ETSIT UPM

dit UPM

Desarrollo de Servicios WEB

Express

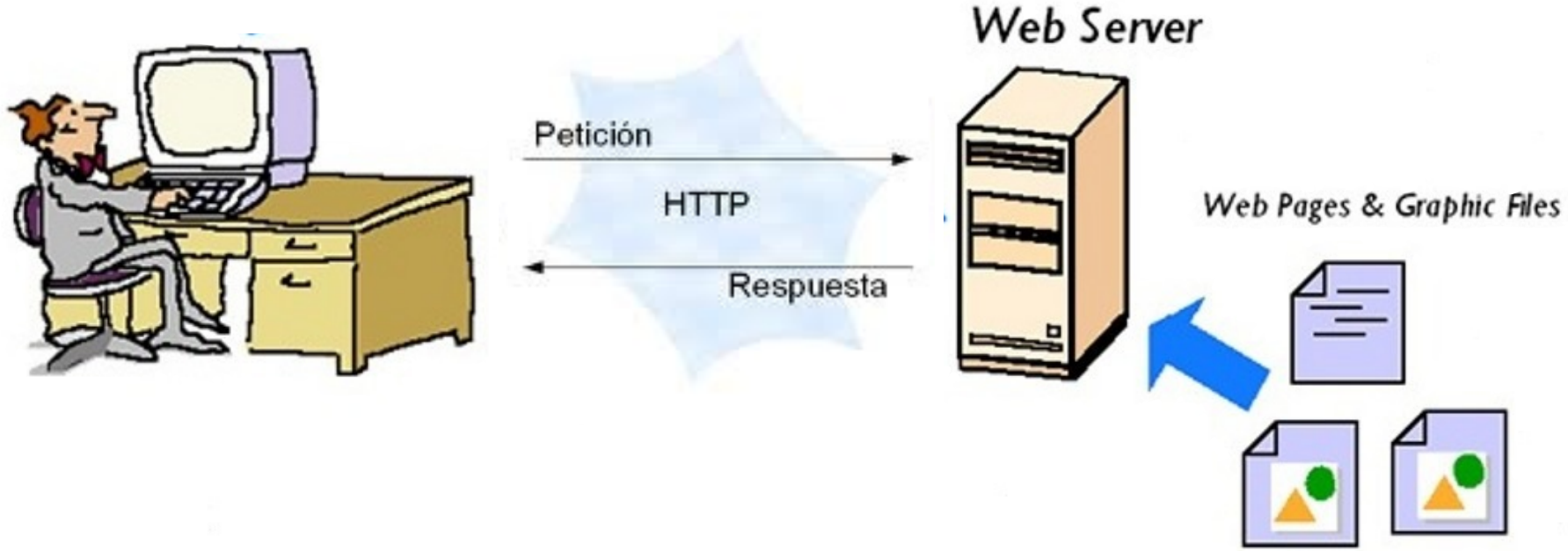
CORE 2018-2019
Santiago Pavón

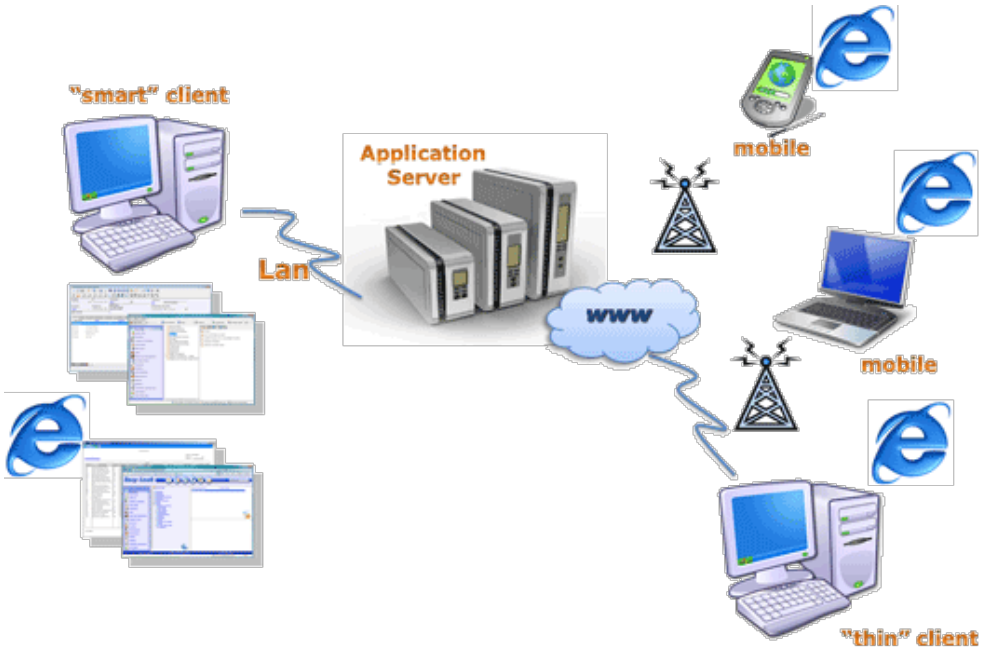
Versión: 2019-03-22

Plataforma Web

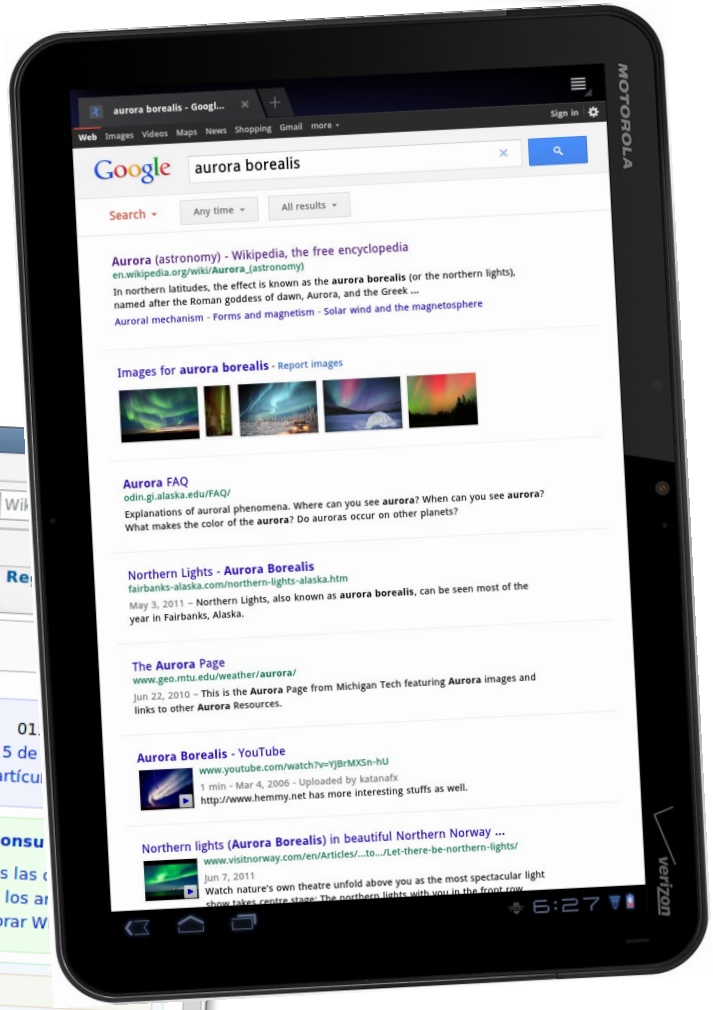
La Web

- Los clientes y servidores son programas hablando HTTP:




"smart" client
Application Server
Lan
mobile
www
mobile
"thin" client


Google
Deutschland

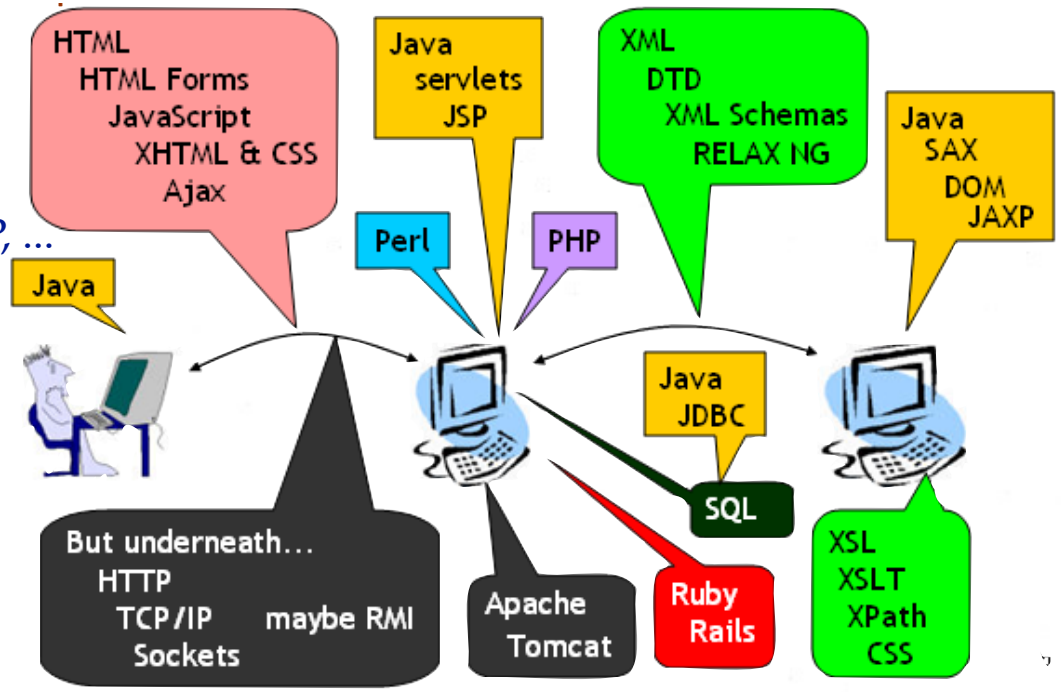

MOTOROLA
aurora borealis - Googl...
Google
aurora borealis
Search
Any time
All results
Aurora (astronomy) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Aurora_(astronomy)
In northern latitudes, the effect is known as the aurora borealis (or the northern lights), named after the Roman goddess of dawn, Aurora, and the Greek ...
Auroral mechanism - Forms and magnetism - Solar wind and the magnetosphere
Images for aurora borealis - Report images
Aurora FAQ
odin.gi.alaska.edu/FAQ/
Explanations of auroral phenomena. Where can you see aurora? When can you see aurora? What makes the color of the aurora? Do auroras occur on other planets?
Northern Lights - Aurora Borealis
fairbanks-alaska.com/northern-lights-alaska.htm
May 3, 2011 – Northern Lights, also known as aurora borealis, can be seen most of the year in Fairbanks, Alaska.
The Aurora Page
www.geo.mtu.edu/weather/aurora/
Jun 22, 2010 – This is the Aurora Page from Michigan Tech featuring Aurora images and links to other Aurora Resources.
Aurora Borealis - YouTube
www.youtube.com/watch?v=YJBrMXSn-hU
1 min - Mar 4, 2006 - Uploaded by katanafx
http://www.hemmy.net has more interesting stuffs as well.
Northern lights (Aurora Borealis) in beautiful Northern Norway ...
www.visitnorway.com/en/Articles/...to.../Let-there-be-northern-lights/
Jun 7, 2011
Watch nature's own theatre unfold above you as the most spectacular light
6:27
verizon


Wikipedia, la enciclopedia libre - Mozilla Firefox
Archivo Editar Ver Historial Marcadores Herramientas Ayuda
http://es.wikipedia.org/wiki/Wikipedia:Portada
Más visitados
Comenzar a usar Fi...
Últimas noticias
WIKIPEDIA
La enciclopedia libre
navegación
Portada
Portal de la comunidad
Actualidad
Cambios recientes
Página aleatoria
Ayuda
Donaciones
buscar
Ir
Buscar
herramientas
portada
discusión
ver código fuente
historial
Mantén Wikipedia: un proyecto sin ánimo de lucro.
Dona ahora »
Bienvenidos a Wikipedia,
la enciclopedia de contenido libre que todos pueden editar.
Libro de visitas · Acceso WAP · Contacto · Donaciones
431.432 artícu
Participación y comunidad
¿Cómo colaborar? · Bienvenida · Primeros pasos
Tutorial · Contenidos de Ayuda
Café · FAQ · Los 5 pilares
Búsquedas y consu
Índice de categorías · Todas las
Índice alfabético · Todos los ar
Portales temáticos · Explorar W
Artículo destacado
La nube de Oort (también llamada nube de Öpik-Oort) es una nube esférica de
Frase del día
« Los manantiales de la abundancia no están en

Desarrollar un Servidor

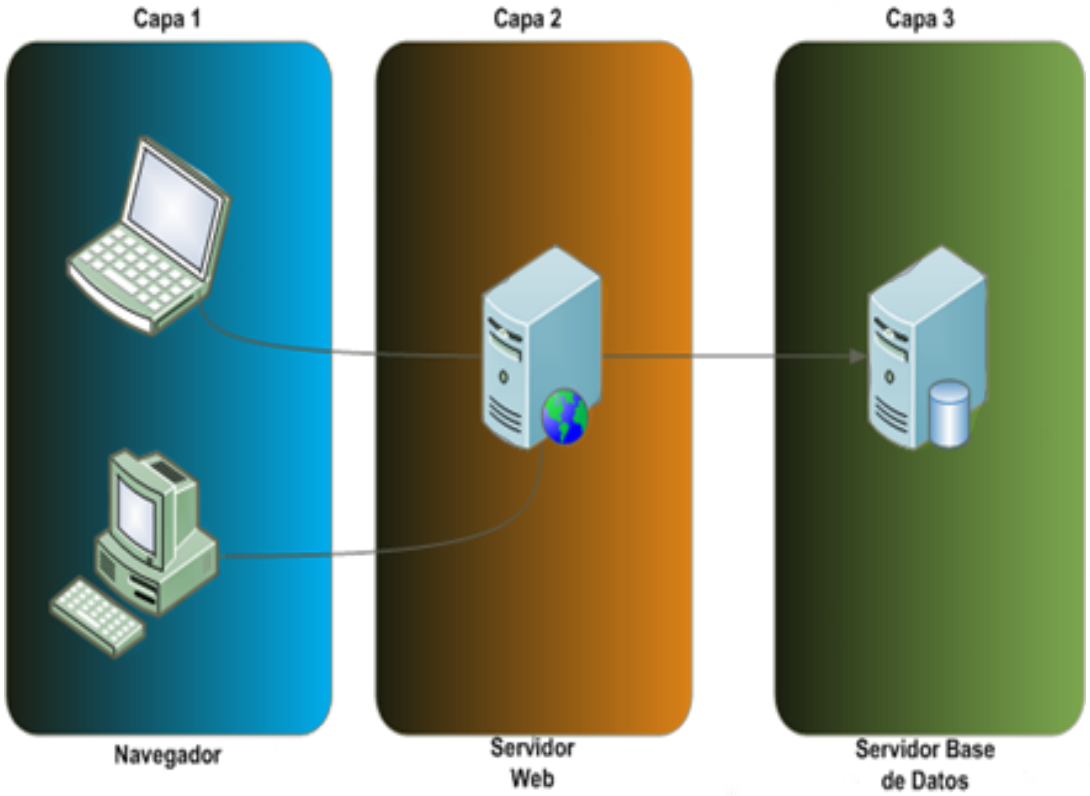
Opciones

- 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**, ...

Servidor 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:

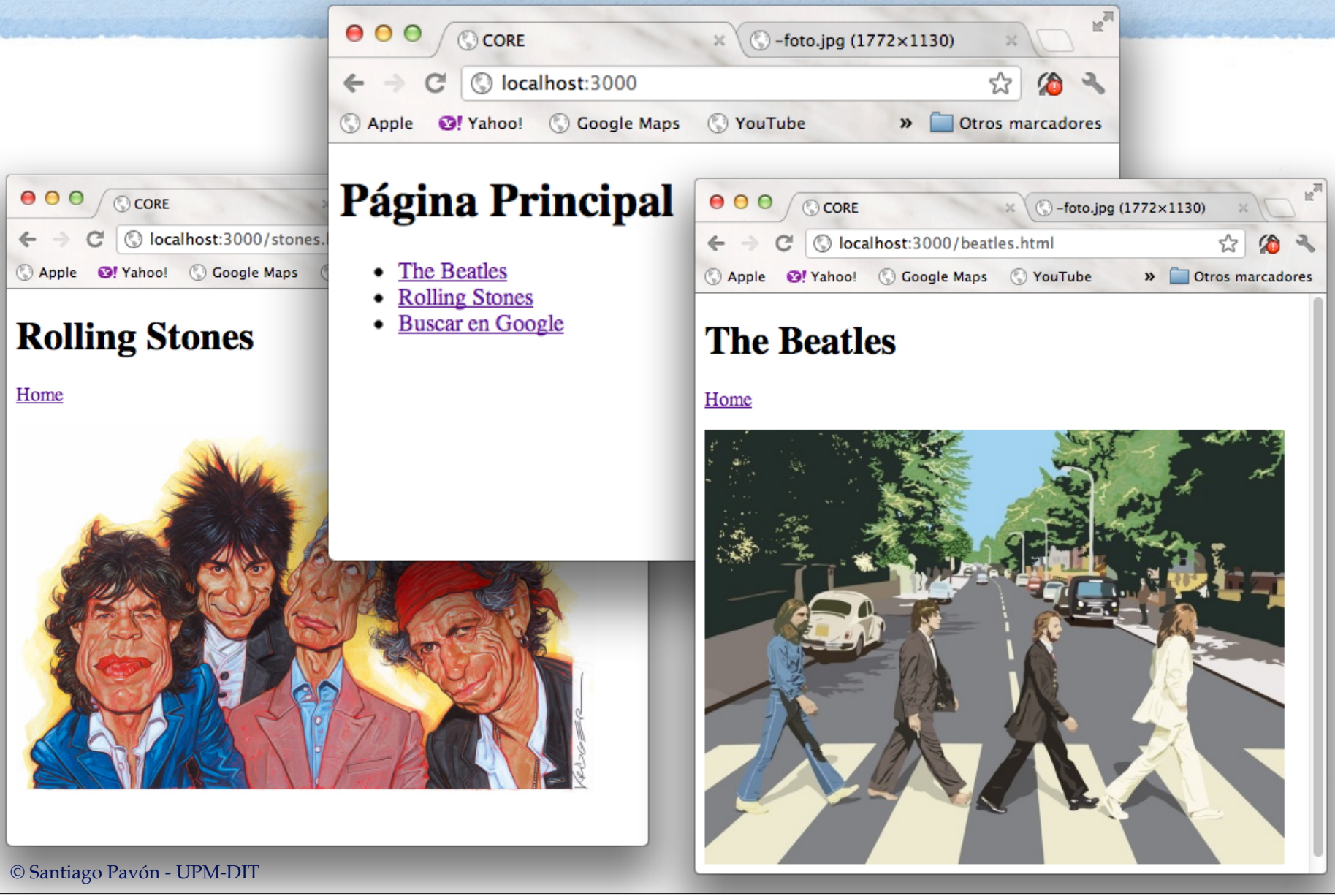
`http://localhost:3000`

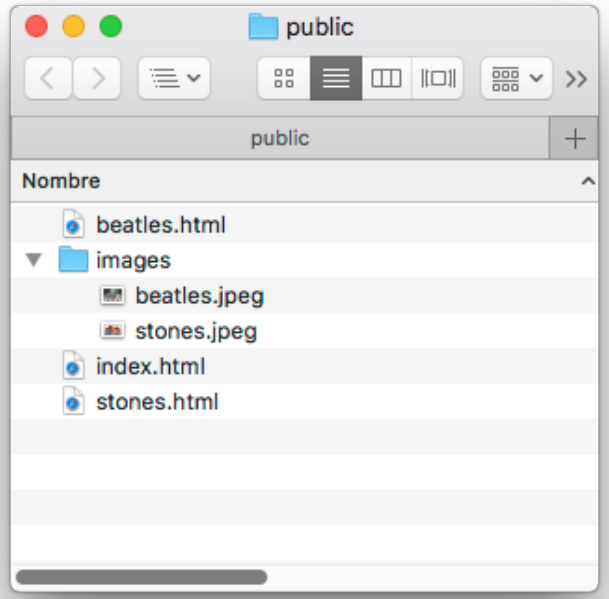
  - Inspeccionar los mensajes intercambiados con las herramientas de desarrollo web del navegador.
- Desde un terminal conectarse con:

```
$ telnet localhost 3000
```

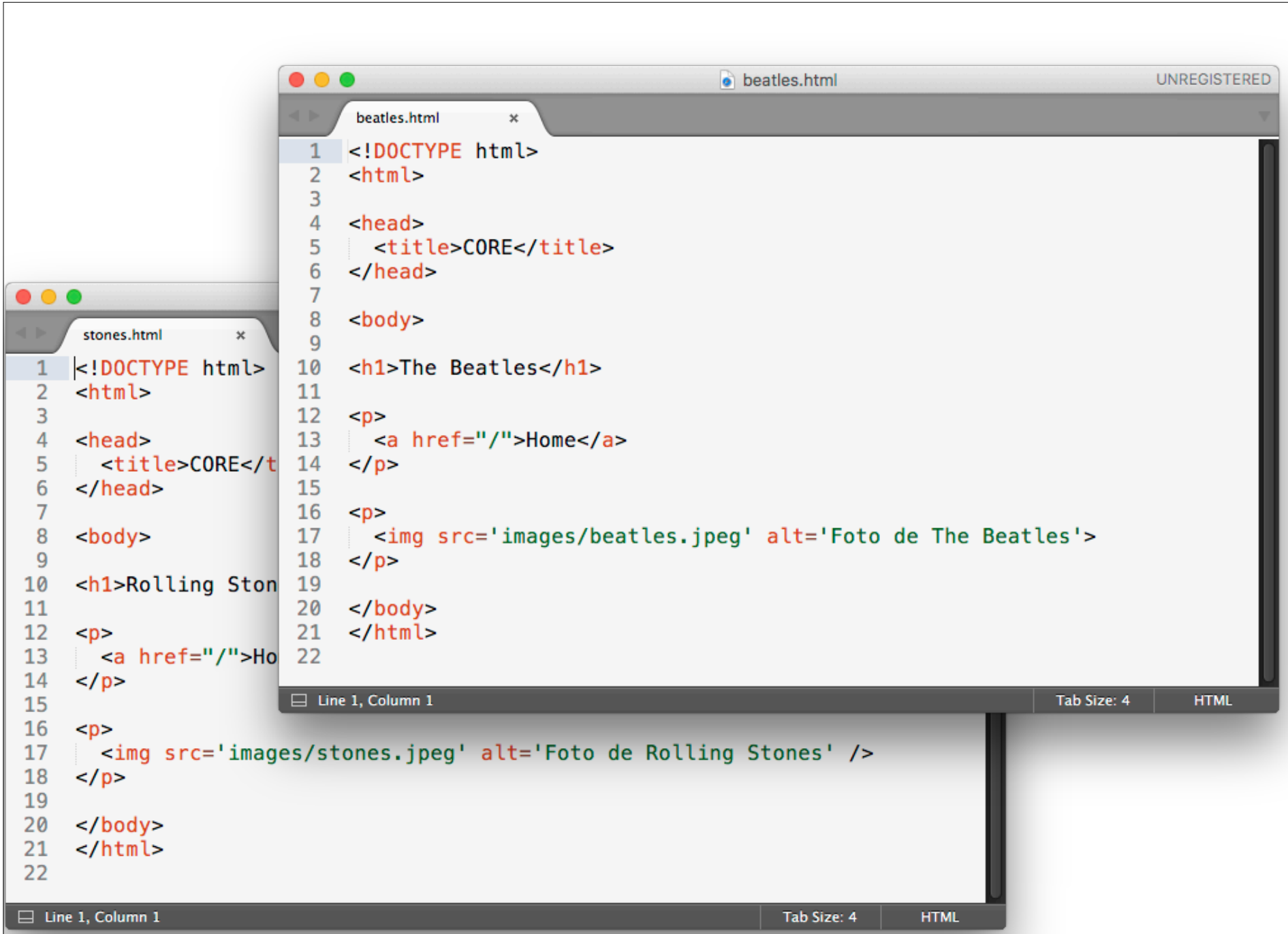
  - Enviar cualquier texto.

Ejemplo: Servidor Páginas Estáticas





beatles.jpeg



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

Servidor 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)

Servidor Web usando Express.js

¿Qué es **express**?

express

- Documentación:

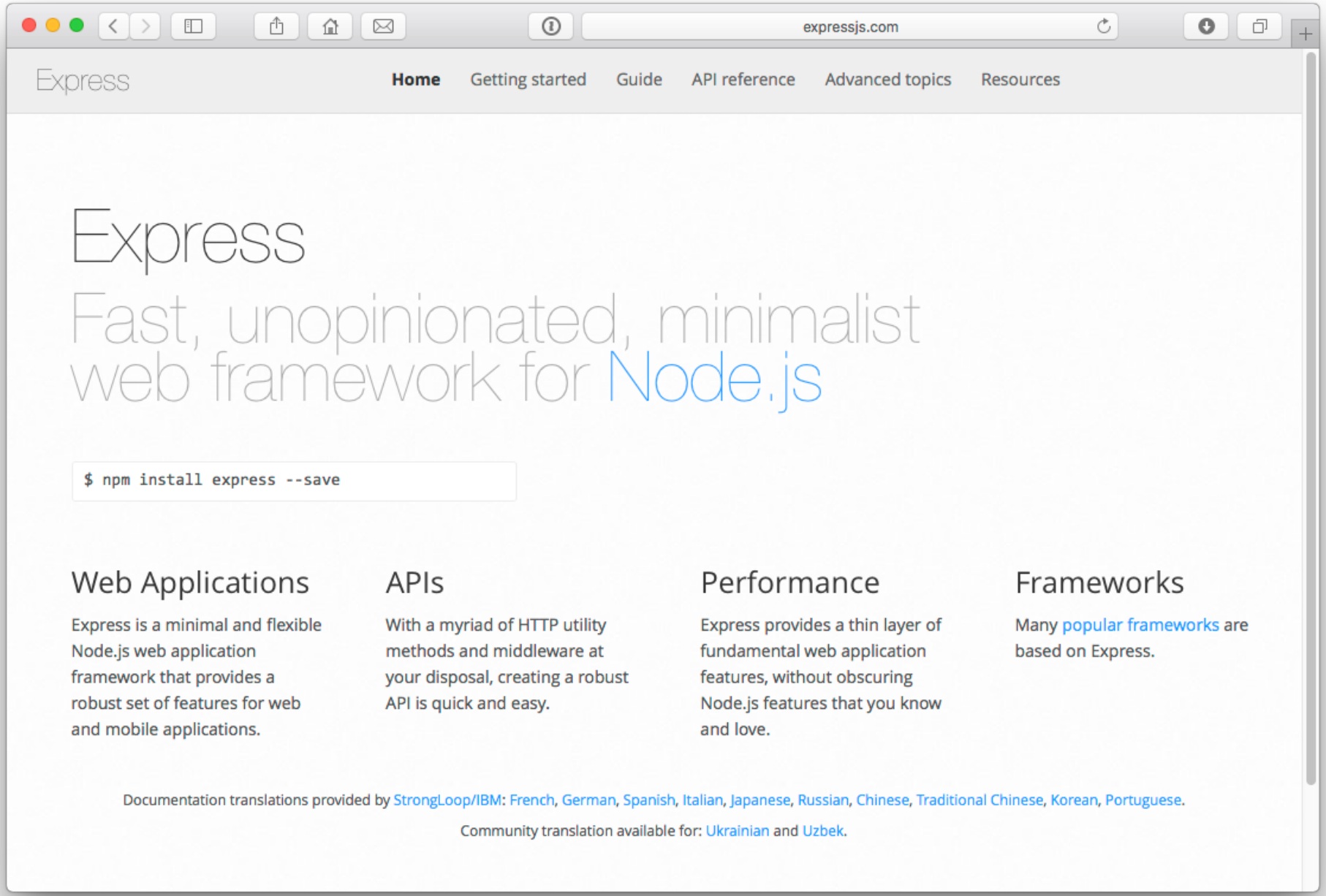
  `http://expressjs.com/guide.html`

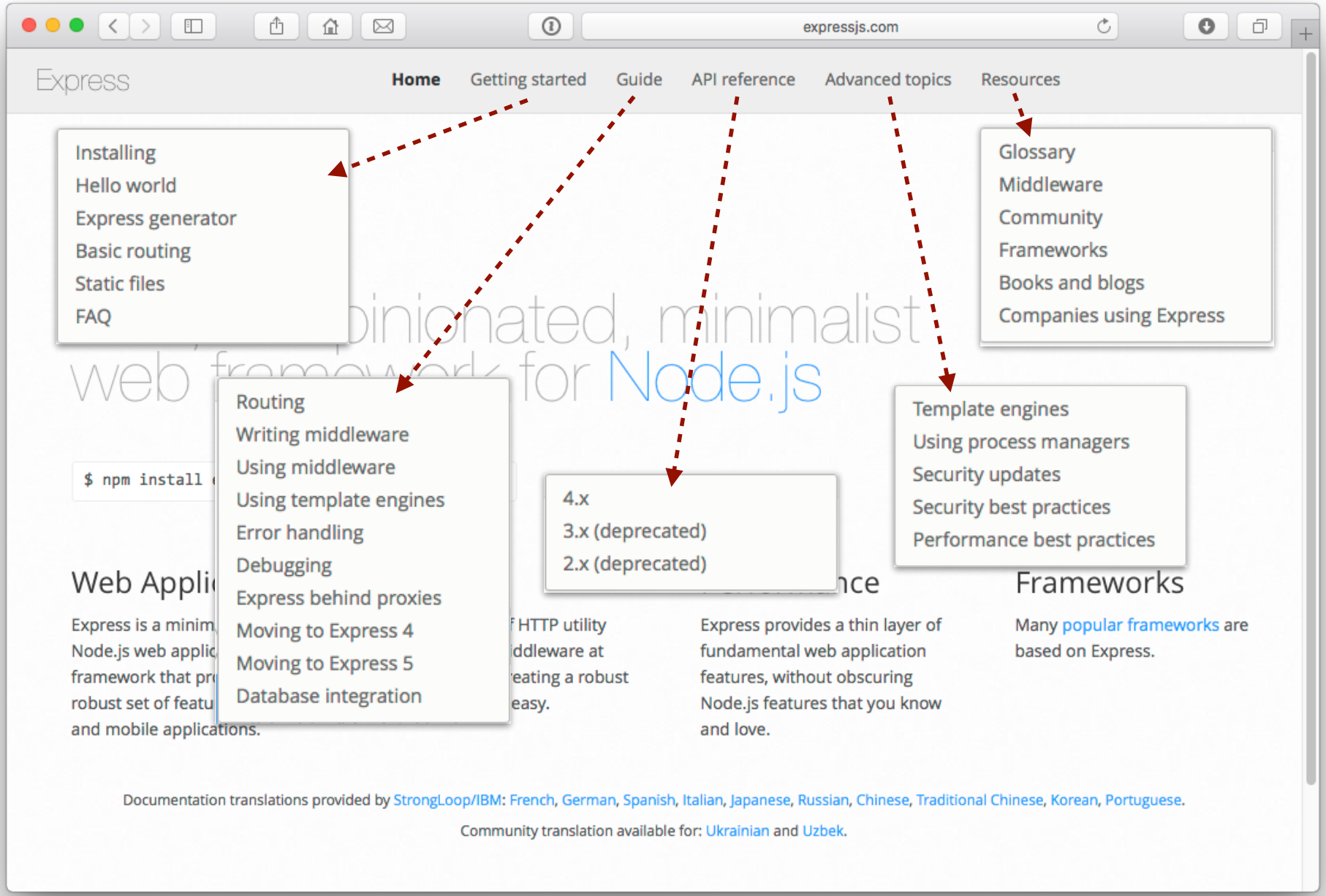
  `https://github.com/visionmedia/express`

- Es un framework para el desarrollo de aplicaciones Web con Node.js.
  - Organizar el código del servidor.

- Características:

  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

Middlewares

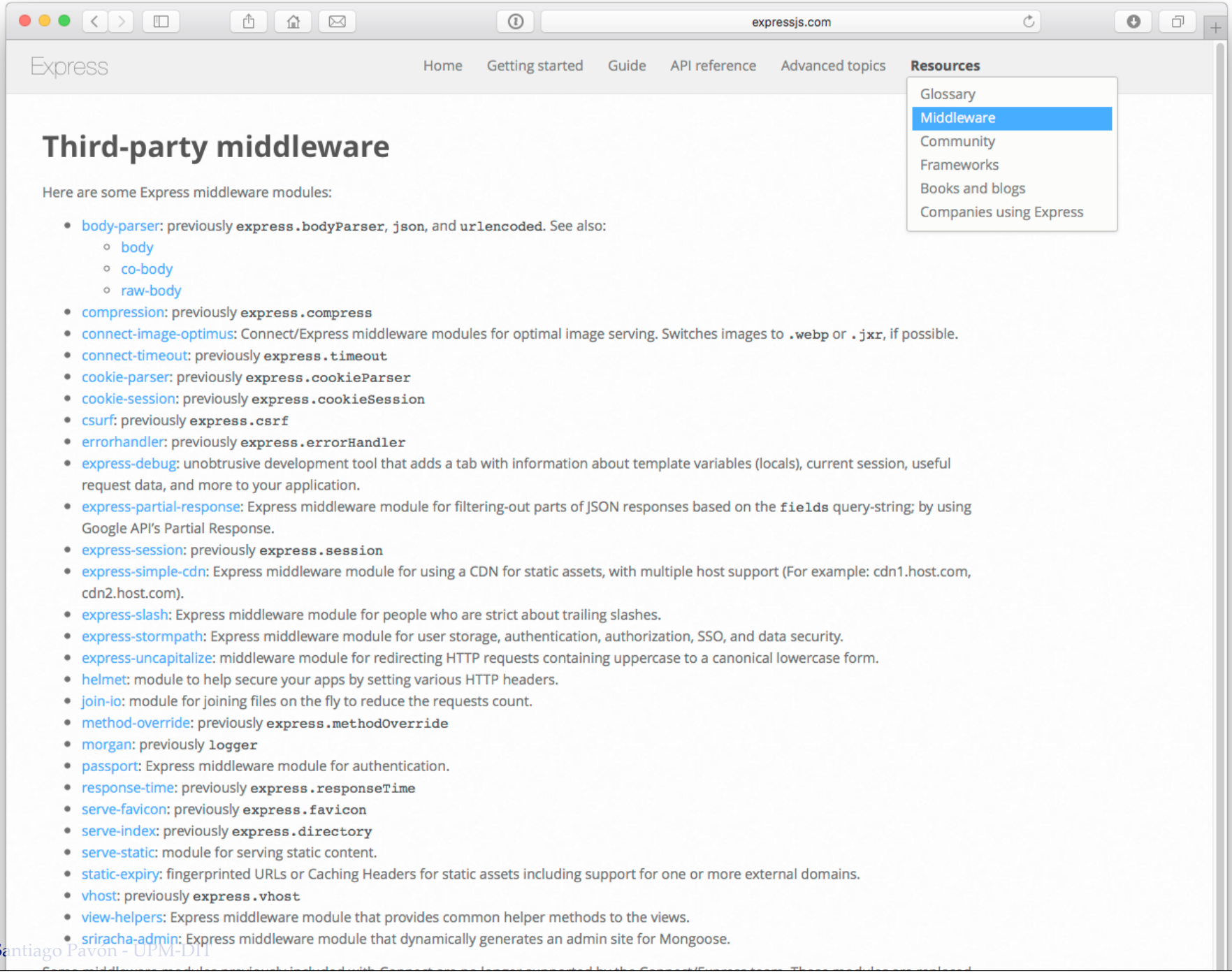
- Las funcionalidades del servidor se programan en **middlewares**.
- Los middlewares son funciones con la misma interface:

```
function(request,response,next)
```

- Cada middleware se encarga de una tarea.
- Los middlewares se registran para definir el orden en que deben ejecutarse.
  - Un middleware **pasa el control** al siguiente middleware llamando a la función **next**() que le han pasado como parámetro.
  - Un middleware **termina** la cadena de llamadas si no llama a **next**().
    - En este caso, debe proporcionar la respuesta HTTP a enviar.
    - No se puede llamar a next y enviar una respuesta HTTP. Una cosa o la otra.
  - Los parámetros **request** y **response** son referencias a los objetos que representan la petición y la respuesta HTTP en curso.
    - Los middlewares suelen añadir a estos objetos **nuevos atributos** como efecto lateral.
      - donde guardan resultados de su ejecución.

Manejo de los Errores

- La ejecución secuencial de los middlewares se interrumpe si:
  - Alguna sentencia lanza una excepción.
  - Nosotros lanzamos una excepción programáticamente:
    ```
    throw new Error('Houston, tenemos un problema');
    ```
  - Llamamos a next pasándole un parámetro:
    ```
    next(new Error('Houston, tenemos otro problema'));
    ```
- En estos casos la ejecución secuencial de middlewares salta al siguiente middleware de manejo de errores.
  - Son los que tienen la signatura: `function(error, req, res, next);`
- El middleware de manejo de errores:
  - Seguramente llamará a res.send, res.end, ... para enviar una respuesta al cliente.
  - O si arregla el problema llamará a next() para seguir con el siguiente middleware.
  - O puede que lance otra excepción o llame a next(error).
    - Se saltará al siguiente middleware de manejo de errores.



Uso

- Crear una **Aplicación Express**:
  - Se crea con la función express()

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

var app = express();
```

- **Instalar Middlewares:**
  - Se instalan con:

```
app.use(<middleware1>);

app.use(<middleware2>);

app.use(<middleware3>, <middleware4>);

app.use(<path>, <middleware3>);
```

  - Se ejecutan en el mismo orden en que han sido instalados.
  - Se ejecutan cada vez que llega una petición HTTP.
  - Documentación: http://expressjs.com/4x/api.html#app.use

35

Rutas

- Instalar middlewares asociándolos con:
  - un path y un método HTTP: GET, PUT, POST, DELETE.
    - MW solo se ejecuta si coinciden el método y el path con la solicitud HTTP.
- Se instalan usando:

```
var router = express.Router();

router.get(path, MW);
router.post(path, MW);
router.put(path, MW);
router.delete(path, MW);
router.all(path, MW);

router.get(path, MW1, MW2, ...); etc...
```

- También así:

```
app.get(path, MW); etc...
```

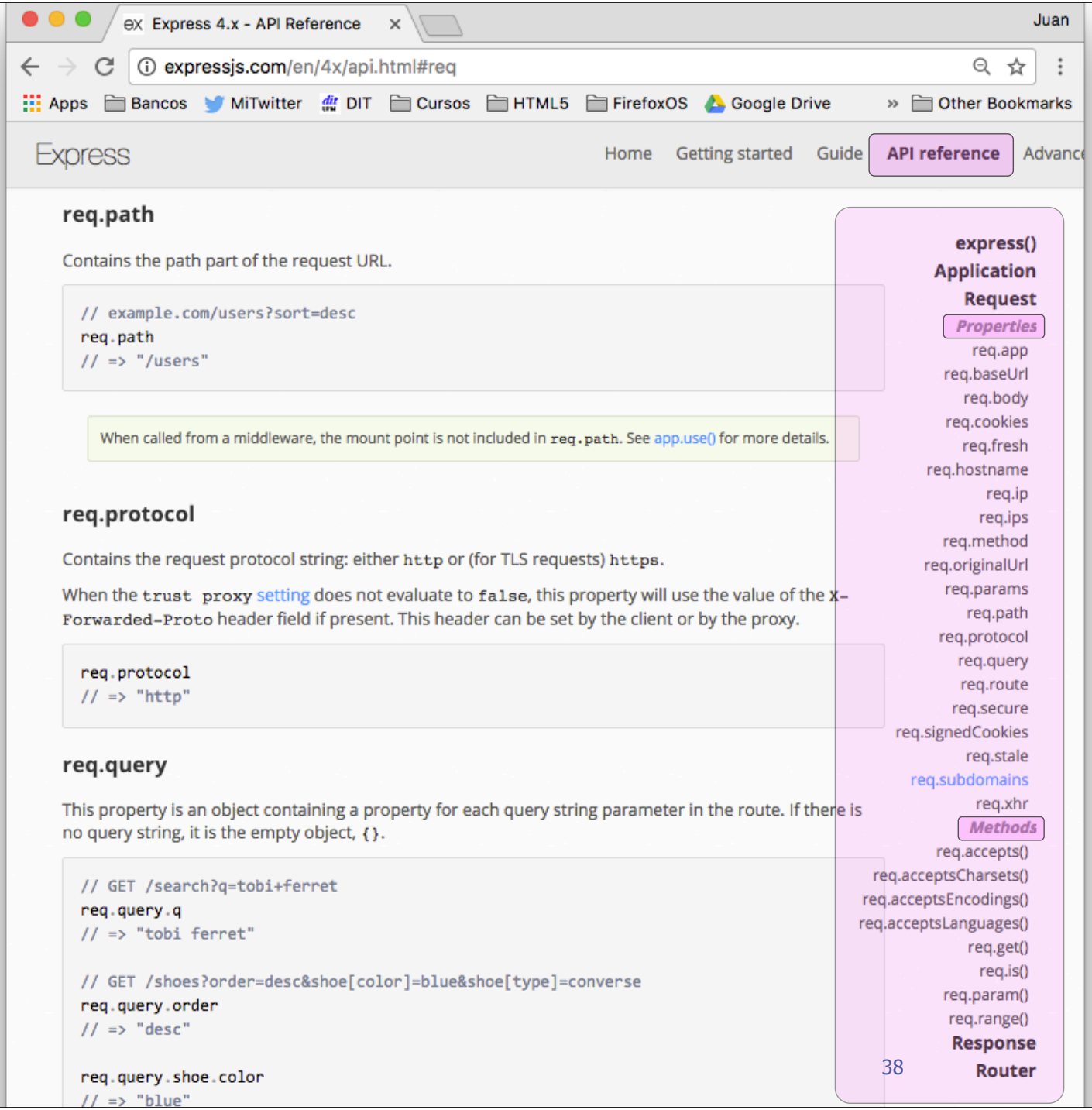

36

Formato de las Rutas:

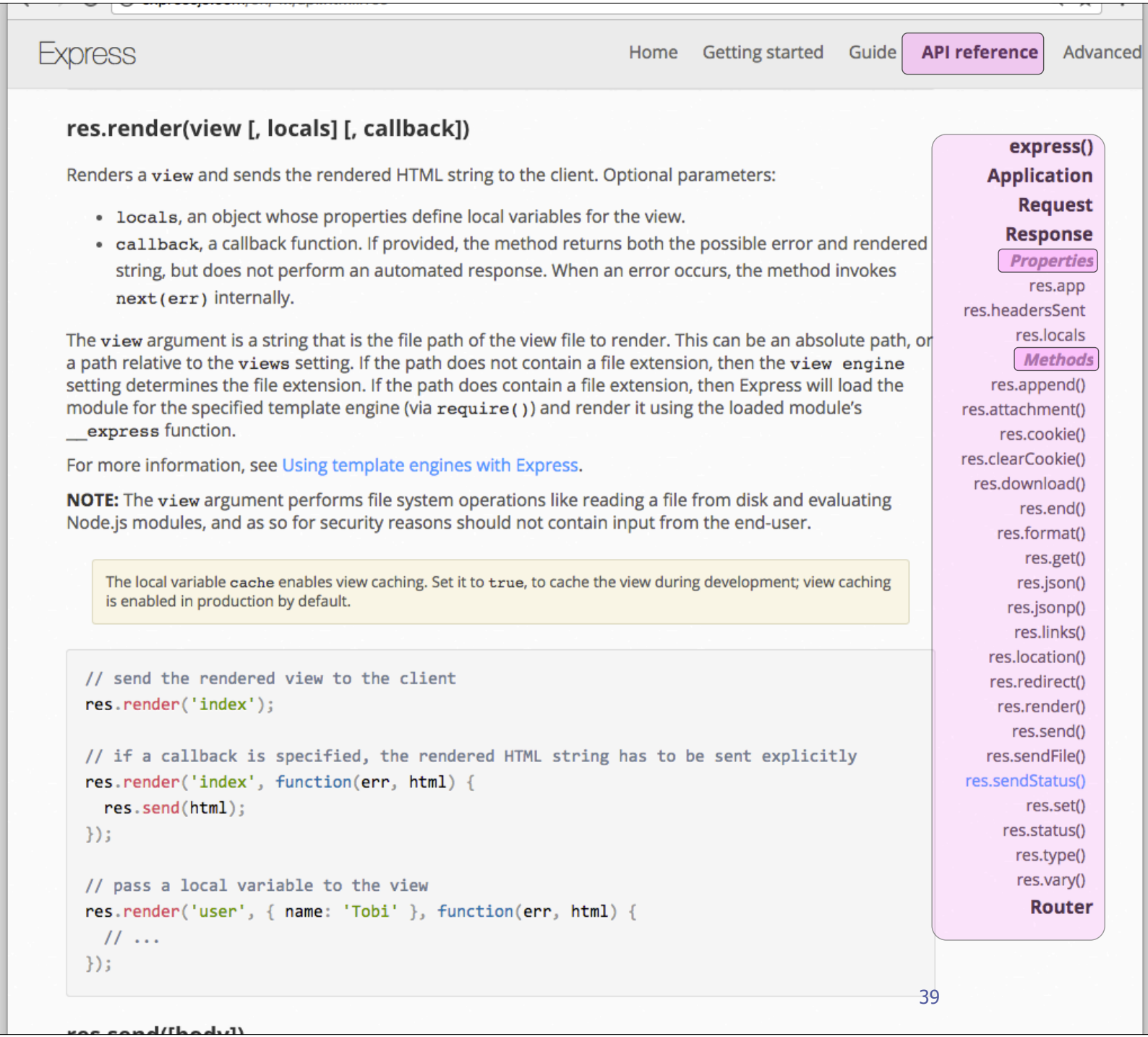
- Se dividen en segmentos separados por **/**
  - Por ejemplo, **/users/25/tasks** tiene 3 segmentos: users, 25 y tasks
- Los segmentos pueden ser:
  - **Literales**: texto con caracteres permitidos en un URL
    - **/client**, **/game**, **/quiz/hola**, **/que**, **/23**, …
  - **Parámetros:** variables que aceptan cualquier texto
    - **/:x**, **/:x1**, **/:x_1**, **/:model**, **……**
    - Para acceder al valor del parámetro: **req.params.x**, **req.params.x1**, **req.params.x_1**, ….
    - El nombre de un parámetro puede tener letras ASCII, dígitos decimales y _
    - Un segmento puede incluir varios parámetros separados por **-** o **.** , por ej. **/:from-:to**, **/:file.:extention**
    - Los **parámetros** pueden **restringirse** con **expresiones regulares**
      - **/:x([0-9]+)** el parámetro **x** solo puede ser un número decimal
      - **/:idioma(es|en|ge|it|fr)** el parámetro **idioma** solo puede contener los strings: es, en, ge, it o fr
      - * - casa con cualquier string, por ej. **/pep*** puede casa con: pep, pepppp, pepito, …
      - ? - 0 o 1 vez, por ej. **/cars?** (casa con car o cars), **/c(ars)?** (casa con **c** o **cars**), ..
      - + - 1 o más veces, por ej. **/cars+** (cars, carss, ..), **/c(ars)+** (**cars**, **carsars**, …), ..

37

Parámetro req



Parámetros res



Express

Home | Getting started | Guide | **API reference** | Advanced

res.render(view [, locals] [, callback])

Renders a `view` and sends the rendered HTML string to the client. Optional parameters:

- `locals`, an object whose properties define local variables for the view.
- `callback`, a callback function. If provided, the method returns both the possible error and rendered string, but does not perform an automated response. When an error occurs, the method invokes `next(err)` internally.

The `view` argument is a string that is the file path of the view file to render. This can be an absolute path, or a path relative to the `views` setting. If the path does not contain a file extension, then the `view engine` setting determines the file extension. If the path does contain a file extension, then Express will load the module for the specified template engine (via `require()`) and render it using the loaded module's `__express` function.

For more information, see Using template engines with Express.

NOTE: The `view` argument performs file system operations like reading a file from disk and evaluating Node.js modules, and as so for security reasons should not contain input from the end-user.

The local variable `cache` enables view caching. Set it to `true`, to cache the view during development; view caching is enabled in production by default.

```
// send the rendered view to the client
res.render('index');

// if a callback is specified, the rendered HTML string has to be sent explicitly
res.render('index', function(err, html) {
  res.send(html);
});

// pass a local variable to the view
res.render('user', { name: 'Tobi' }, function(err, html) {
  // ...
});
```

express()
Application
Request
Response
Properties
res.app
res.headersSent
res.locals
Methods
res.append()
res.attachment()
res.cookie()
res.clearCookie()
res.download()
res.end()
res.format()
res.get()
res.json()
res.jsonp()
res.links()
res.location()
res.redirect()
res.render()
res.send()
res.sendFile()
res.sendStatus()
res.set()
res.status()
res.type()
res.vary()
Router

39

res.send([body])

Enviar la respuesta al cliente

- Métodos de response:
  - **res.send(**string**)**
    - Envía una respuesta HTTP al cliente usando el string pasado como argumento como el body de la respuesta.
      - res.send(**'<html><body><h1> Título </h1></body></html>'**)
  - **res.redirect(**[statuscode], path**)**
    - Le indica al cliente que haga una redirección a la ruta indicada.
      - res.redirect(**'users/22'**)
      - res.redirect(**'http://sitio.com'**)
  - **res.render(**vista**,** opciones**)**
    - Crea una página html con la vista (plantilla) indicada, y la envía al cliente.
      - res.render(**'users/show', {user, title}**)
  - **res.sendStatus(**statuscode**)**
  - **res.sendFile(**path, opciones, cb**)**

Motor de Vistas EJS

- **EJS** = Javascript embebido.
- Los ficheros de vistas EJS contienen:
  - texto HTML.
  - código javascript entre las marcas `<%` y `%>`.
  - expresiones javascript entre las marcas `<%=` y `%>`.
    - El valor de las expresiones se incorpora al texto HTML.
    - Previamente se escapan los caracteres conflictivos para evitar inyección de código:
      - `<` se sustituye por `<`
      - `>` se sustituye por `>`
      - `&` se sustituye por `&`
      - ...
  - expresiones javascript entre las marcas `<%-` y `%>`.
    - El valor de las expresiones se incorpora al texto HTML.
    - No se escapan los caracteres conflictivos.
  - inclusión de ficheros con `<% include` *path_del_fichero_a_incluir* `%>`.
- Documentación: `https://github.com/visionmedia/ejs`

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

views/index.ejs

express-generator
Util para Crear Esqueleto de una Aplicación

- Lo primero es instalar el módulo express-generator:

```
$ npm install express-generator
```

  - *Los paquetes se instalan en el subdirectorio ~/node_modules.*
  - *Para instalarlos a nivel de sistema (global) pasar la opcion -g.*
    - *Se instalarán en /usr/local/lib/node_modules.*

- Crear Esqueleto de una Aplicación:

```
$ express --ejs demo // crea ficheros iniciales (Instalado globalmente)
$ ./node_modules/.bin/express --ejs demo // (Instalado localmente)
$ cd demo
$ npm install  // instala las dependencias declaradas en package.json
```

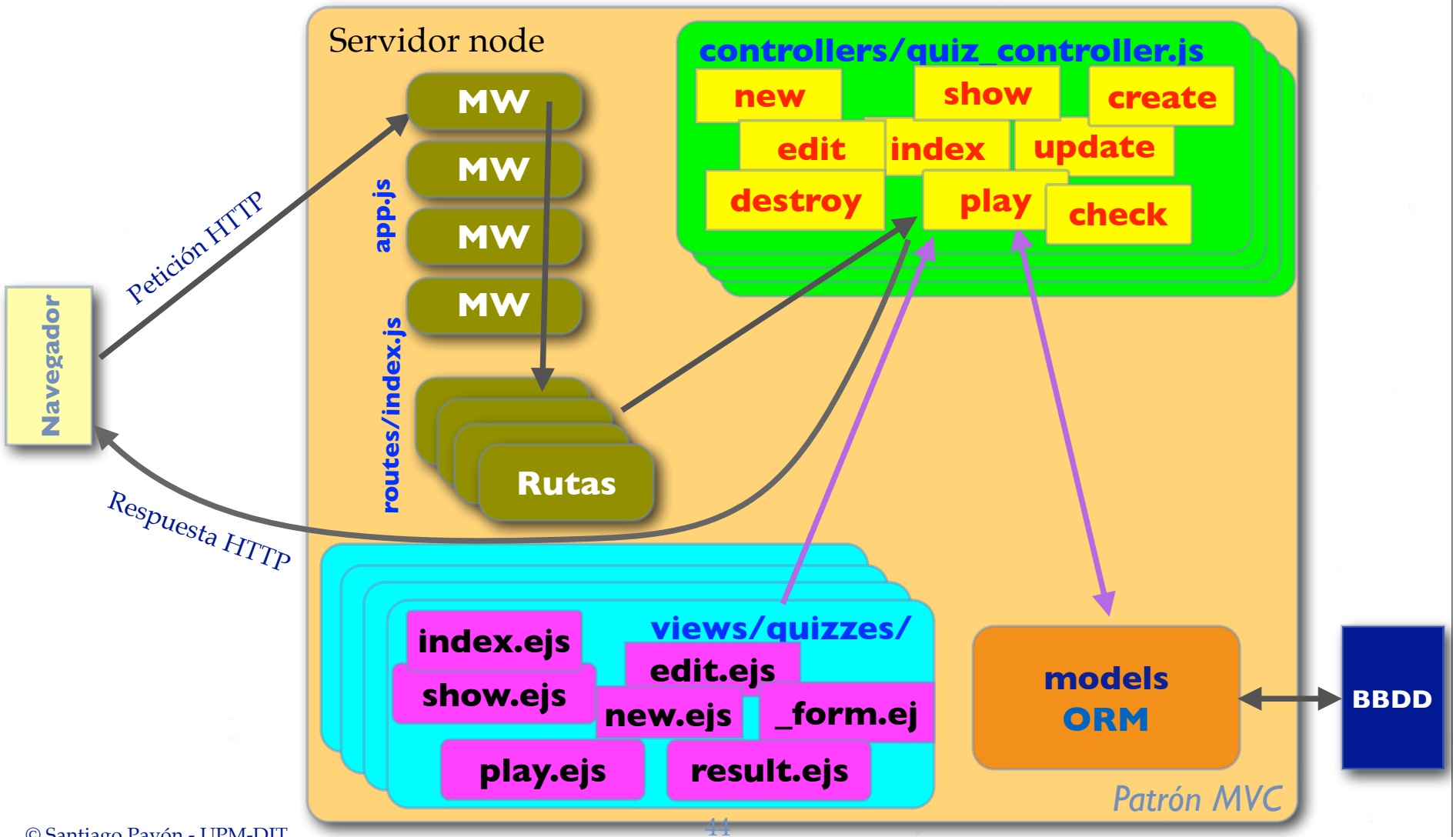
- Ejecutamos la aplicación:

```
$ node bin/www
```

- Y nos conectamos con un navegador a http://localhost:3000.

- También integrado en IDEs (ej: WebStorm)

Arquitectura



Demo: Ejemplo

- **ex1.js** es el básico hola mundo
- **ex2.js** una segunda ruta con un parámetro, coger el parámetro y mostrarlo
- **ex3.js** la segunda ruta solo puede ser numérica, y una tercera ruta que muestra todas
- **ex4.js** middleware checkTime que hace que solo sea accesible la página a ciertas horas
- **ex5.js** mete una ruta * con un default
- **ex6.js** mete un middleware contador de visitas totales

```
const express = require('express');
const path = require('path');

const app = express();

let visitas = 0;
app.use((req, res, next) => {
     console.log(`Visitas = ${++visitas}`);
     console.log(`URL = ${req.url}`);
     next();
});

app.use(express.static(path.join(__dirname, 'public')));

// ****** METER AQUI LAS FUNCIONES DE LA PAGINA SIGUIENTE *******

app.get('/',
      checkTime,
      paginaPrincipalController);
app.get('/ruta/:numerodelaruta([0-9]+)',
      checkTime,
      rutaController);
app.get('/rutas',
      checkTime,
      rutasController);

app.get('*', defaultController);

app.use((err,req,res,next) => {
     res.send(err.toString());
});

app.listen(8000);
```

```
const paginaPrincipalController = function(req, res, next) {
     res.send("<h1>Página Principal</h1>");
}

const rutaController = (req,res,next) => {
     var numerodelaruta = req.params.numerodelaruta;
     res.send("<h1>Número de la ruta: "+numerodelaruta +
               "</h1><img src='/"+numerodelaruta+".jpg'/>");
}

const rutasController = (req, res, next) => {
     let template = "<h1>Rutas disponibles</h1><ul>";
     for (let i = 1; i < 31; i++) {
         template += "<li><a href='/ruta/"+i+"'>Ruta" + i + "</a></li>";
     }
     template +="</ul>";
     res.send(template);
}

const checkTime = (req,res,next) => {
     const hour = 12;
     if ((new Date()).getHours() < hour) {
         res.send(`<h1>Solo accesible a partir de las ${hour}</h1>`);
     } else {
         next();
     }
}

const defaultController = function(req, res, next) {
     res.status(404);
     res.send("<h1>Not found</h1>");
}
```

Ejemplo: Proyecto Quiz

- En el proyecto quiz:

  `https://github.com/CORE-UPM/quiz_2018.git`

- Ver los siguiente ficheros:

  `app.js`

  `controllers/*`

  `routes/index.js`

  `views/*`