



Empresa  
Digitala



Universidad de Deusto  
Deustuko Unibertsitatea

# Full-stack JavaScript: Desarrollo integral de aplicaciones Web con JavaScript

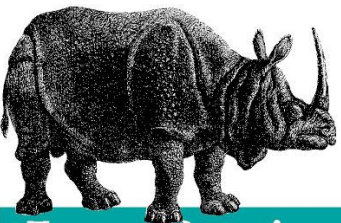
19 Diciembre de 2012, 9:00-14:30

**Dr. Diego López-de-Ipiña González-de-Araza**

[dipina@deusto.es](mailto:dipina@deusto.es)

<http://paginaspersonales.deusto.es/dipina>

<http://www.slideshare.net/dipina>



JavaScript



# Agenda

## 1. Introducción (30')

- RIA, Mobile y Web Apps
- Tecnologías clave
  - HTML5
  - CSS3
  - JavaScript

## 2. JavaScript avanzado (70')

- Repaso de características generales de JavaScript
- jQuery
- JQuery UI y jQuery Mobile

# Agenda

## 3. CouchDB: una BBDD NoSQL basada en JavaScript (30')

- Bases de datos NoSQL
- Instalación y configuración
- Acceso desde JavaScript a CouchDB

## Descanso (20')

## 4. Node.js (180')

- Programación asíncrona orientada a eventos: cambio de paradigma
- Sintaxis de node.js
- Hola Mundo en node.js
- Mi primera aplicación CRUD con Node.js
- Frameworks MVC de desarrollo de aplicaciones web con Node.js
- Despliegue en la nube de aplicaciones JavaScript full-stack



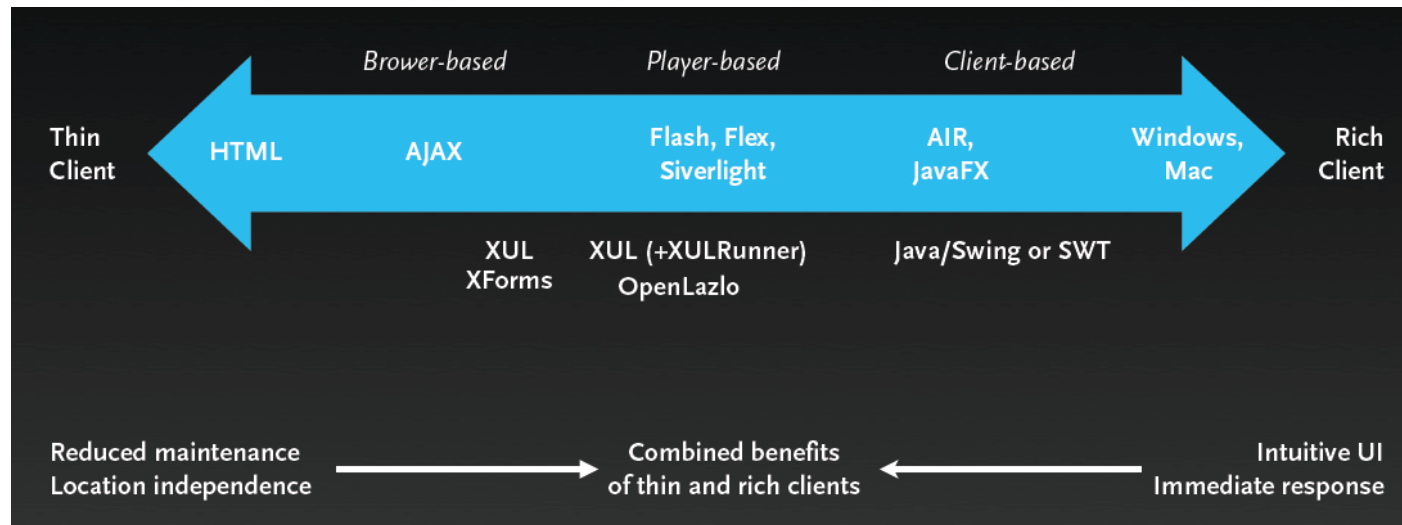
# RIA: Rich Internet Applications

- RIAs son aplicaciones web que tienen características de funcionalidad similar al de las aplicaciones de sobremesa tradicionales.
- Las aplicaciones RIA constituyen una aplicación cliente que mantiene estado y que está desacoplada de la capa de servicios en el back-end
- Las aplicaciones RIA requieren el siguiente entorno de ejecución:
  - Se ejecutan en un navegador web, o no requiere instalación software
  - Se ejecutan en un entorno seguro denominado sandbox
- Beneficios:
  - Las aplicaciones RIA incrementan el retorno en la inversión (ROI) simplificando y mejorando la interacción del usuario
  - Permiten a los usuarios encontrar información más fácilmente
  - Completar tareas rápidamente y de forma precisa
  - Generan ricas visualizaciones de datos para ayudar en la mejora de la toma de decisiones

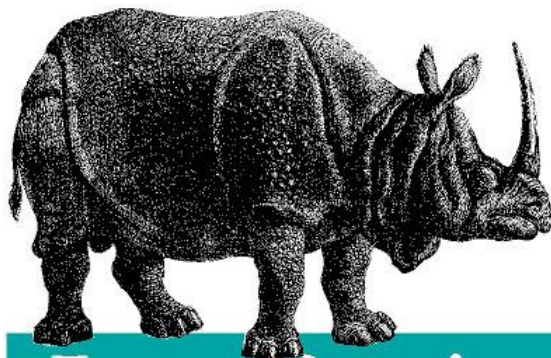


# Taxonomía de aplicaciones RIA

- 3 tipos:
  - **Client-based** – Aplicaciones basadas en un cliente de sobremesa
  - **Player-based** – Aplicaciones que se ejecutan sobre una extensión del navegador
  - **Browser-based** – Aplicaciones web que usan frameworks JavaScript



# Full-Stack JS WebApps = HTML5 + CSS3 + JavaScript + Node.js



JavaScript

HTML



CSS



Enpresa  
Digitala



Universidad de Deusto  
Deustuko Unibertsitatea

# HTML5: The Web as a Platform

- HTML5 es el nuevo estándar HTML
  - Es el resultado de la cooperación entre World Wide Web Consortium (W3C) y el Grupo de Trabajo en Tecnologías de Aplicaciones Web (WHATWG).
- HTML5 es todavía un trabajo en curso, aunque, la mayoría de los navegadores ya soportan muchos de los elementos y APIs de HTML5.
  - Los principios fundacionales de HTML5 son:
    - New features should be based on HTML, CSS, DOM, and JavaScript
    - Reduce the need for external plugins (like Flash)
    - Better error handling
    - More markup to replace scripting
    - HTML5 should be device independent
    - The development process should be visible to the public
- Ningún navegador ofrece un soporte completo
  - Pero todos los navegadores (Safari, Chrome, Firefox, Opera, Internet Explorer) añaden nuevas características HTML5 a sus últimas versiones

# Contribuciones de HTML5



- **Nuevos elementos y atributos**
  - Nuevos elementos para cabeceras, pies, secciones y artículos: `<article>`, `<footer>`, `<header>`, `<nav>`, `<section>`
  - Nuevos elementos de formularios (calendario, fecha, hora, email, URL, búsqueda), atributos, tipos de entradas, validación automática
- **Soporte CSS3 completo**
  - Nuevos selectores y propiedades
  - Animaciones y transformaciones 2D/3D
  - Esquinas redondeadas y efectos de sombra
  - Fuentes descargables
- **Video y Audio, reproducir video y audio es más fácil que nunca**
  - HTML5 `<video>` y HTML5 `<audio>`
- **2D/3D Graphics, dibujar gráficos es más fácil que nunca:**
  - Uso del elemento `<canvas>`
  - Uso de SVG inline
  - Uso de CSS3 2D/3D
- **Soporte para el desarrollo de aplicaciones Web**
  - Almacenamiento local y acceso local a ficheros
  - Bases de datos SQL locales
  - Caché de aplicaciones
  - Hilos JavaScript
  - XMLHttpRequest 2



# El documento HTML5 más básico



- Incluyendo el nuevo simplificado DOCTYPE:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Title of the  
document</title>
```

```
  </head>
```

```
  <body>
```

```
    The content of the document.....
```

```
  </body>
```

```
</html>
```



# Nuevos elementos en HTML5

- HTML5 incluye nuevos elementos para mejorar la estructura, gestión de formularios, dibujado y contenidos multimedia
  - Elementos obsoletos en HTML 4.01 han sido eliminados
- Los siguientes elementos han sido creados para mejorar la estructura:

| Tag          | Description   |
|--------------|---|
| <article>    | Defines an article  |
| <aside>      | Defines content aside from the page content   |
| <bdi>        | Isolates a part of text that might be formatted in a different direction from other text outside it |
| <command>    | Defines a command button that a user can invoke   |
| <details>    | Defines additional details that the user can view or hide   |
| <summary>    | Defines a visible heading for a <details> element   |
| <figure>     | Specifies self-contained content, like illustrations, diagrams, photos, code listings, etc.         |
| <figcaption> | Defines a caption for a <figure> element  |
| <footer>     | Defines a footer for a document or section  |
| <header>     | Defines a header for a document or section  |
| <hgroup>     | Groups a set of <h1> to <h6> elements when a heading has multiple levels                            |
| <mark>       | Defines marked/highlighted text   |
| <meter>      | Defines a scalar measurement within a known range (a gauge)   |
| <nav>        | Defines navigation links  |
| <progress>   | Represents the progress of a task   |
| <ruby>       | Defines a ruby annotation (for East Asian typography)   |
| <rt>         | Defines an explanation/pronunciation of characters (for East Asian typography)                      |
| <rp>         | Defines what to show in browsers that do not support ruby annotations                               |
| <section>    | Defines a section in a document   |
| <time>       | Defines a date/time   |
| <wbr>        | Defines a possible line-break   |

# Nuevos elementos en HTML5

- Nuevos elementos de multimedia:

| Tag      | Description  |
|----------|--|
| <audio>  | Defines sound content  |
| <video>  | Defines a video or movie   |
| <source> | Defines multiple media resources for <video> and <audio>                           |
| <embed>  | Defines a container for an external application or interactive content (a plug-in) |
| <track>  | Defines text tracks for <video> and <audio>  |

- El nuevo elemento para dibujado canvas:

| Tag      | Description   |
|----------|---|
| <canvas> | Used to draw graphics, on the fly, via scripting (usually JavaScript) |

- Nuevos elementos para form:

| Tag        | Description  |
|------------|--|
| <datalist> | Specifies a list of pre-defined options for input controls |
| <keygen>   | Defines a key-pair generator field (for forms)             |
| <output>   | Defines the result of a calculation                        |

## Elementos eliminados:

- <acronym>, <applet>, <basefont>, <big>, <center>, <dir>, <font>, <frame>, <frameset>, <noframes>, <strike>, <tt>

# Tipos de entrada HTML5

- HTML5 tiene nuevos tipos de entrada para formularios. Estas nuevas características permiten mejorar el control de entrada y validación
  - Tipo de entrada: `color`
    - Seleccionar tu color favorito: `<input type="color" name="favcolor">`
  - Tipo de entrada: `date`, permite seleccionar una fecha
    - Cumpleaños: `<input type="date" name="bday">`
  - Tipo de entrada: `datetime`, permite seleccionar una fecha y hora (con zona de tiempo).
    - Cumpleaños: `<input type="datetime" name="bdaytime">`
  - Tipo de entrada: `email`. Utilizado para campos de entrada que deberían tener una dirección de email
    - E-mail: `<input type="email" name="usremail">`
  - Tipo de entrada: `number`
  - Tipo de entrada: `range`, utilizado para campos de entrada que deberían contener un valor de un rango de números
  - Tipo de entrada: `search`, `tel`, `time`, `url`, `week`

# HTML5 Input types HTML5

```
<!DOCTYPE html>
<html>
<body>

<form action="demo_form.asp" method="get">
Points: <input type="range" name="points" min="1" max="10"><input
type="submit"></form>

<form action="demo_form.asp">
  Select your favorite color: <input type="color" name="favcolor"><br>
  <input type="submit"></form>

<form action="demo_form.asp">
  Birthday: <input type="date" name="bday"><input type="submit"></form>

<form action="demo_form.asp">
  E-mail: <input type="email" name="usremail"><br>
  <input type="submit"></form>

</body>
</html>
```

# HMTL5 Form elements

- HTML5 tiene los siguientes elementos nuevos de formulario:
  - `<datalist>`, especifica una lista de opciones para un elemento `<input>`. Usado para proveer un "autocomplete" a elementos
  - `<keygen>`, proporciona una manera segura de autenticar usuarios
  - `<output>`, representa el resultado de un cálculo

- Ejemplo:

```
<input list="browsers" name="browser">
<datalist id="browsers">
  <option value="Internet Explorer">
  <option value="Firefox">
  <option value="Chrome">
  <option value="Opera">
  <option value="Safari">
</datalist>
```

# HTML5: Nuevos atributos de formulario



- `autocomplete` – este atributo especifica si un formulario o campo de entrada debería soportar autocompletado o no
- `novalidate` – especifica que los datos de entrada del formulario (`input`) no deberían ser validados.
- `autofocus` – especifica que un elemento `<input>` debería coger el foco cuando cargue la página.
- `list` – se refiere al elemento `<datalist>` que contiene opciones predefinidas para un elemento `<input>`.
- `pattern` – especifica una expresión regular contra el que el valor del elemento `<input>` es validado.
- `placeholder` – da una pequeña pista que describe el valor esperado de un campo de entrada.
- **Otro atributos:** `formenctype`, `formmethod`, `formnovalidate`, `formtarget`, `min` y `max`, `multiple`, `required`, `step`
  - Más detalles en: [http://www.w3schools.com/html/html5\\_form\\_attributes.asp](http://www.w3schools.com/html/html5_form_attributes.asp)
- **Ejemplo:**

```
<input type="text" name="country_code" pattern="[A-Za-z]{3}"  
title="Three letter country code" autofocus="true"  
placeholder="enter country code">
```



# HTML5 Canvas

- El elemento `<canvas>` se usa para pintar gráficos sobre la marcha.
  - El elemento `<canvas>` es un contenedor rectangular de gráficos, donde la parte superior izquierda tiene la coordenada  $(0, 0)$
  - Debemos utilizar código JavaScript para dibujar los gráficos.
    - Necesarios los atributo `id` (referido en el script), `width` y `height` para definir el tamaño.

- Ejemplo:

- Define el canvas:

```
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #000000;">
</canvas>
```

- Dibuja sobre el mismo con JavaScript:

```
<script type="text/javascript">
var c=document.getElementById("myCanvas");
// The getContext("2d") object is a built-in HTML5 object, with many
// properties and methods for drawing paths, boxes, circles, text,
// images, and more.
var ctx=c.getContext("2d");
ctx.fillStyle="#FF0000"; // make it red
ctx.fillRect(0,0,150,75);
</script>
```



# HTML5 Canvas: Primitivas de dibujo



- Para dibujar líneas usamos los siguientes métodos:
  - `moveTo(x, y)` define el punto de comienzo de la línea
  - `lineTo(x, y)` define el punto de fin de la línea
- Para dibujar un círculo en un canvas, usaremos el siguiente método:
  - `arc(x, y, r, start, stop)`
- Para dibujar texto en un canvas, las propiedades y métodos más importantes son:
  - `font` - define las propiedades de fuentes de texto
  - `fillText(text, x, y)` – rellena el texto en el canvas
  - `strokeText(text, x, y)` – dibuja texto en el canvas (sin relleno)
- Se pueden usar gradientes para rellenar rectángulos, círculos, líneas, texto.
  - `createLinearGradient(x, y, x1, y1)` – crea un gradiente lineal
  - `createRadialGradient(x, y, r, x1, y1, r1)` – crea un radiante radial/circular
- Para digujar una imagen en un canvas, usaremos el siguiente método:  
`drawImage(image, x, y)`

- Ejemplo:

```
var ctx=c.getContext("2d");  
ctx.font="30px Arial";  
ctx.strokeText("Hello World",10,50);
```

# HMTL5 Inline SVG

- HTML5 tiene soporte para **SVG: Scalable Vector Graphics**
  - Usado para definir gráficos vectoriales en la Web
  - Define los gráficos en formato XML
  - Los gráficos no pierden calidad si se hace zoom o amplía su tamaño
  - Cada elemento o atributo en SVG puede ser animado
  - SVG es una recomendación W3C
  - SVG Tutorial: <http://www.w3schools.com/svg/>

- Ejemplo:

```
<!DOCTYPE html>
```

```
<html>
```

```
  <body>
```

```
    <svg xmlns="http://www.w3.org/2000/svg"
version="1.1" height="190">
```

```
      <polygon points="100,10 40,180 190,60 10,60
160,180"
```

```
        style="fill:lime;stroke:purple;stroke-width:5;fill-
rule:evenodd;">
```

```
      </svg>
```

```
    </body>
```

```
</html>
```

# HTML5 Drag and Drop

- Drag and drop permite “agarrar” un objeto y “dejarlo” en una localización diferente.
  - En HTML5 es parte del estándar, todo elemento es arrastrable
- Pasos:
  1. Para hacer un elemento arrastrable, pon su atributo `draggable` a `true`  

```
<img draggable="true">
```
  2. Especifica qué debería ocurrir cuando el elemento es arrastrado a través de la función de callback `ondragstart`:  

```
ev.dataTransfer.setData("Text",ev.target.id);
```
  3. Haz el “drop”, gestionando el evento `drop`

# HTML5 Drag and Drop: Example

```
<!DOCTYPE HTML>
<html>
  <head>
    <style type="text/css">#div1 {width:350px;height:70px;padding:10px;border:1px solid #aaaaaa;}</style>
    <script>
      function allowDrop(ev) { ev.preventDefault(); }
      function drag(ev) {
        ev.dataTransfer.setData("Text",ev.target.id);
      }
      function drop(ev) {
        ev.preventDefault();
        var data=ev.dataTransfer.getData("Text");
        ev.target.appendChild(document.getElementById(data));
      }
    </script>
  </head>
  <body>
    <p>Drag the W3Schools image into the rectangle:</p>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <br>
    
  </body>
</html>
```

# Ejemplo de Vídeo

```
<!DOCTYPE html>
<html>
<body>

<video width="320" height="240" controls="controls"
autoplay="autoplay">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  <source src="movie.webm" type="video/webm">
Your browser does not support the video tag.
</video>

</body>
</html>
```

# Geo-localización HTML5

- Usado para indicar la posición de un usuario:
  - Sólo disponible si el usuario lo aprueba.
  - Utiliza el método `getCurrentPosition()` para obtener la posición del usuario.

```
navigator.geolocation.getCurrentPosition(showPosition);
```

    - Devuelve un objeto de coordenadas a la función especificada en el parámetro
- Algunos atributos del objeto `Geolocation` son:
  - `coords.latitude`: la latitud como un número decimal
  - `coords.longitude`: la longitud como un número decimal
- Otro métodos interesantes del objeto `Geolocation`:
  - `watchPosition()` – devuelve la posición actual del usuario y sigue devolviendo posiciones mientras el usuario se mueve.
  - `clearWatch()` – Para el método `watchPosition()`



```
<!DOCTYPE html>
<html>
<body>
  <p id="demo">Click the button to get your position:</p>
  <button onclick="getLocation()">Try It</button>
  <div id="mapholder"></div>
  <script>
    var x=document.getElementById("demo");
    function getLocation() {
      if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(showPosition,showError);
      } else{x.innerHTML="Geolocation is not supported by this browser.";}
    }
    function showPosition(position) {
      var latlon=position.coords.latitude+","+position.coords.longitude;
      var
      img_url="http://maps.googleapis.com/maps/api/staticmap?center="+latlon+"&zoom=14&size=400x300&sensor=false";
      document.getElementById("mapholder").innerHTML="<img src='"+img_url+"'>";
    }
    function showError(error) {
      switch(error.code) {
        case error.PERMISSION_DENIED:
          x.innerHTML="User denied the request for Geolocation."
          break;
        case error.POSITION_UNAVAILABLE:
          x.innerHTML="Location information is unavailable."
          break;
        case error.TIMEOUT:
          x.innerHTML="The request to get user location timed out."
          break;
        case error.UNKNOWN_ERROR:
          x.innerHTML="An unknown error occurred."
          break;
      }
    }
  </script>
</body>
</html>
```

# HTML5 Web Storage



- Con HTML5, las páginas web pueden almacenar datos localmente dentro del browser del navegador
- Los datos se guardan en pares clave/valor, y una página web sólo puede acceder a datos guardados por ella misma
  - Se guardan como strings
- Hay dos nuevos objetos para guardar datos en el cliente:
  - `localStorage` – guarda datos sin expiración temporal
  - `sessionStorage` – guarda datos para una sesión
- Para comprobar si el navegador lo soporta como sigue:

```
if (typeof (Storage) !== "undefined") {  
    // Yes! localStorage and sessionStorage support!  
    // Some code.....  
} else {  
    // Sorry! No web storage support..  
}
```





# HTML5 Caché de Aplicación



- HTML5 proporciona una caché de aplicación que es accesible cuando la aplicación no tiene acceso a Internet
- Uso:
  - Declarar que elementos son cacheables en un fichero con la extensión `appcache`
  - Referir a tal fichero desde el HTML (`AppCache.html`):  
`<html manifest="demo_html.appcache">`
- Para más detalles:
  - [http://www.w3schools.com/html/html5\\_app\\_cache.asp](http://www.w3schools.com/html/html5_app_cache.asp)

# Web SQL Storage

- Las bases de datos Web son alojadas y hechas persistentes dentro de un navegador de usuario
  - Permiten a los desarrolladores crear aplicaciones con habilidades de consulta avanzadas y la capacidad de trabajar tanto online como off-line
- Opciones:
  - Base de datos Web SQL
    - Almacenamiento web basado en sqlite3, soportado por todos menos Mozilla
      - <http://dev.w3.org/html5/webdatabase/>
    - Un buen ejemplo:
      - <http://www.html5rocks.com/en/tutorials/webdatabase/todo/>
  - IndexedDB
    - Reemplazo de SQL Database, será el estándar
      - <http://www.html5rocks.com/en/tutorials/indexeddb/todo/>
  - ¿Por qué IndexedDB en vez de SQL Database?
    - <http://hacks.mozilla.org/2010/06/beyond-html5-database-apis-and-the-road-to-indexeddb/>

# HTML5 Web Worker

- Un web worker es un proceso JavaScript ejecutándose en background, sin afectar el rendimiento de la página
- Para crear un `WebWorker`, hacer:  

```
w=new Worker("demo_workers.js");
```
- Para comunicar información a la página original se hace uso de:
  - `postMessage()` —envía un mensaje de vuelta a la página HTML
- Revisar el ejemplo: `WebWorker.html`

# HTML5 Server-sent events



- HTML5 Server-Sent Events permiten a una página web obtener actualizaciones de un servidor
- Un server-sent event implica que la página web es automáticamente actualizada desde el servidor
- Por ejemplo: actualizaciones de Facebook/Twitter updates, valores en bolsa, noticias, ...
- El objeto `EventSource` es usado para recibir notificaciones de eventos server-sent :

```
var source=new EventSource ("demo_sse.php");  
source.onmessage=function(event) {  
    document.getElementById("result").innerHTML+=event.data + "<br>";  
};
```

- Se necesita un servidor capaz de enviar actualizaciones de datos de manera continuada
  - Revisar ejemplo `node.js\server-sent\server-sent-server.js`
    - <http://www.html5rocks.com/en/tutorials/eventsource/basics/>
  - La sintaxis de eventos server-sent es sencilla
    - Añade la cabecera `Content-Type: text/event-stream`

```
<?php  
header('Content-Type: text/event-stream');  
header('Cache-Control: no-cache');  
$time = date('r');  
echo "data: The server time is: {$time}\n\n";  
flush();  
?>
```

# Videos en la web

- Hasta ahora, no ha habido un modo estándar de mostrar vídeos y películas en páginas web
  - Casi todos los navegadores usan Flash
- HTML5 define un nuevo elemento que permite especificar un modo estándar de empotrar video en una página web: el elemento `<video>`
  - El atributo `control` añade controles como play, pause, and volume.
  - Es buena idea también incluir los atributos `width` y `height`
  - Los métodos, propiedades y eventos DOM permiten manipular el `<video>` y `<audio>` mediante JavaScript

- Ejemplo:

```
<video width="320" height="240" controls="controls">
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

- Algunos ejemplos en:

- Video: [http://www.w3schools.com/html/tryit.asp?filename=tryhtml5\\_video\\_js\\_prop](http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_video_js_prop)
- Audio: [http://www.w3schools.com/html/tryit.asp?filename=tryhtml5\\_audio\\_all](http://www.w3schools.com/html/tryit.asp?filename=tryhtml5_audio_all)

# CSS

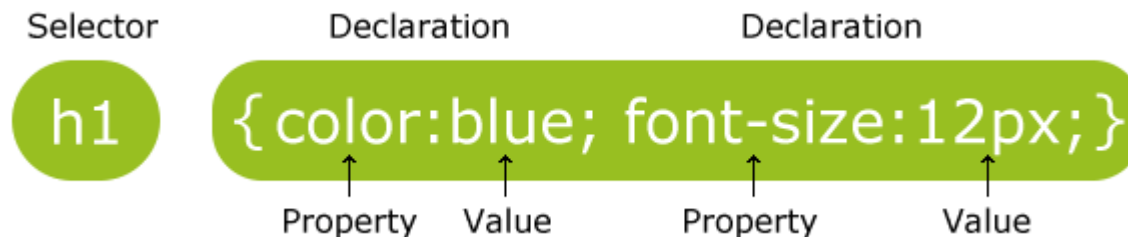


- CSS controla el estilo y formato de las páginas web
- La información de un CSS puede ser provista por:
  - Autor de estilos
    - Un fichero CSS referenciado desde el documento
    - Embebido en el documento
  - Estilo del usuario
    - Un fichero CSS local es especificado por el usuario en su navegador para ser aplicado a todos los documentos
  - Estilo del User-Agent
    - La hoja de estilo por defecto aplicada por el navegador
- Las principales ventajas de CSS son:
  - Información de presentación para una página web completa o una colección de páginas residente en un único lugar donde puede ser actualizada
  - Diferentes usuarios pueden tener sus propias hojas de estilo (fuentes grandes)
  - El código del documento es reducido en tamaño y complejidad, porque no tiene que contener información de presentación

- Los elementos de un documento son seleccionados a través de **selectores**:
  - Todos los elementos → `'*'`
  - Por nombre de elemento → `'p'` ó `'h2'`
  - Descendientes
    - Todos los elementos a descendientes de li `'li a'`
  - Atributos de clase o identificadores
    - `.nombreclase` y/o `#id` para elementos con `class="nombreclase"` or `id="id"`
  - Estilos adyacentes
    - Todos los elementos A precedidos por h2 → `'h2+a'`
- Existen también pseudoclasas que redefinen el comportamiento sobre elementos:
  - `hover` → define un estilo cuando el usuario apunta a un elemento
    - `a:hover` o `#elementid:hover`
  - `:first-line`, `:visited` o `:before`
- Revisar CSS Syntax en:
  - <http://www.w3schools.com/css/default.asp>

# CSS

- Una hoja de estilos contiene un conjunto de reglas
- Cada **regla** consiste de un **selector** y un **bloque declarativo**
- Un bloque declarativo contiene un conjunto de declaraciones separadas por `;`
- Cada declaración consiste de una propiedad, `:` y un valor





# Ejemplo CSS

```
p {
  font-family: "Garamond", serif;
}
h2 {
  font-size: 110%;
  color: red;
  background: white;
}
.note {
  color: red;
  background: yellow;
  font-weight: bold;
}
p#paragraph1 {
  margin: none;
}
a:hover {
  text-decoration: none;
}
```

# Explicación ejemplo CSS



- Hay 5 reglas, con los selectores `p`, `h2`, `.note`, `p#paragraph1` y `a:hover`.
- Un ejemplo de declaración es `color: red`, que asigna el valor rojo a una propiedad.
- Las dos primeras reglas definen estilos para los elementos HTML `p` (párrafo) y `h2` (cabecera de segundo nivel)
  - El elemento párrafo se renderizará en la fuente `Garamond` y si no está disponible en `Serif`.
  - La cabecera será mostrada en rojo sobre fondo blanco.
- La tercera regla define una clase en CSS que puede ser asignada a cualquier elemento del documento, mediante el atributo `class`. Por ejemplo:

```
<p class="note">This paragraph will be rendered in red and bold, with a yellow background.</p>
```
- La cuarta regla afectará al elemento `p` cuyo atributo `id` tenga el valor `paragraph1`. No tendrá márgenes dentro de su caja de renderización
- La última regla define la acción `hover` para todos los elementos `<a>`.
  - Quitará el subrayado cuando el ratón se mueve sobre el elemento enlace.
- Los comentarios en CSS se indican con: `/* comment */`

# CSS3



- CSS3 es la última versión del estándar CSS
- La vieja especificación se ha dividido en partes y otras nuevas han sido añadidas
  - Selectores
  - Modelo Box
  - Fondos y bordes
  - Efectos de texto
  - Layout de múltiples columnas
  - Interfaz de usuario
  - Transformaciones 2D/3D
  - Animaciones

# Bordes & Fondos & Texto en CSS3

- Con CSS3 puedes crear bordes redondeados, añadir sombras a cajas y usar imágenes como bordes
  - `border-radius`, usado para crear bordes redondeados
  - `box-shadow`, usado para añadir sombras a las cajas
  - `border-image`, usa una imagen para crear un borde
- CSS3 contiene varias propiedades de fondo que permiten un mayor control de los elementos de fondo
  - `background-size`, especifica el tamaño de la imagen de fondo
  - `background-origin`, especifica la posición de la imagen de fondo
- CSS3 contiene varias características de texto
  - `text-shadow`, permite especificar la sombra horizontal, la sombra vertical, la distancia de difuminado y el color de la sombra:
  - `word-wrap`, hace que el texto se separe en varias líneas



# CSS3 Múltiples Columnas

- Con CSS3, podemos crear múltiples columnas para colocar texto como en los periódicos:
  - `column-count`
  - `column-gap`
  - `column-rule`
- Tabla con todas las propiedades:

| Property                       | Description  | CSS |
|--------------------------------|--|-----|
| <code>column-count</code>      | Specifies the number of columns an element should be divided into                        | 3   |
| <code>column-fill</code>       | Specifies how to fill columns  | 3   |
| <code>column-gap</code>        | Specifies the gap between the columns  | 3   |
| <code>column-rule</code>       | A shorthand property for setting all the <code>column-rule-*</code> properties           | 3   |
| <code>column-rule-color</code> | Specifies the color of the rule between columns  | 3   |
| <code>column-rule-style</code> | Specifies the style of the rule between columns  | 3   |
| <code>column-rule-width</code> | Specifies the width of the rule between columns  | 3   |
| <code>column-span</code>       | Specifies how many columns an element should span across                                 | 3   |
| <code>column-width</code>      | Specifies the width of the columns   | 3   |
| <code>columns</code>           | A shorthand property for setting <code>column-width</code> and <code>column-count</code> | 3   |





# CSS3 Transformaciones 2D

- Una transformación es un efecto que hace que un elemento cambie su forma, tamaño y posición:
  - `translate()`
  - `rotate()`
  - `scale()`
  - `skew()`
  - `matrix()`
- Por ejemplo:

```
div
{
  transform: rotate(30deg);
  -ms-transform: rotate(30deg); /* IE 9 */
  -webkit-transform: rotate(30deg); /* Safari and
Chrome */
  -o-transform: rotate(30deg); /* Opera */
  -moz-transform: rotate(30deg); /* Firefox */
}
```





# Transiciones CSS3

- Con CSS3 podemos pasar los elementos de un estilo a otro, sin recurrir a Flash o JavaScript.
- Permite hacer que ciertas propiedades cambien gradualmente:
  - Especificar la propiedad CSS a la que queremos añadir un efecto
  - Especificar la duración del efecto
- Por ejemplo:

```
div {  
    width:100px;  
    height:100px;  
    background:red;  
    transition-property:width;  
    transition-duration:1s;  
    transition-timing-function:linear;  
    transition-delay:2s;  
}
```



# Animaciones CSS3

- Con CSS3 podemos crear animaciones que remplazan a las imágenes animadas, animaciones Flash o JavaScript en muchas páginas web
  - Para crear animaciones CSS3 hay que usar la regla `@keyframes`
    - Especificar el estilo CSS en la regla `@keyframes` para que cambie del estado actual al nuevo definido
    - Asociar la animación a un selector especificando dos propiedades CSS3 para la animación: a) su nombre y/o b) su duración

- Por ejemplo:

```
@keyframes myfirst
{
  from {background: red;}
  to {background: yellow;}
}
div
{
  animation: myfirst 5s;
  -moz-animation: myfirst 5s; /* Firefox */
  -webkit-animation: myfirst 5s; /* Safari and Chrome */
  -o-animation: myfirst 5s; /* Opera */
}
```





# El lenguaje JavaScript

- **JavaScript** (a menudo abreviado como JS) es un lenguaje ligero e interpretado con funciones como entidad primaria, sobretodo conocido como lenguaje de scripting para la Web, pero usado en muchos entornos no basados en navegador como node.js o Couchbase
  - Es un lenguaje que soporta los paradigmas orientado a objetos, imperativo y funcional
- Su sintaxis está fuertemente influenciada por C
  - Copia muchos nombres y convenciones de Java, aunque son dos lenguajes muy diferentes
- Netscape lo creó en 1995 como Mocha, renombrándolo luego a LifeScript y finalmente por marketing a JavaScript
  - Desde 1997 es un estándar EMCA, denominado como ECMAScript
    - Todas los navegadores recientes soportan la especificación ECMAScript 5.1
    - Gestionado por Mozilla Foundation: versión actual 1.8.5



# Aplicabilidad

- Principalmente usado en la parte cliente de las aplicaciones web, como parte del navegador Web, con el propósito de ofrecer una interfaz de usuario avanzada y permitir páginas web dinámicas
  - Otorga acceso programático al entorno de hosting (navegador)
- Fuera de las páginas web es usado en documentos PDF, widgets y cada vez más para programación genérica y código en la parte servidora (Node.js)

# JavaScript comparado con Java

- Ambos tienen **sintaxis C-like**. Ambos son object-oriented, se ejecutan en un sandbox (dentro del navegador), y son extensamente usados como parte cliente de aplicaciones web
- Java tiene **static typing**; JavaScript **dynamic typing** (una variable puede tener un objeto de cualquier tipo y ser modificado dinámicamente)
- JavaScript es **weakly typed** (`'0.0000' == 0`, `0 == ""`, `false == ""`) mientras que Java es más strongly typed
- Java genera bytecode (código binario) en compilación; JavaScript es interpretado y cargado como código fuente
- Los objetos en Java son basados en clases; en JavaScript son **prototype-based**
- JavaScript tiene muchas características de **programación funcional** como Scheme

| JavaScript  | Java  |
|---|---|
| <b>Object-oriented.</b> No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically. | <b>Class-based.</b> Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically. |
| Variable data types are not declared ( <b>dynamic typing</b> ).   | Variable data types must be declared ( <b>static typing</b> ).  |



# Características

- **Imperativo y estructurado**

- JavaScript soporta gran parte de la sintaxis de programación estructurada de C (e.g. sentencias `if`, bucles `while`, sentencias `switch`...)
- C-style block-level scoping is not supported (instead, JavaScript has function-level scoping).
  - JavaScript 1.7, however, supports block-level scoping with the `let` keyword

- **Dinámico**

- *Tipado dinámico*: como en la mayoría de los lenguajes de scripting, los tipos están asociados a valores no a las variables
- *Basados en objetos*. Los objetos JavaScript son arrays asociativos
  - Los nombres de propiedades de objetos son claves en formato string:  
`obj.x = 10` y `obj['x'] = 10`
- *Evaluación en tiempo de ejecución*: la función `eval` puede ejecutar sentencias provistas como strings en tiempo de ejecución.



# Características

- **Basada en prototipos**

- Usa prototipos en vez de clases para la herencia
- Las funciones cumplen el doble propósito de actuar como constructores y métodos
  - Si usamos un `new` delante de una llamada a una función, entonces creamos un nuevo objeto que podemos manipular a través de la palabra clave `this`
  - A diferencia de muchos lenguajes orientados a objetos, no hay distinción entre una definición de función y de método
    - Cuando una función es llamada como método de un objeto, la palabra clave `this` permite manipular los atributos del objeto cuyo método ha sido invocado

- **Misceláneo**

- JavaScript depende de un entorno de ejecución (ej. un navegador web) para proporcionar objetos y métodos con los que los scripts puede interactuar con “el mundo externo”
- *Funciones variádicas*: un número indefinido de parámetros pueden ser usados para invocar una función
- JavaScript también ofrece soporte para expresiones regulares al estilo de Perl



# Sintaxis JavaScript

- Es case-sensitive
- El “;” no es necesario al final de cada sentencia
- Los comentarios siguen la misma sintaxis que en C++,  
`/** / y //`
- Las variables se declaran dinámicamente asignándoles un valor o alternativamente precediéndolas con la palabra `var`
  - **Las variables declaradas con `var` dentro de una función son locales**
    - Una variable declarada con `var` sin valor inicial tiene el valor `undefined`.
  - Puedes crear una constante con la palabra clave `const`
    - `const prefix = '212';`



# Expresiones y Operadores

- Expresiones, unidad de código que se resuelve en un valor
  - Asignaciones. Por ejemplo, `x=7`
  - Expresiones. Ej. `3+4`
- Operadores:
  - De asignación:
    - `x += y`, `x >>= y` or `x |= y`
  - De comparación:
    - Strict equal (`===`), Devuelve `true` si los operandos son iguales y del mismo tipo: `3 === var1`
    - Otros: `3 == var1`, `var2 > var1` or `var1 <= var2`
  - Aritméticos:
    - `12 % 5`, `++x`
  - Operadores de bit:
    - `a & b`, `a | b`, `a ^ b`, ...
    - E.g. `15 & 9 // 1111 & 1001 = 1001`
  - Operadores lógicos:
 

```
var a1 = true && true;      // t && t returns true
var o2 = false || true;   // f
var n1 = !true;          // !t returns false
```
  - Operador interrogación:
 

```
var status = (age >= 18) ? "adult" : "minor";
```

# Expresiones regulares

- Patrones que identifican combinaciones de caracteres en strings
- Son objetos en JavaScript
  - Utilizados con los métodos `exec` y `test` de `RegExp`, y con los métodos `match`, `replace`, `search` y `split` de `String`.

- Para crear una expresión regular:

- `var re = /ab+c/;`
- `var re = new RegExp("ab+c");`

- Ejemplo:

```
var re = /\(?\d{3}\)?([-\/\.\])\d{3}\1\d{4}/;
function testInfo(phoneInput) {
    var OK = re.exec(phoneInput.value);
    if (!OK)
        window.alert(RegExp.input + " isn't a phone number
with area code!");
    else
        window.alert("Thanks, your phone number is " + OK[0]);
}
```



# Literales en JavaScript

- Se pueden usar literales para construir
  - Arrays: `var coffees = ["French Roast", "Colombian", "Kona"];`
  - Objetos:

```
var Sales = "Toyota";  
function CarTypes(name) {  
    return (name == "Honda") ?  
        name :  
        "Sorry, we don't sell " + name + "." ;  
}
```

```
var car = { myCar: "Saturn", getCar: CarTypes("Honda"),  
special: Sales };  
console.log(car.myCar); // Saturn  
console.log(car.getCar); // Honda  
console.log(car.special); // Toyota
```

# Estructuras de Datos

- Arrays → mapas de enteros a objetos
  - Define métodos como `join`, `slice` y `push`
  - Tiene la propiedad `length`
  - Ejemplo:

```
myArray = [0,1,,,4,5]; // array with length 6  
and 4 elements
```

```
myArray = new Array(0,1,2,3,4,5); // array  
with length 6 and 6 elements
```

```
myArray = new Array(365); // an empty array  
with length 365
```



# Estructuras de Control

- Son idénticas a las que nos encontramos en C++

- Sentencia `if ... else`

```
if (theForm.mail.value == "" || /^\\w+([\\.-
]?\\w+)*@\\w+([\\.-
]?\\w+)*\\.\\w{2,3})+$/\\.test(theForm.mail.value)) {
    alert(++ Invalid E-mail address! Please re-
enter. ++");
    return false;
}
```

- Operador ?

```
condition ? statement : statement;
```

- Bucle `while`:

```
while (condition) { statements }
```

- `do ... while`

```
do { statements } while (condition);
```



# Estructuras de Control

- For loop

```
for ([initial-expression]; [condition]; [increment-expression]) { statements }
```

- For in loop

```
for (variable in object) { statements }
```

- Switch

```
switch (expression) {  
  case label1 :  
    statements;  
    break;  
  case label2 :  
    statements;  
    break;  
  default :  
    statements;  
}
```



# Funciones

- Una función es un bloque de instrucciones a las que se ha dado un nombre
- Los argumentos de un función pueden ser accedidos a través de la palabra clave `arguments`
- No es necesario invocar a una función con todos los parámetros declarados, los que no se pasan toman el valor `undefined`

- **Ejemplo:**

```
function gcd(segmentA, segmentB) {  
    while (segmentA != segmentB)  
        if (segmentA > segmentB)  
            segmentA -= segmentB;  
        else  
            segmentB -= segmentA;  
    return (segmentA);  
}
```

# Funciones son un tipo de dato primario en JavaScript

- En JavaScript podemos pasar una función como parámetro cuando se llama otra función bien asignando la función a una variable o bien definiendo una función anónima:

```
function say(word) {  
  console.log(word);  
}  
function execute(someFunction, value) {  
  someFunction(value);  
}  
execute(say, "Hello");
```

```
// alternative: anonymous function  
function execute(someFunction, value) {  
  someFunction(value);  
}  
execute(function(word) { console.log(word) }, "Hello");
```

# Funciones especiales

- JavaScript ofrece las siguientes funciones especiales:
  - `eval` – evalúa el contenido de un string como código JavaScript  
`eval(expr);`
  - `isFinite` – evalúa un argumento para determinar si es un número finito  
`isFinite(number);`
  - `isNaN` – evalúa un argumento para determinar si es "NaN" (not a number)  
`isNaN(testValue);`
  - `parseInt` y `parseFloat` – Procesan un string para retornar, en caso correcto, un valor numérico
  - `Number` y `String` – permite convertir un objeto en un número o string
  - `encodeURIComponent`, `decodeURIComponent`, `encodeURIComponent`, and `decodeURIComponent` – para codificar y decodificar pares nombre/valor en URLs



# Manejo de Errores

- A través de las típicas construcciones `try/catch/finally`

- Ejemplo:

```
try {
    // Create an array
    arr = new Array();
    // Call a function that may not succeed
    func(arr);
} catch (...) {
    // Recover from error
    logError();
} finally {
    // Even if a fatal error occurred, we can still free
    our array
    delete arr;
}
```

- La función `eval()` permite evaluar código JavaScript en tiempo de ejecución.



# OOP en JavaScript

- Los objetos en JavaScript toman la forma internamente de un array asociativo, mapean nombres de propiedades a valores
  - Son contenedores para valores y funciones
  - Tiene varios tipos de objetos ya definidos:
    - `Array`, `Boolean`, `Date`, `Function`, `Math`, `Number`, `Object`, `RegExp` y `String`
    - También existen objetos soportados por el entorno de ejecución como `window`, `form`, `links` en el DOM.
- JavaScript es un lenguaje orientado a objetos basado en prototipos.
  - La herencia tiene lugar entre objetos en vez de clases
  - Se pueden añadir nuevas propiedades y métodos a un objeto a través de la palabra clave `prototype`
- No es obligatorio borrar objetos ya que tiene garbage collector
- JavaScript tiene los siguientes tipos de datos primitivos:
  - Números como `42` or `3.14159`
  - Valores lógicos `true` or `false`
  - Strings, como `"Howdy!"`
  - `null`, palabra clave denotando un valor nulo
  - `undefined`, indica que el valor no ha sido definido

# Ejemplo OOP en JavaScript

```
// constructor function
function MyObject(attributeA, attributeB) {
  this.attributeA = attributeA
  this.attributeB = attributeB
}

// create an Object
obj = new MyObject('red', 1000);

// access an attribute of obj
alert(obj.attributeA);

// access an attribute with the associative array notation
alert(obj["attributeA"]);

// add an new attribute
obj.attributeC = new Date();

// remove an attribute of obj
delete obj.attributeB;

// remove the whole Object
delete obj;
```



# Ejemplo Herencia

```
function Base() {
  this.Override = function() {
    alert("Base::Override()");
  }
  this.BaseFunction = function() {
    alert("Base::BaseFunction()");
  }
}
function Derive()
{
  this.Override = function() {
    alert("Derive::Override()");
  }
}
Derive.prototype = new Base();

d = new Derive();
d.Override();
d.BaseFunction();
d.__proto__.Override(); // mozilla only
```

- produce:

```
Derive::Override()
Base::BaseFunction()
Base::Override() // mozilla only 59
```



# Acceso público y privado a variables en JavaScript

JS

```
// declaration
function myClass() {
  // variables
  var privateVar = "I am private";
  this.publicVar = "I am public";

  // functions
  var privateFunction = function() {
    alert("I am private");
  }
  this.publicFunction = function(){
    alert("I am public");
  }
}

// usage
var myInstance = new myClass();

myInstance.publicVar = "new value";
alert(myInstance.privateVar); // undefined!

myInstance.publicFunction();
myInstance.privateFunction(); // error!
```

# Variables privadas y métodos privados

- Otro ejemplo:

```
function Restaurant() {  
  }  
Restaurant.prototype = (function() {  
  var private_stuff = function() {  
    // Private code here  
  };  
  return {  
    constructor:Restaurant,  
    use_restroom:function() {  
      private_stuff();  
    }  
  };  
})();  
var r = new Restaurant();  
r.use_restroom(); // This will work:  
r.private_stuff(); // This will cause an error:
```

- Documentación en:

- <http://javascript.crockford.com/private.html>
- <http://webreflection.blogspot.com.es/2008/04/natural-javascript-private-methods.html>



# Objetos predefinidos

- JavaScript ofrece el objeto `Array` para trabajar con arrays en tu código:

```
var arr = [42];
var arr = Array(42); // Creates an array with no element, but with
arr.length set to 42
// The above code is equivalent to
var arr = [];
arr.length = 42;
var emp = [];
emp[0] = "Casey Jones";
var colors = ['red', 'green', 'blue'];
for (var i = 0; i < colors.length; i++) {
    console.log(colors[i]);
}
```

- El objeto `Array` tiene los siguientes métodos:
  - `concat()` junta dos arrays y devuelve uno nuevo
  - `join(delimitador = ", ")` junta los elementos de un array en un **string**
  - `push()` añade uno o más elementos al final del array y devuelve la **longitud** resultante del mismo
  - `sort()` ordena los elementos del array



# Objetos predefinidos

- El objeto `Boolean` envuelve al tipo primitivo booleano
- El objeto fecha `Date` tiene un número elevado de métodos para manipular fechas:

```
function JSClock() {
  var time = new Date();
  var hour = time.getHours();
  var minute = time.getMinutes();
  var second = time.getSeconds();
  var temp = "" + ((hour > 12) ? hour - 12 : hour);
  if (hour == 0)
    temp = "12";
  temp += ((minute < 10) ? ":0" : ":") + minute;
  temp += ((second < 10) ? ":0" : ":") + second;
  temp += (hour >= 12) ? " P.M." : " A.M.";
  return temp;
}
console.log(JSClock());
```

- El objeto `Function` hace que un string en JavaScript se compile como una función
- El objeto `Math` tiene propiedades y métodos para operaciones matemáticas

```
Math.sin(1.56)
```

- El objeto `String` envuelve al tipo primitivo string.
- More info at: [https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Predefined\\_Core\\_Objects](https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Predefined_Core_Objects)



# Objeto String

- Propiedad `length` devuelve su longitud
- Métodos:
  - `charAt()`
  - `indexOf()`
  - `lastIndexOf()`
  - `split()`
  - `substring()`
  - `toLowerCase()`
  - `toUpperCase()`





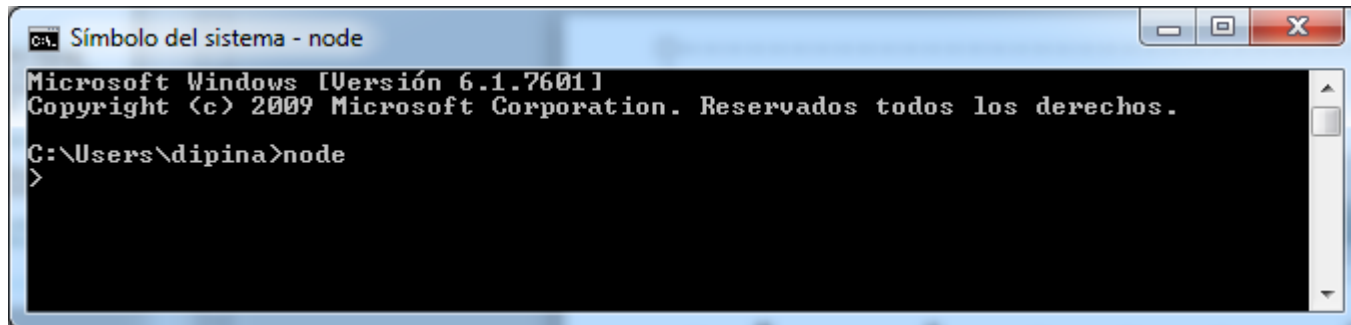
# Motores de JavaScript

- Un motor JavaScript (también conocido como *JavaScript interpreter* or *JavaScript implementation*) es un intérprete que interpreta el código fuente de JavaScript y ejecuta el script del mismo modo
  - El primer motor de JavaScript fue creado por Brendan Eich para el Netscape Navigator
    - El motor, conocido como SpiderMonkey, está implementado en C.
      - Ha sido actualizado (en JavaScript 1.5) para conformar a ECMA-262 Edition 3.
        - » <https://developer.mozilla.org/en-US/docs/SpiderMonkey>
  - El motor Rhino, creado por Norris Boyd es una implementación en Java de JavaScript.
    - Rhino, como SpiderMonkey, es compatible con ECMA-262 Edition 3.
      - <https://developer.mozilla.org/en-US/docs/Rhino>
- Un navegador web es con mucho el motor de ejecución más común de JavaScript
  - Los navegadores web crean "host objects" para representar el Document Object Model (DOM) en JavaScript.



# Instalando JavaScript

- Para probar JavaScript instalaremos el motor de scripting Node.js
  - <http://nodejs.org/download/>
- Seleccionar el fichero `.msi` y asegurarse que su ejecutable, `node`, están en la variable de entorno `PATH`



```
Símbolo del sistema - node
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.
C:\Users\dipina>node
>
```

# Ejemplo: Programa completo en JavaScript

JS

```
/* Finds the lowest common multiple of two numbers
*/
function LCMCalculator(x, y) { // constructor
function
    var checkInt = function (x) { // inner function
        if (x % 1 !== 0) {
            throw new TypeError(x + " is not an
integer"); // throw an exception
        }
        return x;
    };
    this.a = checkInt(x)
    // ^ semicolons are optional
    this.b = checkInt(y);
}
```

# Ejemplo: Programa completo en JavaScript

JS

```
// The prototype of object instances created by a constructor is that constructor's "prototype" property.
LCMCalculator.prototype = { // object literal
  constructor: LCMCalculator, // when reassigning a prototype, set the constructor property appropriately
  gcd: function () { // method that calculates the greatest common divisor
    // Euclidean algorithm:
    var a = Math.abs(this.a), b = Math.abs(this.b), t;
    if (a < b) { // swap variables
      t = b;
      b = a;
      a = t;
    }
    while (b !== 0) {
      t = b;
      b = a % b;
      a = t;
    }
    // Only need to calculate GCD once, so "redefine" this method. (Actually not redefinition—it's defined on the instance
    // itself,
    // so that this.gcd refers to this "redefinition" instead of LCMCalculator.prototype.gcd.) Also, 'gcd' === "gcd", this['gcd']
    === this.gcd
    this['gcd'] = function () { return a; };
    return a;
  },
  // Object property names can be specified by strings delimited by double (") or single (') quotes.
  "lcm" : function () {
    // Variable names don't collide with object properties, e.g. |lcm| is not |this.lcm|.
    // not using |this.a * this.b| to avoid FP precision issues
    var lcm = this.a / this.gcd() * this.b;
    // Only need to calculate lcm once, so "redefine" this method.
    this.lcm = function () { return lcm; };
    return lcm;
  },
  toString: function () {
    return "LCMCalculator: a = " + this.a + ", b = " + this.b;
  }
};
```

# Ejemplo: Programa completo en JavaScript

JS

// Note: Array's map() and forEach() are defined in JavaScript 1.6. They are used here to demonstrate JavaScript's inherent functional nature.

```
[[25, 55], [21, 56], [22, 58], [28, 56]].map(function  
(pair) { // array literal + mapping function  
    return new LCMCalculator(pair[0], pair[1]);  
}).sort(function (a, b) { // sort with this comparative  
function  
    return a.lcm() - b.lcm();  
}).forEach(function (obj) {  
    output(obj + ", gcd = " + obj.gcd() + ", lcm = " +  
obj.lcm());  
});
```



# JavaScript usado fuera de páginas web

JS

- Además de web browsers y servidores, los intérpretes de JavaScript están empotrados en muchas herramientas.
  - Cada una de estas aplicaciones provee su propio modelo de objetos que da acceso a su entorno de ejecución, manteniéndose el core de JavaScript casi idéntico
- Es usado como:
  - Motor de scripting:
    - Desde JDK 1.6, el paquete `javax.script` contiene la implementación de Mozilla Rhino, puedes acceder a los atributos de la aplicación Java desde JavaScript
    - webOS usa la implementación WebKit de JavaScript en su SDK, para permitir a sus desarrolladores crear aplicaciones stand-alone en JavaScript.



# Herramientas de Desarrollo

JS

- Hay debuggers para diferentes procesadores:
  - Las aplicaciones web dentro de Firefox pueden ser depuradas usando Firebug
  - En Google Chrome presiona F12 para mostrar el depurador de JavaScript
- MiniME es un minifier JavaScript open-source, ofuscador, y herramienta de control de código para la plataforma .NET
- Algunas herramientas de depuración pueden ser encontradas en:
  - <http://blog.templatemonster.com/2012/03/09/javascript-debugging-tools/>



# Añadiendo JavaScript a HTML

- Para incluir un fragmento de JavaScript en una página HTML, el código se encierra entre las etiquetas `<script>` y `</script>`, normalmente colocadas en la sección `<head>`

```
<script language="JavaScript"
          type="text/javascript ">
    // tu código
</script>
```

- También se puede incluir un fichero externo de JavaScript con:

```
<script language='JavaScript'
src='MyFunctions.js' />
```

- Además desde cualquier elemento se puede invocar a JavaScript:

```
<a href="javascript:history.back()">go back</a>
<a href="javascript:history.go(-3)">go back</a>
```



# Uso en páginas web

JS

W3C



- El uso más común de JavaScript es para escribir funciones que son empotradas en o incluidas en páginas HTML y que interactúan con el Document Object Model (DOM) de la página
- Ejemplos:
  - Cargando nuevo contenido de página o enviando datos al servidor con AJAX sin recargar la página
  - Animación de elementos de página, ocultándolos y volviéndolos a mostrar, cambiándoles el tamaño o moviéndolos ...
  - Contenido interactivo, juegos, audio y vídeo
  - Validando valores de entrada de un formulario para asegurar que son aceptables antes de ser enviados al servidor
- Como JavaScript se ejecuta localmente en el navegador web (sin acceder a un servidor remoto), el navegador puede responder a acciones del usuario rápidamente.



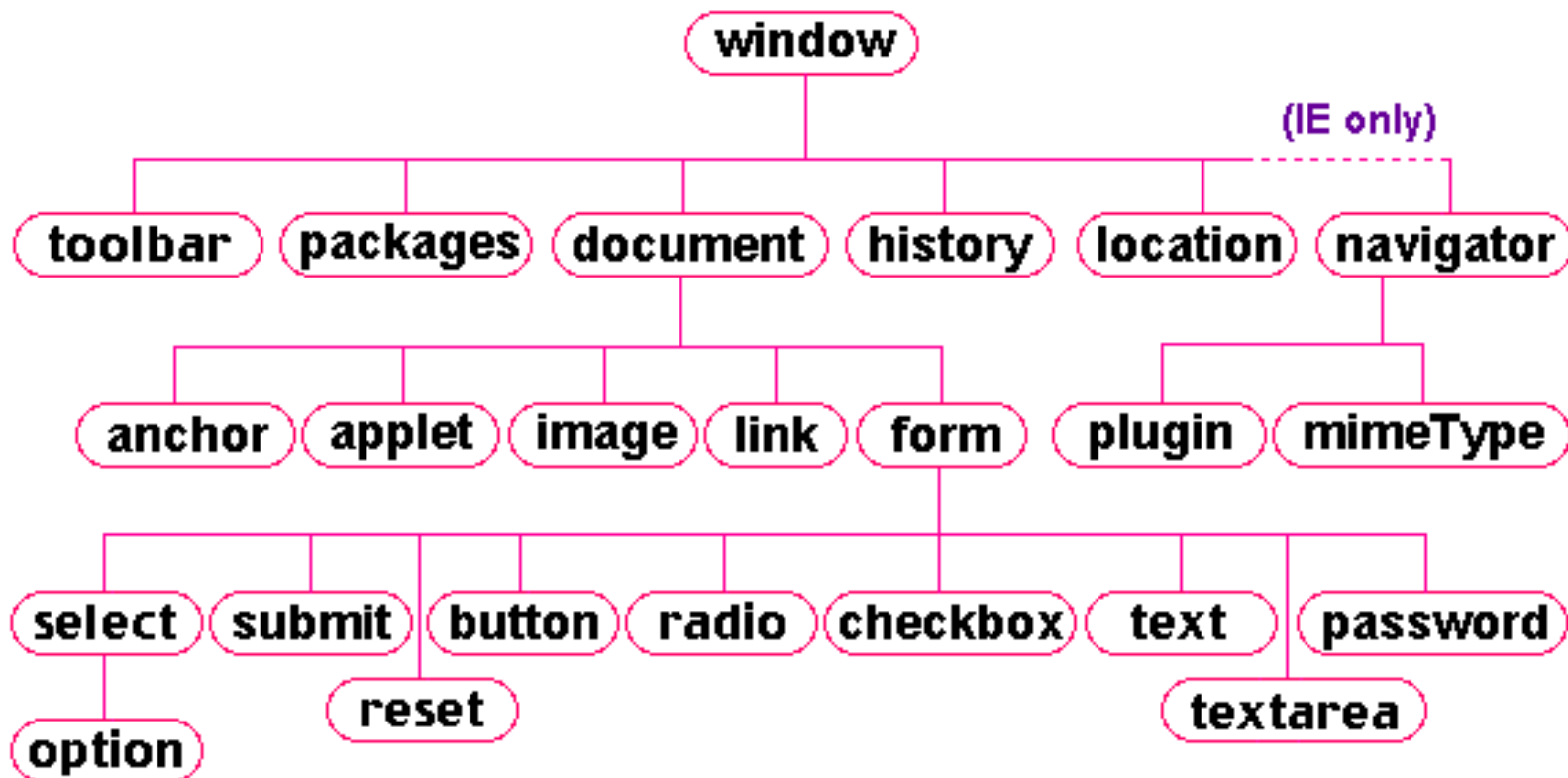
# W3C DOM



- **DOM es un Document Object Model**, un modelo sobre cómo los objetos de un documento están relacionados con otros
  - Describe cómo los elementos en una página HTML, como campos de entrada, imágenes, párrafos, etc. están relacionados con la estructura superior: `document`
  - El Level 1 DOM Recommendation ha sido desarrollado por el W3C para proveer a los lenguajes de programación acceso a cada parte de un documento XML
  - Permite:
    - Leer el texto y atributos de cada etiqueta HTML en un documento, borrar etiquetas y contenidos, incluso crear nuevas etiquetas e insertar en el documento de modo que pueda rescribir páginas sobre la marcha, sin un viaje de vuelta al servidor.



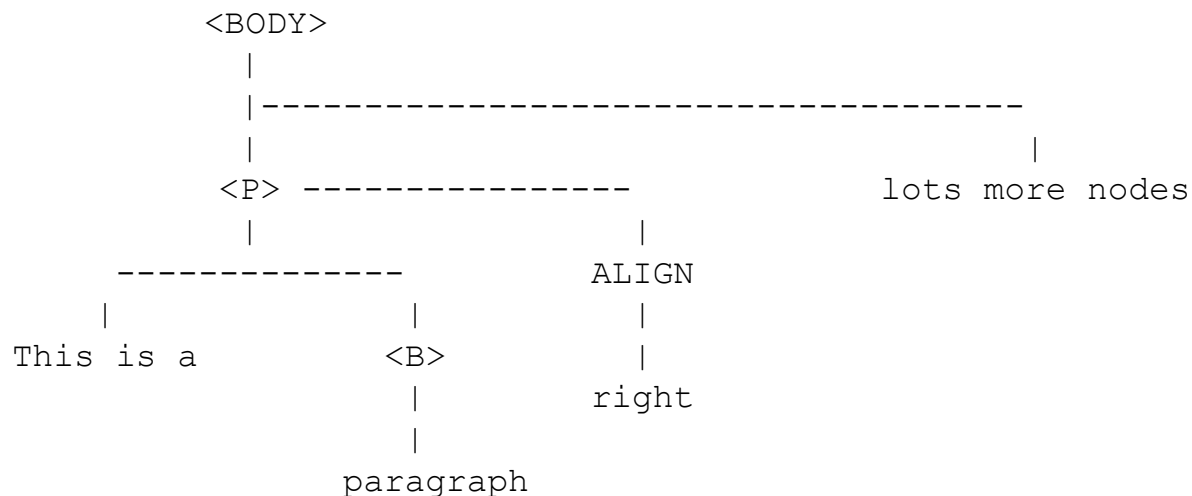
# W3C DOM





# W3C DOM: navegando por el árbol

- Si conoces la estructura de un árbol DOM, puedes navegar por él en busca de un elemento que quieres influenciar
- Asumiendo que el nodo P ha sido guardado en la variable x:
  - `x.parentNode` → accede a su padre
  - `x.childNodes[1]` → accede al 2º hijo del nodo
  - `x.firstChild`, `x.lastChild` → acceden al primero y último hijo de x
- Por ejemplo, para acceder el nodo B puedes usar la expresión:  
`document.firstChild.childNodes[0].childNodes[1];`



# W3C DOM: accediendo a un elemento

JS



- Siempre puedes saltar a un elemento de tu elección directamente
- Puedes acceder a B en el árbol abajo, el primer elemento B encontrado en el documento a través de la expresión:
  - `var x = document.getElementsByTagName('B')[0]`
  - `var x = document.getElementsByTagName('B').item(0)`
- La única manera de llegar al elemento correcto de un árbol DOM es asignar a B un ID:
- Por ejemplo,
  - `<P ALIGN="right">This is a <B ID="hereweare">paragraph</B></P>`
- Para acceder a él en JavaScript usaríamos:
  - `var x = document.getElementById('hereweare');`

# W3C DOM: Propiedades de un nodo

JS



- Cada nodo tiene las siguientes propiedades:

- nodeName → devuelve el nombre de una etiqueta HTML como un string

```
x = document.getElementById("myElement");
if (x.nodeName == "IMG")
    alert("myElement is an image tag");
```

- nodeType → devuelve un entero representando el tipo de un elemento

```
// (1) → nodo elemento, (2) → atributo y (3) texto
x = document.getElementById("myElement");
if (x.nodeType == 1) {
    alert("myElement is an element node")
}
```

- nodeValue → el valor de un nodo

```
a = document.getElementById("myElement");
b = a.childNodes.item(4).nodeValue
```

- parentNode → devuelve el nodo padre del actual

```
x = document.getElementById("myElement");
xParent = x.parentNode ;
```

- childNodes → devuelve los hijos de un nodo

```
a = document.getElementById("myElement");
b = a.childNodes.length;
b = a.childNodes.item(2);
```

- firstChild y lastChild apuntan directamente al primer y último elemento en la colección childNodes



# W3C DOM: modificando atributos de un elemento



- Dado el siguiente fragmento XHTML:

```

```
- Podemos recuperar una referencia al mismo mediante:

```
x = document.getElementById("myImg")
```
- Los siguientes métodos serán disponibles a x:
  - `getAttribute()` → devuelve el valor del atributo pasado

```
myValue = x.getAttribute("src") // devuelve
jabberwocky.gif
```
  - `setAttribute()` → asigna un valor a un atributo

```
x.setAttribute("align", "right")
```
  - `removeAttribute()` → borra el atributo pasado

```
x.removeAttribute("align")
```



# W3C DOM: cambiando un nodo



- Siguiendo el ejemplo anterior, podemos cambiar el texto del elemento B con la siguiente sentencia (realmente hay que cambiar su hijo de tipo texto):

```
document.getElementById('hereweare').firstChild.nodeValue='bold bit of text';
```

- Cambiemos ahora el atributo `align` del elemento P padre de B:

```
function test2(val) {  
    if (document.getElementById && document.createElement)  
    {  
        node = document.getElementById('hereweare').parentNode;  
        node.setAttribute('align',val);  
    }  
    else alert('Your browser doesn\'t support the Level 1 DOM');  
}
```



# W3C DOM: añadiendo y quitando un nodo



- Para crear un nodo tienes que seguir los siguientes pasos:
  1. La creación de un nodo se hace a través de un nodo especial:

```
var x = document.createElement('HR');
```
  2. Inserta el elemento creado en la parte derecha:

```
document.getElementById('inserthrhere').appendChild(x);
```
- Para borrar el elemento creado:

```
var node = document.getElementById('inserthrhere');  
node.removeChild(node.childNodes[0]);
```
- Más documentación sobre DOM en:
  - [https://developer.mozilla.org/en-US/docs/Gecko\\_DOM\\_Reference/Introduction#DOM\\_and\\_Java\\_Script](https://developer.mozilla.org/en-US/docs/Gecko_DOM_Reference/Introduction#DOM_and_Java_Script)
  - <http://www.w3schools.com/html/dom/>

# Ejemplo de Creación de Nodos



```
function buildTable() {
  docBody = document.getElementsByTagName("body").item(0);
  myTable = document.createElement("TABLE");
  myTable.id = "TableOne";
  myTable.border = 1;
  myTableBody = document.createElement("TBODY");
  for (i = 0; i < 3; i++) {
    row = document.createElement("TR");
    for (j = 0; j < 3; j++) {
      cell = document.createElement("TD");
      cell.setAttribute("WIDTH", "50");
      cell.setAttribute("HEIGHT", "50");
      textVal = "Cell" + i + "_" + j;
      textNode = document.createTextNode(textVal);
      cell.appendChild(textNode);
      row.appendChild(cell);
    }
    myTableBody.appendChild(row);
  }
  myTable.appendChild(myTableBody);
  docBody.appendChild(myTable);
}
window.onload = buildTable();
```



# El modelo de objetos HTML DOM utilizado por JavaScript



- Programar en JavaScript supone operar sobre el modelo de objetos del documento (DOM)
  - `window` → el objeto central que representa el navegador, contiene los objetos accesible a través de propiedades:
    - `Location` (`window.location`)
    - `Document` (`window.document`)
    - `History` (`window.history`)
  - Además, contiene una serie of métodos como `alert()`, `confirm()` y `prompt()`
  - No es necesario mencionar este objeto, implícitamente se asume que se invoca algo sobre él





# Objeto Window

- Representa la ventana del navegador, es la raíz de la jerarquía de objetos de JavaScript
- Propiedades:
  - `closed`
  - `defaultStatus`
  - `document`
  - `frames[]`
  - `location`
- Métodos:
  - `alert()`
  - `confirm()`
  - `prompt()`
  - `open()`
  - `setTimeout()`



# Manejo de eventos

- JavaScript añade interactividad a tus páginas web permitiendo escribir código dirigido por eventos
- Ejemplo: conectar un botón a una función que es invocada cuando el usuario hace clic:

```
<input type=button name=send onclick="handler">
```

- **Eventos definidos por algunos controles HTML:**
  - Button → onBlur, onClick, onFocus
  - Form → onReset, onSubmit
  - Frame → onLoad, onUnload
  - Link → onClick, onMouseOut, onMouseOver
  - Select → onBlur, onChange, onFocus
  - Text → onBlur, onChange, onFocus
  - Window → onBlur, onError, onFocus, onLoad, onUnload

# Ejemplo Manejo de Eventos



```
<html>
<head>
<script language="javascript" type="text/javascript">
<!--
function clearBox(box) {
    if(box.value==box.defaultValue) {
        box.value = "";
    }
}
-->
</script>
</head>
<body>
<form>
<input type="text" id="email" name="email" value="enter email
    address" onFocus="clearBox(this);">
<input type="submit" name="Submit" value="Subscribe">
</form>
</body>
</html>
```



# Referenciando Objetos en JavaScript



- Para hacer referencia al documento utilizamos `window.document` o `document`

- Dado el siguiente fragmento de código HTML:

```
<body>
  <form name="boxes">
    Enter first name: <input type="text"
name="fName"><br>
    Enter last name: <input type="text" name="lName">
  </form>
</body>
```

- Para hacer referencia a la primera entrada de texto podríamos hacer:

```
document.form.fName
```

- Alternativamente:

```
document.boxes.fName
```

- Ó:

```
document.forms[0].elements[0]
```





# Seguridad en JavaScript

- JavaScript y el DOM proporcionan el potencial para que autores maliciosos envíen scripts para ser ejecutados en el ordenador cliente vía web.
- Los autores de browsers previenen esos riesgos con dos restricciones:
  1. Los scripts se ejecutan en un sandbox dentro del cual sólo se pueden ejecutar operaciones relacionadas con la web, no tareas de propósito general como creación de ficheros.
  2. Los scripts están restringidos por la **same origin policy**: los scripts de un website no tienen acceso a información como los nombres de usuario, contraseñas o cookies enviadas al otro lado.
- Casi todas las violaciones de seguridad de JavaScript son fisuras a algunas de estas restricciones.



# JSON

- JSON, o JavaScript Object Notation, es un formato de intercambio de datos de propósito general definido como un subconjunto de la sintaxis literal de JavaScript

- Por ejemplo,

- Un objeto es creado conteniendo un sólo miembro "bindings" que contiene un array con 3 objetos, cada uno conteniendo los miembros "ircEvent", "method", y "regex"

```
var myJSONObject = {"bindings": [  
    {"ircEvent": "PRIVMSG", "method": "newURI", "regex": "^http://.*"},  
    {"ircEvent": "PRIVMSG", "method": "deleteURI", "regex": "^delete.*"},  
    {"ircEvent": "PRIVMSG", "method": "randomURI", "regex": "^random.*"}  
    ]  
};
```

- Los miembros pueden ser recuperados usando el operador `.` o los operadores de indexación.

- `myJSONObject.bindings[0].method`

- Para convertir texto JSON en un objeto, puedes evaluar la función `eval()`, que invoca el compilador de JavaScript

- `var myObject = eval('(' + myJSONtext + ')');`

- La función `eval` puede compilar y ejecutar cualquier programa JavaScript, por tanto puede lugar a problemas de seguridad



# JSON

- Es preferible y más seguro usar el parser JSON (<https://github.com/douglascrockford/JSON-js/>) que reconoce sólo texto JSON, rechazando todos los scripts

```
var myObject = JSON.parse(myJSONtext, reviver);
```

- El parámetro opcional `reviver` es una función invocada por cada clave y valor. Cada valor será remplazado por el resultado de invocar la función `reviver`

```
myData = JSON.parse(text, function (key, value) {  
    var type;  
    if (value && typeof value === 'object') {  
        type = value.type;  
        if (typeof type === 'string' && typeof window[type] ===  
'function') {  
            return new (window[type])(value);  
        }  
    }  
    return value;  
});
```

- Un `stringifier` JSON realiza el trabajo opuesto, convierte estructuras de JavaScript en texto JSON:

```
var myJSONText = JSON.stringify(myObject, replacer);
```



# Concepto AJAX

- AJAX (Asynchronous Javascript and XML) es una técnica de desarrollo web que genera aplicaciones web interactivas combinando:
  - XHTML y CSS para la presentación de información
  - Document Object Model (DOM) para visualizar dinámicamente e interactuar con la información presentada
  - XML, XSLT para intercambiar y manipular datos
    - JSON y JSON-RPC pueden ser alternativas a XML/XSLT
  - XMLHttpRequest para recuperar datos asíncronamente
  - Javascript como nexo de unión de todas estas tecnologías
- AJAX es un **patrón de diseño** que propone un nuevo modelo de interacción Web combinando las tecnologías anteriores
- Los navegadores que soportan las tecnologías mencionadas son las plataformas en las que se ejecutan las aplicaciones AJAX (Firefox, IExplorer, Opera, Chrome y Safari)





# ¿Por qué AJAX?

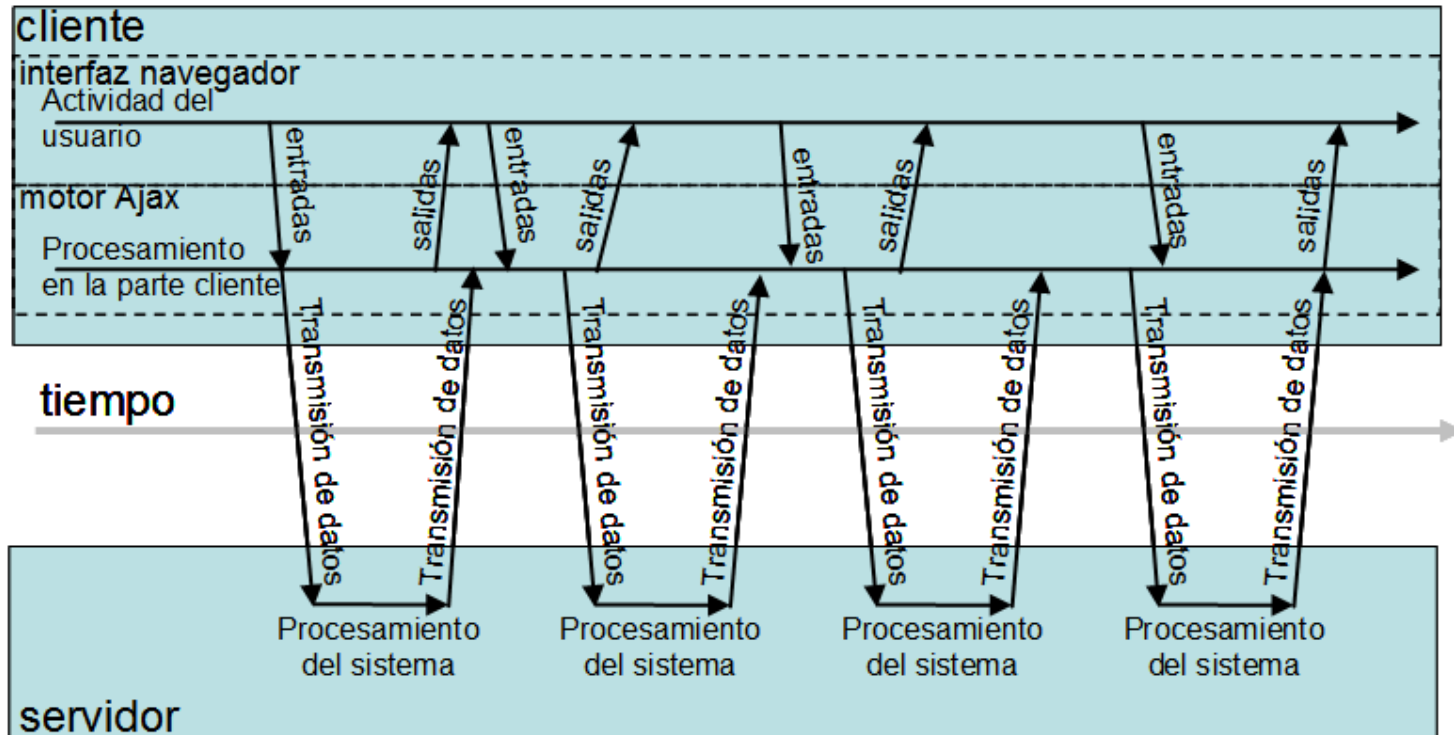
- Hasta hace poco, las aplicaciones web proliferaban debido a su simplicidad, pero:
  - Ofrecen una menor interactividad y usabilidad en comparación con las aplicaciones desktop.
  - La interacción del usuario con una aplicación web se interrumpe cada vez que se necesita algo del servidor
- Varias tecnologías han sido diseñadas para resolver este problema:
  - Java Applets, FLASH
- ¡AJAX es una nueva solución que ha revolucionado las webapps!
  - Término acuñado por Jesse James Garrett en <http://www.adaptivepath.com/publications/essays/archives/000385.php>
  - No requiere plugins o capacidades específicas de ciertos navegadores.





# AJAX vs. Aplicaciones Web Tradicionales

## modelo de aplicación web Ajax (asíncrono)





# La magia de AJAX: El objeto XMLHttpRequest

- Gracias a este objeto los clientes web pueden recuperar y entregar XML, en background → **ASÍNCRONAMENTE.**
- Usando DOM podemos leer el contenido devuelto y modificar el documento presente en el navegador
- Las innovaciones introducidas en XMLHttpRequest Level 2 están detalladas en:
  - <http://www.html5rocks.com/en/tutorials/file/xhr2/>





# Creación del XMLHttpRequest



- Desafortunadamente en su creación hay que tener en cuenta las diferencias entre navegadores:
- En Mozilla y Safari se hará:

```
var req = new XMLHttpRequest();
```
- En el ActiveX de Internet Explorer haríamos:

```
var req = new ActiveXObject("Microsoft.XMLHTTP");
```
- ...o en las últimas versiones de IE:

```
xmlreq = new ActiveXObject("Msxml2.XMLHTTP");
```
- Este objeto puede trabajar de manera asíncrona al usuario







# Métodos del XMLHttpRequest



- `abort()` → para la invocación actual
- `getAllResponseHeaders()` → devuelve todas las cabeceras (nombre-valor) como un string
- `getResponseHeader("headerLabel")` → devuelve el valor asociado a una cabecera
- `open("method", "URL" [, asyncFlag [, "userName" [, "password"]]])` → abre una conexión remota
  - Usar POST para invocaciones que envían datos al servidor, de un tamaño superior a 512 bytes
  - Usar GET para recuperar datos del servidor
  - Si `asyncFlag` es 'true' hay que definir un método que recibirá un callback cuando se genere el evento `onreadystatechange`
- `send(content)` → envía la petición con un contenido opcional
- `setRequestHeader("label", "value")` → añade una cabecera a la petición





# Propiedades Métodos del XMLHttpRequest



- Propiedades del objeto XMLHttpRequest:
  - `onreadystatechange` → asocia el manejador de eventos invocado cada vez que cambie el estado de una petición
  - `readyState` → entero con estado de objeto:
    - 0 = no inicializado
    - 1 = cargando
    - 2 = cargado
    - 3 = interactivo
    - 4 = completado
  - `responseText` → contenido textual de la respuesta enviada por el servidor
  - `responseXML` → documento DOM correspondiente a la respuesta del servidor
  - `status` → código numérico devuelto por el servidor
  - `statusText` → mensaje de texto acompañando al código de status

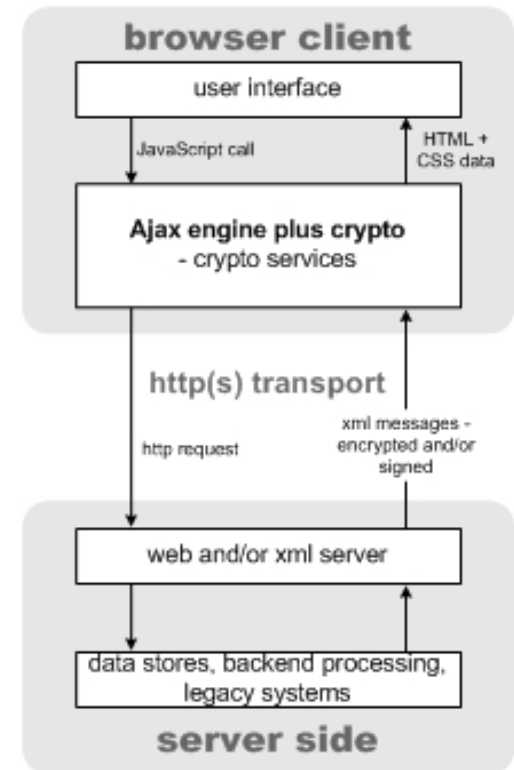




# Seguridad en AJAX

- Por motivos de seguridad las invocaciones a recursos vía HTTP solamente pueden efectuarse a páginas alojadas en el mismo dominio que el de la página que contenía el script
- Si queremos que la información de nuestras peticiones no sea visible deberemos utilizar un canal HTTPS.
- El same origin policy puede ser resuelto con:
  - CORS y JSONP  
(<http://en.wikipedia.org/wiki/JSONP>)

```
response.writeHead(200, {  
  'Content-Type': 'text/plain',  
  'Access-Control-Allow-Origin'  
: '*'  
});
```





# Nuestra primera aplicación AJAX

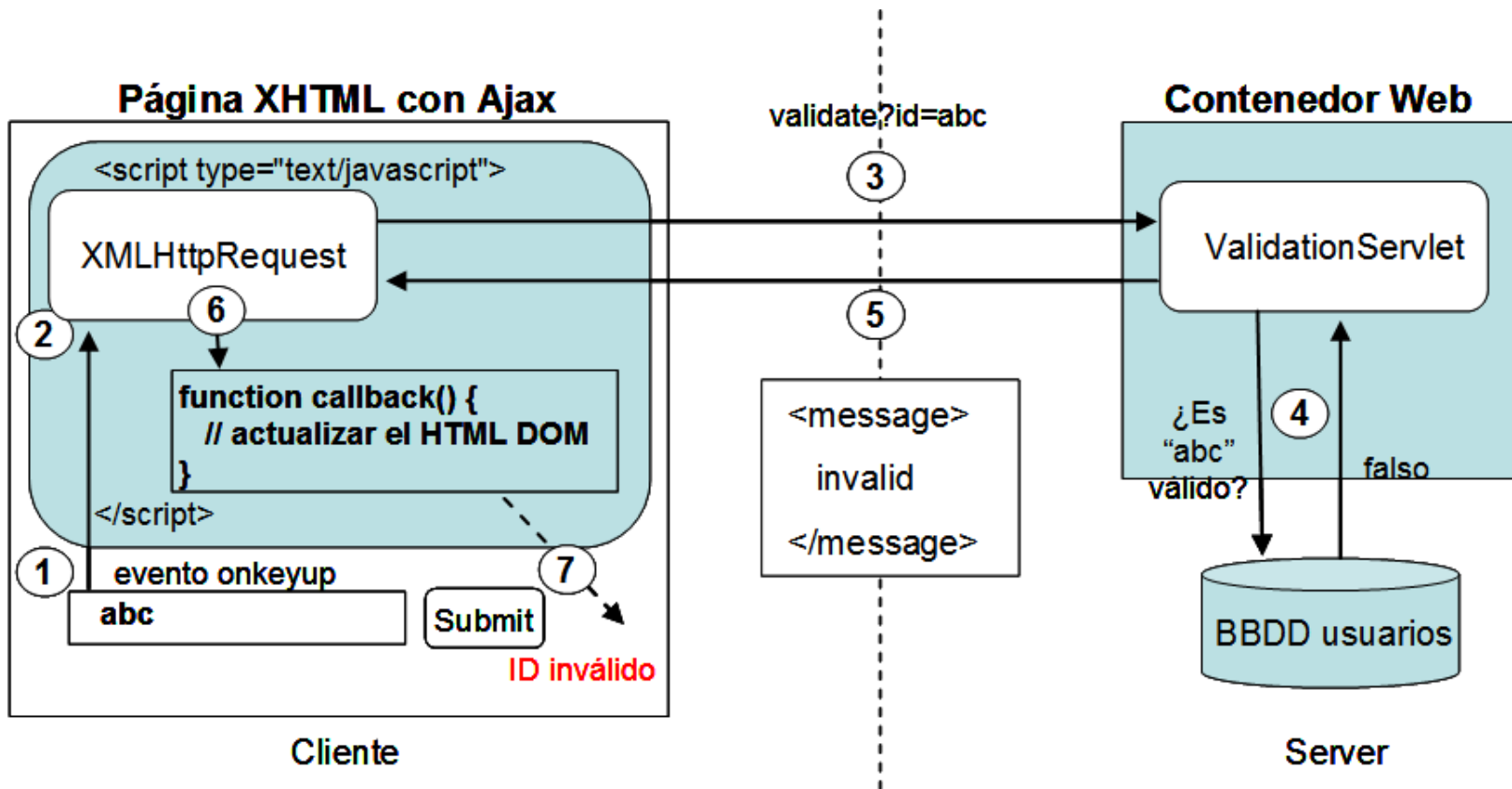


- Vamos a considerar una aplicación web con un formulario que requiere validación en el servidor sin necesidad de refrescar la página completamente
  - Es decir, realizar un HTTP POST de la página implícitamente
- La figura en la siguiente transparencia ilustra el funcionamiento de esta aplicación





# Nuestra primera aplicación AJAX





# Nuestra primera aplicación AJAX



- Los flujos de control serían:
  1. Un evento ocurre
  2. Un objeto XMLHttpRequest es creado e inicializado
  3. Se realiza una invocación remota a través del objeto XMLHttpRequest
  4. La petición es procesada por el componente servidor ValidationServlet
  5. Este servlet devuelve un documento XML con el resultado
  6. El objeto XMLHttpRequest invoca la función callback() y procesa el resultado
  7. El árbol DOM de la página web es actualizado



# Un evento ocurre



- Como resultados de eventos generados por la interacción del usuario funciones JavaScript son invocadas
- Ejemplo:

```
<input type="text"  
  size="20"  
  id="userID"  
  name="id"  
  onkeyup="validate();" >
```





# El objeto XMLHttpRequest es creado



```
var req;
function validate() {
    var idField = document.getElementById("userID");
    var url = "validate?id=" + escape(idField.value);
    if (window.XMLHttpRequest) {
        req = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        req = new ActiveXObject("Microsoft.XMLHTTP");
    }
    req.open("GET", url, true);
    req.onreadystatechange = callback;
    req.send(null);
}
```

- Los tres parámetros del método `open()` del objeto `XMLHttpRequest` indican el método HTTP utilizado, la `url` de destino y si el método se invocará de manera asíncrona o no
- `req.onreadystatechange = callback;` indica la función a invocar cuando la petición sea resuelta





# El objeto XMLHttpRequest realiza una invocación remota



- `req.send(null)` ; realiza la invocación, se pasa `null` porque un HTTP GET no tiene cuerpo
- Si se quiere utilizar un HTTP POST entonces:
  - Deberíamos enviar la cabecera `Content-Type`
- Ejemplo HTTP POST:

```
req.setRequestHeader("Content-Type",  
    "application/x-www-form-urlencoded");  
req.send("id=" +  
    escape(idTextField.value));
```





# La petición es procesada por ValidationServlet



```
// compile: javac -classpath %TOMCAT_HOME%\lib\servlet-api.jar ValidationServlet.java
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class ValidationServlet extends HttpServlet {
    private ServletContext context;
    private HashMap users = new HashMap();

    public void init(ServletConfig config) throws ServletException {
        this.context = config.getServletContext();
        users.put("solop", "account data");
        users.put("solop", "account data");
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        String targetId = request.getParameter("id");
        if ((targetId != null) && !users.containsKey(targetId.trim())) {
            response.setContentType("application/xml");
            response.setHeader("Cache-Control", "no-cache");
            response.getWriter().write("<message>invalid</message>");
        } else {
            response.setContentType("text/xml");
            response.setHeader("Cache-Control", "no-cache");
            response.getWriter().write("<message>valid</message>");
        }
    }
}
```





# La petición es procesada por ValidationServlet



- La respuesta necesariamente tiene que ser en XML, para evitar caching de respuestas se utiliza la cabecera Cache-Control:

```
response.setContentType("text/xml");  
response.setHeader("Cache-Control",  
    "no-cache");  
response.getWriter().write("<message  
>valid</message>");
```





## El objeto XMLHttpRequest invoca la función `callback` y procesa su resultado

- La función `callback` es invocada cuando se producen cambios en el `readyState` del objeto XMLHttpRequest. Si `readyState == 4` significa que la invocación ha acabado:
- Una vez recibida la respuesta la representación DOM de un documento XHTML puede ser modificado incluyendo el contenido de la respuesta.
  - `req.responseText` → devuelve el contenido textual de la respuesta
  - `req.responseXML` → la respuesta XML a la que se le pueden aplicar métodos del estándar DOM





# El objeto XMLHttpRequest invoca la función callback y procesa su resultado



```
function callback() {
  if (req.readyState == 4) {
    if (req.status == 200) {
      // update the HTML DOM based on whether or not message is valid
      parseMessage();
    }
  }
}

function parseMessage() {
  var message = req.responseXML.getElementsByTagName("message")[0];
  setMessage(message.childNodes[0].nodeValue);
}

function setMessage(message) {
  var userMessageElement = document.getElementById("userIdMessage");
  userMessageElement.innerHTML = "<font color=\"red\">" + message + "
  </font>";
}
```





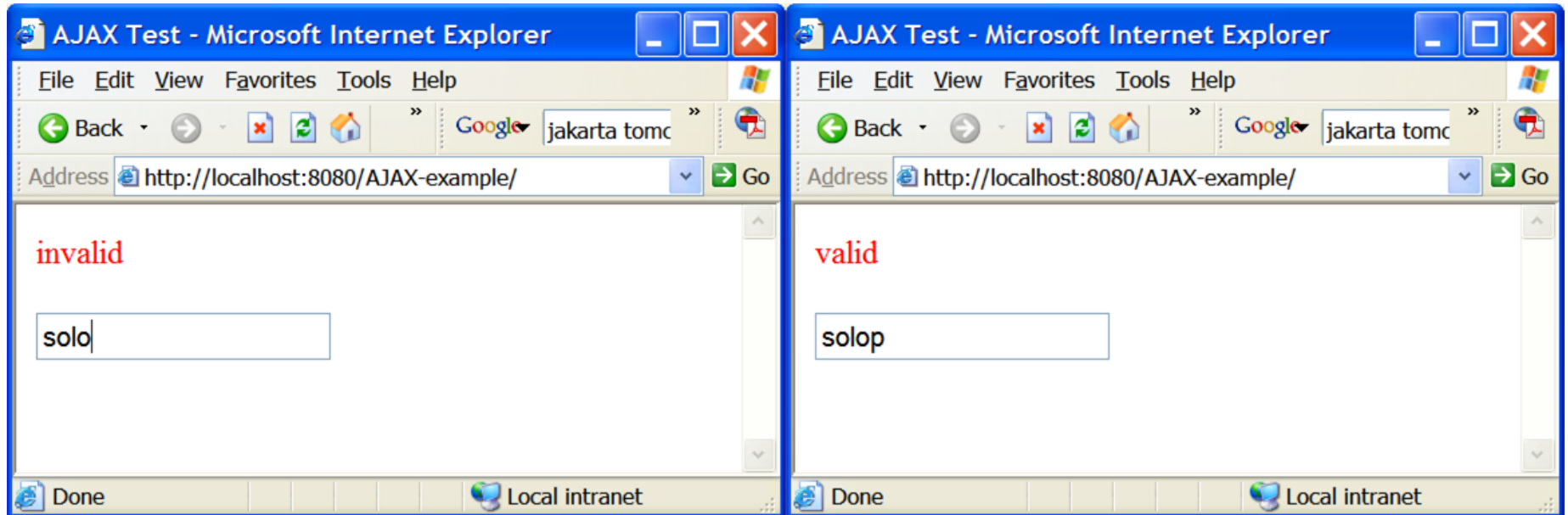
# El XHTML DOM es actualizado

- JavaScript puede recoger una referencia a un elemento en el HTML DOM
- Por ejemplo, a través de la invocación `document.getElementById("userIdMessage")`
- Con una referencia a un elemento:
  - Añadir, modificar o eliminar atributos
  - Añadir, modificar o eliminar elementos





# Aplicación en Ejecución





- jQuery es una librería JavaScript cuyo slogan es: "write less, do more"
  - Su propósito es simplificar el uso de JavaScript en tu web app
- Es por muchos considerada como la framework JavaScript más popular y extendida
  - Simplifica tareas comunes que requieren varias líneas de código JavaScript y permite invocarlas en una sola línea
    - Por ejemplo, llamadas a AJAX o manipulación DOM
- La librería jQuery contiene las siguientes características:
  - Manipulación HTML/DOM y CSS
  - Gestión de eventos HTML
  - Efectos y animaciones
  - Invocación remota asíncrona con AJAX
  - Otras utilidades
- Site: <http://jquery.com/>
  - Version actual 1.8.3



# Usando jQuery

- Para usarlo incluye una referencia a la librería jQuery de la página donde lo quieres usar
  - Descargable de:
    - <http://jquery.com/download>
  - Se recomienda revisar el ‘starter kit’ que podemos encontrar en nuestro disco duro:
    - <http://docs.jquery.com/mw/images/c/c7/Jquery-starterkit.zip>
- También puedes acceder siempre a un CDN (Content Delivery Network) que cachea de modo escalable y distribuido copias de la librería
  - Google y Microsoft alojan jQuery. Por ejemplo,

```
<script  
src="http://ajax.googleapis.com/ajax/libs/jquery/1.  
8.0/jquery.min.js">  
</script>
```

# jQuery Syntax



- Con jQuery seleccionas (query) los elementos HTML y ejecutas "actions" sobre ellos
- Basado en la sintaxis: `$ (selector) .action ()`
  - Un `$` para definir/acceder a la librería jQuery
  - Un `(selector)` para "consultar (o seleccionar)" elementos HTML
  - Una jQuery `action ()` a ser ejecutada sobre los elementos
- Ejemplos:
  - `$(this).hide ()` – esconde el elemento actual
  - `$("p").hide ()` – esconde todos los elementos `<p>`
    - Selector de elementos de un tipo
  - `$(".test").hide ()` – esconde todos los elementos con atributo `class="test"`.
    - Selector de elementos con una clase concreta
  - `$("#test").hide ()` – esconde el elemento con `id="test"`
    - Selector de elemento concreto
- jQuery define el evento `ready` del documento para prevenir que el código jQuery se ejecute antes de haber cargado completamente el documento

```
$(document).ready(function() {  
    // jQuery methods go here...  
});
```



# Hola Mundo en jQuery



- Copiar a `index.html` el siguiente código:

```
<html>
  <head>
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      // The following is a shortcut for the
      $(document).ready(callback)
      // $(function() { // code to execute when the DOM is ready });
      $(document).ready(function() {
        $("a").click(function() { alert("Hello world!"); });
      });
    </script>
  </head>
  <body>
    <a href="">Link</a>
  </body>
</html>
```

- Recuerda que la sentencia `$(document).ready` espera a que el DOM esté listo

- Sin jQuery se podría hacer del siguiente modos:

```
<a href="" onclick="alert('Hello World!')">Link</a>
```



# Usando selectores

- Dos enfoques para seleccionar elementos:
  1. Una combinación de selectores CSS y XPath pasados como strings al constructor de jQuery (e.g. `$("#div > ul a")`)
    - CSS selectors: [http://docs.jquery.com/DOM/Traversing/Selectors#CSS\\_Selectors](http://docs.jquery.com/DOM/Traversing/Selectors#CSS_Selectors)
    - Xpath selectors: [http://docs.jquery.com/DOM/Traversing/Selectors#XPath\\_Selectors](http://docs.jquery.com/DOM/Traversing/Selectors#XPath_Selectors)
  2. Diferentes métodos del objeto jQuery: `$("#orderedlist").find("li")`
    - API: <http://api.jquery.com/>

| Goal                   | CSS 3               | XPath                       |
|------------------------|---------------------|-----------------------------|
| All Elements           | *                   | //*                         |
| All P Elements         | p                   | //p                         |
| All Child Elements     | p > *               | //p/*                       |
| Element By ID          | #foo                | //*[@id='foo']              |
| Element By Class       | .foo                | //*[contains(@class,'foo')] |
| Element With Attribute | *[title]            | //*[@title]                 |
| First Child of All P   | p > *:first-child   | //p/*[0]                    |
| All P with an A child  | <i>Not possible</i> | //p[a]                      |
| Next Element           | p + *               | //p/following-sibling::*[0] |



# Ejemplos selectores

- Documentación detallada de los diferentes tipos de selectores puede ser encontrada en:
  - <http://docs.jquery.com/DOM/Traversing/Selectors>
- Para seleccionar un elemento `<ul>` con el ID "orderedlist":
  - En JavaScript clásico: `document.getElementById("orderedlist")`
  - En jQuery: `$("#orderedlist").addClass("red");`
    - Añade un fondo rojo a la lista identificada con `orderedlist`
- Para añadir clases a los elementos hijo de una lista:  
`$("#orderedlist > li").addClass("blue");`
- Para añadir y quitar la clase cuando el usuario mueve el ratón sobre el elemento `li`, pero sólo en el último elemento de la lista:

```
$("#orderedlist li:last").hover(function() {  
    $(this).addClass("green");  
},function(){  
    $(this).removeClass("green");  
});
```

# Ejemplos selectores

- Iterar sobre los descendientes de un nodo y añadir una clase a ellos:

```
$("#orderedlist").find("li").each(function(i) {  
    $(this).append( " BAM! " + i );  
});
```

- Limpiar todos los formularios de la página cuando se haga click con el ID `reset`

```
$("#reset").click(function() {  
    $("form").each(function() {  
        this.reset();  
    });  
});
```

- Seleccionar sólo algunos elementos del grupo, `filter()` reduce la selección de los elementos que cumplen la expresión de filtro, `not()` hace lo contrario y elimina los elementos que cumplen la expresión:

- Todos los elementos `li` obtienen un borde, excepto aquellos que NO tienen un subelemento `ul`  
`$("li").not(":has(ul)").css("border", "1px solid black");`

- La sintaxis `[expression]` se toma de XPath y puede ser usada para filtrar atributos

- Seleccionar todos los elementos de tipo `<a>` que tienen un atributo `name` y cambiar su fondo:  
– `$("a[name]").css("background", "#eee" );`

# Ejemplos selectores

- Puedes seleccionar no sólo el elemento actualmente selecciona y sus hijos sino que también los padres y descendientes

- Por ejemplo, FAQ donde las respuestas están escondidas y se muestran cuando el usuario hace click sobre ellas:

```
$('#faq').find('dd').hide().end().find('dt').click(function() {  
    $(this).next().slideToggle();  
});
```

- El encadenado puede ser usado para reducir el tamaño del código y ganar más rendimiento, dado que '#faq' es seleccionado sólo una vez:

- Usando `end()`, el primer `find()` es deshecho, de modo que podemos lanzar otro `find` sobre `#faq` en vez de los elementos hijo `dd`:

- `$(this).next()` es usado para encontrar los siguientes hijos empezando por el elemento `dt`:

```
$('#faq').find('dd').hide().end().find('dt').click(function() {  
    $(this).next().slideToggle();  
});
```

- Para seleccionar los elementos hijo (también conocidos como antecesores), puedes usar la expresión `parents`

- Selecciona el párrafo que es el padre del enlace sobre el que mueve el ratón el usuario:

```
$("a").hover(function() {  
    $(this).parents("p").addClass("highlight");  
},function() {  
    $(this).parents("p").removeClass("highlight");  
});
```

# Compendio selectores

| Syntax                                  | Description  |
|---|--|
| <code>\$("*")</code>                    | Selects all elements   |
| <code>\$(this)</code>                   | Selects the current HTML element   |
| <code>\$("#p.intro")</code>             | Selects all <code>&lt;p&gt;</code> elements with <code>class="intro"</code>  |
| <code>\$("#p:first")</code>             | Selects the first <code>&lt;p&gt;</code> element   |
| <code>\$("#ul li:first")</code>         | Selects the first <code>&lt;li&gt;</code> element of the first <code>&lt;ul&gt;</code>                                 |
| <code>\$("#ul li:first-child")</code>   | Selects the first <code>&lt;li&gt;</code> element of every <code>&lt;ul&gt;</code>                                     |
| <code>\$("#[href]")</code>              | Selects all elements with an <code>href</code> attribute   |
| <code>\$("#a[target='_blank']")</code>  | Selects all <code>&lt;a&gt;</code> elements with a target attribute value equal to <code>"_blank"</code>               |
| <code>\$("#a[target!='_blank']")</code> | Selects all <code>&lt;a&gt;</code> elements with a target attribute value NOT equal to <code>"_blank"</code>           |
| <code>\$("#:button")</code>             | Selects all <code>&lt;button&gt;</code> elements and <code>&lt;input&gt;</code> elements of <code>type="button"</code> |
| <code>\$("#tr:even")</code>             | Selects all even <code>&lt;tr&gt;</code> elements  |
| <code>\$("#tr:odd")</code>              | Selects all odd <code>&lt;tr&gt;</code> elements   |



# Gestionando eventos

- Los gestores de eventos son métodos llamados cuando “algo ocurre” en HTML
- Por cada evento `onxxx` disponible, como `onclick`, `onchange`, `onsubmit`, hay un equivalente jQuery
  - Otros eventos como `ready` y `hover`, son provistos como métodos convenientes para algunas tareas
    - Para un listado completo, revisar:  
<http://api.jquery.com/category/Events/>

# Gestionando eventos

- Es común poner código jQuery en el gestor de eventos en la sección `<head>`
  - Por ejemplo: una función es invocada cuando el evento `click` del botón es lanzado

```
<head>
<script src="jquery.js"></script>
<script>
$ (document) .ready (function () {
    $("button").click(function () {
        $("p").hide ();
    });
});
</script>
</head>
```

- Es conveniente colocar las funciones en ficheros aparte
  - Si tu portal tiene muchas páginas y quieres hacer que tus funciones jQuery sean fáciles de mantener, pon tu código en un fichero `.js` aparte

```
<head>
<script src="jquery.js"></script>
<script src="my_jquery_functions.js"></script>
</head>
```

# Ejemplos gestores de eventos

| Event Method                                  | Description  |
|---|--|
| <code>\$(document).ready(function)</code>     | Binds a function to the <code>ready</code> event of a document (when the document is finished loading) |
| <code>\$(selector).click(function)</code>     | Triggers, or binds a function to the <code>click</code> event of selected elements                     |
| <code>\$(selector).dblclick(function)</code>  | Triggers, or binds a function to the <code>double click</code> event of selected elements              |
| <code>\$(selector).focus(function)</code>     | Triggers, or binds a function to the <code>focus</code> event of selected elements                     |
| <code>\$(selector).mouseover(function)</code> | Triggers, or binds a function to the <code>mouseover</code> event of selected elements                 |

# Efectos en jQuery

- Permite Esconder, Mostrar, Intercambiar, Desplazar, Difuminar o Animar elementos HTML
  - Documentación en: <http://docs.jquery.com/Effects>
  - Con jQuery, puedes esconder y mostrar elementos HTML con los métodos `hide()` y `show()`, respectivamente:
    - Sintaxis: `$(selector).hide(speed, callback)`, donde la velocidad puede ser:
      - `slow`, `fast` o `milliseconds`
      - `callback` es la función a ejecutar tras el `hide`

```
$("#hide").click(function() {  
    $("p").hide();  
});
```

- Con jQuery, puedes también cambiar de visible a oculto con el método `toggle()`

- Por ejemplo:

```
$("#button").click(function() {  
    $("p").toggle();  
});
```

# Ejemplo efectos en jQuery

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <script src="http://code.jquery.com/jquery-latest.js"></script>
  <script>
    $(document).ready(function() {
      $("button").click(function () {
        $("p").toggle();
      });
    });
  </script>
</head>
<body>
  <button>Toggle</button>
  <p>Hello</p>
  <p style="display: none">Good Bye</p>
</body>
</html>
```



# Más efectos en jQuery

- Para hacer desaparecer y emerger elementos se pueden usar:
  - jQuery `fadeIn()` para ocultar un elemento progresivamente:
    - `$(selector).fadeIn(speed,callback);`
  - jQuery `fadeOut()` para visualizar un elemento.
  - jQuery `fadeToggle()` para cambiar el estado entre `fadeIn()` y `fadeOut()`
  - The jQuery `fadeTo()` method allows fading to a given opacity (value between 0 and 1).
- Con jQuery, puedes también definir un efecto de desplazamiento de controles con:

```
$(selector).slideDown(speed,callback):  
$("#flip").click(function() {  
    $("#panel").slideDown();  
});
```

- jQuery `slideUp()` es usado para hacer emerger un elemento
- jQuery `slideToggle()` para cambiar la dirección entre `slideDown()` y `slideUp()`.
- jQuery `animate()` permite crear animaciones personalizadas.
- Ejemplos y documentación adicional en:

[http://www.w3schools.com/jquery/jquery\\_ref\\_effects.asp](http://www.w3schools.com/jquery/jquery_ref_effects.asp)



# Métodos para manipular HTML/CSS



- jQuery contiene varios métodos para cambiar y manipular los elementos y atributos HTML
  - Una parte importante de jQuery es la posibilidad de manipular DOM
    - `text()` – asocia o devuelve texto contenido en los elementos seleccionados
    - `html()` – asocia o devuelve el contenido HTML de los elementos seleccionados
    - `val()` – asocia o devuelve el valor de los campos de un formulario
  - Ejemplo:

```
$("#btn1").click(function(){
    alert("Text: " + $("#test").text());
});
$("#btn2").click(function(){
    alert("HTML: " + $("#test").html());
});
```
  - Los tres métodos mencionados vienen con funciones de callback:
    - Reciben dos parámetros: el índice del elemento actual en los elementos seleccionados y el valor original
    - Devuelven el string que quiere asociarse como valor desde la función



# Métodos para manipular HTML/CSS



- Con jQuery es fácil añadir elementos/contenido:
  - `append()` – inserta contenido al final de los elementos seleccionados
    - `$("#p").append("Some appended text.");`
  - `prepend()` – inserta contenido antes de los elementos seleccionados
    - `$("#p").prepend("Some prepended text.");`
  - `after()` – inserta contenido después del elemento indicado
    - `$("#img").after("Some text after");`
  - `before()` – inserta contenido antes del elemento indicado
- Con jQuery es fácil eliminar elementos HTML
  - `remove()` – elimina el elemento seleccionado y sus hijos
    - `$("#div1").remove();`
  - `empty()` – elimina los hijos del elemento seleccionado
    - `$("#div1").empty();`
    - `$("#p").remove(".italic"); // filter the elements to be removed`





# Métodos para manipular HTML/CSS



- jQuery tiene varios métodos para manipulación CSS:
  - `addClass()` – añade una o más clases a los elementos seleccionados
  - `removeClass()` – elimina una o más clases de los elementos seleccionados
    - `$("#h1,h2,p").removeClass("blue");`
  - `toggleClass()` – cambia de añadir/eliminar clases de los elementos seleccionados
    - `$("#h1,h2,p").toggleClass("blue");`
  - `css()` – asigna o devuelve el estilo de un atributo
    - `$("#p").css("background-color"); // gets`
    - `$("#p").css("background-color", "yellow");`

- Ejemplo:

- CSS:

```
.important {  
    font-weight:bold;  
    font-size:xx-large;  
}
```

```
.blue {  
    color:blue;  
}
```

- Código jQuery:

```
$("#button").click(function(){  
    $("#h1,h2,p").addClass("blue");  
    $("#div").addClass("important");  
});
```



# Dimensiones jQuery

- Con jQuery es posible trabajar con dimensiones de elementos y de la ventana del navegador:
  - `height()` – asigna o recupera la altura de los elementos seleccionados
    - `$("#div").height("150px");`
  - `width()` - asigna o recupera la anchura de los elementos seleccionados
  - `innerHeight()` – devuelve la altura de un elemento (incluye relleno pero no borde)
  - `innerWidth()` - devuelve la anchura de un elemento (incluye relleno pero no borde)
  - `outerHeight()` – devuelve la altura de un elemento (incluye relleno y borde)
  - `outerWidth()` - devuelve la anchura de un elemento (incluye relleno y borde)

# jQuery AJAX

- jQuery ofrece una librería de métodos para desarrollo AJAX = Asynchronous JavaScript and XML.
  - Permite crear páginas web rápidas y dinámicas
    - Es posible actualizar parte de una página
- Con jQuery AJAX, puedes solicitar texto, HTML, XML o JSON de un servidor remoto mediante HTTP GET y HTTP POST
  - **Y puedes cargar datos remotos directamente en los elementos HTML seleccionados de tu página web!**
- Documentación: <http://api.jquery.com/category/ajax/>



# jQuery AJAX

- Método jQuery `load()`
  - Es una función AJAX que carga datos del servidor y los coloca en el elemento AJAX seleccionado
  - Sintaxis: `$(selector).load(URL,data,callback);`
    - El parámetro opcional `data` indica los datos a enviar al servidor junto con la petición.
    - El parámetro `callback` es el nombre de la función a ser ejecutado después que la petición es completada.
  - Ejemplo: `$("#div").load('test1.txt');`
- Método jQuery `$.ajax()`
  - Documentación: <http://api.jquery.com/jQuery.ajax/>
  - Ejecuta una petición AJAX
  - Sintaxis: `$.ajax({name:value, name:value, ... })`
  - Ofrece más funcionalidad que `load()`, `get()`, y `post()`, pero es más complicada de usar:
    - El parámetro `options` toma pares `name:value` definiendo la URL, contraseñas, tipos de datos, filtros, conjuntos de caracteres, timeout y funciones de error
    - Por ejemplo,:

```
$.ajax({url:"test1.txt",success:function(result){
    $("#div").html(result);
}});
```
- Una referencia completa a los métodos de AJAX puede encontrarse en:  
[http://www.w3schools.com/jquery/jquery\\_ref\\_ajax.asp](http://www.w3schools.com/jquery/jquery_ref_ajax.asp)

# Opciones de \$.ajax()

| Name                                  | Value/Description  |
|---------------------------------------|--|
| async                                 | A Boolean value indicating whether the request should be handled asynchronous or not. Default is true                                  |
| beforeSend( <i>xhr</i> )              | A function to run before the request is sent   |
| cache                                 | A Boolean value indicating whether the browser should cache the requested pages. Default is true                                       |
| complete( <i>xhr, status</i> )        | A function to run when the request is finished (after success and error functions).  |
| contentType                           | The content type used when sending data to the server. Default is: "application/x-www-form-urlencoded"                                 |
| context                               | Specifies the "this" value for all AJAX related callback functions   |
| data                                  | Specifies data to be sent to the server  |
| dataFilter( <i>data, type</i> )       | A function used to handle the raw response data of the XMLHttpRequest  |
| dataType                              | The data type expected of the server response.   |
| error( <i>xhr, status, error</i> )    | A function to run if the request fails.  |
| global                                | A Boolean value specifying whether or not to trigger global AJAX event handles for the request. Default is true                        |
| ifModified                            | A Boolean value specifying whether a request is only successful if the response has changed since the last request. Default is: false. |
| jsonp                                 | A string overriding the callback function in a jsonp request   |
| jsonpCallback                         | Specifies a name for the callback function in a jsonp request  |
| password                              | Specifies a password to be used in an HTTP access authentication request.  |
| processData                           | A Boolean value specifying whether or not data sent with the request should be transformed into a query string. Default is true        |
| scriptCharset                         | Specifies the charset for the request  |
| success( <i>result, status, xhr</i> ) | A function to be run when the request succeeds   |
| timeout                               | The local timeout (in milliseconds) for the request  |
| traditional                           | A Boolean value specifying whether or not to use the traditional style of param serialization  |
| type                                  | Specifies the type of request. (GET or POST)   |
| url                                   | Specifies the URL to send the request to. Default is the current page  |
| username                              | Specifies a username to be used in an HTTP access authentication request   |
| xhr                                   | A function used for creating the XMLHttpRequest object   |

# Usando AJAX

- Imagina un site en el que podemos valorar cosas, hecho en PHP:

```
$(document).ready(function() {  
    // generate markup  
    var ratingMarkup = ["Please rate: "];  
    for(var i=1; i <= 5; i++) { ratingMarkup[ratingMarkup.length] = "<a href='#'>" + i  
+ "</a> "; }  
    var container = $("#rating"); // add markup to container  
    container.html(ratingMarkup.join('')); // add click handlers  
    container.find("a").click(function(e) {  
        e.preventDefault(); // send requests  
        $.post("rate.php", {rating: $(this).html()}, function(xml) { // format result  
            var result = [ "Thanks for rating, current average: ",  
                $("average", xml).text(),  
                ", number of votes: ",  
                $("count", xml).text()  
            ]; // output result  
            $("#rating").html(result.join(''));  
        } );  
    });  
});
```

# Usando AJAX

- El código servidor en PHP sería:

```
/* This snippet generates five anchor elements and appends them to the container element with the id "rating". Afterwards, for every anchor inside the container, a click handler is added. When the anchor is clicked, a post request is send to rate.php with the content of the anchor as a parameter. The result returned as a XML is then added to the container, replacing the anchors.*/
```

```
<?php  
define('STORE', 'ratings.dat');  
function put_contents($file,$content) {  
    $f = fopen($file,"w");  
    fwrite($f,$content);  
    fclose($f);  
}  
  
if(isset($_REQUEST["rating"])) {  
    $rating = $_REQUEST["rating"];  
    $storedRatings = unserialize(file_get_contents(STORE));  
    $storedRatings[] = $rating;  
    put_contents(STORE, serialize($storedRatings));  
    $average = round(array_sum($storedRatings) / count($storedRatings), 2);  
    $count = count($storedRatings);  
    $xml = "<ratings><average>$average</average><count>$count</count></ratings>";  
    header('Content-type: text/xml');  
    echo $xml;  
}  
?>
```

# jQuery UI



- jQuery UI es una librería de widgets e interacción construida encima de la librería JavaScript jQuery
  - Puede ser usada para construir aplicaciones web muy interactivas
- El siguiente site ofrece demostraciones de controles provistos en forma de Interacciones, Widgets y Efectos
  - <http://jqueryui.com/demos/>
- La herramienta jQuery UI's download builder permite generar una librería personalizada de componentes a descargar
  - <http://jqueryui.com/download/>
    - El resultado será un fichero `jquery-ui-1.8.24.custom.min.js`
    - La aplicación ThemeRoller permite personalizar tus controles:
      - <http://jqueryui.com/themeroller/>
- Documentación: [http://docs.jquery.com/UI/Getting\\_Started](http://docs.jquery.com/UI/Getting_Started)





# Usando jQuery UI

- Después de descargar jQuery UI se obtiene .zip que incluye demos, documentación y librerías
  - Puedes revisar sus contenidos para aprender jQuery UI

- Uso:

1. Incluir 3 ficheros para usar jQuery UI widgets e interactions:

```
<link type="text/css" href="css/themename/jquery-ui-1.8.24.custom.css" rel="Stylesheet" />
```

```
<script type="text/javascript" src="js/jquery-1.8.2.min.js"></script>
```

```
<script type="text/javascript" src="js/jquery-ui-1.8.24.custom.min.js"></script>
```

2. Añadir jQuery widgets a tu página

- Para añadir un widget datepicker, coloca un elemento de entrada de texto a tu página y luego invoca sobre él .datepicker();

```
<input type="text" name="date" id="date" />  
$('#date').datepicker();
```

# Customizando jQuery UI

- jQuery UI puede ser customizado en diferentes modos:
  1. El download builder te permite crear una copia de jQuery UI que incluye sólo lo que deseas
  2. Puedes diseñar un jQuery UI theme con la aplicación ThemeRoller app:  
<http://jqueryui.com/themeroller/>
  3. Cada plugin en jQuery UI tiene una configuración por defecto para los casos de uso más comunes
    - Podemos sobrescribir las configuraciones por defecto usando "options".
    - Ejemplo: el widget slider tiene una opción de orientación que permite especificar si el slider debería ser horizontal o vertical

```
$('#mySliderDiv').slider({  
  orientation: 'vertical',  
  min: 0,  
  max: 150,  
  value: 50 });
```

- Documentación jQuery UI: <http://jqueryui.com/demos/>
- Demos de jQuery UI: <http://jqueryui.com/demos/>

# jQuery Mobile UI

- jQuery Mobile es a los móviles lo que jQuery UI a la sobremesa
  - jQuery mobile es una framework primero para móviles pero también funciona para ayudar a construir apps desde navegadores web
    - Las plataformas soportadas pueden revisarse en: <http://jquerymobile.com/gbs/>
- Es una “Touch-Optimized Web Framework for Smartphones & Tablets”
  - Una interfaz HTML5 para dispositivos móviles construida sobre jQuery y jQuery UI
    - Resuelve los problemas que encuentras entre navegadores de diferentes positivos cuando despliegas una aplicación web
      - Última versión estable: 1.2.0
      - Site: <http://jquerymobile.com>
- Todas las páginas en jQuery Mobile se construyen sobre el principio “**clean, semantic HTML**” para garantizar la compatibilidad con cualquier dispositivo web
  - Hace uso del atributo **data-\*** para asignar roles a los elementos
    - Más info en: <http://html5doctor.com/html5-custom-data-attributes/>

# Características

- **Built on jQuery core** for familiar and consistent jQuery syntax and minimal learning curve and leverages jQuery UI code and patterns.
- **Compatible with all major mobile, tablet, e-reader & desktop platforms** - iOS, Android, Blackberry, Palm WebOS, Nokia/Symbian, Windows Phone 7, MeeGo, Opera Mobile/Mini, Firefox Mobile, Kindle, Nook, and all modern browsers with graded levels of support.
- **Lightweight size** and minimal image dependencies for speed.
- **Modular architecture** for creating custom builds that are optimized to only include the features needed for a particular application
- **HTML5 Markup-driven configuration** of pages and behavior for fast development and minimal required scripting.
- **Progressive enhancement** approach brings core content and functionality to all mobile, tablet and desktop platforms and a rich, installed application-like experience on newer mobile platforms.
- **Responsive design** techniques and tools allow the same underlying codebase to automatically scale from smartphone to desktop-sized screens
- **Powerful Ajax-powered navigation system** to enable animated page transitions while maintaining back button, bookmarking and and clean URLs though `pushState`.
- **Accessibility** features such as WAI-ARIA are also included to ensure that the pages work for screen readers (e.g. VoiceOver in iOS) and other assistive technologies.
- **Touch and mouse event support** streamline the process of supporting touch, mouse, and cursor focus-based user input methods with a simple API.
- **Unified UI widgets** for common controls enhance native controls with touch-optimized, themable controls that are platform-agnostic and easy to use.
- **Powerful theming framework** and the ThemeRoller application make highly-branded experiences easy to build.



# Elementos de jQuery Mobile

- jQm nos ofrece un conjunto notable de controles con estilos que pueden ser fácilmente modificables
  - Las listas se crean como siempre en HTML con el añadido del atributo `data-role` asignado al valor **listview**

```
<ul data-role="listview" data-inset="true">  
  <li><a href="#">Acura</a></li>  
  <li><a href="#">Audi</a></li>  
</ul>
```
  - También se proporcionan diferentes controles para formularios:
    - Campos de entrada, text areas, sliders, toggle switches, checkboxes y botones de radio buttons usados como siempre dentro de `<form>`
      - Los botones se pueden crear de modo fácil asignado `data-role="button"` a enlaces o botones
  - El estilo asociado a cada elemento puede ser modificado a través del atributo `data-theme`

# ¿Cómo usar jQuery Mobile?

- jQuery Mobile es muy fácil de usar, solamente es necesario incluir los ficheros de jQuery Mobile en tu cabecera y añadir algunos atributos a tu marcado
  - Todo el “estilado” es hecho por jQuery a través del fichero CSS

- A continuación, el proceso para usar jQm:

- **Paso 1. Incluir los ficheros jQuery Mobile Files en la cabecera**

- Puedes descargar jQuery Mobile de su sitio o cargar los documentos de su CDN, lo cual reduce su ancho de banda y mejora la velocidad del site

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0a1/jquery.mobile-1.0a1.min.css" />
```

```
<script type="text/javascript" src="http://code.jquery.com/jquery-1.4.3.min.js"></script>
```

```
<script type="text/javascript" src="http://code.jquery.com/mobile/1.0a1/jquery.mobile-1.0a1.min.js"></script>
```

# ¿Cómo usar jQuery Mobile?

## – Paso 2. Usa el atributo data en el código HTML

- Aplica el atributo data-n a tu HTML, donde n es la característica a añadir, por ejemplo:

```
<div data-role="content">
  <ul data-role="listview" data-inset="true" data-theme="c" data-
  dividertheme="a">
    <li data-role="list-divider">Transition Effects</li>
    <li><a href="effects.php?id=slide" data-
  transition="slide">Slide</a></li>
    <li><a href="effects.php?id=slideup" data-
  transition="slideup">Slide Up</a></li>
    <li><a href="effects.php?id=slidedown" data-
  transition="slidedown">Slide Down</a></li>
    <li><a href="effects.php?id=pop" data-
  transition="pop">Pop</a></li>
    <li><a href="effects.php?id=flip" data-
  transition="flip">Flip</a></li>
    <li><a href="effects.php?id=fade" data-
  transition="fade">Fade</a></li>
  </ul>
</div>
```

# ¿Cómo usar jQuery Mobile?

- » **data-role** – especifica la naturaleza del elemento contenedor: `page`, `header`, `content`, `footer`.
  - » **data-position** – especifica si el elemento debería ser *fijo*, en cuyo caso se mostrará en la parte superior (para la header) o parte inferior (for footer)
  - » **data-inset** – especifica si el elemento debería estar dentro los márgenes del contenido o fuera (flush or with margin) – debe tener los valores *true* o *false*
  - » **data-transition** – especifica qué transición utilizar para cargar nuevas páginas, puede ser asociado a los valores: `slide`, `slideup`, `slidedown`, `pop`, `flip` o `fade`
  - » **data-theme** – especifica qué diseño de tema usar para los elementos dentro del contenedor: `a`, `b`, `c`, `d`, `e`
  - » **data-dividertheme** – especifica el tema para los divisores de lista usando las mismas opciones que `data-theme`
- **Paso 3. Añade contenido a una plantilla de aplicación**
    - Usando los atributos en el elemento HTML 5, usando bien la plantilla [jquerymobile.com](http://jquerymobile.com) o tu propio diseño, puebla los bloques con otros contenidos y sube los ficheros
      - » jQueryMobile template: <http://jquerymobile.com/demos/1.0a1/#docs/pages/docs-pages.html>
  - **Paso 4. Añade meta etiquetas viewport**
  - **Paso 5. Aplicación simple o multi-página**
    - Usa para ello `data-role="page"`.



# Ejemplo de aplicación jQuery Mobile App



## 1. Crear un plantilla básica

### – En el head:

- Una meta-etiqueta `viewport` asigna el tamaño de la pantalla (anchura, altura) al tamaño del dispositivo
- Añade enlaces a jQuery, jQuery Mobile y el tema de estilo móvil (stylesheet) del CDN.

```
<meta name="viewport" content="width=device-width, initial-scale=1"> <link  
rel="stylesheet" href="http://code.jquery.com/mobile/1.0.1/jquery.mobile-  
1.0.1.min.css" />
```

```
<script src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
```

```
<script src="http://code.jquery.com/mobile/1.0.1/jquery.mobile-  
1.0.1.min.js"></script>
```

### – En el body, un elemento `div` con un `data-role` de `page` es el wrapper usado para demarcar la página y la barra de cabecera (`data-role="header"`) y contenidos (`data-role="content"`) se añaden dentro para incluir páginas básicas

- Estos atributos HTML5 `data-` son usados por jQuery Mobile para transformar markup básico en un estilo de widgets mejorado

```
<div data-role="page">  
  <div data-role="header">  
    <h1>My Title</h1>  
  </div><!-- /header -->  
  <div data-role="content">  
    <p>Hello world</p>  
  </div><!-- /content -->  
</div><!-- /page -->
```

## 2. Dentro de tu contenedor de contenidos, puedes añadir elementos HTML estándar como cabeceras, listas, párrafos, etc.



# Ejemplo de aplicación

## jQuery Mobile App



### 3. Hagamos un listview

```
<ul data-role="listview" data-inset="true" data-filter="true">
  <li><a href="#">Acura</a></li>
  <li><a href="#">Audi</a></li>
  <li><a href="#">BMW</a></li>
  <li><a href="#">Cadillac</a></li>
  <li><a href="#">Ferrari</a></li>
</ul>
```

### 4. Ahora coloquemos un slider. jQm tiene diversos **form elements** que están mejorados para dispositivos táctiles. Aquí tenemos un slider hecho con el nuevo tipo de entrada HTML5 range, no es necesario usar data-role

```
<form>
  <label for="slider-0">Input slider:</label>
  <input type="range" name="slider" id="slider-0" value="25" min="0" max="100" />
</form>
```

### 5. Aprende a hacer un botón de comienzo con un enlace y añade el atributo data-role="button" al mismo

- Puedes añadir un icono con el atributo data-icon y opcionalmente establecer su posición con el atributo data-iconpos

```
<a href="#" data-role="button" data-icon="star">Star button</a>
```

### 6. jQuery Mobile tiene una framework de temas robusta que soporta hasta 26 estilos para elementos o "swatch".

- Simplemente añade un atributo data-theme="e" a cualquier elemento de la página y su apariencia cambiará

```
<a href="#" data-role="button" data-icon="star" data-theme="a">Button</a>
```

# CouchDB

- CouchDB es una base de datos open source orientada a documentos, accesible mediante una API RESTful que hace uso extensivo de JavaScript Object Notation (JSON)
  - **"Couch"** es el acrónimo de **"Cluster Of Unreliable Commodity Hardware"**
    - Su misión es ser muy escalable, con alta disponibilidad y robustez, incluso cuando se ejecuta en hardware convencional
  - Creada como "database of the Web"

*"Django may be built for the Web, but CouchDB is built of the Web. I've never seen software that so completely embraces the philosophies behind HTTP. CouchDB makes Django look old-school in the same way that Django makes ASP look outdated."*

—Jacob Kaplan-Moss, Django developer

# Características de CouchDB

- CouchDB es una base de datos orientada a documentos JSON escrita en Erlang.
  - Parte de la generación de bases de datos NoSQL
  - Es un proyecto open source de la fundación Apache
- Es altamente concurrente, diseñada para ser replicada horizontalmente, a través de varios dispositivos y tolerante a fallos.
- Permite a las aplicaciones guardar documentos JSON a través de una interfaz RESTful
- Utiliza map/reduce para indexar y consultar la base de datos
- Permite escribir una aplicación cliente que habla directamente vía HTTP con CouchDB sin necesidad de una capa servidora intermedia
- Guarda datos en local en la propia máquina cliente para reducir latencia
  - Gestiona la replicación a la nube por ti

# CouchDB: BBDD orientada a documentos



- Una BBDD *document-oriented* está compuesta de una serie de documentos
  - Son libres de esquema; no existe un esquema definido a priori, antes de usar la BBDD
    - Si un documento necesita un nuevo campo, puedes incluirlo, sin afectar a otros documentos en la BBDD
- CouchDB no tiene una funcionalidad de auto-increment o secuencia
  - Asigna un Universally Unique Identifier (UUID) a cada documento, haciendo casi imposible que otra base de datos seleccione el mismo identificador
- No soporta JOINS como las bases de datos relacionales
  - Una característica denominada *vista* permite crear relaciones arbitrarias entre documentos que no son definidas en las propias bases de datos.
- CouchDB ofrece una alternativa a todos aquellos proyectos donde un modelo orientado a documentos encaja mejor que una base de datos relacional: wikis, blogs y sistemas de gestión documental



# Ventajas de CouchDB

- **Documentos JSON** – todo lo que se guarda en CouchDB son simplemente documentos JSON.
- **Interfaz RESTful** – desde la creación a la replicación a la inserción de datos, toda la gestión de datos en CouchDB puede ser realizada vía HTTP.
- **Replicación N-Master** – puedes hacer uso de un número ilimitado de ‘masters’, dando lugar a topologías de replicación muy interesantes.
- **Escrita para ejecutarse offline** – CouchDB puede replicarse en dispositivos (e.j. teléfonos Android) que pueden quedarse sin conexión y gestionar sincronización de datos cuando el dispositivo está online de nuevo
- **Filtros de replicado** – puedes filtrar de modo preciso los datos que quieres replicar a distintos nodos.
  - [http://wiki.apache.org/couchdb/Replication#Filtered\\_Replication](http://wiki.apache.org/couchdb/Replication#Filtered_Replication)

# Conceptos clave en CouchDB:

## Documentos

- Una base de datos en CouchDB es una **colección de documentos**, donde cada uno está identificado por un ID y contiene un conjunto de campos nombrados:
  - Los campos pueden ser strings, números, fechas o incluso listas ordenadas y diccionarios.
  - Ejemplos de documentos serían:

```
"Subject": "I like Plankton",  
"Tags": ["plankton", "baseball",  
"decisions"]
```

# Conceptos clave en CouchDB:

## Documentos

- Las BBDD CouchDB guardan documentos nombrados de modo unívoco y proporcionan una API RESTful JSON que permite a las aplicaciones leer y modificar estos documentos
  - Cada documento puede tener campos no definidos en otros documentos:
    - Los documentos no están asociados a un esquema de bases de datos estricto
- Cada documento contiene metadatos (datos sobre datos) como el **identificador unívoco del documento** (`id`) y su **número de revisión** (`rev`)
- Los campos de un documento pueden ser de varios tipos como strings, números, booleanos, colecciones, etc.
- Cuando se hacen cambios sobre un documento CouchDB se crea una nueva versión del documento, denominado revisión
  - Se mantiene un historial de modificaciones gestionado automáticamente por la BBDD
- CouchDB no dispone de mecanismos de bloqueo (locking) ante escrituras



# Conceptos clave en CouchDB:



## Vistas

- Son el mecanismo para añadir estructura a datos semi-estructurados
- El modelo de vistas en CouchDB usa JavaScript para describirlas
- Las vistas son el método para agregar y realizar informes sobre los documentos de un repositorio, siendo creados en demanda para agregar y agregar documentos.
- Las vistas se construyen dinámicamente y no afectan el documento subyacente, puedes tener tantas representaciones de vistas de los mismos datos como gustes.



# Conceptos clave en CouchDB:

## Vistas

- CouchDB es desestructurado en naturaleza, adolece de un esquema estricto pero provee beneficios en términos de flexibilidad y escalabilidad, explotar sus datos en aplicaciones reales a veces puede hacerse complicado
  - Los datos se guardan en un espacio de almacenamiento plano, algo así como un repositorio de datos desnormalizados.
  - Proporciona un modelo de vistas para añadir estructura a los datos de modo que pueda agregarse para añadir significado útil
- Las vistas se crean en demanda y son utilizadas para agregar, enlazar y reportar sobre documentos en la base de datos
  - Se definen en documentos de diseño y pueden ser replicadas a través de varias instancias
  - Estos documentos de diseño contienen funciones JavaScript que pueden ejecutar consultas mediante el concepto de MapReduce.
    - La **función Map** de la vista recibe un documento como argumento y realiza una serie de cálculos para determinar qué datos deberían ser disponibles en la vista
    - Si la vista tiene una **función Reduce**, es usada para agregar los resultados. A partir de un conjunto de pares clave/valor devuelve un sólo valor.



# Ejemplo de Vista en CouchDB

```
map: function(doc) {  
  if (doc._attachments) {  
    emit("with attachment", 1);  
  }  
  else {  
    emit("without attachment", 1);  
  }  
}
```

```
reduce: function(keys, values) {  
  return sum(values);  
}
```

# Conceptos clave: Distribuida

- CouchDB es un sistema distribuido de base de datos basado en nodos
  - Un número variable de nodos CouchDB (servidores y clientes offline) pueden tener “copias de réplicas” independientes de la misma BBDD, donde las aplicaciones pueden tener interactividad completa con la BBDD (consultar, añadir, editar y borrar)
    - Cuando vuelven a estar online o de modo planificado, los cambios de las bases de datos son replicados bidireccionalmente.
- CouchDB tiene gestión de conflictos incorporada de serie, haciendo que el proceso de replicación sea incremental y rápido, copiando sólo documentos y campos individuales modificados desde la última replicación.
  - Utiliza Multi-Version Concurrency Control (MVCC)

# Detalles técnicos

- Un servidor CouchDB gestiona bases de datos bajo un nombre, que almacenan **documentos**:
  - Cada documento tiene un nombre único en la BBDD y CouchDB proporciona una API HTTP RESTful para leer y modificar (añadir, editar y borrar) documentos de la BBDD.
  - Los documentos son la unidad de datos primaria en CouchDB y consisten de un número variable de campos y adjuntos
  - Las modificaciones sobre documentos (añadir, editar, borrar) son del todo o de nada, o se modifican completamente o fallan completamente.
  - El modelo de modificación de documentos de CouchDB es optimista y no hace uso de locks.
  - Más detalles genéricos en:  
<http://CouchDB.apache.org/docs/overview.html>



# Modelo de Vistas

- Para añadir estructura a datos no estructurados o semi-estructurados, CouchDB incorpora el modelo de vistas
  - Se crean dinámicamente y no afectan al documento subyacente
  - Se definen dentro de documentos de diseño
  - Se replican a otras instancias de la base de datos como si fueran documentos convencionales
    - En CouchDB sólo se replican datos, aunque esos datos a menudo (código JavaScript) puede corresponder a aplicaciones.
- Para garantizar un alto rendimiento, el motor de vistas mantiene índices de sus vistas e incrementalmente las actualiza para reflejar los cambios en la base de datos.

# Map/Reduce en CouchDB

- Usar Map/Reduce tiene ventajas sobre consultas SQL porque pueden ser distribuidas entre varios nodos, algo que no puede hacerse con RDBMS.
- Las bases de datos NoSQL utilizan map/reduce para consultar e indexar la BBDD
  - `map` consiste en extraer los datos a procesar
  - `reduce` se centra en la agregación de los mismos.

# Instalación y Gestión de CouchDB

- Download CouchDB 1.2 de <http://couchdb.apache.org/>
  - En Windows, descarga el fichero .mdi y sigue los pasos del Wizard.
- Futon: [http://localhost:5984/\\_utils/](http://localhost:5984/_utils/)
  - Hace uso internamente de la librería [http://127.0.0.1:5984/\\_utils/script/jquery.couch.js](http://127.0.0.1:5984/_utils/script/jquery.couch.js)
  - Lo primero que hay que hacer es hacer click en Fix Me para asegurarnos que sólo usuarios autorizados pueden acceder a CouchDB
    - Nosotros usaremos la combinación `admin/enpresadigitala`



# La API RESTful JSON

- CouchDB ofrece una API como mecanismo para recuperar datos de una BBDD.
  - Donde siguiendo la convención REST: (si no aparece la sabes para crear POST y sino PUT)
    - **POST** – crea un nuevo registro
    - **GET** – lee registros
    - **PUT** – actualiza un registro
    - **DELETE** – borra un registro
- Esta API es accesible vía HTTP GET y POST y retorna datos en el formato de objetos JavaScript mediante JSON.
  - Una ventaja de este enfoque es que puede usarse una framework AJAX como Prototype o jQuery para crear una aplicación web, sin necesidad de hacer uso de un lenguaje de parte servidora
  - La herramienta de línea de comando CURL pueden ser usada como cliente de línea de comandos HTTP:
    - Descargable de: <http://curl.haxx.se/>
    - Permite realizar peticiones GET, POST, PUT, y DELETE, mostrando la respuesta HTTP recibida del servidor web

# Probando la API RESTful de CouchDB con curl



- `$ curl http://127.0.0.1:5984/`
  - Respuesta: `{"couchdb": "Welcome", "version": "1.2.0"}`
- O explícitamente define el tipo de petición realizada a través del parámetro `-X` de curl:
  - `$ curl -X GET http://127.0.0.1:5984/_all_dbs`
    - Respuesta:  
`["_replicator", "_users", "nerekurtsoak", "test_suite_reports", "testdb", "users"]`
- Para crear dos nuevas bases de datos, ejecutaríamos los comandos:
  - `$ curl -uadmin:enpresadigitala -X PUT http://127.0.0.1:5984/fruit`
    - Respuesta: `{"ok": true}`
  - `$ curl -uadmin:enpresadigitala -X PUT http://127.0.0.1:5984/vegetables`
- Si ejecutamos ahora:
  - `$ curl -X GET http://127.0.0.1:5984/_all_dbs`
  - Obtendríamos:  
`["_replicator", "_users", "fruit", "nerekurtsoak", "test_suite_reports", "testdb", "users"]`
- Si intentamos volver a crear una BBDD ya existente, recibimos un error:
  - `$ curl -X PUT http://127.0.0.1:5984/fruit`
    - Respuesta: `{"error": "file_exists", "reason": "The database could not be created, the file already exists."}`

# Probando la API RESTful de CouchDB con curl



- Podemos borrar una base de datos con el siguiente comando:
  - `$ curl -uadmin:enpresadigitala -X DELETE http://127.0.0.1:5984/vegetables`
  - Respuesta: `{"ok":true}`
- Para crear un documento:
  - `$ curl -uadmin:enpresadigitala -X PUT http://127.0.0.1:5984/fruit/apple -H "Content-Type: application/json" -d {}`
  - Respuesta: `{"ok":true,"id":"apple","rev":"1-967a00dff5e02add41819138abb3284d"}`
- Para recuperarlo:
  - `$ curl -X GET http://127.0.0.1:5984/fruit/apple`
  - Respuesta: `{"_id":"apple","_rev":"1-967a00dff5e02add41819138abb3284d"}`
- Para recuperar información de la BBDD:
  - `$ curl -X GET http://127.0.0.1:5984/fruit`
  - Respuesta:  
`{"db_name":"fruit","doc_count":1,"doc_del_count":0,"update_seq":1,"p  
urge_seq":0,  
"compact_running":false,"disk_size":4179,"instance_start_time":"1321  
991208171560","disk_format_version":5,"committed_update_seq":1}`

# Programando CouchDB

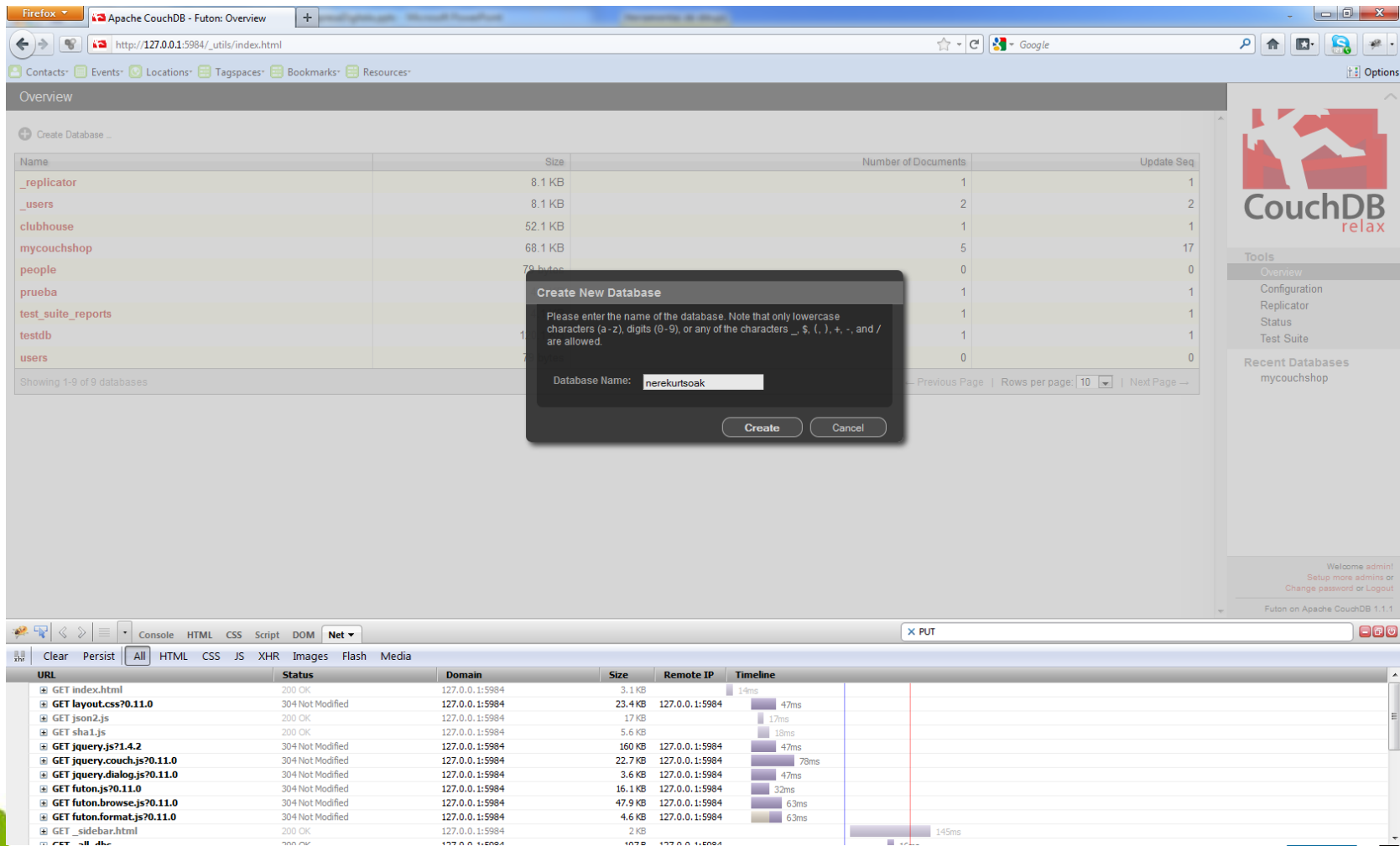
- Gracias a la API RESTful, los desarrolladores pueden conectarse a CouchDB usando cualquier software que soporte HTTP
- La mayoría de los lenguajes modernos ofrecen algún tipo de interfaz HTTP, implicando que CouchDB puede ser usada en cualquier proyecto de desarrollo.
  - Revisar la siguiente página para diferentes clientes programáticos a CouchDB:
    - [http://wiki.apache.org/CouchDB/Related\\_Projects](http://wiki.apache.org/CouchDB/Related_Projects)

# Primeros pasos con CouchDB

- Vamos a seguir el tutorial en:  
<http://net.tutsplus.com/tutorials/getting-started-with-CouchDB/>
  - Asegúrate de crear una cuenta de `admin/enpresadigitala`
- Otros tutoriales:
  - <http://www.catswhocode.com/blog/getting-started-with-CouchDB-tutorial-a-beginners-guide>



# Primeros pasos en CouchDB



The screenshot displays the Apache CouchDB Futon web interface. A modal dialog titled "Create New Database" is open, asking for a database name. The name "nerekurtsok" is entered in the text field. The dialog also includes instructions: "Please enter the name of the database. Note that only lowercase characters (a-z), digits (0-9), or any of the characters \_ \$ ( ) + , - and / are allowed." Below the text field are "Create" and "Cancel" buttons.

In the background, the "Overview" page shows a table of databases:

| Name               | Size     | Number of Documents | Update Seq |
|--------------------|----------|---------------------|------------|
| _replicator        | 8.1 KB   | 1                   | 1          |
| _users             | 8.1 KB   | 2                   | 2          |
| clubhouse          | 52.1 KB  | 1                   | 1          |
| mycouchshop        | 68.1 KB  | 5                   | 17         |
| people             | 70 bytes | 0                   | 0          |
| prueba             |          | 1                   | 1          |
| test_suite_reports |          | 1                   | 1          |
| testdb             |          | 1                   | 1          |
| users              |          | 0                   | 0          |

At the bottom of the browser window, a network log is visible, showing various GET requests and their response times.

# Primeros Pasos en CouchDB

- En Futon:
  1. Crearemos la base de datos haciendo click en “Create Database”, de nombre "nerekurtsiak"
  2. Hacer click en “New Document”
    - Vete añadiendo campos a través de “ADD Field”
      - "name" : "Curso NoSQL"
      - "price" : 30
    - Guarda cada campo
    - Guarda el documento
  3. Modificar el documento ya existente añadiendo el campo "type" con valor "course"
    - La nueva versión del documento debería empezar por 2



# Primeros Pasos en Curl

1. Creemos un documento `persona.json` con el siguiente contenido:

```
{  
  "forename": "Diego",  
  "surname":  "Lopez-de-Ipina",  
  "type":     "person"  
}
```

2. Usamos el siguiente comando de CURL para subir el documento a CouchDB:

```
- curl -X POST http://127.0.0.1:5984/nerekurtsoak/ -d  
  @persona.json -H "Content-Type: application/json"
```

- Se devolvería:

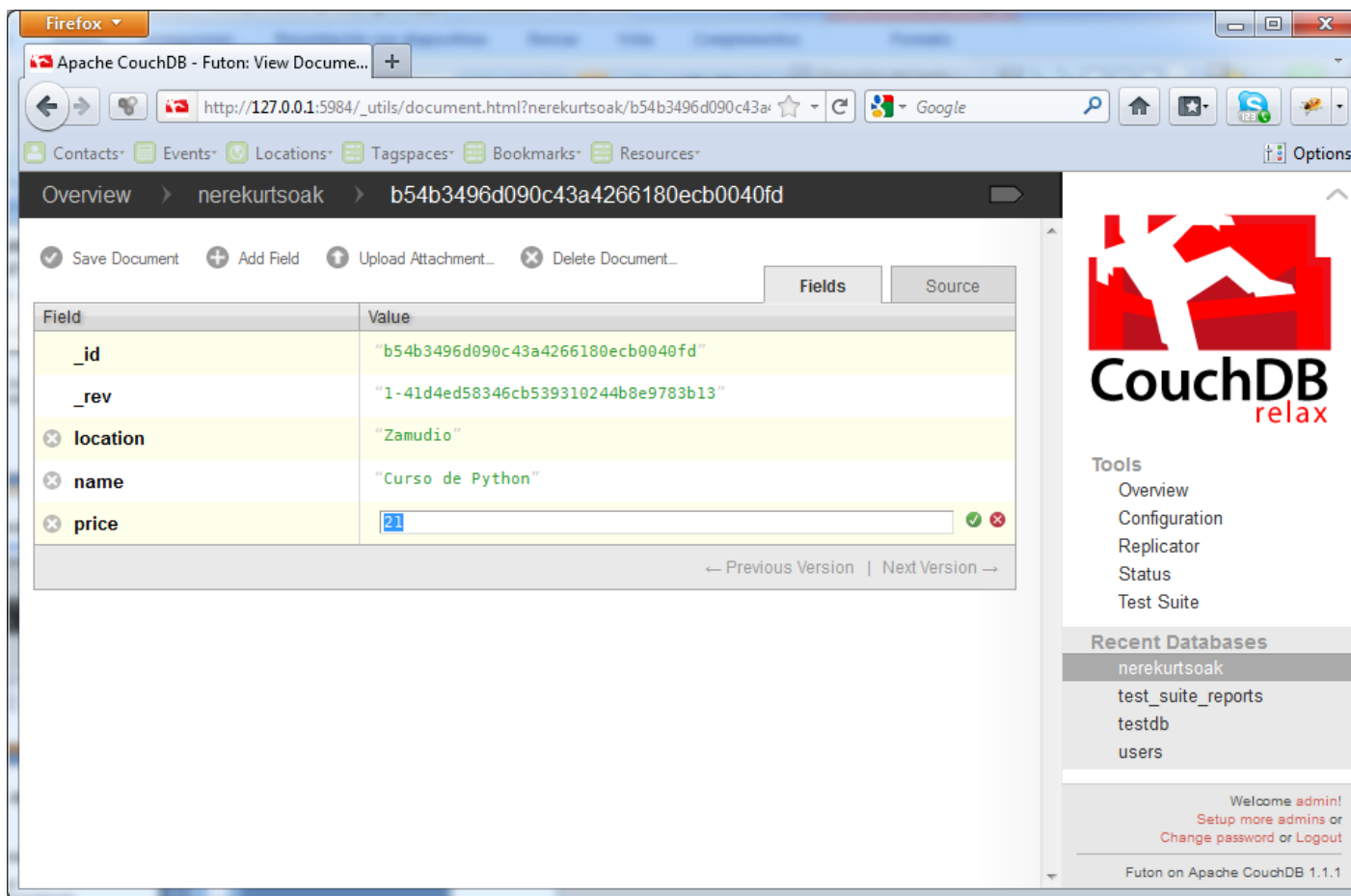
- {"ok":true,"id":"b54b3496d090c43a4266180ecb002a92","rev":"1-93c73d298af443f623db6861f90e9f6e"}

3. Para recuperar todos los documentos:

```
- curl -X GET http://127.0.0.1:5984/nerekurtsoak/_all_docs
```



# Vista de un documento en Futon



The screenshot shows the Apache CouchDB Futon interface in a Firefox browser window. The address bar shows the URL: `http://127.0.0.1:5984/_utils/document.html?nerekurtsoak/b54b3496d090c43a4266180ecb0040fd`. The breadcrumb navigation shows: Overview > nerekurtsoak > b54b3496d090c43a4266180ecb0040fd. The document is displayed in a table with the following fields and values:

| Field                 | Value                                |
|-----------------------|--------------------------------------|
| <code>_id</code>      | "b54b3496d090c43a4266180ecb0040fd"   |
| <code>_rev</code>     | "1-41d4ed58346cb539310244b8e9783b13" |
| <code>location</code> | "Zamudio"                            |
| <code>name</code>     | "Curso de Python"                    |
| <code>price</code>    | 21                                   |

On the right side, there is a sidebar with the CouchDB relax logo, a 'Tools' menu (Overview, Configuration, Replicator, Status, Test Suite), and a 'Recent Databases' list (nerekurtsoak, test\_suite\_reports, testdb, users). At the bottom of the sidebar, it says 'Welcome admin! Setup more admins or Change password or Logout' and 'Futon on Apache CouchDB 1.1.1'.

# Creando una función Map

- Seleccionar Temporary View en the Drop View dentro de Futon
  - La función map tendría el siguiente código:

```
function (doc) {  
    if (doc.type == "course" && doc.name) {  
        emit(doc.name, doc);  
    }  
}
```

- Accede a ella a través del navegador como:
  - [http://127.0.0.1:5984/\\_utils/database.html?nerekurtsoak/\\_design/courses/view/courses](http://127.0.0.1:5984/_utils/database.html?nerekurtsoak/_design/courses/view/courses)

# Creando un Reduce

- Nos aseguraremos de tener al menos dos cursos con precio
- Definir la función de mapeo como:

```
function (doc) {  
  if (doc.type === "course" && doc.price) {  
    emit(doc.id, doc.price);  
  }  
}
```

- Definir la función reduce como:

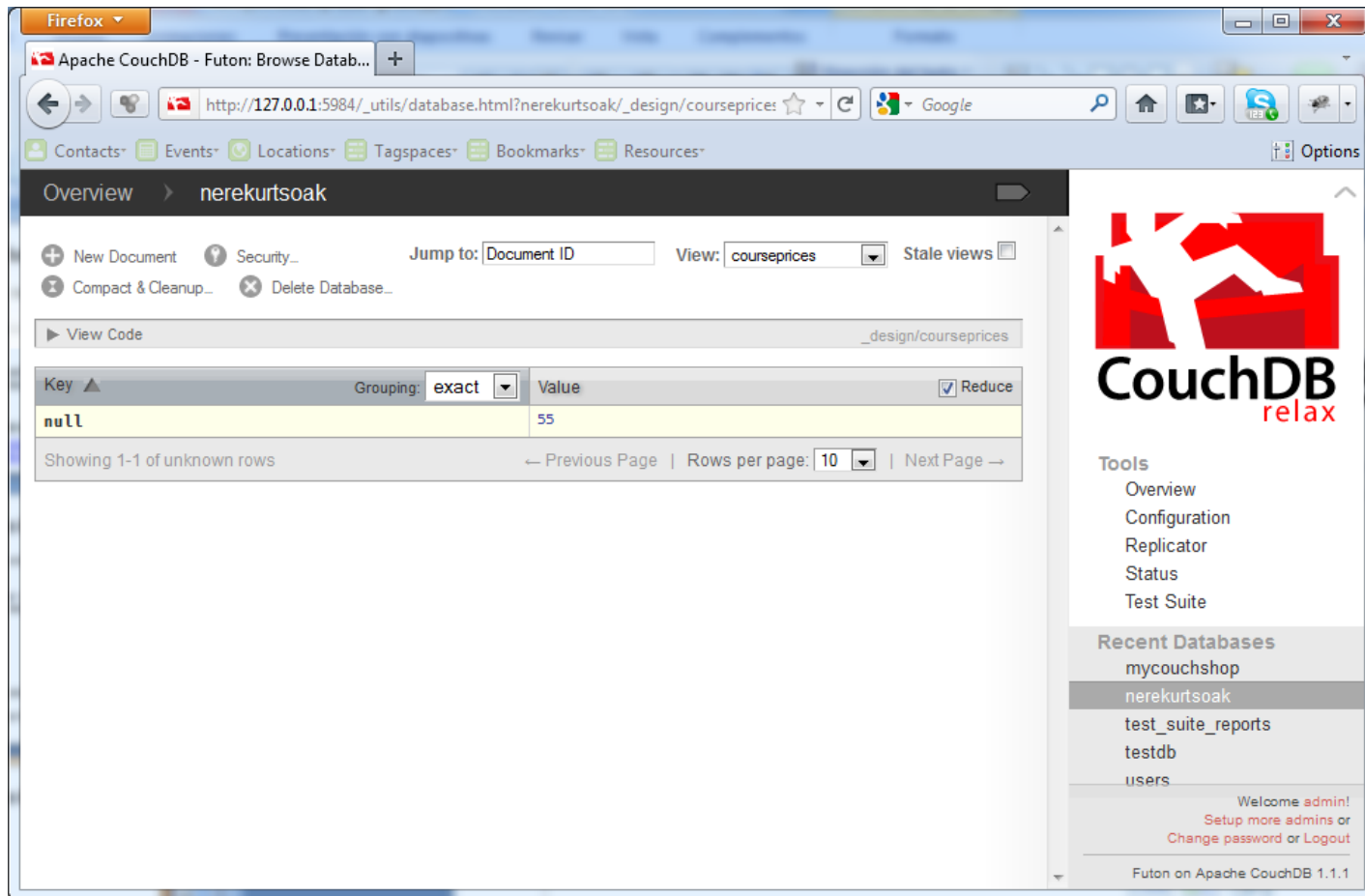
```
function (keys, prices) {  
  return sum(prices);  
}
```

- Guardarlo como `coursePrices`, asegurarse de hacer click en Reduce

- Ir a

[http://127.0.0.1:5984/\\_utils/database.html?nerekurtsoak/\\_design/courseprices/\\_view/courseprices](http://127.0.0.1:5984/_utils/database.html?nerekurtsoak/_design/courseprices/_view/courseprices)

# Creando un Reduce



The screenshot shows the Apache CouchDB Futon interface in a Firefox browser window. The address bar shows the URL: `http://127.0.0.1:5984/_utils/database.html?nerekurtoak/_design/courseprices`. The interface displays the 'Overview' page for the 'nerekurtoak' database. The 'View' is set to 'courseprices'. The main content area shows a table with one row of data:

| Key  | Value |
|------|-------|
| null | 55    |

The table has a 'Grouping' dropdown set to 'exact' and a 'Reduce' checkbox checked. Below the table, it says 'Showing 1-1 of unknown rows'. The right sidebar contains the CouchDB logo, a 'Tools' menu with options like Overview, Configuration, Replicator, Status, and Test Suite, and a 'Recent Databases' list including 'mycouchshop', 'nerekurtoak', 'test\_suite\_reports', 'testdb', and 'users'. At the bottom of the sidebar, it says 'Welcome admin!' and 'Futon on Apache CouchDB 1.1.1'.

# Documentos de Diseño en CloudDB



- Los documentos de diseño son un tipo especial de documento en CouchDB que contiene código de aplicación:
  - Vistas MapReduce, validaciones, funciones `show`, `list` y `update`
- Se suele crear un documento de diseño por cada aplicación
- El documento de diseño es un documento CouchDB con un ID que comienza con `_design/`:
  - Se replica como otros documentos en la BBDD y soporta gestión de conflictos a través del parámetro `rev`
    - CouchDB mira las vistas y otras funciones de aplicación en él
    - Los contenidos estáticos de la aplicación aparecen como `_attachments` en el documento de diseño

# Estructura interna de un Documento de Diseño

- Están compuestos de:
  - Funciones de validación
  - Definición de vistas
  - Funciones show, list y update
  - Attachments
- Una BBDD CouchDB puede tener varios documentos de diseño.
  - `_design/calendar`
  - `_design/contacts`
- Para recuperar un documento de diseño haz un GET con el siguiente patrón de URL:
  - `http://localhost:5984/mydb/_design/calendar`
  - `http://127.0.0.1:5984/mydb/_design/contacts`

```
1 {
2   "_id": "_design/sofa", ← Determines the app URL
3   "_rev": "3157636749",
4
5   "language": "javascript", (for the web)
6
7   "validate_doc_update": "function (newDoc, oldDoc, userCtx) { ... }",
8                               Application is stored as JSON data
9
10  "views": { ← Views field stores incremental
11             comments: {map reduce functions
12             "map": "function(doc) { ... };",
13             "reduce": "function(keys, values, rereduce) { ... };",
14             }
15  },
16
17  "shows": { ← Shows functions transform
18             documents into any format
19             "post": "function(doc, req) { ... }"
20  },
21
22  "_attachments": { ← Attachments show
23                   up as stubs
24                   "jquery.couchapp.js": {
25                     "stub": true,
26                     "content_type": "text/javascript",
27                     "length": 7539
28                   }
29  },
30
31  "signatures": { ← CouchApp traces attachments here
32                 for faster deployments
33                 "jquery.couchapp.js": "80078849ad6ca281f6993bd012c708f5",
34  },
35
36  "lib": { ← CouchApp can include
37           library code and data
38           in your functions
39           "templates": {
40             "post": "<!DOCTYPE html> ... </html>"
41           }
42  }
43 }
```

# Vistas: funciones map

- Las funciones vistas en CouchDB son strings guardados en el campo `views` del documento de diseño
- Los resultados de un vista se guardan en un B-tree, al igual que todo documento
  - Estos árboles permiten realizar búsquedas de filas por clave y recuperar rangos de filas
- Todas las funciones `map` tienen un único parámetro `doc`:

```
function (doc) {  
  if(doc.date && doc.title) {  
    emit(doc.date, doc.title);  
  }  
}
```

- Las funciones `map` son funciones libres de efectos laterales que toman un documento como argumento y emiten pares clave/valor:
  - Generan una lista ordenada por la clave de las filas
- Tenemos varios parámetros para recuperar resultados:
  - Una sola fila: `_design/docs/_view/date?key="2009/01/30 18:04:11"`
  - Varias filas: `_design/docs/_view/date?startkey="2010/01/01 00:00:00"&endkey="2010/02/00 00:00:00"&descending=true`

# Vistas: funciones `reduce`

- Las funciones `reduce` operan sobre las filas ordenadas emitidas por las funciones `map`
- Dada la manera en que los árboles B-tree están estructurados, podemos cachear los resultados intermedios de un `reduce` en los nodos no-hoja
- El formato de una función `reduce` en una vista es el siguiente:

```
function(keys, values, rereduce) {  
    return sum(values)  
}
```
- La función `reduce` se ejecuta sobre cada nodo del árbol para calcular el resultado final que es un valor escalar:
  - Cuando se ejecuta sobre las hojas del árbol (que contiene filas del mapa), el parámetro `rereduce` es `false`
  - Cuando se ejecuta en los nodos internos del árbol, el valor de `rereduce` es `true`



# Configuración de vistas

- Se definen en el documento de diseño como sigue:

```
{
  "_id": "_design/application",
  "_rev": "1-C1687D17",
  "views": {
    "viewname": {
      "map": "function(doc) { ... }",
      "reduce": "function(keys, values) { ... }"
    },
    "anotherview": {
      "map": "function(doc) { ... }",
      "reduce": "function(keys, values) { ... }"
    }
  }
}
```

- Para acceder a ellas se realiza un HTTP GET a  
`/database/_design/application/_view/viewname`



# Recetas: De SQL a MapReduce

- `SELECT field FROM table WHERE value="searchterm"`

– La función map sería:

```
function(doc) {  
    if(doc.age && doc.name) {  
        emit(doc.age, doc.name);  
    }  
}
```

– Y la consulta:

```
/ladies/_design/ladies/_view/age?key=5
```

- Documentación:

<http://guide.CouchDB.org/draft/cookbook.html>

# Seguridad en CouchDB

- CouchDB define autorización a través de diferentes tipos de usuarios:
  - Miembros de la BBDD, todo menos acceso a documentos de diseño
  - Administradores de la BBDD, todo menos crear y borrar bases de datos
  - Administradores del servidor, pueden hacer todo
    - [http://wiki.apache.org/couchdb/Security\\_Features\\_Overview](http://wiki.apache.org/couchdb/Security_Features_Overview)
- Funciones `validate_doc_update`, cada petición de crear o actualizar un documento pasa por esta función  
`function(newDoc, oldDoc, userCtx, secObj)`
  - Referencia: [http://wiki.apache.org/couchdb/Document\\_Update\\_Validation](http://wiki.apache.org/couchdb/Document_Update_Validation)
- Soporte nativo de SSL desde la versión 1.1.0
  - [http://wiki.apache.org/couchdb/How\\_to\\_enable\\_SSL](http://wiki.apache.org/couchdb/How_to_enable_SSL)



# Node: del client-side al server-side

- La mayoría de nosotros ha usado JavaScript como un mecanismo para añadir interactividad a las páginas web
- Con la aparición de jQuery, Prototype y otros nos hemos dado cuenta que es un lenguaje que sirve para algo más que hacer un `window.open()`
  - Sin embargo, todas estas innovaciones eran para JavaScript como lenguaje en la parte cliente
- **Node.js** nace para permitir usar JavaScript también en la parte servidora
  - Sin embargo, tu enfoque de desarrollo tiene que cambiar radicalmente
    - Va a estar basado en un **enfoque asíncrono guiado por eventos**



# Server-side JavaScript

- JavaScript es un lenguaje completo con el que puedes hacer cualquier cosa que puedes hacer con otros lenguajes
- Node.js te permite ejecutar código en el back-end, fuera del navegador
- Node.js utiliza Google V8 VM, el mismo entorno de ejecución para JavaScript de Chrome
  - Además incorpora varios módulos para no hacer todo de fuera, por tanto ...
    - **Node.js = entorno de ejecución + librería**

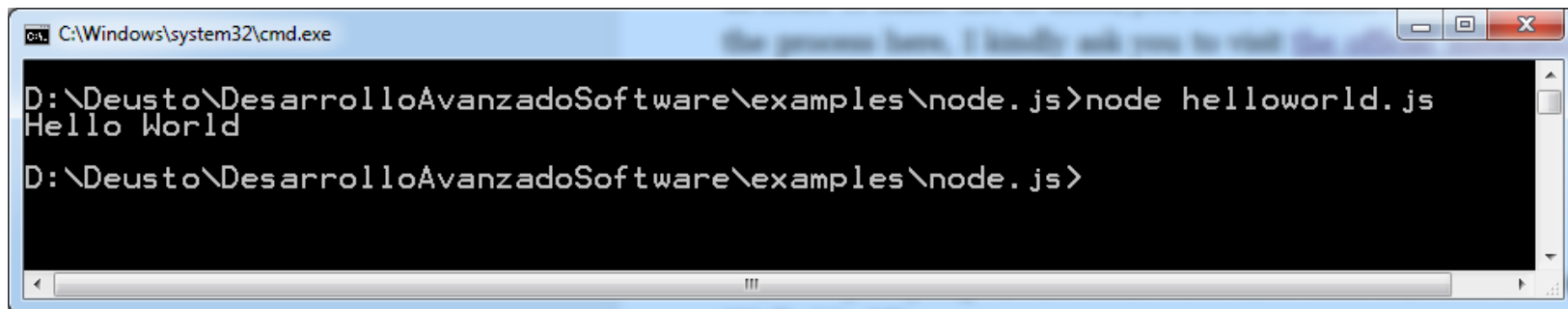


# Instalación de Node.js

- Los detalles de instalación para los diferentes sistemas operativos pueden encontrarse en:
  - <https://github.com/joyent/node/wiki/Installation>
- Para instalarlo en Windows simplemente elige el último msi para tu arquitectura x86 o x64 de:
  - <http://nodejs.org/dist/latest/>
    - Por ejemplo: [node-v0.8.8-x86.msi](#)
- Sigue los pasos del asistente de instalación para proceder a desplegar Node.js en tu sistema.
  - Por defecto, se instala en: `C:\Program Files (x86)\nodejs`
- Comprueba que está instalado ejecutando el comando `node` en línea de comandos
  - Si no funciona, modifica tu variable de entorno PATH para que apunte al ejecutable de `node`, en mi caso: `C:\Program Files (x86)\nodejs`

# “Hola Mundo” en Node.js

- Crea un fichero de nombre `helloworld.js` donde colocarás la siguiente sentencia que imprime "hello world" en consola:
  - `console.log("Hello World");`
- Guarda el fichero y ejecuta el comando:
  - `node helloworld.js`
- El resultado sería:



```
C:\Windows\system32\cmd.exe
D:\Deusto\DesarrolloAvanzadoSoftware\examples\node.js>node helloworld.js
Hello World
D:\Deusto\DesarrolloAvanzadoSoftware\examples\node.js>
```

- Documentación API node.js: <http://nodejs.org/docs/latest/api/>

# Estructura de la aplicación web en Node.js

- La aplicación a desarrollar consistirá de:
  1. Un servidor HTTP
  2. Un router que dependiendo en la URL de la petición dirigirá ésta a diferentes request handlers
  3. Un conjunto de gestores de peticiones que den respuesta a las peticiones solicitadas
  4. El router también tendrá que recoger y preparar los datos que recibe de un POST para remitírselos a los request handlers
  5. Cierta lógica de vista servirá para mostrar los resultados generados por los request handlers
  6. Lógica programática especial para gestionar el dominio de aplicación (por ejemplo, upload de ficheros)





# Creando una aplicación web para subir y visualizar ficheros

- Caso de uso:
  - La aplicación tendrá una página desde la que subir imágenes
  - La subida de un fichero dará lugar a otra página en la que se muestre la página subida
- Aparte de enseñar las APIs de Node.js para escribir una aplicación web, lo mas IMPORTANTE, aprenderemos ...
  - **cómo desarrollar aplicaciones totalmente asíncronas dirigidas por eventos**
    - Paradigma de programación que propone Node.js
    - Tenemos que cambiar radicalmente el enfoque síncrono de programación al que estamos acostumbrados todos nosotros.



# Paso 1. Creando el servidor web

- Algo peculiar de Node.js es que no sólo desarrollamos la aplicación web sino también el servidor web que la contiene
  - <http://thomashunter.name/blog/php-vs-nodejs/>
- En Node.js como con otros lenguajes de programación se puede dividir el código en diferentes módulos
  - Así crearemos un fichero principal que se invocará desde Node.js y un conjunto de módulos que podrán ser invocados desde él
    - Llamaremos `index.js` a nuestro fichero principal
    - `server.js` será el nombre del fichero conteniendo la lógica del servidor



# Paso 1. Creando el servidor web

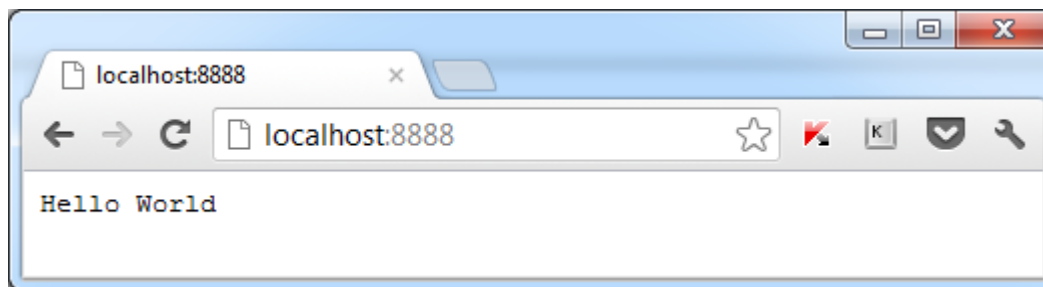
- Coloca el siguiente código en `server.js`:

```
var http = require("http");  
http.createServer(function(request, response) {  
  response.writeHead(200, {"Content-Type": "text/plain"});  
  response.write("Hello World");  
  response.end();  
}).listen(8888);
```

- Ejecuta el código con el siguiente comando y accede desde un navegador web a:

<http://localhost:8888>

```
node server.js
```



# Paso 1. Creando el servidor web: explicación del código

- La primera línea hace accesible a través de la variable `http`, el módulo homónimo distribuido de serie con Node.js
- La función `http.createServer` devuelve una instancia del objeto `http.Server` que tiene un método `listen` a través del que se puede indicar el puerto en el que se van a escuchar a peticiones web, en este caso 8888
- En una aplicación desarrollada con un código convencional haríamos código como el que sigue que quedaría bloqueado a la espera de peticiones:

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

```
var server = http.createServer();  
server.listen(8888);
```

- Sin embargo, en Node.js nutrimos a la función `createServer` con un parámetro que es una función JavaScript
  - Tal función será ejecutada cada vez que se reciba una petición HTTP por parte del servidor



# ¿Por qué complicarse la vida programando?: event-driven asynchronous callbacks

- El modelo de ejecución que subyace a Node.js es diferente al de Python, Ruby, PHP o Java
- La latencia de la siguiente operación tendría efectos diferentes en Node.js frente a otros lenguajes populares de desarrollo web:

```
var result = database.query("SELECT * FROM  
hugetable");  
console.log("Hello World");
```

- Aunque esta función tuviera una gran latencia en PHP no sería un grave problema, dado que el servidor web arranca un nuevo proceso por cada petición entrante
  - El retardo por una petición lenta lo experimentaría un único usuario
- En Node.js sólo hay un proceso, si se ejecuta una parte lenta afecta al proceso y por tanto a todos los clientes cuyas peticiones son servidas a través de este proceso
  - JavaScript y Node.js introducen el concepto de **callbacks asíncronas y dirigidas a eventos a través de un event loop**
    - Explicación: “Understanding the node.js event loop”

» <http://blog.mixu.net/2011/02/01/understanding-the-node-js-event-loop/>



# Event loops

- Para entender este concepto considera el siguiente código:

```
database.query("SELECT * FROM hugetable",  
function(rows) { var result = rows; });  
console.log("Hello World");
```

- Aquí le estamos pasando una función anónima al método `query` en vez de quedarnos bloqueados a que responda.
  - Node.js está compuesto por un conjunto de librerías asíncronas
    - En este caso coge la consulta, la envía a la BBDD y es avisado por el runtime cuando la ejecución iniciada concluye.
  - Node.js inicia tareas y asocia funciones que serán invocadas cuando el evento finalización de la operación sea desencolado de la cola de eventos interna de Node.js
    - Node itera continuamente sacando eventos de la cola y ejecutando el cuerpo de los eventos de callback



# Paso 1. Nuevo ejemplo del código del servidor web

- En el siguiente código se imprimiría “Server has started” antes de los otros mensajes de logeo

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

function onRequest(request, response) {
  console.log("Request received.");
  response.writeHead(200, {"Content-Type":
"text/plain"});
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);

console.log("Server has started.");
```



## Paso 2. Gestionando peticiones

- Cuando la callback es ejecutada nuestra función `onRequest()` es invocado, pasándosele dos parámetros:
  - `Request`: objeto que contiene todos los detalles de la petición entrante
  - `Response`: objeto que contiene los detalles de la salida de la petición
    - Los siguientes métodos sobre este objeto permiten modificar su estado:
      - `response.writeHead()` → permite escribir las cabeceras HTTP de la respuesta
      - `response.write()` → sirve para escribir la respuesta en el cuerpo de HTTP
      - `response.end()` → acaba la respuesta a la petición entrante





# Creando módulos en node.js

- En Node.js cuando usamos módulos externos, creamos una variable local con el nombre del modulo importado a través de la sentencia `require`:

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

```
...
```

```
http.createServer (...);
```

- Hacer que cierto código se transforme en un módulo implica exportar aquellas partes de su funcionalidad que queremos proveer a scripts que usen nuestro módulo

- Ejemplo:

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

```
function start() {  
  function onRequest(request, response) {  
    console.log("Request received.");  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
  }  
}
```

```
http.createServer(onRequest).listen(8888);  
console.log("Server has started.");  
}
```

```
exports.start = start;
```

- Y para usarlo desde `index.js` haríamos:

```
var server = require("./server");  
server.start();
```



# Paso 3: enrutado de peticiones

- Es decir, hacer que diferentes peticiones HTTP apunten a diferentes partes de nuestro código
  - Para ello, miraremos el contenido de la URL de la petición HTTP así como los parámetros GET y POST de las peticiones
    - Toda esta información la provee la clase `http.ServerRequest`
    - Además, para facilitar su procesamiento usaremos las clases `URL` y `querystring`
- Creamos un nuevo módulo `route` que será quien trabaje sobre el `path` de la petición y delegamos desde `server.js`, coordinando todo ello desde `index.js` a través de `dependency injection`

– En `route.js`:

```
function route(pathname) {
  console.log("About to route a request for " + pathname);
}
exports.route = route;
```

– En `server.js`:

```
var http = require("http");
var url = require("url");
function start(route) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
exports.start = start;
```

– En `index.js`:

```
var server = require("./server");
var router = require("./router");
server.start(router.route);
```

# Paso 4: creando los gestores de peticiones

- Para esta aplicación queremos gestionar el contexto `/start` and `/upload`, crearemos dos request handlers.
  - Creamos un nuevo módulo denominado `requesthandlers.js`
  - Para desacoplar totalmente los gestores de peticiones del código del router vamos también a usar dependency injection para especificar los mapeos entre contextos y funciones
    - *In JavaScript, objects are just collections of name/value pairs - think of a JavaScript object as a dictionary with string keys, where the values can perfectly be functions*

- Ejemplo:

- Código en `requesthandlers.js`

```
function start() {
  console.log("Request handler 'start' was called.");
}
function upload() {
  console.log("Request handler 'upload' was called.");
}
exports.start = start;
exports.upload = upload;
```

- Código en `index.js` para desacoplar los gestores del router:

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");
var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
server.start(router.route, handle);
```

# Paso 4: creando los gestores de peticiones

- Modificando `server.js` para que propague los gestores:

```
var http = require("http");
var url = require("url");
function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(handle, pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
exports.start = start;
```

- Cambiando `router.js` para que gestione cada contexto, determine si hay un handler asociado y en caso afirmativo lo solicite:

```
function route(handle, pathname) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname]();
  } else {
    console.log("No request handler found for " + pathname);
  }
}
exports.route = route;
```

# Operaciones bloqueantes y no bloqueantes en Node.js



- Si pusiéramos el siguiente código en `requesthandlers.js` lo que ocurriría es que cada petición a `start` bloquearía el servidor por 10s y otras peticiones no serían atendidas hasta que ésta acabara:

```
function start() {
  console.log("Request handler 'start' was called.");
  function sleep(milliSeconds) {
    var startTime = new Date().getTime();
    while (new Date().getTime() < startTime + milliSeconds);
  }
  sleep(10000);
  return "Hello Start";
}
function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello Upload";
}
exports.start = start;
exports.upload = upload;
```

- Sin embargo las operaciones no bloqueantes (por defecto) en Node.js también presentan problemas, ya que los resultados de las mismas se producen de modo asíncrono. En el siguiente ejemplo el resultado sería `empty` para la variable `content` y no el listado de ficheros que esperaríamos:

```
var exec = require("child_process").exec;
function start() {
  console.log("Request handler 'start' was called.");
  var content = "empty";
  exec("ls -lah", function (error, stdout, stderr) {
    content = stdout;
  });
  return content;
}
function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello Upload";
}
exports.start = start;
exports.upload = upload;
```



# Paso 5. Generando resultados en los requestHandlers



- La solución consiste en inyectar el objeto response de cada petición HTTP desde `server.js` hasta `requesthandlers.js`
- Ejemplo:

- Hacemos que `server.js` pase el objeto response al router:

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(handle, pathname, response);
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
```

```
exports.start = start;
```

- Y del router lo propagamos a los handlers:

```
function route(handle, pathname, response) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}
```

```
exports.route = route;
```

# Paso 5. Generando resultados en los requestHandlers



- Finalmente en `requestHandlers.js`, hacemos uso del modulo `child_process` que permite a través de `exec` ejecutar de modo asíncrono comandos del sistema o ejecutables:

```
var exec = require("child_process").exec;
function start(response) {
  console.log("Request handler 'start' was called.");
  exec("dir", function (error, stdout, stderr) {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write(stdout);
    response.end();
  });
}
function upload(response) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Upload");
  response.end();
}
exports.start = start;
exports.upload = upload;
```

```
localhost:8888/start
localhost:8888/start

El volumen de la unidad D es Data
El número de serie del volumen es: 70E8-2F8B

Directorio de D:\Deusto\DesarrolloAvanzadoSoftware\examples\node.js\fileuploadapp-step3

31/08/2012 09:59 <DIR> .
31/08/2012 09:59 <DIR> ..
31/08/2012 09:34 305 index.js
31/08/2012 10:00 556 requesthandlers.js
31/08/2012 10:00 415 router.js
31/08/2012 10:00 408 server.js
          4 archivos 1.684 bytes
          2 dirs 45.800.165.376 bytes libres
```

# Paso 6. Gestionando la subida de ficheros

- Seguiremos dos pasos:
  - Gestionaremos peticiones POST
  - Usaremos un módulo externo para gestionar uploads
- Para hacer que el proceso sea no bloqueante, Node.js sirve al código los datos del POST en pequeños fragmentos a través de callbacks que gestionan los eventos *data* (un nuevo fragmento de datos llega) y *end* (todos los fragmentos han sido recibidos).

```
request.addListener("data", function(chunk) {  
  // called when a new chunk of data was received  
});
```

```
request.addListener("end", function() {  
  // called when all chunks of data have been received  
});
```

- En el servidor recogeremos los fragmentos a través del callback para el evento *data* y llamaremos al router cuando recibamos el evento *end*
  - [http://nodejs.org/api/http.html#http\\_event\\_data](http://nodejs.org/api/http.html#http_event_data)
  - [http://nodejs.org/api/http.html#http\\_event\\_end](http://nodejs.org/api/http.html#http_event_end)



# Paso 6. Gestionando la subida de ficheros

```
var http = require("http");
var url = require("url");
function start(route, handle) {
  function onRequest(request, response) {
    var postData = "";
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");
    request.setEncoding("utf8");
    request.addListener("data", function(postDataChunk) {
      postData += postDataChunk;
      console.log("Received POST data chunk '" +
        postDataChunk + "'.");
    });
    request.addListener("end", function() {
      route(handle, pathname, response, postData);
    });
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
exports.start = start;
```

# Paso 6. Gestionando la subida de ficheros

- En `router.js` recibimos el `postData` capturado en `server.js` y lo propagamos a `requestHandlers.js`:

```
function route(handle, pathname, response, postData) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, postData);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}
exports.route = route;
```

# Paso 6. Gestionando la subida de ficheros

- Y ahora modificamos requesthandlers para que en la función `upload` devuelva como respuesta el contenido introducido en la caja de texto:

```
function start(response, postData) {
  console.log("Request handler 'start' was called.");
  var body = '<html>'+
    '<head>'+
    '<meta http-equiv="Content-Type" content="text/html; '+
    'charset=UTF-8" />'+
    '</head>'+
    '<body>'+
    '<form action="/upload" method="post">'+
    '<textarea name="text" rows="20" cols="60"></textarea>'+
    '<input type="submit" value="Submit text" />'+
    '</form>'+
    '</body>'+
    '</html>';
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write(body);
  response.end();
}

function upload(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  //response.write("You've sent: " + postData);
  response.write("You've sent the text: " + querystring.parse(postData).text);
  response.end();
}

exports.start = start;
exports.upload = upload;
```

# Paso 6. Gestionando la subida de ficheros

- Vamos ahora a habilitar la subida de ficheros de imágenes y su visualización como una página:
  - Para facilitar el proceso vamos a instalar y usar una librería externa, a través del comando: `npm install formidable@latest`
    - Más información sobre este módulo en: <https://github.com/felixge/node-formidable>
- Para poder mostrar el fichero subido haremos uso del módulo `fs` que nos permite renombrar y leer el contenido de ficheros
  - Por el camino cambiaremos la aplicación significativamente porque ahora necesitamos hacer llegar el objeto `request` a `requesthandlers`.
- Ejemplo:
  - Añadimos a `index.js` un nuevo par clave a función de gestión de petición (`show`):

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");
var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
handle["/show"] = requestHandlers.show;
server.start(router.route, handle);
```

# Paso 6. Gestionando la subida de ficheros

- En `requesthandlers.js`, en la función `upload` hacemos uso del módulo `formidable` para tratar los ficheros subidos y alojarlos en el directorio de trabajo actual:
  - `form.parse()` procesa la petición HTTP recibida
  - `fs.rename()` renombra el fichero subido a `test.png`
  - Se crea un enlace a una imagen asociada a la URL: `/show`, es decir, se invoca el handler `show`, que con la llamada `fs.readFile()` carga el contenido del fichero en la variable `file`

```
var querystring = require("querystring"),
    fs = require("fs"),
    formidable = require("formidable"),
    util = require('util');

function start(response) {
  console.log("Request handler 'start' was called.");
  var body = '<html>'+
    '<head>'+ '<meta http-equiv="Content-Type" '+ 'content="text/html; charset=UTF-8" />'+
  '</head>'+
    '<body>'+
    '<form action="/upload" enctype="multipart/form-data" '+ 'method="post">'+
    '<input type="file" name="upload" multiple="multiple">'+
    '<input type="submit" value="Upload file" />'+
    '</form>'+
    '</body>'+
    '</html>';
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write(body);
  response.end();
}
```

# Paso 6. Gestionando la subida de ficheros

```
function upload(response, request) {
  console.log("Request handler 'upload' was called.");

  var form = new formidable.IncomingForm();
  form.keepExtensions = true;
  form.uploadDir = process.cwd();
  console.log("about to parse");
  form.parse(request, function(error, fields, files) {
    console.log("parsing done");

    /* Possible error on Windows systems:
       tried to rename to an already existing file */
    fs.rename(files.upload.path, "test.png", function(err) {
      if (err) {
        fs.unlink("/tmp/test.png");
        fs.rename(files.upload.path, "test.png");
      }
    });
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write("received image:<br/>");
    response.write("<img src='/show' />");
    response.end();
    //response.end(util.inspect({fields: fields, files: files}));
  });
}
```

# Paso 6. Gestionando la subida de ficheros

```
function show(response) {
  console.log("Request handler 'show' was called.");
  fs.readFile("test.png", "binary", function(error, file) {
    if(error) {
      response.writeHead(500, {"Content-Type": "text/plain"});
      response.write(error + "\n");
      response.end();
    } else {
      response.writeHead(200, {"Content-Type": "image/png"});
      response.write(file, "binary");
      response.end();
    }
  });
}

exports.start = start;
exports.upload = upload;
exports.show = show;
```

# Final de la aplicación

- Hemos escrito una aplicación completa en Node.js, tratando los siguientes temas:
  - Server-side JavaScript
  - Functional programming
  - Blocking and non-blocking operations
  - Callbacks & events
  - Custom, internal and external modules
- Otros temas de interés serían:
  - Hablar con BBDD
  - Tests de unidad
  - Módulos externos instalables vía NPM
  - Alojamiento en la Cloud ...





# npm

- `npm` es el gestor de paquetes de Node.js
  - Puedes usarlo para instalar nuevos programas
  - Simplifica el linchado y especificación de dependencias
- Algunos comandos de ejemplo:
  - `npm help` – para obtener ayuda sobre esta utilidad
  - `npm install [<módulo>]` – instala la última dependencia de un módulo o sin argumento lo que le indique `package.json`
    - El fichero `package.json` instruye a `npm` qué hay que instalar para poder ejecutar exitosamente la aplicación
      - Para revisar el formato visita: <https://npmjs.org/doc/json.html>
  - `npm ls` – muestra qué hay en tu sistema instalado
- Documentación: <http://howtonode.org/introduction-to-npm>



# Sirviendo contenido estático y dinámico



- Buenos artículos explicando cómo hacerlo:
  - Fully Functional NodeJS static + Dynamic Server
    - <http://roadtobe.com/supaldubey/creating-fully-functional-nodejs-static-dynamic-server/>
  - Extending Our NodeJS static + Dynamic Server
    - <http://roadtobe.com/supaldubey/extending-our-nodejs-static-dynamic-server/>



# Express.js

- Express es una framework flexible y minimalista para el desarrollo de aplicaciones web en Node.js que permite escribir aplicaciones web sencillas y multi-página
  - Proporciona una capa ligera de características fundamentales para una aplicación web, todavía sin esconder capacidades avanzadas de Node.js
- URL:
  - Guía Express.js: <http://expressjs.com/guide.html>
  - Website: <http://expressjs.com/>
  - Documentación API: <http://expressjs.com/api.html>
- Express es distribuido a través del ejecutable `express`
  - Si instalas `express` con `npm` estará ya disponible para toda la máquina:
    - Esta herramienta proporciona un mecanismo para preparar el esqueleto de la aplicación
    - Escribe `express --help` para ver sus opciones



# Usando express.js

- Una vez instalado node, crea un directorio en tu máquina:

```
- mkdir hello-world
```

- Crearemos un paquete de aplicación que incluye el fichero `package.json`

```
- npm init --yes
```

- Los contenidos del fichero `package.json` serán:

```
{
  "name": "hello-world",
  "description": "hello world test app",
  "version": "0.0.1",
  "private": true,
  "dependencies": {
    "express": "3.0.0"
  }
}
```

- Con `package.json` puedes instalar las dependencias escribiendo: `npm install`

- Crea un fichero de nombre `app.js` y crea una nueva aplicación con `express()`:

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

```
var app = express();
```

- Con la instancia de la nueva aplicación puedes definir rutas con `app.VERB()`:

```
- req y res son los mismos objetos que node te proporciona, tú sólo tienes que invocar res.pipe(), req.on('data', callback) y cualquier cosa que harías sin Express
```

```
app.get('/', function(req, res){ res.send('Hello World'); });
```

- Ahora hace un `bind` y un `listen` para esperar a peticiones `app.listen()`, aceptando los mismos argumentos que con

```
net.Server#listen():
```

```
app.listen(3000);
```

```
console.log('Listening on port 3000');
```

# Jade – node template engine

- Jade es un motor de plantillas muy influenciado por Haml e implementado en JavaScript para node
- El lenguaje de plantillas Jade es descrito a continuación
  - Para más detalles visitar: <http://naltatis.github.com/jade-syntax-docs/>
- Documentación sobre cómo usarlo en Node.js puede revisarse en:
  - <https://github.com/visionmedia/jade#readme>

```
doctype 5
html(lang="en")
  head
    title= pageTitle
    script(type='text/javascript')
      if (foo) {
        bar()
      }
  body
    h1 Jade - node template engine
    #container
      if youAreUsingJade
        p You are amazing
      else
        p Get on it!
```



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar()
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container">
      <p>You are amazing</p>
    </div>
  </body>
</html>
```

# Uso de templates en Node.js

- EJS: Embedded JavaScript, es una librería de plantillas en JavaScript
- Se usa para generar strings HTML desde datos JSON. EJS funciona del siguiente modo:



- Queremos evitar lo siguiente:

```
var html = "<h1>" + data.title + "</h1>"
html += "<ul>"
for(var i=0; i<data.supplies.length; i++) {
  html += "<li><a href='supplies/' + data.supplies[i] + '>"
  html += data.supplies[i] + "</a></li>"
}
html += "</ul>"
```

- Para incluir EJS necesitamos añadirlo a nuestro código HTML:

```
<script type="text/javascript" src="ejs_production.js"></script>
```

- Site: <http://embeddedjs.com/>

# Más EJS: Embedded JavaScript

- EJS limpia el HTML de tu JavaScript con plantillas de parte cliente.
  - EJS combina datos y plantillas para producir HTML.
    - Como en ERB, el JavaScript entre `<% %>` es ejecutado
    - JavaScript entre `<%= %>` añade HTML a tu resultado
- Ejemplo de sintaxis:

```
// load a template file, then render it with data
html = new EJS({url:
  '/template.ejs'}).render(data)
```

```
// update element 'todo' with the result of a
// template rendered with data from a JSON request
new
EJS({url: '/todo.ejs'}).update('todo', '/todo.json')
```



# Uso de templates en node.js

- Una plantilla es un fichero donde guardas el mark-up de tu presentación
  - El código entre `<% %>` es ejecutado mientras que las expresiones entre `<%= %>` son insertadas en el HTML generado

```
<h1><%= title %></h1>
<ul>
  <% for(var i=0; i<supplies.length; i++) { %>
    <li>
      <a href='supplies/<%= supplies[i] %>'>
        <%= supplies[i] %>
      </a>
    </li>
  <% } %>
</ul>
```

- A continuación, añadimos JavaScript para controlar la carga y renderización de la plantilla

```
// load the template file, then render it with data
var html = new EJS({url: 'cleaning.ejs'}).render(data);
```

- EJS viene con un conjunto de funciones de ayuda para generar marcado típico como links y forms.

```
<li>
  <%= link_to(supplies[i], 'supplies/'+supplies[i]) %>
</li>
```

- Si tienes un error con EJS, EJS mostrará la línea de error exacta . Todo lo que tienes que hacer es importar el fichero JavaScript `ejs_jslint.js`





# Uso de EJS en Express

- Para usar EJS en Express usar los siguientes pasos:
  1. Instalar ejs con el comando `npm install ejs`
  2. Cambiar el view engine: `app.set('view engine', 'ejs');`
  3. Asociar el directory views: `app.set('views', __dirname + '/views');`
  4. Crear el armazón de todas las páginas que es la plantilla de nombre `layout` en el directorio de view: `layout.ejs`
    - Contiene código HTML estándar con dos excepciones:
      - `<%- body %>` – este hueco se rellenará con el contenido del fichero ejs pasado como parámetro a `response.render()`
        - » `res.render('index', { title: 'The index page!' })`
        - » Donde `index.ejs` es:

```
<div class="title"> <%= title %> </div>
```
      - `<%- partial('header.ejs') %>` – son bloques que se incorporan al layout donde `header.ejs` tendría la forma:

```
<header> My awesome header! </header> <hr>
```
- Documentación:
  - <http://robdodson.me/blog/2012/05/31/how-to-use-ejs-in-express/>



# Desplegando Node.js en la nube

- Nodejitsu
  - Steps:
    - `npm install jitsu -g`
    - `jitsu signup`
    - `jitsu login`
    - `jitsu install helloworld`
    - `jitsu deploy`
  - Documentación:
    - <http://nodejitsu.com/paas/getting-started.html>
    - <http://docs.jit.su/>
- Amazon Web Services
  - Documentación: <http://iconof.com/blog/how-to-install-setup-node-js-on-amazon-aws-ec2-complete-guide/>
  - CloudFormation template: <http://iconof.com/blog/node-js-amazon-cloudformation-template/>
    - Más detalles sobre CloudFormation: <http://aws.amazon.com/es/cloudformation/>
- Alternativas:
  - <https://my.joyentcloud.com/>
  - <http://www.cloudfoundry.com/>



# Desplegando Node.js en Heroku

- Documentación en: <https://devcenter.heroku.com/articles/nodejs#visit-your-application>
- Registrarse en heroku: <https://api.heroku.com/signup/devcenter>
- Instalar heroku toolbelt: <https://toolbelt.heroku.com/>
  - Esto garantiza acceso a la línea de comandos de Heroku, Foreman y al sistema de gestión de revisiones Git
- Escribe: `heroku login`
- Crea el fichero: `web.js`:

```
var express = require('express');
var app = express.createServer(express.logger());
app.get('/', function(request, response) {
    response.send('Hello World!');
});
var port = process.env.PORT || 5000;
app.listen(port, function() {
    console.log("Listening on " + port);
});
```



# Desplegando Node.js en Heroku

- Declara las dependencias para npm con package.json:

```
{  
  "name": "node-example",  
  "version": "0.0.1",  
  "dependencies": {  
    "express": "2.5.x"  
  },  
  "engines": {  
    "node": "0.8.x",  
    "npm": "1.1.x"  
  }  
}
```

- Utiliza npm para instalar tus dependencias: `npm install`
- Crea un Procfile para la aplicación sencilla en la que estamos trabajando:
  - `web: node web.js`
- Puedes ahora arrancar tu aplicación localmente con Foreman (): `foreman start`
- Usa git para crear un repositorio local:
  - `git init`
  - `git add .`
  - `git commit -m "init"`
- Despliega tu aplicación a Heroku
  - `heroku create`
  - `git push heroku master`
- Visita tu aplicación
  - `heroku ps:scale web=1`
  - `heroku open`

# Módulo cradle para conectarse a CouchDB

- Podemos acceder a una BBDD en la nube si nos logueamos en IrisCouch: <http://www.iriscouch.com/>
- El acceso a CouchDB desde node.js puede ser hecho a través de Cradle:
  - Cradle – es un cliente CouchDB de alto nivel que incluye caching para Node.js
    - <https://github.com/cloudhead/cradle>

- Ejemplos:

- Conectándose a la BBDD:

```
var cradle = require('cradle');
var connection = new(cradle.Connection)('https://dipina.iriscouch.com', 443, {
  auth: { username: 'admin', password: 'enpresadigitala' }
});

var db = connection.database('conversions');
db.exists(function (err, exists) {
  if (err) {
    console.log('error', err);
  } else if (exists) {
    console.log('the force is with you.');
```

```
  } else {
    console.log('database does not exists.');
```

```
    db.create();
    /* populate design documents */
  }
});
```

# Cradle para conectarse a CouchDB

- Guardando la BBDD:

```
db.save(idConversion, {
  'username': username,
  'operation': operation,
  'value': value,
  'result': result,
  'timestamp': ts
}, function (err, res) {
  if (err) {
    // Handle error
    console.log("Error produced: " +
err);
  } else {
    // Handle success
    console.log("Información guardada: "
+ err);
  }
});
```

# Otros puntos de interés

- OpenShift: <https://openshift.redhat.com>
- A complete JavaScript stack:
  - Hosting in Joyent for free: <http://no.de>
  - Hosting of IrisCouch for CouchDB: <http://www.iriscouch.com/>
  - HTML5 charts: <http://www.highcharts.com/>
- Express web application framework for node: <http://expressjs.com/>
- 40+ resources for the Node.js developer: <http://architects.dzone.com/articles/40-resources-nodejs-developer>
- Request – simplified HTTP request method: <https://github.com/mikeal/request> y <http://www.jmarshall.com/easy/http/>
- Nano – minimalist CouchDB client for nodejs: <http://writings.nunojob.com/2011/08/nano-minimalistic-couchdb-client-for-nodejs.html>
- Flatiron is an adaptable framework for building modern web applications. It was built from the ground up for use with **Javascript** and **Node.js**: <http://flatironjs.org/>

# socket.io

- **Socket.IO** permite realizar aplicaciones en tiempo real en cualquier navegador y dispositivo móvil, difuminando las diferencias entre mecanismos de transporte
- Instalación: `npm install socket.io`
- URL:
  - Site principal: <http://socket.io/>
  - Documentación: <http://socket.io/#how-to-use>
- Ejemplos:

- `App.js` (parte servidora)

```
var app = require('express')()
, server = require('http').createServer(app)
, io = require('socket.io').listen(server);

server.listen(3000);
app.get('/', function (req, res) {
  res.sendFile(__dirname + '/index.html');
});
io.sockets.on('connection', function (socket) {
  socket.emit('news', { hello: 'world' });
  socket.on('my other event', function (data) {
    console.log(data);
  });
});
```



# socket.io

## – index.html (parte cliente)

```
<script
src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect('http://localhost');
  socket.on('news', function (data) {
    console.log(data);
    socket.emit('my other event', { my:
'data' });
  });
</script>
```



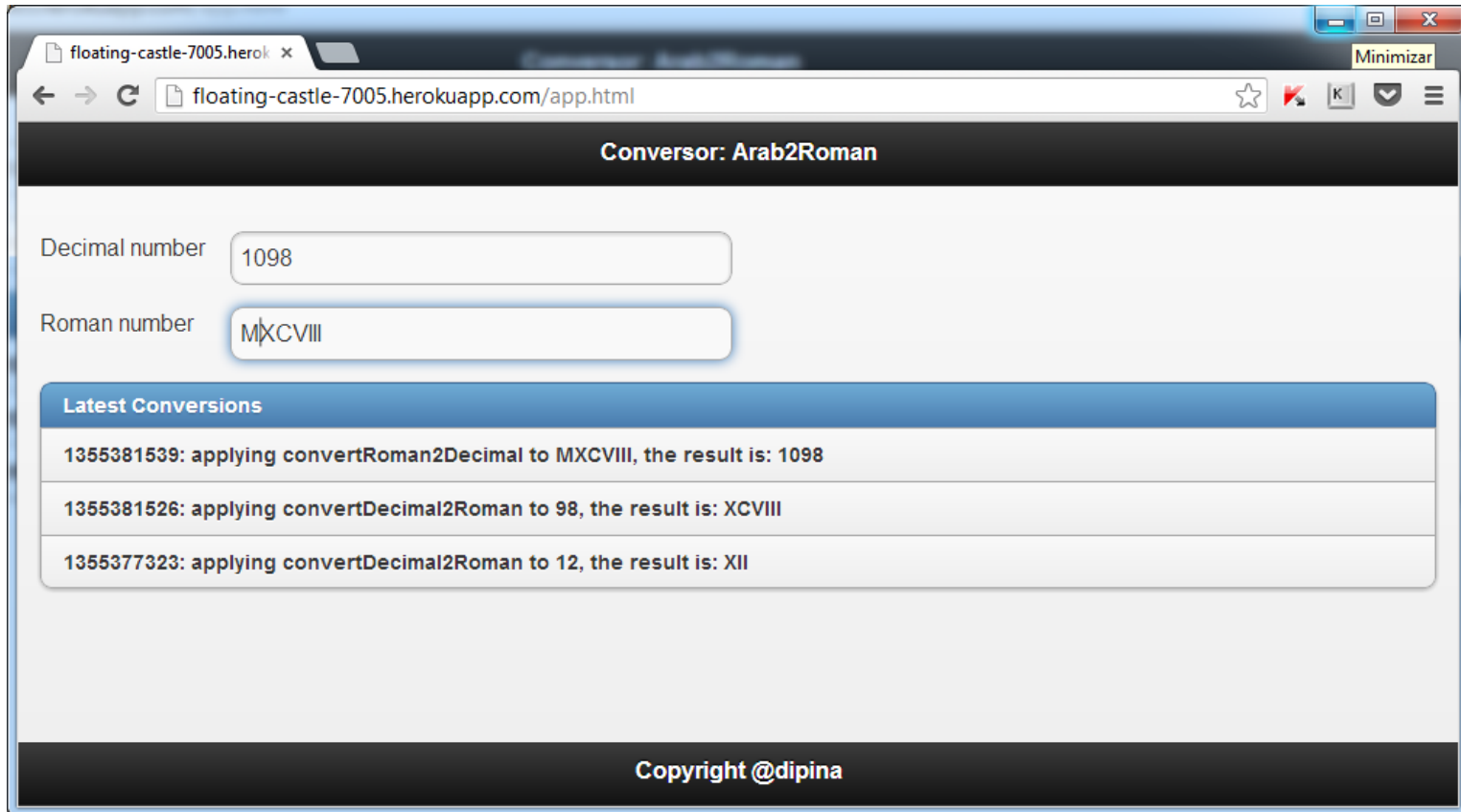
# Creando un Full-Stack JavaScript App

- Vamos a hacer un conversor árabes a romanos, que recuerda las últimas conversiones realizadas:
  1. Hacer que el código use jQuery, basar en ejemplo conversor árabes romanos en AJAX
  2. Convertir el código de la aplicación a express + EJS
  3. Subir la aplicación a <http://www.heroku.com>
  4. Usar una BBDD NoSQL (CouchDB) para guardar un historial de conversiones
    - `<timestamp>`: You converted XXX to 30
    - `<timestamp2>`: You converted 31 to XXXI
  5. Uso de plantilla EJS



# Creando un Full-Stack JavaScript App

- <http://floating-castle-7005.herokuapp.com/app.html>

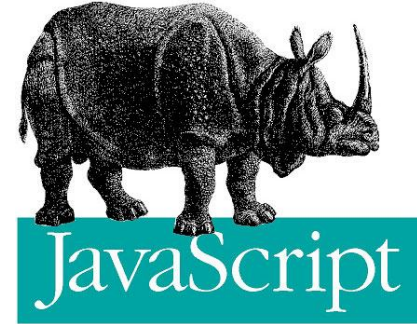




# Referencias

- HTML5 Introduction
  - [http://www.w3schools.com/html/html5\\_intro.asp](http://www.w3schools.com/html/html5_intro.asp)
- CSS3 Tutorial
  - <http://www.w3schools.com/css3/default.asp>

# Referencias



- JavaScript wikipedia page
  - <http://en.wikipedia.org/wiki/JavaScript>
- JavaScript syntax
  - [http://en.wikipedia.org/wiki/JavaScript\\_syntax](http://en.wikipedia.org/wiki/JavaScript_syntax)
- JavaScript Guide
  - <https://developer.mozilla.org/en-US/docs/JavaScript/Guide>
- JavaScript examples
  - [http://www.w3schools.com/js/js\\_examples.asp](http://www.w3schools.com/js/js_examples.asp)
- W3C DOM – Introduction
  - <http://www.quirksmode.org/dom/intro.html>
- JSON in JavaScript
  - <http://www.json.org/js.html>
- Why embed JavaScript?
  - <http://spiderape.sourceforge.net/why/>





# Referencias

- AJAX (programming):
  - [http://en.wikipedia.org/wiki/Ajax\\_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming))
- AJAX Tutorial:
  - <http://www.w3schools.com/ajax/default.asp>
- AJAX Examples:
  - [http://www.w3schools.com/ajax/ajax\\_examples.asp](http://www.w3schools.com/ajax/ajax_examples.asp)

# Referencias



# jQuery

*write less, do more.*

- jQuery Tutorial w3schools
  - <http://www.w3schools.com/jquery/default.asp>
- jQuery API documentation
  - <http://api.jquery.com/>
- jQuery User Interface (UI)
  - <http://jqueryui.com/>
- jQuery UI Demos
  - <http://jqueryui.com/demos/>
- jQuery Mobile UI
  - <http://jquerymobile.com/>
- jQuery Mobile Tutorials:
  - Beginner guide to a mobile app using the jQuery Mobile JavaScript Framework
    - <http://ht.ly/cM7QS>
  - The jQuery Mobile Tutorial
    - <http://the-jquerymobile-tutorial.org/index.php>
  - Mobile Web Development Part 2: Creating a Simple App Using jQuery Mobile
    - <http://devgrow.com/mobile-web-dev-using-jquery-mobile/>
- jQuery Mobile Viewports: <http://www.quirksmode.org/mobile/viewports.html>
- jQuery Mobile Demos: <http://jquerymobile.com/demos/1.0.1/>



# Referencias



- Exploring CouchDB, Joe Lennon,
  - <http://www.ibm.com/developerworks/opensource/library/os-CouchDB/index.html>
- CouchDB tutorial
  - <http://net.tutsplus.com/tutorials/getting-started-with-couchdb/>
- CouchDB for geeks:
  - [http://www.slideshare.net/svdgraaf/CouchDB-for-geeks?from=share\\_email](http://www.slideshare.net/svdgraaf/CouchDB-for-geeks?from=share_email)
- CouchDB site:
  - <http://CouchDB.apache.org/>
- CouchApp.org: The 'Do It Yourself' Evently Tutorial
  - <http://couchapp.org/page/evently-do-it-yourself>
- CouchApp.org: What the HTTP is CouchApp?
  - <http://wiki.couchapp.org/page/what-is-couchapp>
- Tutorial: Using JQuery and CouchDB to build a simple AJAX web application
  - <http://blog.edparcell.com/using-jquery-and-CouchDB-to-build-a-simple-we>
- CouchApp site:
  - <http://couchapp.org/page/getting-started>



# Referencias

- Node.js installation
  - <https://github.com/joyent/node/wiki/Installation>
- The Node Beginner Book
  - <http://www.nodebeginner.org/>
- Understanding Node.js
  - <http://debuggable.com/posts/understanding-node-js:4bd98440-45e4-4a9a-8ef7-0f7ecbdd56cb>
- NodeBeginnerBook code repository
  - <https://github.com/ManuelKiessling/NodeBeginnerBook/tree/master/code/application>
- Node.js v.8.8 Manual & Documentation
  - <http://nodejs.org/docs/latest/api/>
- Principios de JavaScript
  - <http://manuel.kiessling.net/2012/03/23/object-orientation-and-inheritance-in-javascript-a-comprehensive-explanation/>

# Referencias



- Express – getting started
  - <http://expressjs.com/guide.html>
- Creating a Basic Site With node.js and Express
  - <http://shapedshed.com/creating-a-basic-site-with-node-and-express/>
- How to use EJS in Express
  - <http://robdodson.me/blog/2012/05/31/how-to-use-ejs-in-express/>
- Introduction to npm
  - <http://howtonode.org/introduction-to-npm>





Empresa  
Digitala



Universidad de Deusto  
Deustuko Unibertsitatea

# Full-stack JavaScript: Desarrollo integral de aplicaciones Web con JavaScript

19 Diciembre de 2012, 9:00-14:30

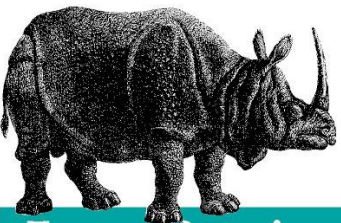
**Dr. Diego López-de-Ipiña González-de-Araza**

[dipina@deusto.es](mailto:dipina@deusto.es)

<http://paginaspersonales.deusto.es/dipina>

<http://www.slideshare.net/dipina>

Descargar ejemplos de: <http://bit.ly/UmmY2H>



JavaScript

