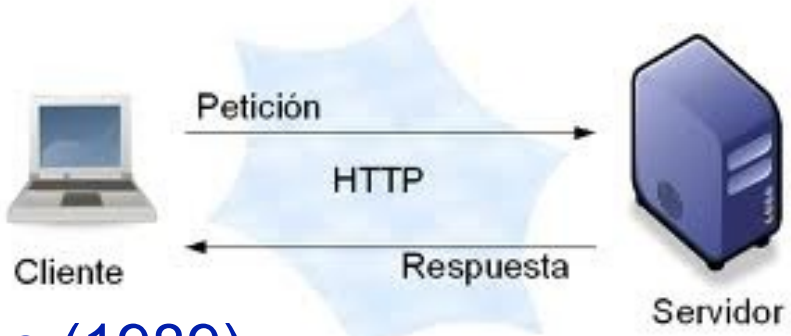
Introducción a la navegación Web:
Cliente, Servidor, URL, HTTP y HTML

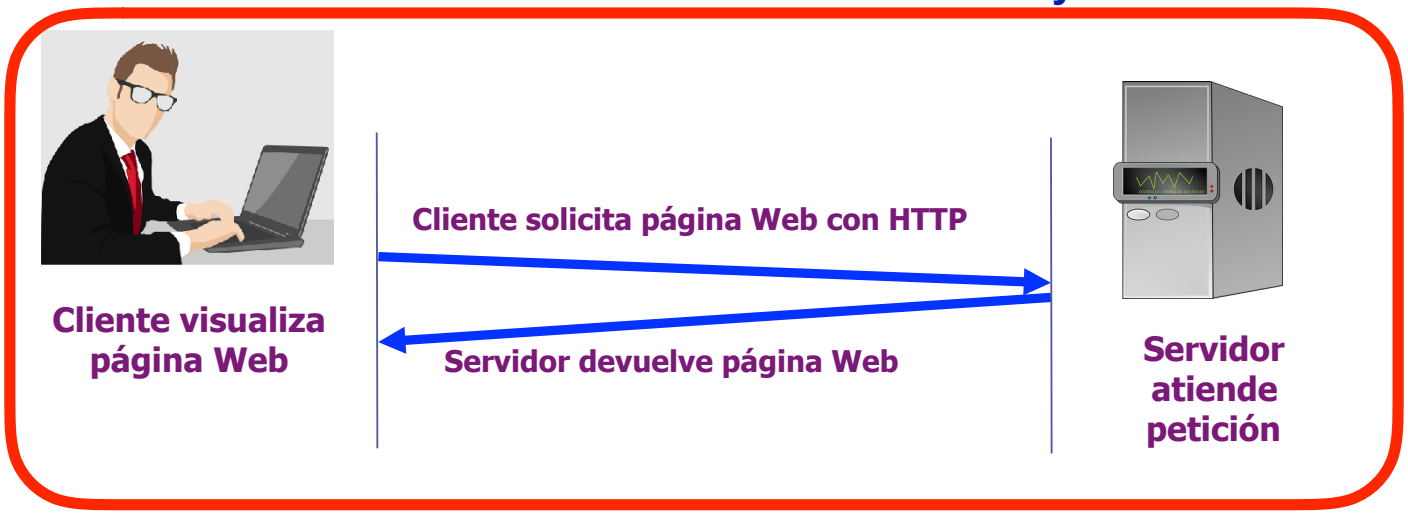
Juan Quemada, DIT - UPM

La Web



- Inventada por Tim Berners Lee (1989)
  - Servicio de publicación de documentos hipertexto en Internet

- Es el almacén de contenidos que necesitaba la red
  - Transforma Internet en una "**Red de distribución de contenidos**"
    - Crece continuamente -> es **descentralizada y escalable**

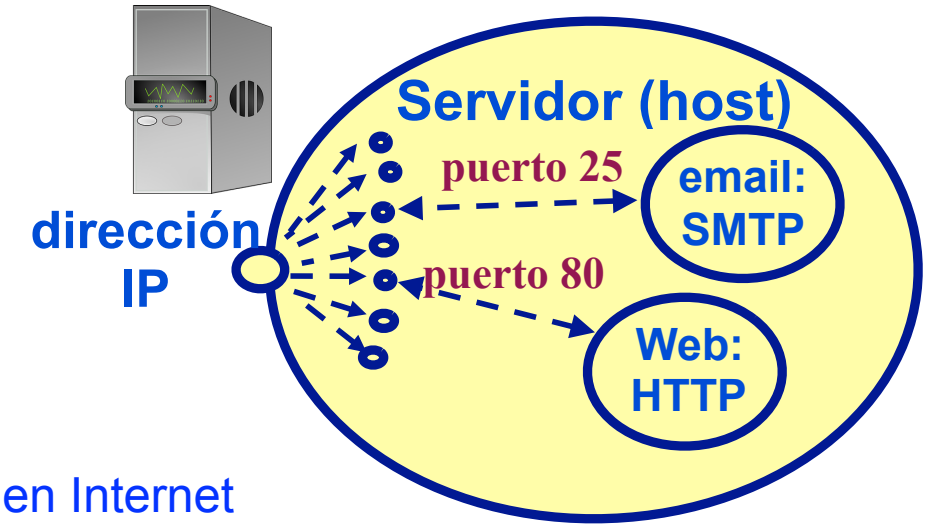


Clientes: dispositivos y navegadores

- Los clientes o usuarios acceden a los servicios de Internet
  - Utilizando dispositivos u ordenadores (llamados a veces clientes)
    - PCs, portátiles, tabletas, teléfonos o relojes inteligentes, etc
- **Cliente**: **programa** que da acceso a un servicio de Internet
  - El **cliente** de acceso a **Web** se denomina **navegador** (browser)
    - p.e. Chrome, Firefox, Internet Explorer, Opera, Safari, …

- El navegador trae **páginas Web** de un servidor y las muestra
  - Pide las páginas Web con transacciones **GET** del protocolo **HTTP**
    - El servidor devuelve páginas Web codificadas en **HTML** en la respuesta
  - La página Web tiene su estructura definida en **HTML**
    - HTML permite mostrarla al usuario con tipografías y formatos tipo documento impreso

Servidor y máquina servidora

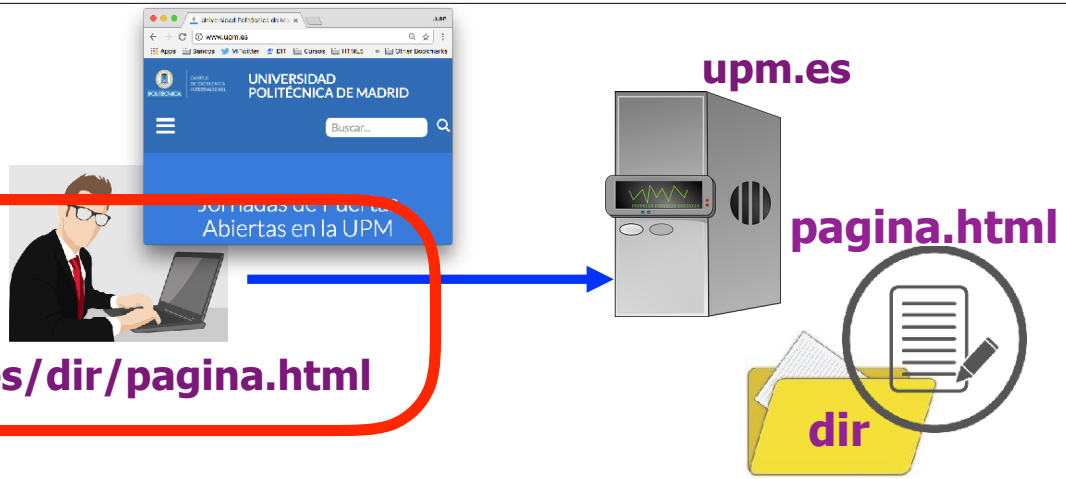


- **Máquina servidora**
  - Ordenador que alberga servidores en Internet
    - Tiene una **dirección "conocida"** en Internet que se utiliza para acceder
      - **Dirección simbólicas** (de dominio): google.com, upm.es, localhost (mi máquina), …
      - **Direcciones numéricas IP:** 192.9.0.144 (v4), 2001:db8::8a2e:370:7334 (v6), ..
- **Servidor**
  - **Programa** que atiende un servicio en un **puerto TCP**

Demo: nc, curl, ...

    - Los servidores son pasivos, esperan solicitudes de los clientes
      - Cada servicio tiene un protocolo de aplicación asociado: HTTP, SMTP, SSH, ..
- Algunos servicios típicos de Internet
  - Web: protocolo HTTP (puerto 80), HTTPS (puerto 443)
  - Email: protocolo SMTP (puerto 25), POP3 (110), IMAP (143)
  - Shell segura: protocolo SSH (puerto 22)
  - ….

URL

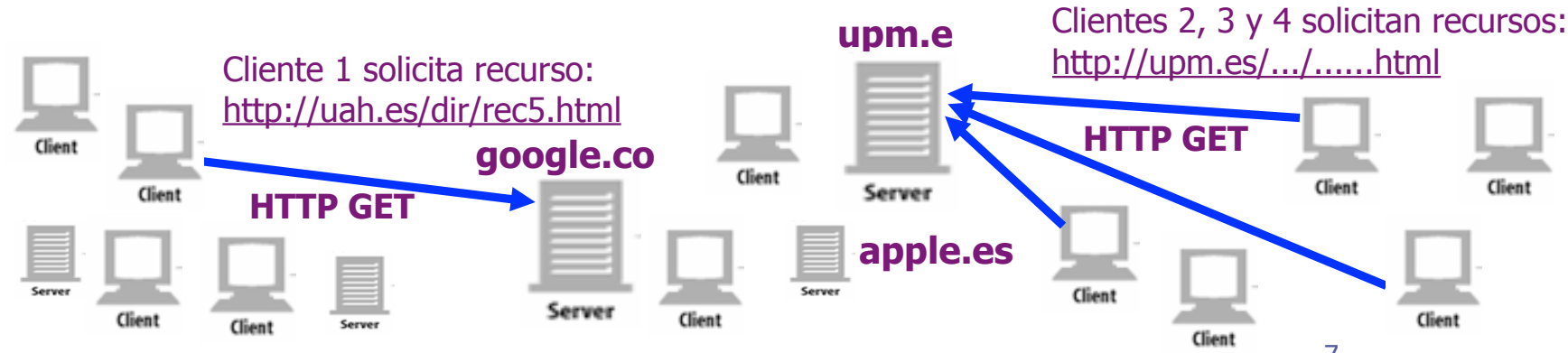


- **URL (Uniform Resource Locator)**
  - **Dirección** utilizada por un cliente para acceder a un recurso/servicio
- El URL apareció para acceder a servidores Web
  - Pero se ha generalizado a casi todos los servicios de Internet
- Algunos ejemplos de URLs para distintos servicios
  - **URL Web**: utiliza HTTP para acceder a recursos, incluye
    - Por ejemplo: **http://upm.es/dir/pagina.html**
  - **URL de correo** (email): identifica el buzon de usuario, incluye
    - Por ejemplo: **mailto:pepe_garcia@gmail.com**
  - Y muchos otros

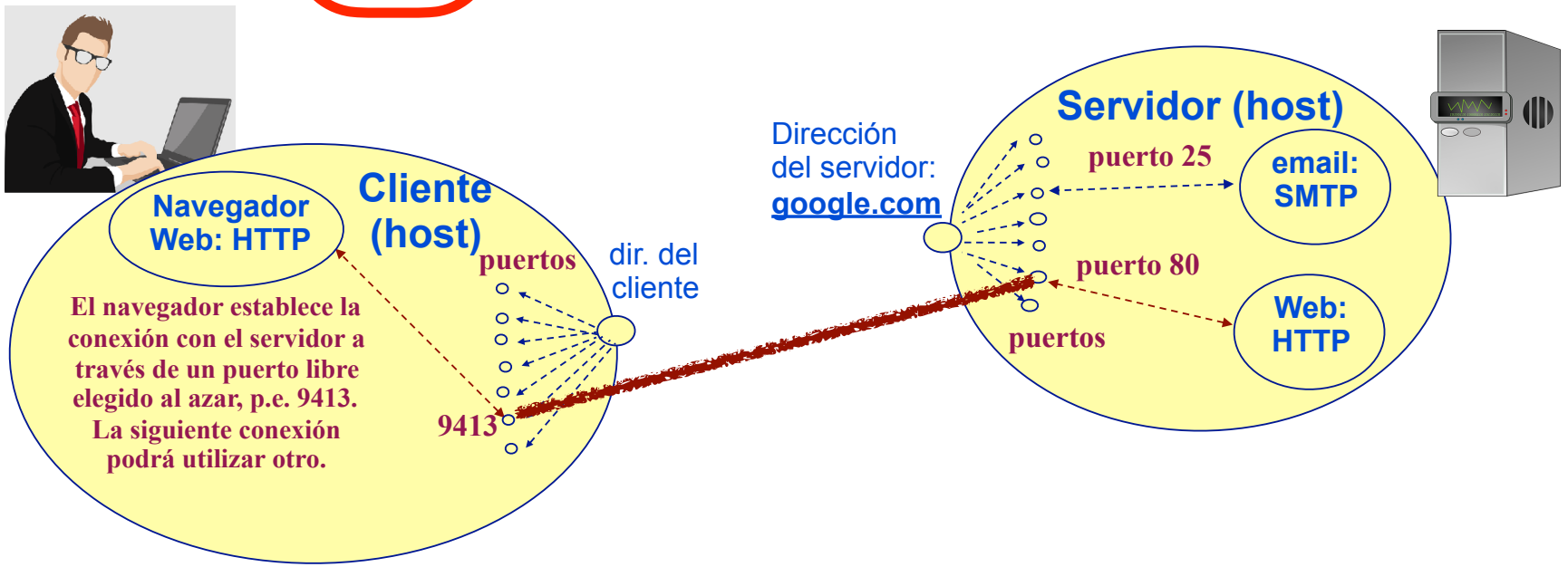

```
┌─────────────────────────────────────────────────────────────────────────────────────────┐
│                                          href                                           │
├──────────┬┬─────────────┬──────────────────────┬──────────────────────────────┬──────────┤
│ protocol ││    auth     │         host         │             path             │   hash   │
│          ││             ├──────────┬───────────┼──────────┬───────────────────┤          │
│          ││             │ hostname │   port    │ pathname │      search       │          │
│          ││             │          │           │          ├─┬─────────────────┤          │
│          ││             │          │           │          │ │      query      │          │
"  http:    // user:pass @ host.com : 8080    /p/a/t/h   ?   query=string    #hash "
│          ││             │          │           │          │ │                 │          │
└──────────┴┴─────────────┴──────────┴───────────┴──────────┴─┴─────────────────┴──────────┘
(all spaces in the "" line should be ignored -- they are purely for formatting)
```

HTTP (HiperText Transfer Protocol)

- Protocolo del Web
  - Procesa recursos identificados por un URL en un servidor remoto
- Métodos o comandos principales de HTTP
  - **GET**: trae al cliente (lee) un recurso identificado por un **URL**
  - POST: crea un recurso identificado por un URL
  - PUT: actualiza un recurso identificado por un URL
  - DELETE: borra un recurso identificado `pr un URL
  - ..... (hay mas comandos)



Conexión HTTP entre cliente y servidor

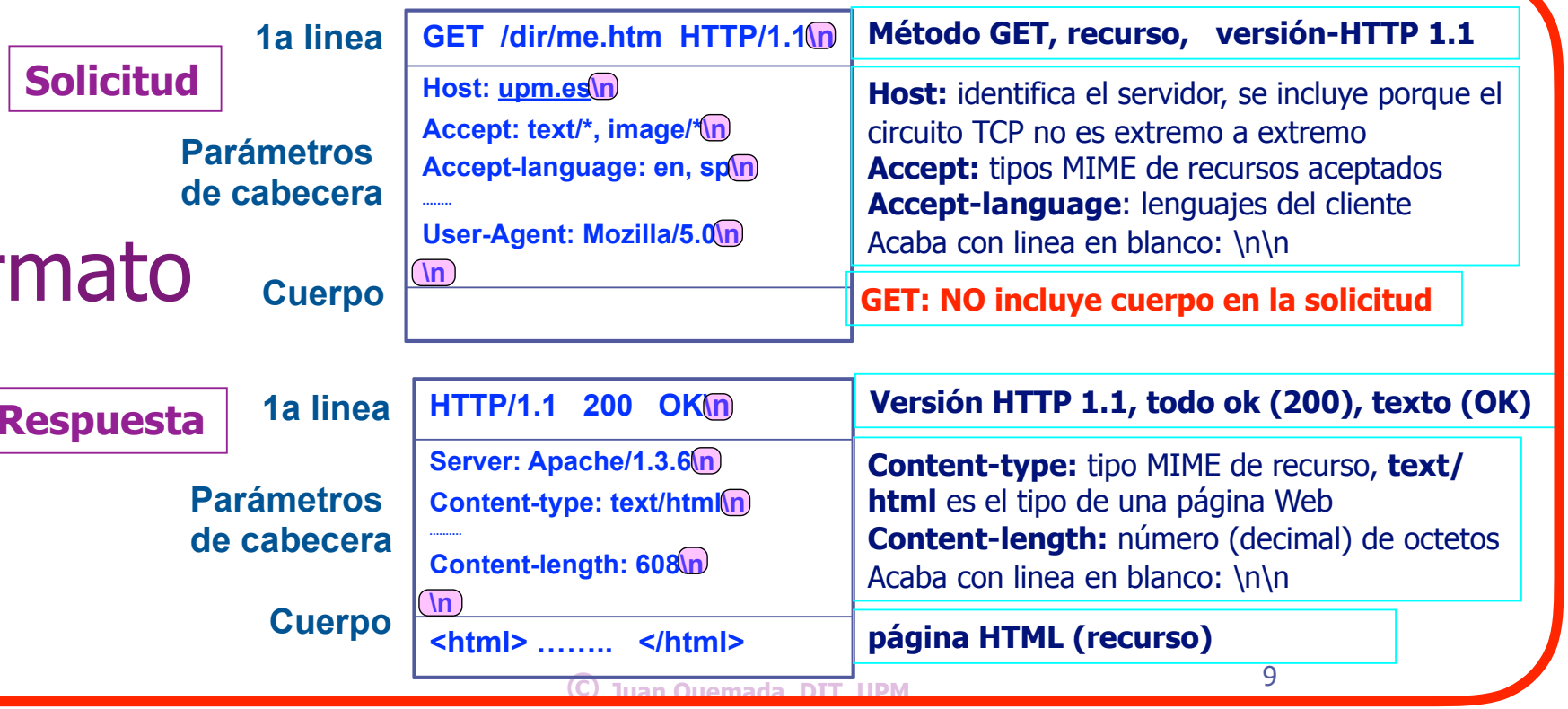


- El cliente establece una **conexión HTTP** con el servidor
  - El **circuito virtual TCP** es la forma más sencilla de conexión HTTP
    - El puerto del cliente se suele elegir **al azar**
    - El servidor se conecta al **puerto por defecto** (conocido), para HTTP es el **puerto 80**
      - Se puede conectar a **otro puerto** (p.e. **6000**) incluyéndolo así: google.com:6000
- Cliente y servidor interaccionan a través de la conexión HTTP
  - El cliente envía **Solicitudes** y el servidor devuelve **Respuestas**
    - **Solicitudes** y **Respuestas** son bloques de octetos, la mayor parte es texto (string)

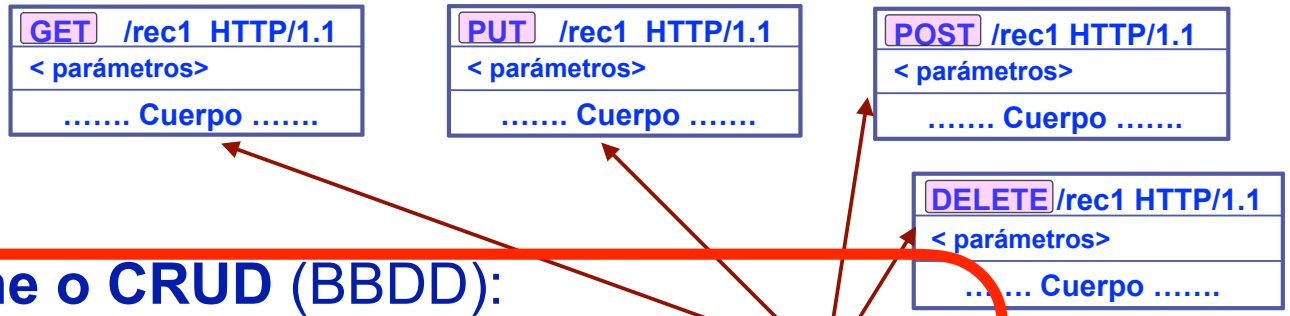
8

- **Mensajes** (solicitud o respuesta)**:** constan de **Cabecera** y **Cuerpo**.
- **Cabecera: string** formado por **1a linea** y **parámetros**. Acaba con una línea en blanco (\n\n).
  - **Primera línea de la solicitud:** incluye el **método**, la **ruta o path** que identifica el recurso en el servidor y la **versión** de HTTP utilizada por el cliente.
  - **Primera línea de la respuesta:** incluye **versión** HTTP del servidor, **código** y un **mensaje** de texto explicativo de la respuesta.
  - **Parámetros de la cabecera:** Cada parámetro es un string con el formato: **Nombre: valor**, que ocupa una línea (acaba con \n).
- **Cuerpo**: incluye el recurso enviado que puede ser de cualquier tipo, p.e. string, imagen, ….

Formato



Métodos HTTP



Interfaz Uniforme o CRUD (BBDD):

- **POST**: Crear un recurso en el servidor (**C**reate)
- **GET**: Traer un recurso al servidor (**R**ead)
- **PUT**: Modificar un recurso del servidor (**U**pdate)
- **DELETE**: Borrar un recurso del servidor (**D**elete)

La **interfaz uniforme** o **CRUD** es la base de las **arquitecturas orientadas a recursos** (ROA) y de las interfaces **REST**.

Las **aplicaciones de servidor** utilizan habitualmente la **interfaz uniforme** con transacciones HTTP **POST**, **GET**, **PUT** y **DELETE** para gestionar recursos en un servidor remoto. Los **recursos** pueden ser **ficheros** o **datos en una BBDD**. El **programador** decide como atiende cada **solicitud**.

más métodos

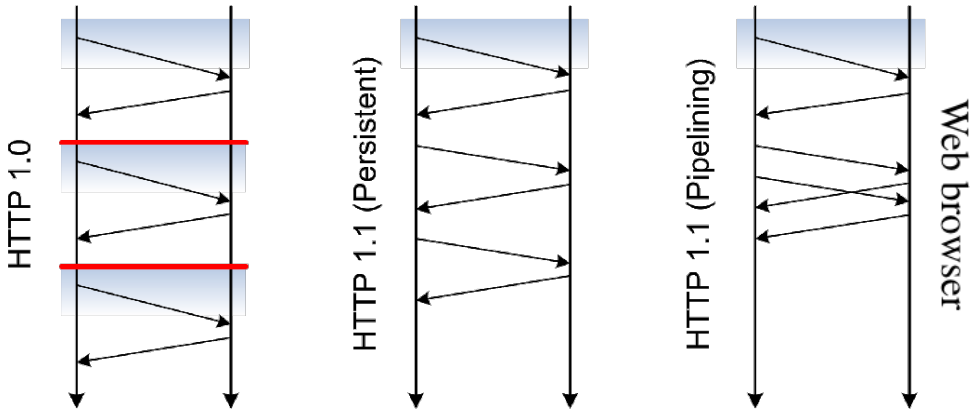
- HEAD: similar a GET, pero solo pide cabecera al servidor
- OPTIONS: Determinar qué métodos acepta un servidor

TRACE: Trazar proxies, caches, … hasta el servidor

CONNECT: Conectar a un servidor a través de un proxy

……

10



Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Códigos de estado de un servidor Web

- Respuestas informativas (1xx)
  - 100 Continue // Continuar solicitud parcial
- Solicitud finalizada (2xx)
  - 200 OK // Operación GET realizada satisfactoriamente, recurso servido
  - 201 Created // Recurso creado satisfactoriamente con POST, PUT
  - 206 Partial Content // para uso con GET parcial
- Redirección (3xx)
  - 301 Moved Permanently // Recurso se ha movido, cliente debe actualizar el URL
  - 303 See Other // Envía la URI de un documento de respuesta
  - 304 Not Modified // Cuando el cliente ya tiene los datos
- Error de cliente (4xx)
  - 400 Bad request // Comando enviado incorrecto
  - 404 Not Found // Recurso no encontrado, no hay ningún fichero con ese path
  - 405 Method Not Allowed // Método no permitido, p.e. se solicita método POST, PUT, ….
  - 409 Conflict // Existe conflicto con el estado del recurso en el servidor
  - 410 Gone // Recurso ya no esta
- Error de Servidor (5xx)
  - 500 Internal Server Error // El servidor tiene errores, p.e. error lectura disco, ….

Un servidor Web estático suele utilizar las respuestas marcadas

Un servidor Web estático suele utilizar las respuestas marcadas

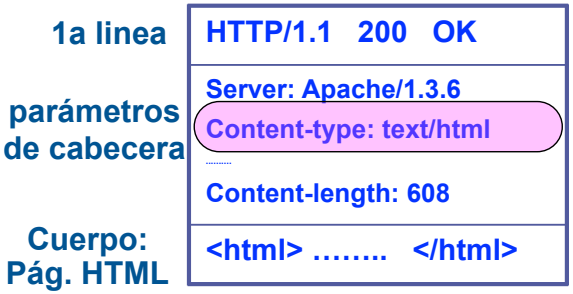
Un servidor Web estático suele utilizar las respuestas marcadas

Tipos MIME

Solicitud HTTP GET

- Tipos MIME: definen el tipo de un recurso
  - Aparecieron en email para tipar ficheros adjuntos
    - Su uso se ha extendido a otros protocolos y en particular a HTTP
      - Tipos: http://www.iana.org/assignments/media-types/media-types.xhtml
- Un tipo MIME tiene 2 partes **tipo / subtipo,**
  - Tipos: application, audio, example, image, message, model, multipart, text, video
- Ejemplos:
  - **image/gif**, **image/jpeg**, **image/png**, **image/svg**, …..
  - **text/plain**, **text/html**, **text/css**, ……
  - **application/javascript**, **application/msword**, …..
  - ……..

Respuesta HTTP GET



- HTTP utiliza el tipo mime para tipar el contenido del cuerpo (body)
  - Cabecera Request: **"Accept: text/html, image/png, ..."**
  - Cabecera Response: **"Content-type: text/html"**

13

URL HTTP

URLs de páginas Web

URL: dirección de un **recurso** de Internet.

Un recurso es cualquier cosa que necesite ser referenciada en Internet.

El URL tiene el siguiente formato:

<schema:><//><authority></path><?query><#fragment>

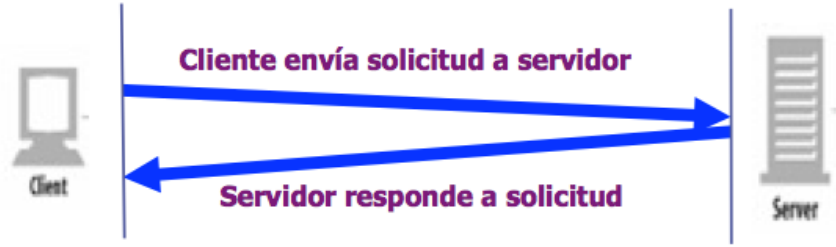
<schema:> = protocolo de acceso al recurso
<authority> = **<UserInfo@><host><:port>**
</path> = fichero, incluyendo camino en el servidor
<?query> = parámetros
<#fragment> = fragmento o parte del recurso

Algunos ejemplos de URLs de páginas Web

http://dit.upm.es/core/programa.html#tema3
http://192.168.1.42:8080/core/programa.html#tema3

HTTP

- Protocolo de **acceso a recursos**
  - identificados con un **URL**
    - utilizando una **conexión TCP**
      - entre cliente y servidor



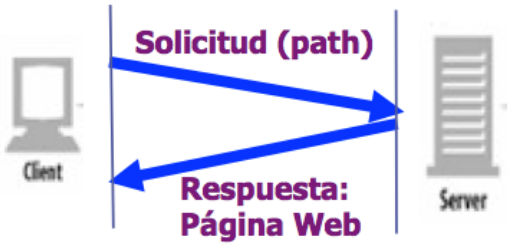
- Transacciones HTTP
  - Son de tipo: **Solicitud - Respuesta**
    - Cabecera es un **string de texto** (igual que en email)
- Métodos o comandos de HTTP
  - **GET**, **POST**, **PUT**, **DELETE,** HEAD, TRACE, ...
- Protocolo **extensible** que ha evolucionado
  - HTTP 0.9, 1.0, 1.1 (1999) y 2.0 (en preparación)

GET

URL: http://upm.es/dir/hola.html

GET: comando HTTP que permite traer un recurso Web (identificado con un URL) desde un servidor.

El cliente envía **solicitud** al servidor que devuelve **respuesta**, ambas con el siguiente formato:



Solicitud

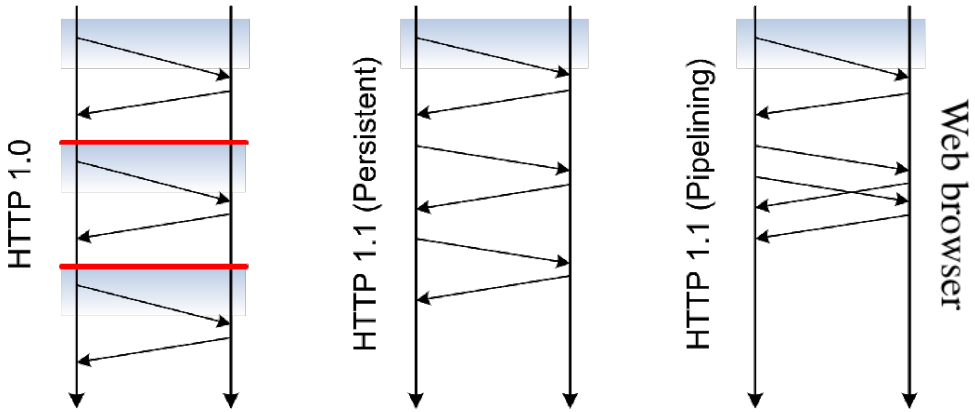
1a linea	GET /dir/hola.html HTTP/1.1	Comando, path, versión-HTTP
Cabecera	Host: upm.es Accept: text/*, image/* Accept-language: en, sp ..... User-Agent: Mozilla/5.0	Cada línea es un parámetro con formato "Nombre: valor" Cabecera acaba con línea en blanco
Cuerpo		GET: NO incluye cuerpo en la solicitud

Respuesta

1a linea	HTTP/1.1 200 OK	Versión-HTTP, resultado, mensaje
Cabecera	Server: Apache/1.3.6 Content-type: text/html ..... Content-length: 608	Cada línea es un parámetro con formato "Nombre: valor" Cabecera acaba con línea en blanco
Cuerpo	<html> …….. </html>	Cuerpo con fichero (página HTML)

Códigos Respuestas HTTP

- Respuestas informativas (1xx)
  - 100 Continue // Continuar solicitud parcial
- Solicitud finalizada (2xx)
  - 200 OK // Operación GET realizada satisfactoriamente, recurso servido
  - 201 Created // Recurso creado satisfactoriamente con POST, PUT
  - 206 Partial Content // para uso con GET parcial
- Redirección (3xx)
  - 301 Moved Permanently // Recurso se ha movido, cliente debe actualizar el URL
  - 303 See Other // Envía la URI de un documento de respuesta
  - 304 Not Modified // Cuando el cliente ya tiene los datos
- Error de cliente (4xx)
  - 400 Bad request // Comando enviado incorrecto
  - 404 Not Found // Recurso no encontrado, no hay ningún fichero con ese path
  - 405 Method Not Allowed // Método no permitido, p.e. se solicita método POST, PUT, ….
  - 409 Conflict // Existe conflicto con el estado del recurso en el servidor
  - 410 Gone // Recurso ya no esta
- Error de Servidor (5xx)
  - 500 Internal Server Error // El servidor tiene errores, p.e. error lectura disco, ….

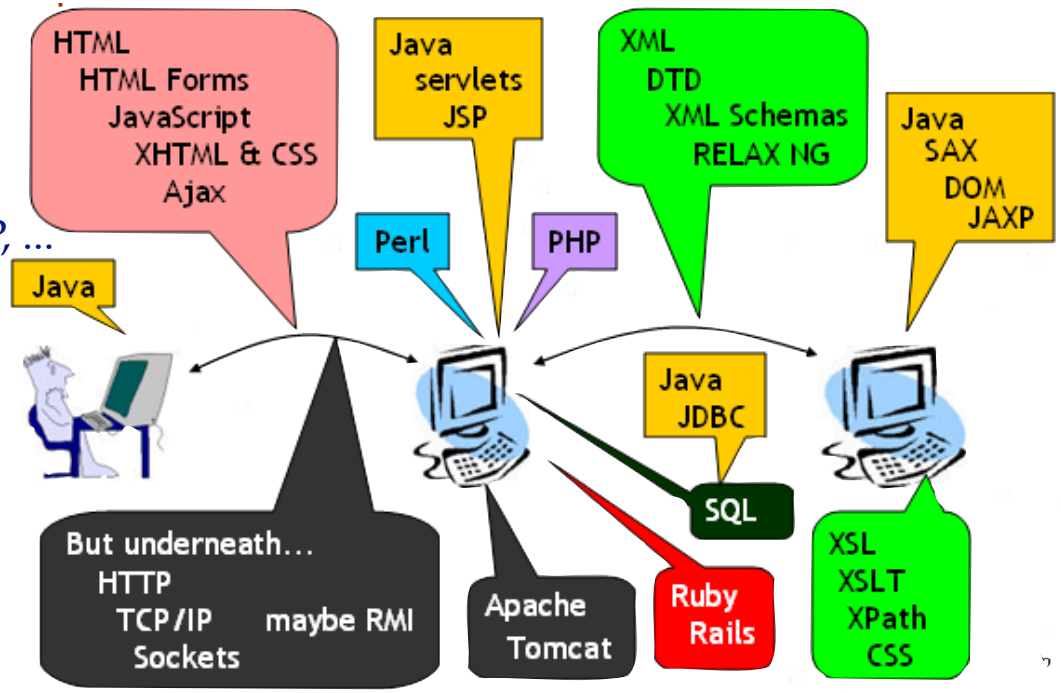


Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Desarrollar un Servidor

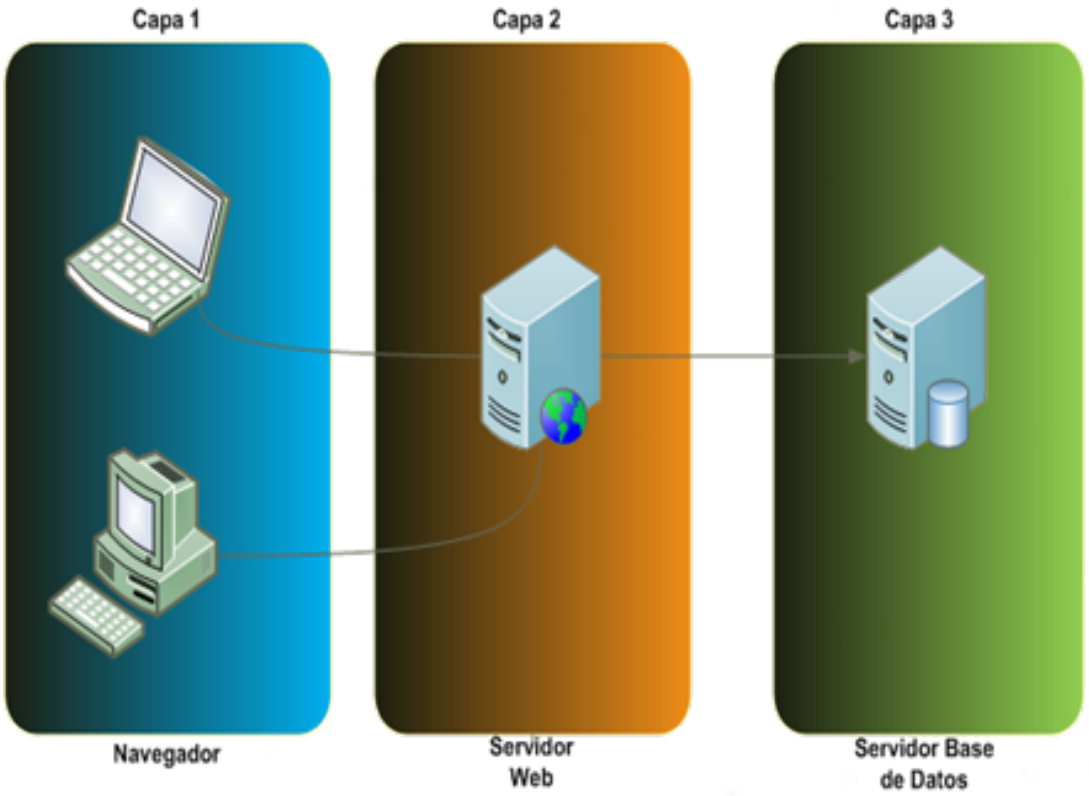
Opciones

- Servidores Web
  - Apache, Ngnix, ...
- Servidor de aplicaciones:
  - Java EE, Rails, Sinatra, Nodejs, PHP, ...
- Frameworks:
  - expressjs, angularjs, ...
- Vistas:
  - JSP, ERB, EJS, Jade, ...
- Bases de datos
  - NoSQL: MongoDB, CouchDB
  - SQL: SQLite, MySQL, Postgres, Oracle
- Despliegue:
  - Heroku, Joyent, Nodejitsu, ...



Servidor: Arquitectura en Tres Capas

- Frontend
  - Las vistas
- Middleware
  - La lógica de la aplicación
- Backend
  - Persistencia de la información



Node.js

http://nodejs.org

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Nodejs: net > http > express

- Desarrollo de un servidor con Nodejs:
  - Podemos desarrollar un servicio web usando módulos de bajo nivel:
    - **Net** o **HTTP**.
      - Es un nivel demasiado bajo que nos obliga a escribir mucho código.
      - Y ese código es igual/repetido en todos los servicios web.
  - Mejor usar algún framework de más alto nivel:
    - **Express**.
      - Proporciona rutas, middlewares, …
  - O de más alto nivel:
    - **Sails**, **Loopback**, ...

Servicios Web usando módulo **Net**

Usando el Módulo **Net**

- El módulo **Net** permite crear un servidor que atiende las conexiones TCP realizadas por los clientes.
- Nosotros tenemos que implementar el protocolo HTTP sobre esto.
  - Implementar escuchadores para los eventos: **connection**, **data**, **error**, **end**, ...
  - Analizar los datos recibidos:
    - Método de la petición HTTP,
    - Versión del protocolo,
    - URL (ruta, query, ...)
    - Cabeceras (tipo de contenidos, codificación, autenticación, cookies, tamaño, caches, ...),
    - Datos
  - Devolver una respuesta HTTP para cada petición HTTP.
    - Código de respuesta, cabeceras, datos.

Ejemplo: Servidor HolaMundo

```
var net = require('net');

var body = '<html><head><title>Hola Mundo</title></head>'+
           '<body>Hola Mundo</body></html>';

net.createServer(function(socket) {

    // No miro nada.
    // Al recibir cualquier cosa: log, contesto y cierro.

    socket.on('data',function(data) {

       console.log(data.toString());

       socket.write('HTTP/1.1 200 OK\n');
       socket.write('Content-Length: '+ body.length+'\n');
       socket.write('Content-Type: text/html\n');
       socket.write('\n');
       socket.end(body);
    });
}).listen(3000);
```

Falta analizar los datos que se vayan recibiendo para contestar adecuadamente: Cabeceras, formatos, codificaciones, url, query, datos, versiones, etc...

Función a ejecutar cada vez que se conecta un cliente.

Función a ejecutar cuando socket genera el evento 'data'

Envío cabecera

Termino enviando el body.

Petición y respuesta HTTP intercambiadas en el ejemplo anterior

- **Petición HTTP:**

```
GET / HTTP/1.1
Host: localhost:3000
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4)
       AppleWebKit/536.11 (KHTML, like Gecko)
       Chrome/20.0.1132.57 Safari/536.11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
       */*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: es-ES,es;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

- **Respuesta HTTP:**

```
HTTP/1.1 200 OK
Content-Length: 74
Content-Type: text/html

<html><head><title>Hola Mundo</title></head>
       <body>Hola Mundo</body></html>
```

Probar el Servidor

- Crearse un fichero con el código del servidor.
  - Llamar al fichero `HolaMundo.js`
- Desde un terminal lanzar el servidor:

```
$ node HolaMundo.js
```

- Desde un navegador conectarse a:

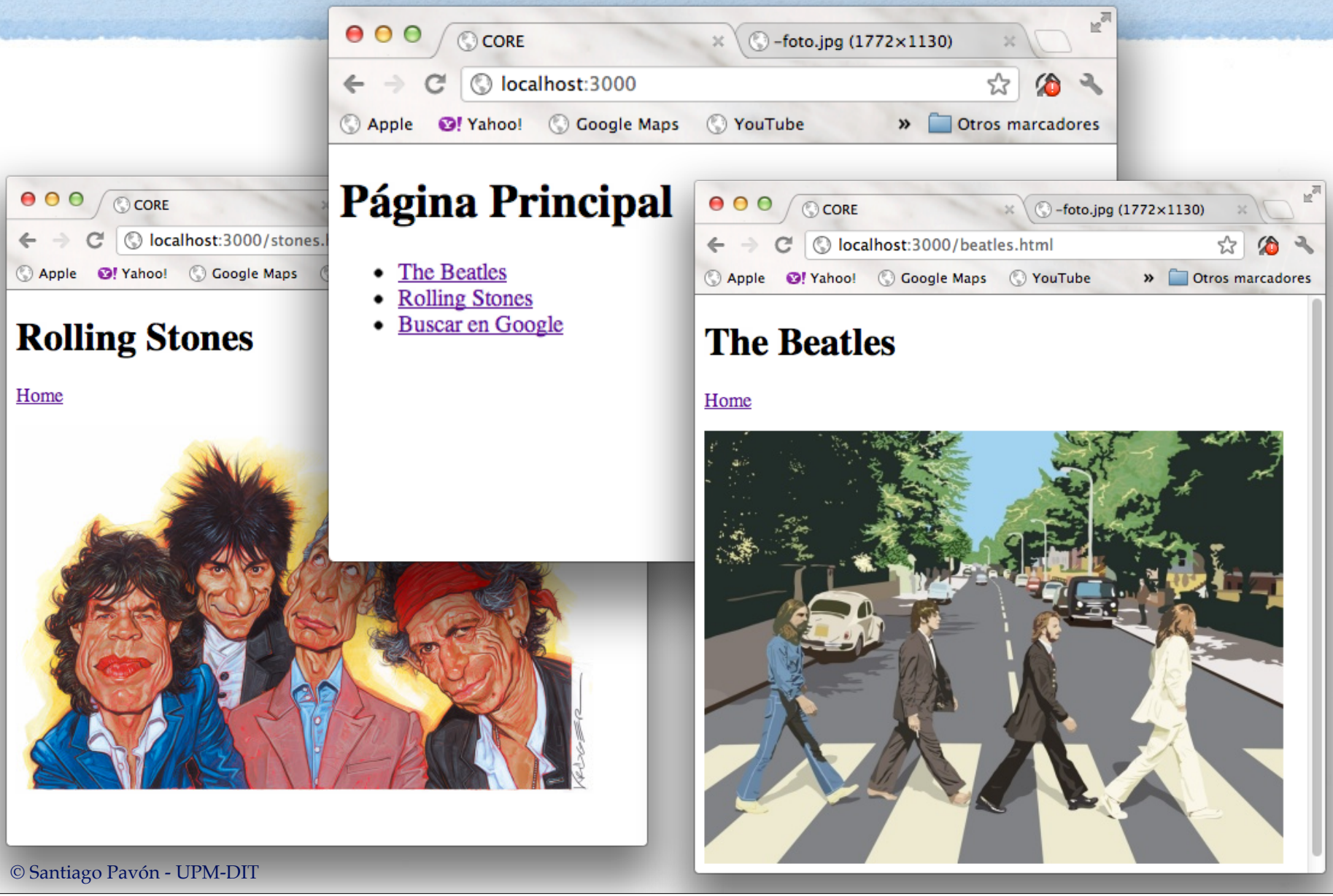
```
http://localhost:3000
```

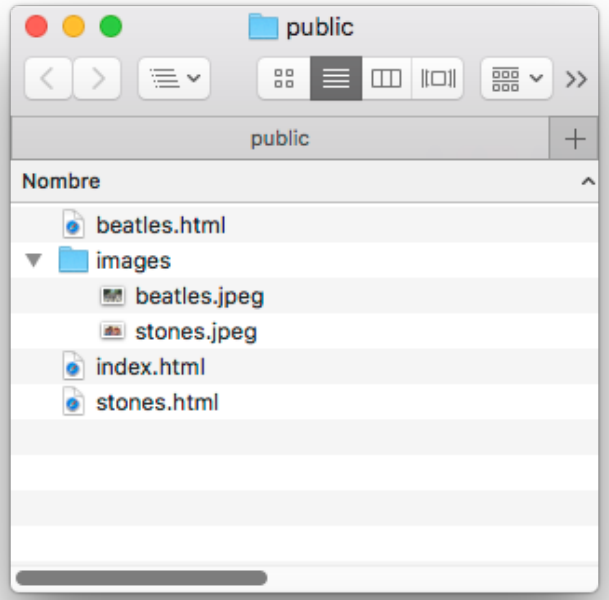
  - Inspeccionar los mensajes intercambiados con las herramientas de desarrollo web del navegador.
- Desde un terminal conectarse con:

```
$ telnet localhost 3000
```

  - Enviar cualquier texto.

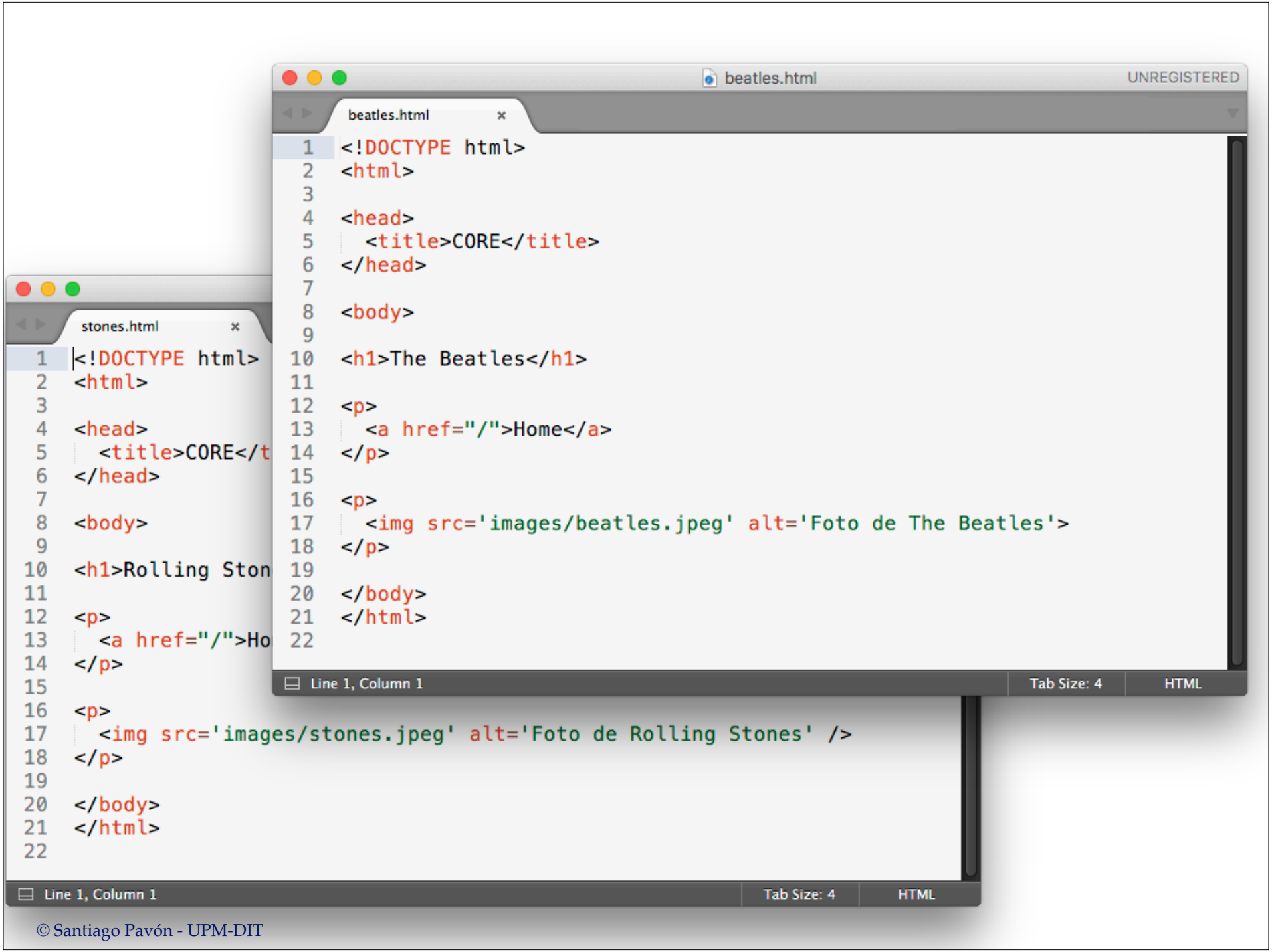
Ejemplo: Servidor Páginas Estáticas





beatles.jpeg

stones.jpeg



Ejemplo: Servidor Páginas Estáticas

```
var net = require("net");
var path = require("path");
var fs = require("fs");
var url = require('url');

var mimeTypes = {
    "html": "text/html",
    "jpeg": "image/jpeg",
    "jpg": "image/jpeg",
    "png": "image/png",
    "js": "text/javascript",
    "css": "text/css"
};

// Crear socket de servidor
net.createServer(function(socket) {

  socket.on('data', function(data){
        // PROCESAR LOS DATOS RECIBIDOS
  });
})
.listen(3000);
```

Función a ejecutar cada vez que se conecta un cliente.

Función a ejecutar cuando socket genera el evento 'data'

// PROCESAR LOS DATOS RECIBIDOS (1/2)

```
// Extraer metodo, url y version HTTP:
var request = data.toString();

var matches = request.match(/^(\S+)\s(\S+)\s(\S+)\s/);
var req_method  = matches[1];
var req_url     = matches[2];
var req_version = matches[3];

// Solo acepto GET
if (req_method != 'GET') {
    socket.write(req_version + ' 405 Method Not Allowed\n');
    socket.write('Allow: GET\n\n');
    socket.end();
    return;
}

var filename = url.parse(req_url).pathname;
if (filename == '/') filename = '/index.html';
filename = path.join("public", filename);
```

Envío respuesta

Se ignoran cabeceras.

// PROCESAR LOS DATOS RECIBIDOS (2/2)

```
fs.exists(filename, function(exists) {

    if (!exists) {

        socket.write(req_version + ' 404 Not Found\n\n');
        socket.end();

    } else {

        var mt = mimeTypes[path.extname(filename).split(".")[1]];

        socket.write(req_version + ' 200 OK\n');
        socket.write('Content-Type: '+mt+'; charset=UTF-8\n\n');

        var rs = fs.createReadStream(filename);
        rs.pipe(socket);

        rs.on('error',function(error) {
           socket.close();
        });
    }
});
```

Envío respuesta

Envío respuesta

Servicios Web usando módulo **HTTP**

Usando el Módulo **HTTP**

- El módulo **HTTP** nos ayuda con algunas tareas del protocolo HTTP.
- Recibe el flujo de datos y lo separa en cabeceras y cuerpo
  - Pero no parsea su contenido, sólo lo separa.
- Crea objetos que representan las peticiones y las respuestas HTTP.
  - Disponemos de métodos para manejar las cabeceras, código de respuesta, los flujos, codificación, etc...
- Tenemos nuevos eventos:
  - Ejemplo: **request** se dispara cada vez que llega una nueva petición, y nos proporciona los objetos **request** y **response**.

- Cada vez que llegue una petición HTTP hay que:
  - Analizar el método HTTP, el URL y las cabeceras de la petición que nos proporcionan en un objeto **IncommingMessage**.
  - Leer los datos del cuerpo.
  - Responder utilizando el objeto **ServerResponse** que nos proporcionan.
    - Poner un status code.
    - Ajustar cabeceras.
    - Enviar datos.

Ejemplo: Hola Mundo

```
var http = require('http');

var body = '<html><head><title>Hola Mundo</title></head>'+
           '<body>Hola Mundo</body></html>';

http.createServer(function(request, response) {

    console.log('Nueva peticion.');

    if (request.method != 'GET') {
        response.writeHead(405, {'Allow': 'GET'});
        response.end();
        return;
    }

    response.writeHead(200, {
       'Content-Type': 'text/html',
       'Content-Length': body.length
    });

    response.end(body);

}).listen(3000);
```

Función invocada para cada petición recibida. Me pasan objetos **ServerRequest** y **ServerResponse**.

Sólo acepto GET

Código de respuesta y cabeceras.

Envío datos y termino.

Ejemplo: Servidor Ficheros Estáticos

```
var http = require('http');
var path = require("path");
var fs = require("fs");
var url = require('url');

http.createServer(function(request, response) {
    if (request.method != 'GET') {
        response.writeHead(405, {'Allow': 'GET'});
        response.end();
        return;
    }

    var filename = url.parse(request.url).pathname;
    if (filename == '/') filename = '/index.html';
    filename = path.join("public", filename);

    var rs = fs.createReadStream(filename);

    rs.pipe(response);

    rs.on('error', function(error) {
        response.end('Error leyendo '+request.url);
    });
}).listen(3000);
```

Sólo acepto GET

Ruta raíz

Intercambio asíncrono entre un readStream (rs) y un writeStream (response)

Express.js

¿Qué es **express**?

- Documentación:

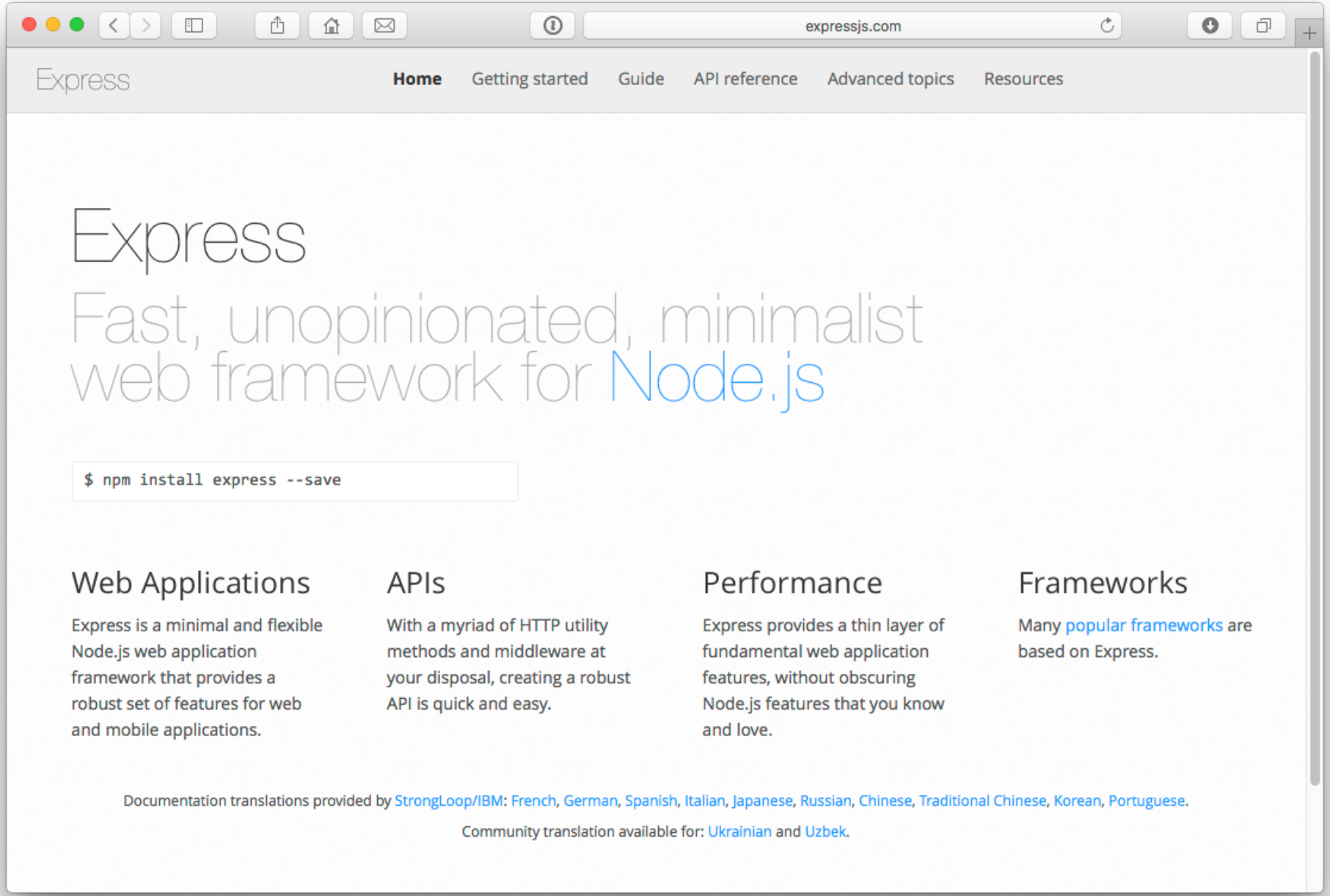
  `http://expressjs.com/guide.html`

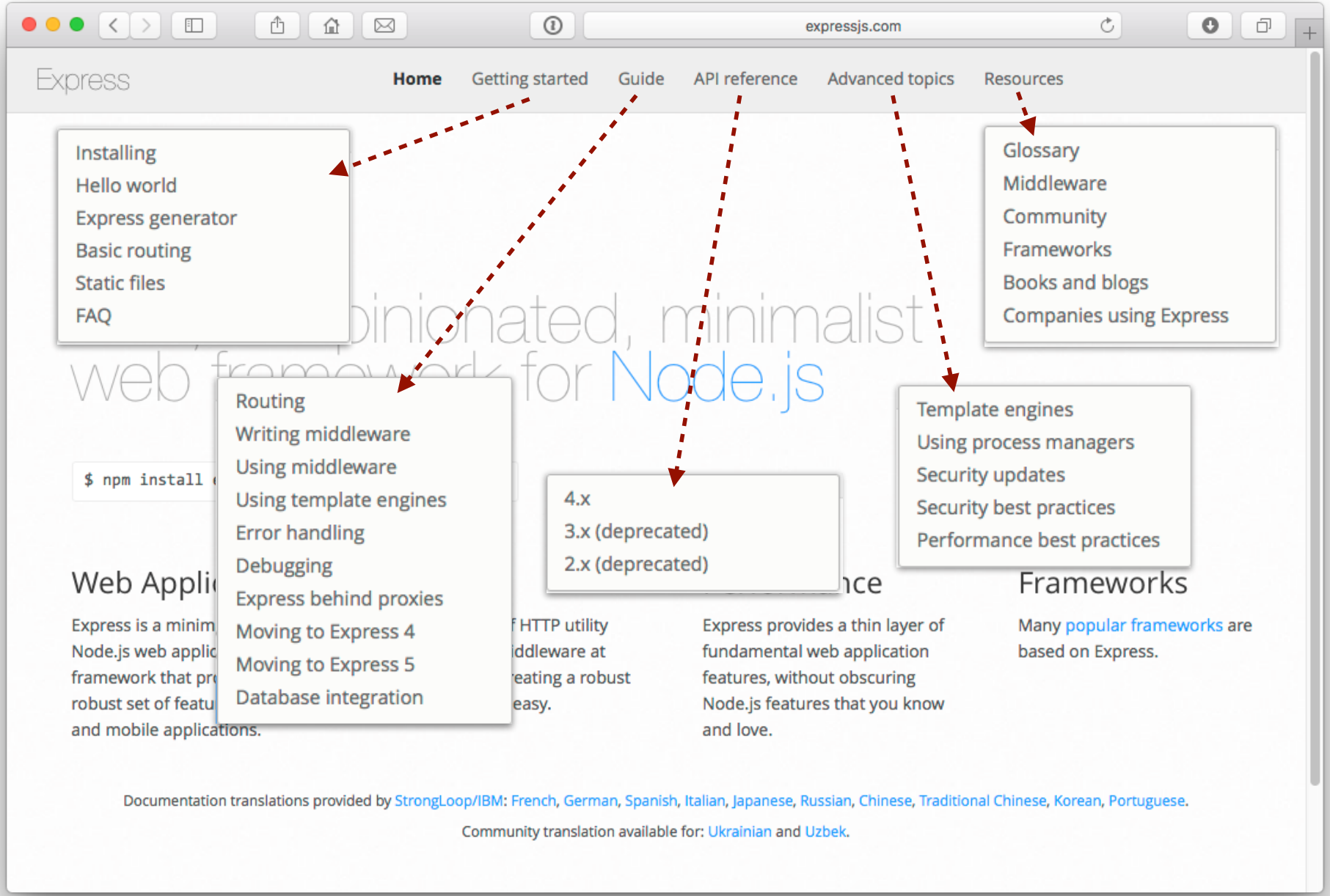
  `https://github.com/visionmedia/express`

- Es un framework para el desarrollo de aplicaciones Web con Node.js.

- Características:

  Extiende connect (uso de middlewares), manejo de **rutas**, soporte de múltiples motores de **plantillas** para la generación de vistas, negociación del **formato** de los contenidos, configurable para entornos de producción/desarrollo/pruebas, módulos adicionales para crear rápidamente una versión inicial de la aplicación, etc.





Hay que instalar módulo express:

$ npm install express