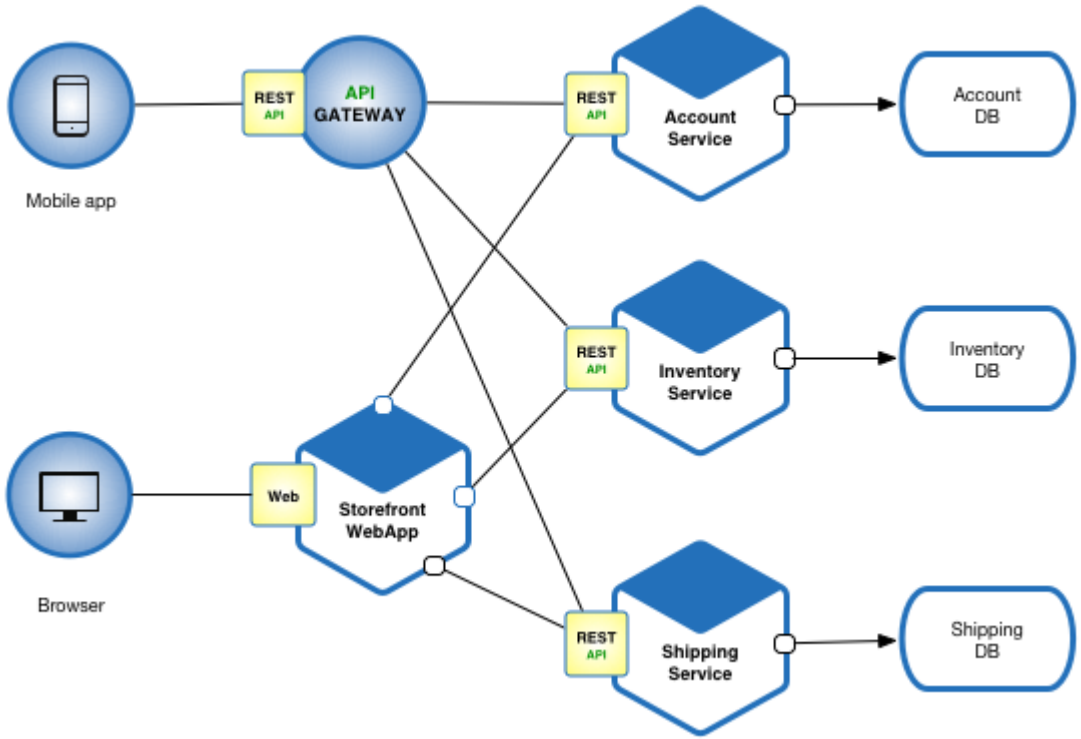
- Cómo funciona:
  - Los usuarios invocan la parte **cliente** de la aplicación
  - Ésta construye una solicitud para ese servicio y se la envía al **servidor** de la aplicación, usando habitualmente, TCP/IP como transporte.
  - El **servidor** recibe una solicitud, realiza el servicio requerido y devuelve los resultados en forma de una respuesta.



Componentes

Componentes

- Servidor: software que ofrece una funcionalidad o información.
  - Función que realiza.
  - Detalles técnicos de implementación: protocolos, lenguajes, BD,
  - Otras características: autenticación, seguridad, capacidad, etc.
- Cliente: software que consume dicha funcionalidad o información.
  - Debe estar construido según las especificaciones del servicio
- **Protocolo**: Conjunto de normas – técnicas y/o de negocio – que definen cómo se lleva a cabo la solicitud y prestación del servicio.
  - Mecanismo técnico: FTP, HTTP, etc.
  - Sintaxis y semántica de la comunicación
- Arquitectura: Estructura del servicio.



Algunos Servicios de uso común

Servicios:

- WWW
- Email
- Transferencia de archivos
- Conexión a equipo remoto
- Mensajería instantánea

Protocolos:

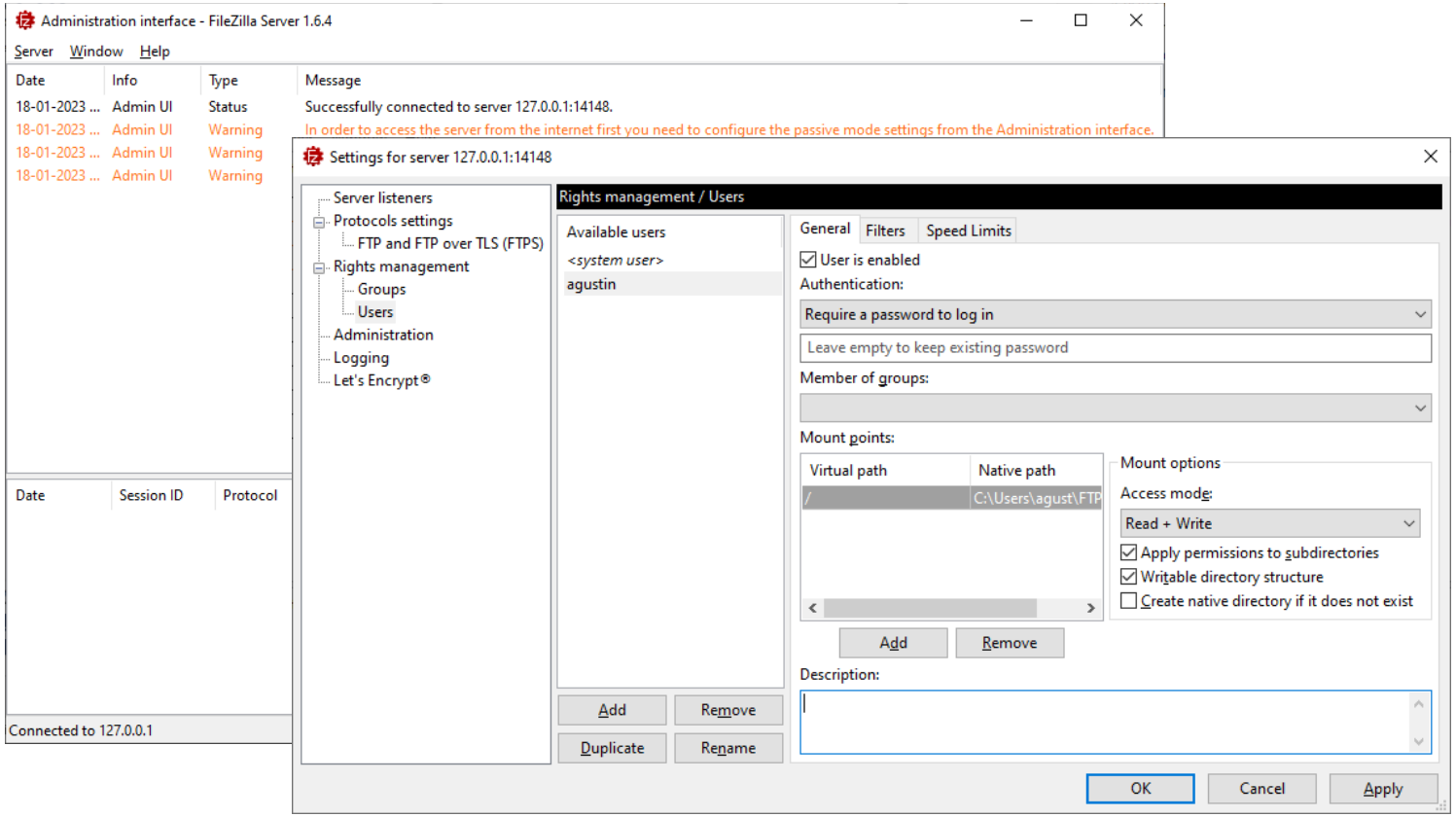
- FTP (File Transfer Protocol - Protocolo de transferencia de archivos)
- DNS (Domain Name System - Sistema de nombres de dominio).
- DHCP (Dynamic Host Configuration Protocol - Protocolo de configuración dinámica de anfitrión).
- HTTP (HyperText Transfer Protocol) para acceso a páginas web.
- HTTPS (Hypertext Transfer Protocol Secure)
- POP (Post Office Protocol) para recuperación de correo electrónico.
- SMTP (Simple Mail Transport Protocol) para envío de correo electrónico.
- SSH (Secure SHell)
- TELNET para acceder a equipos remotos.
- TFTP (Trivial File Transfer Protocol).
- LDAP (Lightweight Directory Access Protocol).
- XMPP, (Extensible Messaging and Presence Protocol) - Protocolo estándar para mensajería instantánea.
- WS (Websockets)

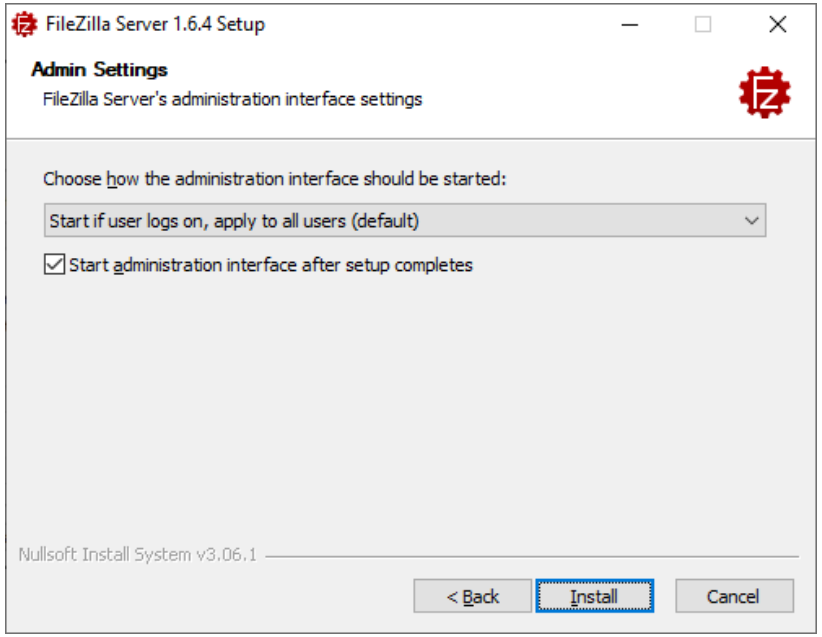
Servicio: FTP (File Transfer Protocol)

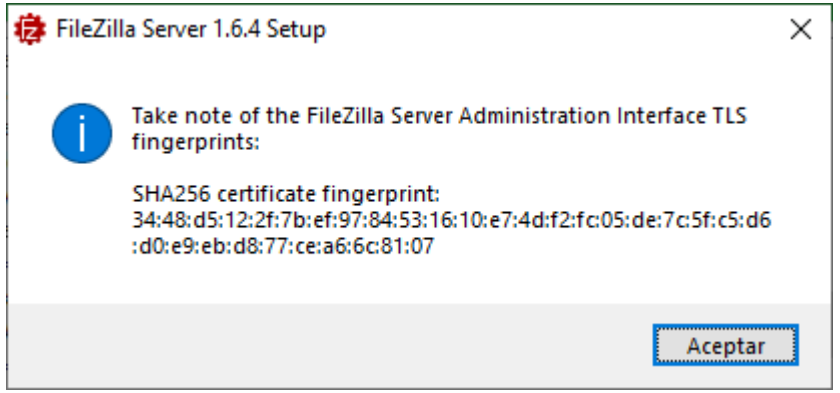
- Servicio que permite fácilmente transferir ficheros de un equipo a otro sobre una red TCP/IP.
- Es más eficiente que otros servicios para este propósito.
- Establece una conexión que **mantiene información del estado**: usuario, directorio de conexión, etc.
- Usos más típicos: Distribución de documentación, Backup.
- Tipos de acceso
  - FTP Anónimo.
  - Protegido por usuario.
  - FTP seguro sobre SSL/TLS, comunicación segura encriptada.

Servidor FTP

- Instalación y configuración de Filezilla Server
  - Instalación con *wizard* (Windows)
  - Al final de la instalación ofrece la huella del certificado del servidor
  - Interfaz gráfico de administración







Clientes FTP desde programa (1)

Librería **ftplib**

- https://docs.python.org/3/library/ftplib.html

Funciones más usadas

```
import ftplib

ftp = ftplib.FTP(FTP_HOST)   # FTP_TLS
ftp = ftplib.FTP(FTP_HOST, FTP_USER, FTP_PASS)

ftp.login(FTP_USER, FTP_PASS)
ftp.login(passwd="anonymous@")

ftp.encoding = "utf-8"
msg = ftp.getwelcome()

ftp.pwd()
ftp.cwd('pub/ubuntu/dists/xenial')

ftp.quit()
```

```
ftp.dir(funcion_callback)
    response = []
    ftp.dir(response.append)


ftp.retrlines("LIST")
# RETR, LIST, NLST devuelve solo nombres
ftp.storlines("STOR")
# Sube un fichero en modo no binario

ftp.retrbinary('RETR ' + filename, file.write, 1024)
ftp.storbinary(f"STOR {filename}", file)
```

Clientes FTP desde programa (2)

Librería **ftputil**

- https://ftputil.sschwarzer.net/documentation
- Funciones más usadas

```
import ftputil

ftp = ftputil.FTPHost(FTP_HOST, FTP_USER, FTP_PASS)

ftp.curdir
ftp.listdir(directorio)

ftp.download(remoto, local)
ftp.upload(local, remoto)

ftp.close()
```

FTP

Ejercicio 1

- Conectarse de forma anónima a ftp.osuosl.org
- Cambiarse a la carpeta 'pub/ubuntu/dists/xenial'
- Listar el contenido
- Bajarse el documento 'Release.gpg'

Ejercicio 2

- Conectarse de a mi equipo con usuario "alumno" // pwd: "ciud4d"
- Subir el fichero del programa del ejercicio 1
- Bajarse el fichero "calendario.pdf"

Email

- https://docs.python.org/3/library/smtplib.html
- https://docs.python.org/3/library/imaplib.html

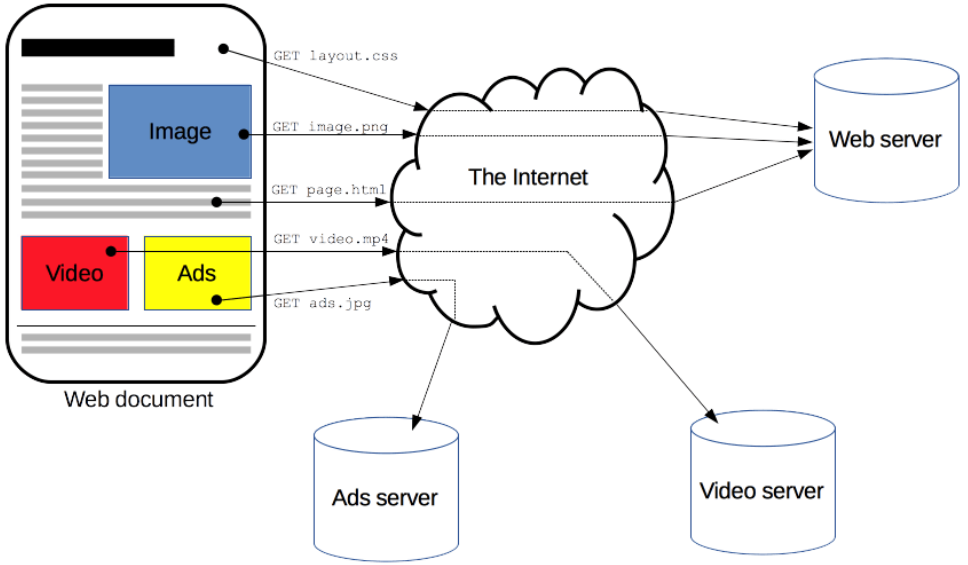
Entendiendo más HTTP

HTTP es un protocolo de la capa de aplicación para la transmisión de documentos HTML.

Fue diseñado para la comunicación entre los navegadores y servidores web, aunque se puede utilizar para otros propósitos también.

Sigue el clásico modelo cliente-servidor, en el que un cliente establece una conexión con el servidor, realiza una petición y espera hasta que recibe una respuesta del mismo.

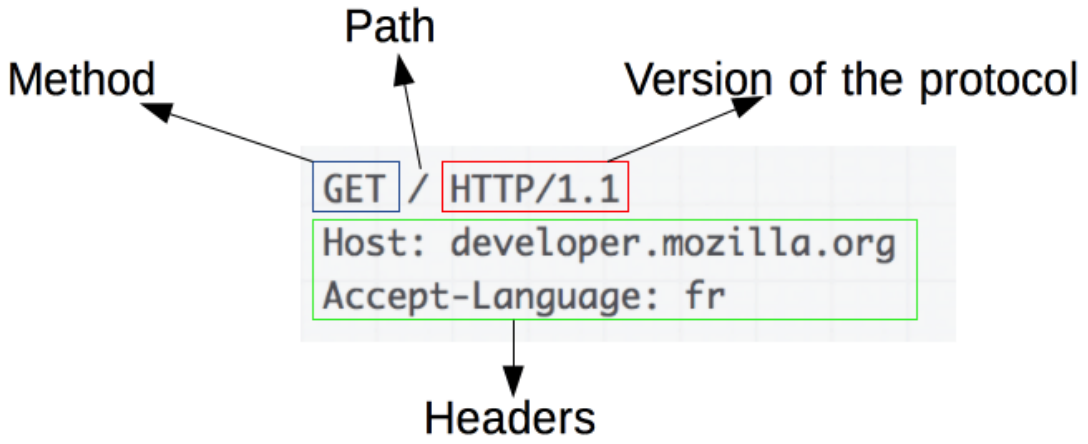
- HTTP es sencillo
- HTTP es un protocolo con sesiones, pero sin estados
  - El uso de HTTP cookies permite relacionar peticiones
  - con el estado del servidor.



Fuente: https://developer.mozilla.org/es/docs/Web/HTTP/Overview

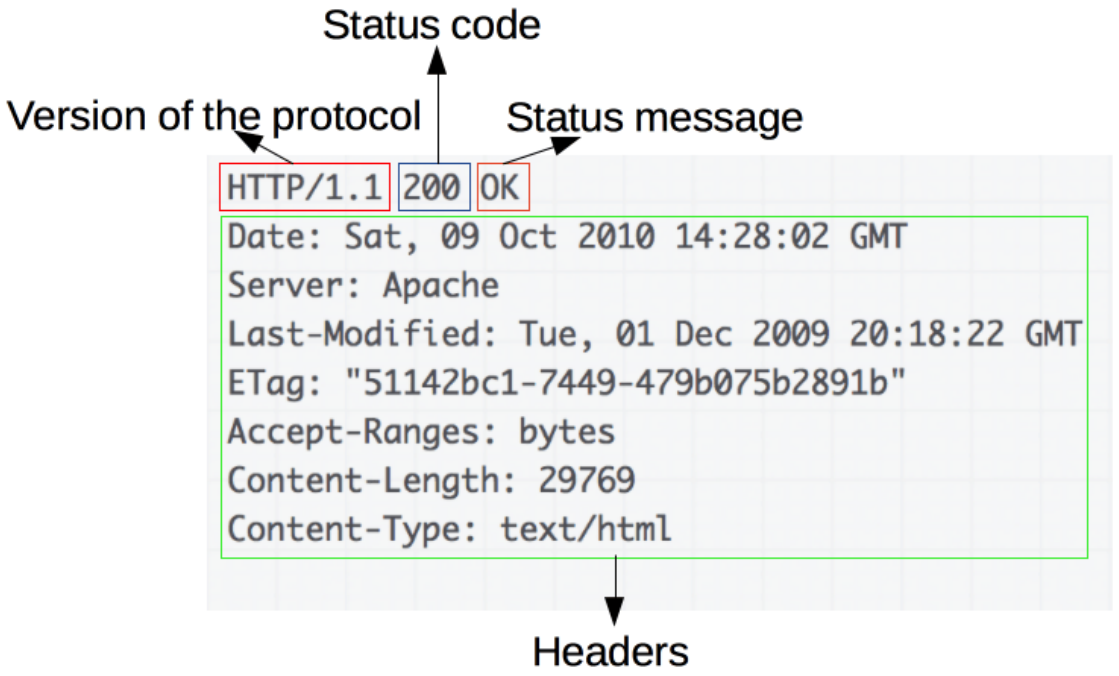
HTTP: Peticiones

- Un método HTTP
- La dirección del recurso pedido, URL del recurso (path)
- La versión del protocolo HTTP.
- Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
- Cuerpo de mensaje, en algún método, como puede ser POST/PUT, en el que se envía la información para el servidor.



HTTP: Respuestas

- La versión del protocolo HTTP que están usando.
- Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que.
- Un mensaje de estado, una breve descripción del código de estado.
- Cabeceras HTTP, como las de las peticiones.



Métodos

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado.

- **GET**: El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- **POST**: El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **PUT**: El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

Menos usados:

- HEAD: El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
- DELETE: El método DELETE borra un recurso en específico.
- CONNECT: El método CONNECT establece un túnel hacia el servidor identificado por el recurso.
- OPTIONS: El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
- TRACE: El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
- PATCH: El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

Códigos de respuesta HTTP

100 Series: Situaciones temporales

- 100 Continue
- 101 Switching Protocols
- 102 Processing

200 Series: Éxito

- 200 – OK
- 201 – Created
- 202 – Accepted
- 203 – Non-Authoritative Information
- 204 – No Content
- 205 – Reset Content
- 206 – Partial Content
- 207 – Multi-Status
- 208 – Already Reported
- 226 – IM Used

- **300 Series: Redirección URL**
  - 300 – Multiple Choices
  - 301 – Moved Permanently
  - 302 – Found
  - 303 – Check Other
  - 304 – Not Modified

- **300 Series: Redirección URL (cont)**
  - 305 – Use Proxy
  - 306 – Switch Proxy
  - 307 – Temporary Redirect
  - 308 – Permanent Redirect

- **400 Series: Error en la llamada.**
  - 400 – Bad Request
  - 401 – Unauthorised
  - 402 – Payment Required
  - 403 – Forbidden
  - 404 – Not Found
  - 405 – Method Not Allowed
  - 406 – Not Acceptable
  - 407 – Proxy Authentic
  - 408 – Request Timeout
  - 409 – Conflict
  - 410 – Gone
  - 411 – Length Required
  - 412 – Precondition Failed
  - 413 – Payload Too Large
  - 414 – URI Too Long
  - 415 – Unsupported Media Type

400 Series (cont)

- 416 – Range Not Satisfiable
- 417 – Expectation Failed
- 418 – I'm a teapot
- 421 – Misdirected Request
- 422 – Unprocessable Entity
- 423 – Locked
- 424 – Failed Dependency
- 426 – Upgrade Required
- 428 – Precondition Required
- 429 – Too Many Requests
- 431 – Request Header Fields Too Large
- 451 – Unavailable For Legal Reasons

- **500 Series: Error de servidor**
  - 500 – Internal Server Error
  - 501 – Not Implemented
  - 502 – Bad Gateway
  - 503 – Service Unavailable
  - 504 – Gateway Timeout
  - 505 – HTTP Version Not Supported
  - 506 – Variant Also Negotiates
  - 507 – Insufficient Storage
  - 508 – Loop Detected
  - 510 – Not Extended
  - 511 – Network Authentication Required

Trabajando con HTTP desde programa

Librería **requests**

- Permite hacer llamadas HTTP: páginas web, API REST, …
- https://requests.readthedocs.io/en/latest/
- No confundir con request: antigua librería que ya no existe.

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

```
>>> r = requests.get("http://www.marca.com")
>>> r.status_code
200
>>> r.encoding
'iso-8859-15'
>>> r.text
```

Squeezed text (9108 lines).

```
>>> r.json()
Traceback (most recent call last):
  File "C:\Program Files\Python312\Lib\site-packages\requests\models.py", line 9
74, in json
    return complexjson.loads(self.text, **kwargs)
  File "C:\Program Files\Python312\Lib\json\__init__.py", line 346, in loads
    return _default_decoder.decode(s)
  File "C:\Program Files\Python312\Lib\json\decoder.py", line 337, in decode
    obj, end = self.raw_decode(s, idx=_w(s, 0).end())
  File "C:\Program Files\Python312\Lib\json\decoder.py", line 355, in raw_decode
    raise JSONDecodeError("Expecting value", s, err.value) from None
json.decoder.JSONDecodeError: Expecting value: line 1 column 2 (char 1)
```

BeautifulSoup

Librería para poder trabajar de una forma estructurada con el texto de una página web

- https://beautiful-soup-4.readthedocs.io/en/latest/

Funcionamiento

- Instalación: pip install python3-bs4
- Parseamos el documento:

```
soup = BeautifulSoup(textoDelHTML, 'html.parser’)
```

Resultado de una llamada con requests

```
response = requests.get(url)
textoDelHTML = response.text
```

- A partir de aquí tenemos acceso a los elementos de la página de forma estructurada.

```
soup.title
soup.title.name
soup.title.string
soup.title.parent.name
soup.head
soup.pretify()
```

Campos y comandos de BS4

Conseguir el primer elemento de un determinado tipo

- soup.p
- soup.a

- Funciones find y find_all

  Devuelve una lista o [] si no encuentra nada

```
soup.find_all('a')
soup.find_all(True)
```

  Devuelve 1 resultado o None si no encuentra nada

```
soup.find(id="link3")
soup.find("head")
```

API – Application Programming Interface

Además de la navegación WWW el uso más extendido de HTTP es el soporte a las arquitecturas basadas en API

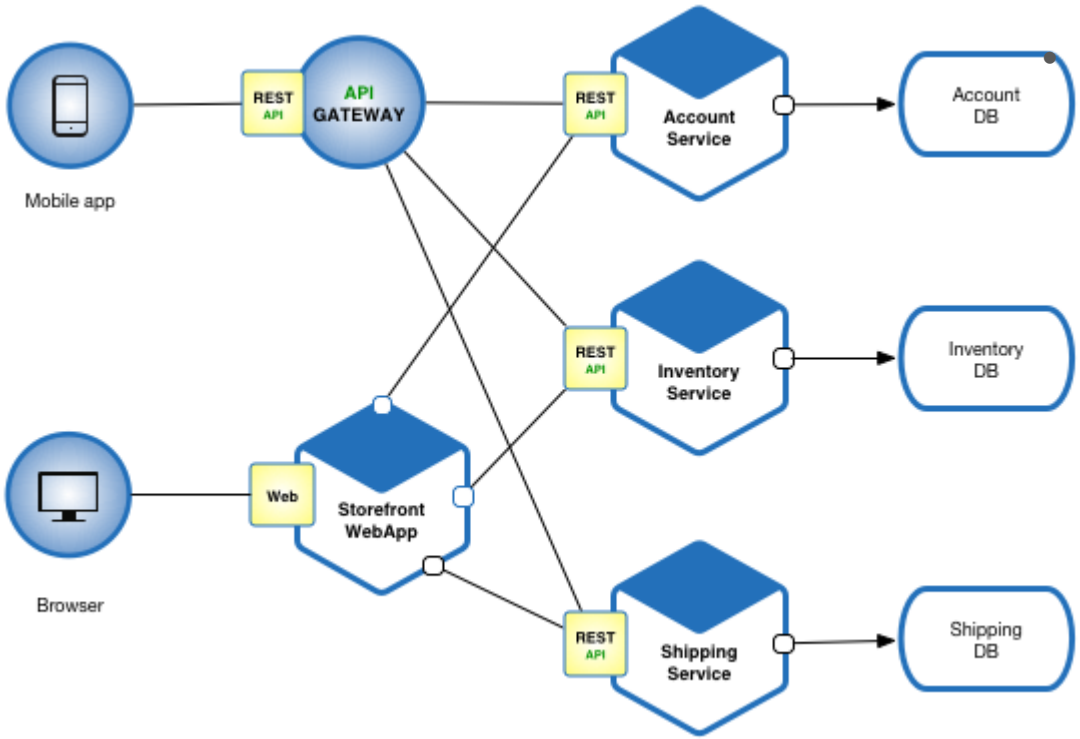
- Estas arquitecturas permiten estructurar las aplicaciones a partir de la composición de recursos propios o ajenos.
- Facilitan el reuso, escalabilidad y su mantenimiento.
- También permiten de forma más sencilla incorporar políticas de seguridad, autenticación y control

Dentro de los distintos tipos de API el mas usado es API REST

Tenemos un documento específico sobre este tema.

Arquitecturas de Microservicios

- https://microservices.io/patterns/microservices.html



- Ver el caso de uso en la web

Arquitecturas de Microservicios

- Estilo de arquitectura para desarrollar aplicaciones.
- Gracias a los microservicios, una aplicación grande puede separarse en partes independientes más pequeñas, cada una con su propio dominio de responsabilidad. Para servir una única solicitud de usuario,
- Una aplicación basada en microservicios puede llamar a muchos microservicios internos con los que preparar su respuesta.
- Cada microservicio implementa un servicio único creado para desempeñar una función de la aplicación y gestionar tareas independientes.
- Cada microservicio se comunica con los otros servicios a través interfaces sencillas y estandarizadas.

Fin Unidad 4