



DISEÑO DE INTERFACES WEB

UT 6
Implementación de la Usabilidad en la Web



Implementación de la Usabilidad en la Web

Objetivos:

- Conocer los factores que determinan la velocidad de acceso a una web
- Conocer el funcionamiento del servicio web las CDN
- Presentar herramientas de optimización de interfaces
- Conocer herramientas online para comprobar la optimización de sitios web
- Presentar herramientas de testeo de interfaces
- Presentar el concepto de integración continua y conocer dos herramientas básicas

Implementación de la Usabilidad en la Web

Contenidos:

- Velocidad de conexión
- Factores determinantes para la velocidad y rendimiento Web
- Optimización de interfaces
- Herramientas online de comprobación de rendimiento
- Herramientas y test de verificación
- Testing desde navegadores
- Testing desde programas
- Integración continua

Implementación de la Usabilidad en la Web

Resumen:

- En este capítulo nos centraremos en la usabilidad desde el punto de vista de la optimización de las interfaces, las técnicas para analizar y mejorar la navegación de sitio web y herramientas para la verificación y optimización del acceso a webs

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.1. Factores determinantes en la velocidad

La velocidad en la que una página o una aplicación se carga ante el usuario es un elemento determinante en el éxito de un proyecto.

Hace una década se afirmaba que una página web que tardase más de siete segundos en cargarse estaba condenada a ser abandonada por otra. Hoy en día ese margen se ha reducido a lo mínimo, considerando siete segundos como poco menos que una eternidad. Por lo tanto, para ofrecer una buena experiencia de usuario hay que empezar por la agilidad desde el inicio.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.1. Factores determinantes en la velocidad

¿Qué factores pueden influir a la hora de cargar la página o una aplicación web? Obviamente el tamaño de la misma, pero casi con la misma importancia está la cantidad de ficheros que compone la misma. Las páginas se componen de hojas de estilos, ficheros JavaScript, imágenes, etc...

Si abrimos determinadas páginas, aunque parezca que se cargan rápido, basta con echar un vistazo a través de las herramientas de desarrollador en la pestaña network para comprobar la ingente cantidad de ficheros que son necesarios únicamente para una simple web.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.1. Factores determinantes en la velocidad

Ya no solo hay que cargar los recursos propios, en ocasiones, se deben cargar ficheros de terceras partes como el analytics, el código para los banners, librerías JavaScript extra, etc. El protocolo HTTP generalmente abre y cierra una conexión de red por cada uno de los ficheros que precisa; así que el navegador tiene que efectuar tantas conexiones como recursos necesite, lo que lleva a un retardo inevitable, teniendo en cuenta que luego, además, hay que iniciar, juntar y ejecutar todo lo que se descarga el navegador.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

Otros factores que afectan al rendimiento son la proximidad o retardo desde el cliente al recurso web. La proximidad geográfica provincial no trae necesariamente una mayor velocidad de acceso, ya que la infraestructura de red puede estar organizada de tal manera que las conexiones sean más rápidas atravesando redes que se encuentran más alejadas. Pero a un nivel nacional generalmente la velocidad de acceso si se puede notar.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

En todos los países existe al menos un punto neutro de internet donde todos los grandes y medianos operadores, ISPs y las empresas que ofrecen hosting o contenidos intercambian tráfico de red. Estar presente en uno de esos puntos neutros aproxima los datos a los usuarios finales.

Un CDN es una red de contenidos que básicamente acerca los contenidos al usuario de una forma muy obvia: manteniendo una red de servidores que este presente en la mayor parte de lugares del planeta posibles. Los CDN ofrecen servicios para que las empresas puedan distribuir contenido y que este sea accesible de forma optima a una mayor parte de los usuarios.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

En todos los países existe al menos un punto neutro de internet donde todos los grandes y medianos operadores, ISPs y las empresas que ofrecen hosting o contenidos intercambian tráfico de red. Estar presente en uno de esos puntos neutros aproxima los datos a los usuarios finales.

Un CDN es una red de contenidos que básicamente acerca los contenidos al usuario de una forma muy obvia: manteniendo una red de servidores que este presente en la mayor parte de lugares del planeta posibles. Los CDN ofrecen servicios para que las empresas puedan distribuir contenido y que este sea accesible de forma optima a una mayor parte de los usuarios.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

CLOUDFLARE

MAXCDN

AKAMAI

Implementación de la Usabilidad en la Web

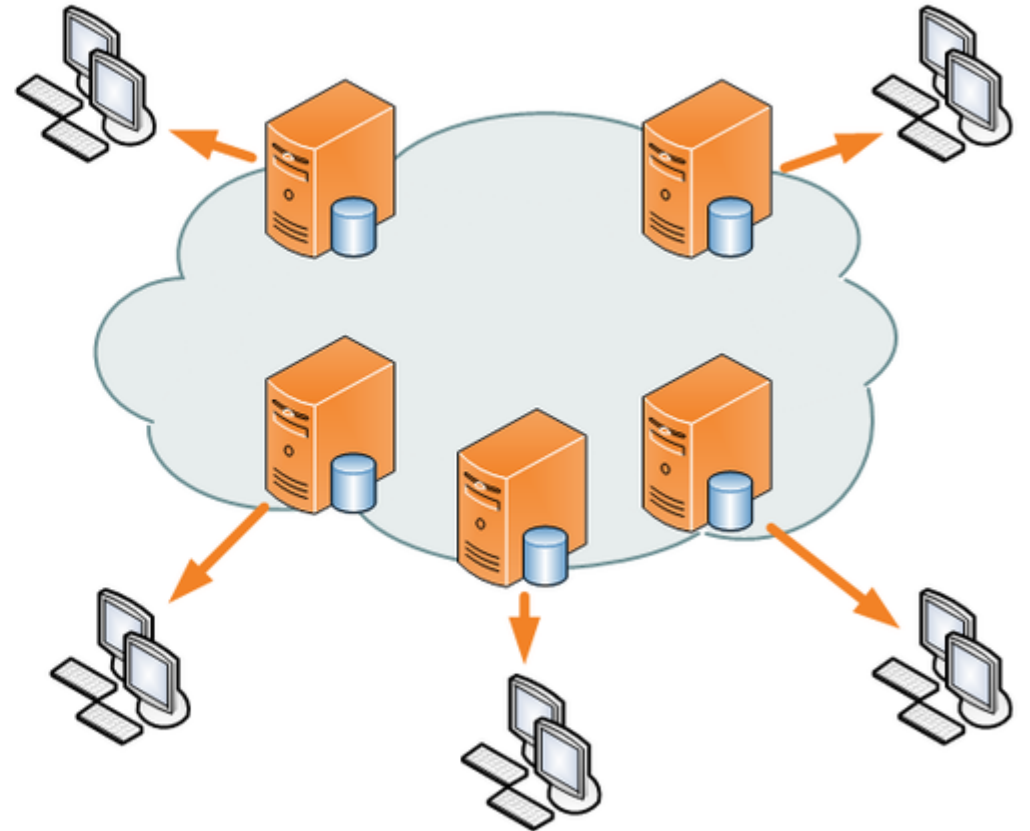
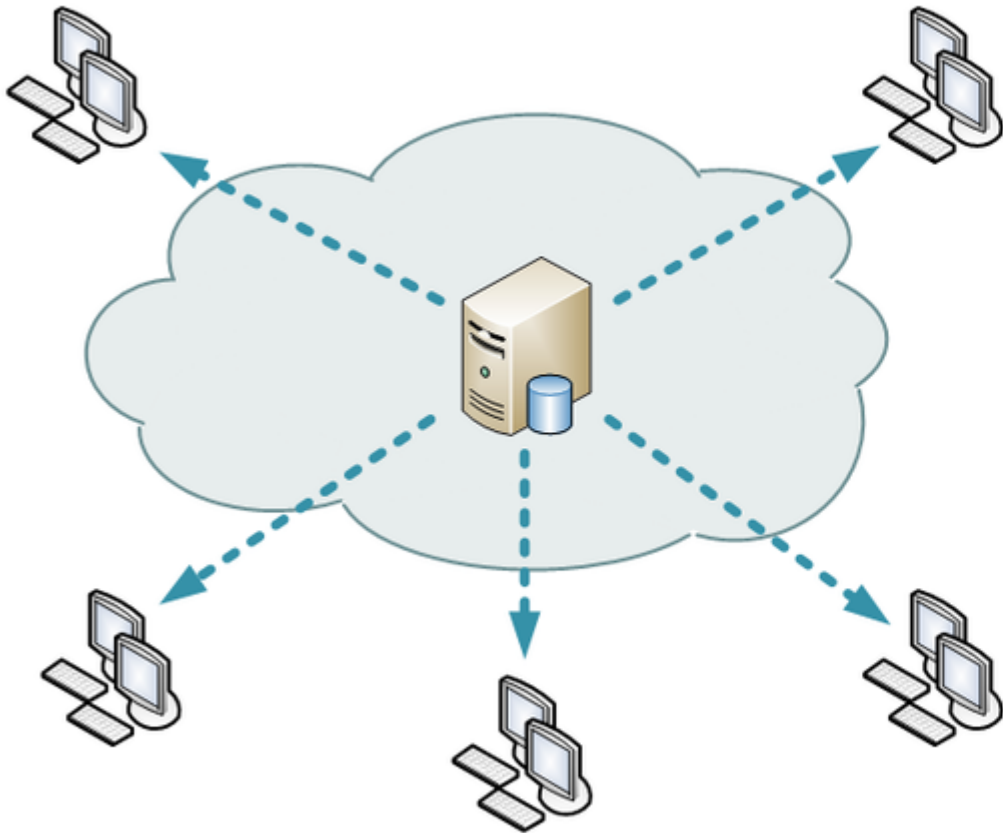
6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

Por ejemplo, si una web contiene elementos estáticos como el html, imágenes, hojas de estilo y JavaScript, estos pueden ser distribuidos a través del CDN. Gráficamente, la diferencia entre usar o no usar un CDN es la siguiente:

Implementación de la Usabilidad en la Web

6.1 VELOCIDAD DE CONEXIÓN



Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.2. CDN: Content Delivery Network

De hecho, existen muchos frameworks que se ofrecen a través de CDNs, por lo que no tendremos que incluir más que una referencia a los mismos para usarlos.

Por ejemplo:

- jQuery
- Angular.js
- Bootstrap
- Fuentes Web, fuentes de iconos, etc.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

Un acceso optimizado a una página web es un requisito indispensable por varios motivos:

- Las debilidades de rendimiento del protocolo HTTP.
- Las interfaces web tienen muchas áreas de mejora.
- La velocidad es una mejora en la usabilidad.
- Es una ventaja competitiva frente a otros.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

Por mucho que haya un ancho de banda mayor y que los terminales tengan mayor capacidad de proceso, la solución no la tenemos de forma automática. De hecho, no solo hay avances en las infraestructuras y en el hardware, las aplicaciones también son más complejas y requieren de más prestaciones. Por lo tanto, siempre será necesario optimizar al máximo las aplicaciones.

Antes de dar las pautas de optimización, conviene conocer los fundamentos del protocolo HTTP, así como el comportamiento de los navegadores. Ambos son determinantes para entender como se deben aplicar las optimizaciones sin caer en errores.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

HTTP es el protocolo de transferencia de hipertexto que utilizan los navegadores para comunicarse con los servidores web. Es, por tanto, un protocolo cliente/servidor, con un servidor centralizado que atiende las peticiones de uno o más clientes de forma simultánea,

A nivel de transporte es un protocolo TCP, por lo tanto, orientado a conexión, es decir, abre una conexión al puerto del servidor y la mantiene abierta hasta que la transferencia de contenido finalice. ¡Pero ojo! Esa conexión y esa transferencia es única por cada fichero que utiliza una web, por lo tanto por cada elemento o fichero que contenga la web, el navegador deberá abrir una conexión distinta:

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

- Una conexión por la página HTML.
- Una conexión por cada hoja de estilos.
- Una conexión por cada fichero JavaScript.
- Una conexión por cada fichero.
- Una conexión por cualquier otro tipo de elemento al que haga referencia la páginaHTML
- Peticiones AJAX a recursos REST.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

En definitiva, para que el usuario vea una simple página, deben abrirse multitud desconexiones al servidor web, a veces incluso a servidores diferentes debido a mecanismos de analítica, banners y contenidos de terceros. Esto puede observarse fácilmente en la pestaña network de las herramientas de desarrollador de cualquier navegador conocido..

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Cualquier web cotidiana supera fácilmente las 100 peticiones. Los navegadores pueden llevara cabo peticiones en paralelo, normalmente hasta 8, así que para llegar hasta números como 100 o más las peticiones se acaban escalonando, con la carga extra que eso supone.

Por mucho que el contenido de cada fichero sea reducido, cada petición HTTP tiene una penalización de tiempo importante. En primer lugar por el uso del propio TCP en sí mismo, y en segundo lugar por las cabeceras de HTTP, que se repiten una y otra vez, puesto que cada peticiones independiente.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Primera conclusiones

Estas son algunas de las primeras conclusiones que extraemos si queremos mejorar el rendimiento de una web. Algunas tienen matices sobre las que se profundiza más adelante.

- Hay que reducir el número de peticiones al servidor.
- Hay que reducir el tamaño de ficheros.
- Hay que unir los ficheros CSS y JS en ficheros únicos de CSS y JS respectivamente.
- Hay que integrar el contenido CSS y JS en el propio HTML.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Primera conclusiones

A primera vista parece una clara regresión a una manera de desarrollar interfaces donde se crea un fichero que tiene todo mezclado, algo difícilmente mantenible. Obviamente, gracias a herramientas de building como Grunt o Gulp, se puede editar de forma separada y luego se integra todo de forma automática.

Esas herramientas son también las que nos pueden permitir reducir los ficheros con plugins como htmlmin, cssmin, uglify y unirlos mediante concatenación, para posteriormente generar un fichero HTML que contenga todo de una forma que funcione.

En principio se dispone de todo tipo de herramientas de optimización. Pero hay que aplicarlas al extremo?

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Optimización de imágenes

Con las imágenes pueden aplicarse diferentes optimizaciones, alguna de ellas ya mencionadas. Lo normal es que tengamos muchas imágenes, lo que se trata de evitar es que el navegador haga una petición por cada una de las mismas. Por lo tanto:

- Podemos crear una imagen única que contenga todas y mostrarlas con propiedades deshojas de estilos.
- Podemos optimizar su tamaño, aplicando compresión o bajando la calidad a un ratio razonable.
- Podemos serializar las imágenes e integrarlas dentro del HTML o en el CSS.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Optimización de imágenes

La serialización de imágenes consiste en codificar un archivo de imagen a formato base64, de manera similar a como se hace en los correos electrónicos que contienen imágenes incrustadas. A la hora de añadirlas en el HTML lo ponemos así:

```

```

De esta manera, incrustando esa imagen nos evitamos una petición a la web por cada imagen.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Comportamiento de los navegadores

Los navegadores son la parte cliente del protocolo HTTP. El proceso que siguen para mostrar una página es el siguiente:

1. El usuario indica una url en la barra de navegación y pulsa enter.
2. El navegador trata de resolver el nombre de host a través de la IP para poder conectarse al servidor.
3. Una vez resuelta la IP, el navegador abre una conexión TCP/IP al puerto 80 (o 443 si es https) del servidor remoto.
4. Una vez abierto el puerto, le indica la petición según el protocolo HTTP por ejemplo: GET /

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Comportamiento de los navegadores

5. El servidor responde con una serie de cabeceras y el contenido HTML.

6. El navegador recorre el HTML y por cada elemento referenciado repite el mismo proceso (CSS, JavaScript, imágenes).

Como se puede apreciar, por mucho que esto se haga en milisegundos, cada fichero requiere de una petición compleja. Y si ese elemento está en otra web, habrá que añadir el paso de resolución de nombres.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Comportamiento de los navegadores

Como se ha dicho antes, el navegador es capaz de realizar peticiones simultáneas pero hasta cierto punto. Una web de un medio de comunicación puede mostrar el siguiente tráfico de red en la pestaña network de la herramienta del desarrollador:

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Comportamiento de los navegadores

Las cifras en este caso son espeluznantes:

- 524 peticiones, algunas a url externas.
- 8MB de transferencia.
- 1 minuto para cargar toda la página.

Si bien la carga del DOM ocurre en nueve segundos, tal y como se aprecia en el timeline de peticiones, el navegador debe ir encadenando peticiones por mucho que pueda hacer varias en paralelo.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Comportamiento mejorado

Debido a que el protocolo HTTP no ha evolucionado a la velocidad que evolucionan el resto de actores de la Web, los navegadores modernos procuran comportarse de manera eficiente para mejorar la navegación sea como sea la página que se esté visitando

Una de ellas es la posibilidad de iniciar las descargas o al menos la resolución de los elementos referenciados dentro de la página, también conocido como prefetching. De esa manera se ahorra tiempo en el caso de que el usuario solicite esas páginas.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

JavaScript

En el caso de ficheros JavaScript sucede algo peculiar. Los navegadores bloquean las peticiones hasta que el JavaScript este descargado. Si las referencias a JavaScript están al principio de las páginas o en la sección head, esto supondrá un obstáculo para la carga del resto de la página. Por ese motivo, conviene incluir todas las referencias a JavaScript al final

Pero esto no es suficiente. Poner el script al final hace que la renderización de la página resulte más ágil pero al final el script tiene que acabar cargándose. Lo que se puede aplicar en el caso de JavaScript, en especial en el caso de que se cargue JavaScript de otros sitios (jQuery, Google Analytics, Adsense,...), es una carga asíncrona del mismo.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

JavaScript

Este problema lo resuelve Facebook en el código JavaScript que ofrece para añadir un botón "me gusta" en cualquier web. Si cargásemos ese script de una forma tradicional lo haríamos así:

```
<script src="http://connect.facebook.net/en_US/sdk.js"></script>
```

Pero tal y como propone Facebook, el script se añade de esta manera:

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

JavaScript

Pero tal y como propone Facebook, el script se añade de esta manera:

```
<script> (function(d, s, id) {  
    var js, fjs = d.getElementsByTagName(s)[0];  
    if (d.getElementById(id)) return;  
    js = d.createElement(s); js.id = id;  
    js.src "//connect.facebook.net/en_US/sdk.js#xfbml=1";  
    fjs.parentNode.insertBefore(js, fjs);  
} (document, 'script', facebook-jssdk)); </script>
```


Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

JavaScript

Básicamente se trata de una función que se auto invoca y que agrega el JavaScript de manera dinámica al documento. La cuestión es que esa forma de cargar el JavaScript es asíncrona y no deja el navegador bloqueado.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Conclusiones

A primera vista, se podría pensar que la opción óptima sería, en resumen, intentar que toda la web quepa en una única petición. Y por tanto, además de concatenar todos los CSS y el código JavaScript, habría que codificar imágenes y meter todo ese conjunto incrustado en el HTML. Sin duda se reducirían las peticiones a la mínima expresión. ¿Pero es verdaderamente la mejor opción?

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Conclusiones

Ni mucho menos. En primer lugar porque no todas las páginas webs son iguales, no todas hacen el mismo uso de JavaScript, o de CSS o de frameworks más o menos pesados. Y en caso de hacerlo, no todas lo harán con el mismo nivel de funcionalidad. Además de esta obviedad hay que tener en cuenta algunas consideraciones por las cuales hacer todo el contenido incrustado no es necesariamente la mejor opción:

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Conclusiones

- Se desperdicia la capacidad de cacheo del navegador. Al estar todo integrado, si cambia el contenido de la página siempre será distinta por mucho que el JavaScript o el CSS sea el mismo.
- Se desperdician las ventajas que pueden aportar los CDNs. Si se utilizan librerías o frameworks populares como jQuery o Angular, nuestra web no se podrá beneficiar de la inmediatez que ofrecen los CDNs sirviendo ese tipo de ficheros.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Conclusiones

- Se retrasa el prefetching o preloading que hacen los navegadores de los contenidos que aparecen en la página.
- Una página puede no cargar contenido mientras el usuario no haga scroll o no se abra un cuadro de diálogo o bloque invisible. Si todo el contenido ya está precargado, se invalida la optimización de carga de contenido bajo demanda.
- El uso de imágenes inline también impide aprovecharse del sistema de cache. Una posible mejora para ello es utilizar imágenes inline como fondos en CSS, ya que de esa manera si se guardan en caché.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Mejoras

En resumen, las mejoras razonables que se pueden aplicar en una interfaz y más en concreto en interfaces web son las siguientes:

- Minimizar las peticiones.
- Concatenar y minimizar las hojas de estilos y servir las al principio de la página.
- Concatenar y minimizar el JavaScript desarrollado por nosotros y servirlo al final de la página
- Minimizar el HTML.

Implementación de la Usabilidad en la Web

6.1. VELOCIDAD DE CONEXIÓN

6.1.3. Rendimiento web

El protocolo HTTP

Mejoras

- Unir imágenes tipo icono en mapas de imágenes.
- Reducir calidad de imágenes.
- Hacer imágenes inline o serializadas, siempre que no superen el tamaño de 2048bytes.
- Utilizar módulos de compresión en el servidor web.
- Meter contenido CSS o JavaScript inline solamente si es muy pequeño, aproximadamente menor a 4K bytes.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

La optimización de interfaces requiere de muchas operaciones sobre todos los ficheros que forman parte de la misma: HTML, CSS, JavaScript, imágenes, etc. Y además hay que hacerlo de forma continuada, cada vez que se modifica algo.

La única forma razonable de poder llevar a cabo todas esas tareas de forma coordinada es utilizando ejecutores de tareas task-runners. Se trata de ficheros con instrucciones, scripts o frameworks que se configuran para hacer esas tareas de forma automatizada. Desde los ficheros makefile de unix, a los ficheros xml de ant, para la web contamos con dos herramientas hechas en JavaScript que se ejecutan con Node.js: Grunt y Gulp. Ambas consiguen el mismo resultado, pero utilizan distintas filosofías. En capítulos anteriores ya hemos comprobado la utilidad de Grunt para compilar Bootstrap, ahora nos centraremos en las herramientas de optimización.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Grunt es un task-runner que se caracteriza por basarse en un fichero de configuración: Gruntfile.js. Si bien el fichero es un programa JavaScript, la mayor parte del mismo consiste en configurar los plugins de tareas, y apenas tiene código.

Veamos un ejemplo completo de optimizaciones aplicadas a un proyecto Web. Por lo general se parte de una web sin optimizaciones y se genera una web en una carpeta diferente con las optimizaciones aplicadas. La convención general suele ser que la carpeta de origen o fuente sea src y la fuente destino sea dist.

A veces puede incluso haber una carpeta intermedia que se necesita entre una optimización y otra, y se le suele llamar stage.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Veremos por separado las tareas en el orden en que son ejecutadas:

Inicio

En el inicio de Grunt es donde debemos indicar qué módulos vamos a utilizar entre nuestras tareas Grunt. El hecho de que estén ahí no significa que se vayan a ejecutar todas, pero si deben estar declaradas las que queramos utilizar.

En este caso el orden en que aparecen en esa lista no es determinante, y tampoco lo es la listade configuraciones. En cualquier caso no está de más hacerlo:

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

```
// Gruntfile.js
module.exports=function (grunt) {
// Loading
// One task: grunt.loadNpr Tasks('grunt-contrib-jshint');
// Many tasks
'grunt-contrib-jshint',
'grunt-contrib-esslint',
'grunt-html',
'grunt-contrib-clean',
'grunt-contrib-concat',
'grunt-contrib-cssmin',
'grunt-contrib-html.min',
'grunt-contrib-uglify',
'grunt-processhtml',
'grunt-hashres'
].forEach(function (g) {
grunt. loadNpmTasks (g);
});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Una vez declaradas las tareas pasamos a la configuración de cada una de las tareas que se encuentra dentro de:

```
// Configuration  
grunt.initConfig({
```

Generalmente las tareas tienen un formato similar:

- nombre: el nombre de la tarea.
- opciones: características que queremos utilizar de la herramienta.
- origen: fichero o directorios de origen que deseamos analizar, convertir, etc.
- destino: destino de esos ficheros.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Jshint

jshint es una herramienta basada en jslint que busca problemas o posibles fuentes de errores en el código. No se trata tanto de un compilador, sino más bien un corrector de estilo. Obviamente su uso es totalmente opcional, pero generalmente sus comprobaciones se basan en buenas prácticas bien conocidas y, por tanto, da buenos consejos que conviene tener en cuenta:

Este sería el aspecto que tendría la configuración básica de jshint:

```
jshint: {  
  options: {  
    curly: true,  
    eqeqeq: true  
  },  
  target1: ['Gruntfile.js', 'src/js/js', '!src/**/*.js']  
},
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Jshint

Opciones: jshint tiene multitud de opciones y estas son algunas de las más destacables:

- eqeqeq: obliga a que los operadores de igualdad/desigualdad === y !== sean más estrictos y comparen el tipo: === y !==
- curly: indica si hay algún bloque sin las llaves.
- esversion: indica con qué estándar ECMAScript se debe cumplir: 3,5 o 6.
- freeze: para detectar sobrescrituras de prototipos de los objetos por defectos: Array, Math, Date...
- funscope: avisa si alguna variable declarada dentro de una estructura de control se utiliza fuera de ella. Es algo posible, pero es una mala práctica.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Jshint

- maxdepth: el nivel de profundidad que se puede alcanzar al anidar estructuras de control.
- maxerr: número máximo de errores que debe tolerar jshint.
- maxparam: número máximo de parámetros que permitimos por método.
- strict: trata de que el código cumpla con ECMAScript 5.
- unused: alerta sobre variables no utilizadas.
- var: alerta sobre el uso de declaraciones var. Debería usarse let o const.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Htmlint

Al igual que jshint, htmlint es la herramienta correspondiente para detectar problemas en los ficheros html.

Este sería el aspecto que tendría la configuración básica:

```
htmlint:{
  options: {
    path: false,
    reportpath: false // output to console },
  src: [
    'src/*.html', // Include all HTML files in this directory.
    '!src/*.min.html' // Exclude any files ending with
    .min.html
  ]
}
```


Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Htmlint

Opciones: Algunas de las opciones más interesantes son las siguientes:

- `force: true | false`, para indicar si ante un error debe detenerse el build de grunt.
- `attr-name-style: true | false`, para obligar a que los atributos estén en minúsculas.
- `attr-quote-style: double single quoted false`, para establecer el tipo de comillas, si se pone false se ignoraría.
- `doctype-html5`: obliga a iniciar el documento con la declaración de doctype de html5.
- `head-req-title`: obliga a poner un título en el head.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Htmlint

- href-style: absolute relative, obliga a un tipo u otro de enlaces.
- id-no-dup: controla que no haya ids duplicados.
- img-req-alt: true false, obliga a meter el atributo alt en las imágenes.
- indent-style: tabs spaces nonmixed false, para controlar el modo de indentar el código Se puede combinar con indent-width, que controla el número de caracteres utilizados.
- tag-names-lowercase: true false, controla si se usan mayúsculas/minúsculas en las etiquetas
- tag-self-close: controla las etiquetas que se autocierran como img, br, input....

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Csslint

También las hojas de estilos disponen de una herramienta para depurar, sugerir y corregir las reglas aplicadas. Este sería el aspecto que tendría la configuración básica de csslint:

```
csslint: {  
    options:{},  
    src: [ 'src/css/*.css', // Include all HTML files in this  
directory.  
        '!src/css/*.min.css' // Exclude any files ending with  
'..min.html'  
    ],  
}
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Csslint

Opciones: Algunas de las opciones más interesantes son las siguientes:

- empty-rules: para evitar que haya reglas vacías.
- duplicate-properties: evita reglas duplicadas.
- known-properties: obliga a utilizar solo propiedades conocidas.
- box-model: evita problemas de uso incorrecto de dimensiones en elementos según el modelo de caja.
- adjoining-classes: para permitir o no selectores de clases pegadas: .foo var.
- vendor-prefix: si se usa algún prefijo específico de navegador -moz, -webkit, ... Indica que al final hay que meter la propiedad estándar.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Csslint

- font-faces: para no utilizar demasiadas fuentes.
- zero-units: para los valores o se deberían quitar las unidades (px, em....)
- floats: para no permitir demasiados valores con decimales, como en alpha, dimensiones.
- Import: habilitar o deshabilitar el uso de @imports.
- universal-selector: para permitir o no el uso del selector universal o *
- regex-selectors: habilitar o deshabilitar los elementos que cumplan con un determinado patrón.
- overqualified-elements: para evitar elementos a los que se hace referencia de distintas maneras.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Concat

Su uso resulta fundamental, une un conjunto de ficheros en uno solo. Es frecuente que tanto hojas de estilos como ficheros JavaScript se editen de forma separada para facilitar el mantenimiento y, de hecho, es conveniente hacerlo así. Pero a la hora de cargar la página, el hecho de tener muchos ficheros por separado tiene un alto coste en cuanto a la velocidad de carga de la página, y eso sin duda afecta a la experiencia de usuario. Por lo tanto, unir todos esos ficheros es crucial, ya que además no afectará al funcionamiento ni de las hojas de estilo ni del código JavaScript

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Concat

Este seria el aspecto que tendría la configuración básica de jshint:

```
concat: {options:{separator: ';'},
dist:{src:['src/js/jquery.js', 'src/js/bootstrap.js', 'src/js/question.js',
'src/js/quiz.js','src/js/game.js','src/js/app.js','!src/**/*.min.js'].
dest: 'stage/js/app.nin.js' }},
clean : {target1 : {src: I['src/**/*.-', 'dist/*', 'stage/*'] }},
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Cssmin

Este módulo minimiza los ficheros CSS de tal manera que ocupen lo menos posible sin perder su funcionalidad. Algo tan simple como la eliminación de espacios en blanco superfluos o los saltos de línea reducirá el tamaño de los mismos.

Este sería el aspecto que tendría la configuración básica de cssmin:

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Cssmin

Cssmin:

```
minify: {files: [{  
  expand: true, cwd: 'src/css',  
  src: ['**/*.css', '!**/*.min.css'],  
  dest: 'dist/css', ext: '.min.css' } ]},  
options: {  
  shorthandCompacting: false,  
  rounding Precision: -1},  
combine: {  
  files: { 'dist/css/app.min.css': ['!dist/css/*.min.cas',  
    'dist/css/*.css'] } }},
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Htmlmin

También los ficheros html son susceptibles de ser miniaturizados y además en gran medida. Aparte de eliminar todos los caracteres invisibles (espacios, tabulaciones, saltos) también quita comillas, comentarios y lo que le indiquemos. Este sería el aspecto que tendría la configuración básica de htmlmin:

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Htmlmin

```
htmlmin:{dist:{      //Task
  options:{  // Target options
    removeComments: true,
    collapseWhitespace: true },
  files: {    // Dictionary of files
    'dist/index.html': 'stage/index.html' // 'destination': 'source'
  }
},
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Uglify

Uglify tiene en cierto modo dos funciones: ofusca el código y lo minimiza. Es una forma de "compilar" el JavaScript porque lo hace irreconocible, pero lo interesante es que hace que el tamaño del mismo se reduzca considerablemente, quitando caracteres invisibles, renombrando variables, etc. Uglify además es capaz de llevar a cabo la tarea de concatenación de ficheros. Este sería el aspecto que tendría la configuración básica de processhtml:

```
uglify:{report: 'min',target1: {files :{
`dist/js/app.min.js': ['stage/js/app.min.js']
}}},
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Processhtml

Cuando se aplican optimizaciones en las hojas de estilo y JavaScript se acaban uniendo los ficheros, pero queda una cosa que hacer: en la página HTML que hace referencia a esos ficheros hay que cambiar de referencia individual a cada fichero css y js a una nueva referencia al fichero minificado y único de css y js. De esto se encarga processhtml. Para que sea capaz de hacer ese cambio, processhtml nos obliga simplemente a añadir una marca en el HTML para saber dónde hacer el cambio. Este sería el aspecto que tendría la configuración básica de processhtml:

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Processhtml

```
procesahtml: {  
    options: {},  
    dist : {  
        files: { 'stage/index.html': ['src/index.html']  }}}  
// Cerramos el conjunto de configuraciones de tareas,  
});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Ejecución de tareas

Por último debemos registrar las tareas, que en cierto modo sirve para crear conjuntos de tareas bajo un identificador. Como mínimo debemos tener una tarea llamada default, aunque podemos crear las combinaciones que necesitemos.

// Default task

```
grunt.registerTask('default', ['clean', 'cssmin', 'concat', 'uglify',  
'processhtml', 'htmlmin']);  
grunt.registerTask('hard', ['clean', 'jshint', 'csslint', 'htmlhint', 'concat']);  
};
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.1. Aplicando optimizaciones con Grunt

Ejecución de tareas

Si ejecutamos simplemente:

Grunt

Se ejecutará la tarea default, para ejecutar cualquier otra debemos indicar su nombre:

grunt hard

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Gulp es otra herramienta de ejecución de tareas, con que se puede conseguir lo mismo que con Grunt pero aplicando un estilo totalmente diferente: el fichero gulpfile.js tiene forma de programa más que de fichero de configuración.

La instalación es más simple ya que simplemente nos basta con hacer:

```
npm install -g gulp
```

A partir de ahí podemos editar un fichero gulpfile.js con las definiciones de tareas.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

El fichero `gulpfile.js`

Este fichero se suele poner en la raíz del proyecto y es el que Gulp utiliza como referencia para ejecutar las tareas. Si ejecutamos simplemente:

Gulp

Este buscará el fichero `gulpfile.js` y ejecutará la tarea llamada `default`. Si quisiéramos ejecutar otra tarea podríamos hacer algo así como:

```
gulp nombre-de-tarea
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

El fichero `gulpfile.js`

Este sería un ejemplo mínimo de fichero `gulpfile.js`, con una única tarea que muestra un

```
var gulp = require('gulp');  
gulp.task('say_hello', function () {  
    console.log('Gulp says hello');  
});  
gulp.task('default', ['say_hello']);
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

El fichero `gulpfile.js`

Como se puede apreciar, el fichero es puro código JavaScript para Node.js. Primero se incluye la librería gulp, luego se define una tarea llamada say hello y luego se configura el grupo de tareas default, cuya única tarea es precisamente say hello.

Lo normal es tener más de una tarea en el default o incluso tener distintos grupos de tareas.

El formato de una tarea o task siempre es el mismo: se le da un nombre, se toman unos ficheros de origen, se les aplican uno o más cambios a través del método pipe, y el resultado se traslada a un destino. Este por ejemplo, es un task que simplemente mueve los ficheros de un directorio a otro:

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

El fichero `gulpfile.js`

```
gulp.task('just_move', function () {  
    return gulp.src('/src/**/*')  
        .pipe(gulp.dest('dist/'));  
});
```

La clave de las tareas Gulp es el encadenamiento de streams a los que se les van aplicando procesos: concatenación, minimización, etc. Como veremos en el siguiente ejemplo de optimización

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Optimizando una aplicación web

En el siguiente ejemplo se muestra la optimización de una web que se encuentra dentro de una carpeta src con la siguiente estructura típica:

Por una parte tiene un único fichero html. En cuanto a CSS, incluye Bootstrap, además del CSS de theme de Bootstrap y algo de CSS personalizado en custom.css. En JavaScript tienen la librería jQuery además del soporte para los diálogos modal de Bootstrap y algo de JavaScript propio en custom.js.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Limpiar dist

Lo primero que haremos en el proceso es limpiar el directorio destino, el cual dejaremos como nuevo cada vez que ejecutemos Gulp:

```
gulp.task('clean', function()  
  return del('dist/**/*');
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Mover las fuentes

A las fuentes web no les aplicamos ninguna optimización y simplemente las movemos a dist:

```
gulp.task('movefonts', function () {  
  return gulp.src('src/fonts/*.*)  
    .pipe(gulp.dest('dist/fonts'));});
```


Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Optimizar el JavaScript

Todos los ficheros JavaScript son concatenados y posteriormente miniaturizados o afeados con uglify. En este caso se especifican los ficheros JavaScript uno a uno, porque el orden es determinante para el correcto funcionamiento del código. jQuery debe ir primero, ya que el resto depende de él.

```
gulp.task('minifycss', function()  
return gulp.src('src/js/jquery.js', 'src/js/nodal.js', 'src/js/custom.js').  
pipe(concat('all.min.js'))  
.pipe(uglify())  
.pipe(gulp.dest('dist/js'))});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Optimizar las hojas de estilos

Todos los ficheros CSS son concatenados y miniaturizados. En este caso no se indica ningún orden concreto, por tanto, la concatenación se hará en orden alfabético, según el nombre que tengan los ficheros:

```
gulp.taskminifycss', function(){  
  return gulp.src('src/css/*.css').  
    .pipe(concat('all.min.css'))  
    .pipe(cssnano())  
    .pipe(gulp.dest('dist/css'));
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Reescribir referencias y minificar el HTML

Una parte crucial del proceso es cambiar las referencias a los múltiples ficheros CSS Y JavaScript y cambiarlos por un único fichero. Para eso hay que modificar el HTML y de eso se encarga el plugin useref.

Por ejemplo, en el fichero HTML original tenemos lo siguiente en la sección CSS, donde deben ponerse unos comentarios específicos para que useref funcione correctamente

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Reescribir referencias y minificar el HTML

```
<!-- build:css css/all.min.css -->  
<link rel="stylesheet" href="css/bootstrap.css" type="text/css">  
<link rel="stylesheet" href="css/bootstrap-theme.css" type="text/css">  
<link rel="stylesheet" href="css/custom.css" type="text/css">  
<!-- endbuild -->
```

Lo cual useref convertirá en:

```
<link rel="stylesheet" href="css/all.min.css" type="text/css">
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Reescribir referencias y minificar el HTML

Hay que tener cuidado porque useref aplica concatenación por defecto, de hecho se puede aplicar procesos de miniaturización junto con useref, pero en este caso se prefiere mostrar como tareas separadas. Así es como quedaria:

```
gulp.task('minifyhtml', function (){  
  return gulp.src ('src/*.html')  
    .pipe (useref({noAssets: true}))  
    .pipe (htnimin ({collapseWhitespace: true}))  
    .pipe (gulp.dest('dist'))});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Sincronizar con un navegador

Si queremos ver los cambios reflejados en un navegador, conforme modificamos los contenidos podemos utilizar este plugin el cual crea un pequeño servidor web y lo sincroniza común navegador. Basta con decirle que muestre el contenido de dist/:

```
gulp. task ('browserSync', function(){  
  browser Sync.init({  
    server :(  
      baseDir: 'dist'  
    }  
  });});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

Watch

Esta tarea básica permite vigilar los ficheros que se editan en la web para que se les apliquen los procesos anteriores cada vez que se modifica el contenido de cualquiera de ellos. En cuanto se cambia se ejecutan los procesos que se indiquen en un array

```
gulp.task('watch', function() {  
  // and now my watch begins...  
  gulp.watch('src/js/*.js', [minifyjs]);  
  gulp.watch('src/css/*.css', ['minifycss']);  
  gulp.watch('src/index.html', ['minifyhtml']);  
});
```

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

La tarea por defecto

Por último, la tarea por defecto queda de la siguiente forma:

```
gulp.task('default',  
[clean', 'movefonts", minaryjs', 'minifycss', 'minifyhta', 'browserSync', 'watch']  
);
```

Una vez puesto en marcha con Gulp, el proceso quedará abierto debido al watch, y cada cambio provocará que se apliquen las optimizaciones pertinentes y se recargue el navegador.

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.2. Optimizaciones con Gulp

¿Grunt o Gulp?

A primera vista, la diferencia entre Grunt y Gulp es prácticamente la misma que hay Maven y Gradle: uno se basa en configuraciones y otro es un script (Maven utiliza xml y Gradle el lenguaje Groovy). Gulp y Gradle han surgido posteriormente, y quizá su estilo sean más del gusto de algunos desarrolladores o encajen más en cierto tipo de proyectos. Por ejemplo, Android ahora utiliza Gradle de forma oficial. Quizá no haya ninguna respuesta única, pero lo que está claro es que si requerimos añadir más funcionalidades personalizadas al build quizá nos convenga el enfoque de Gulp.

Implementación de la Usabilidad en la Web

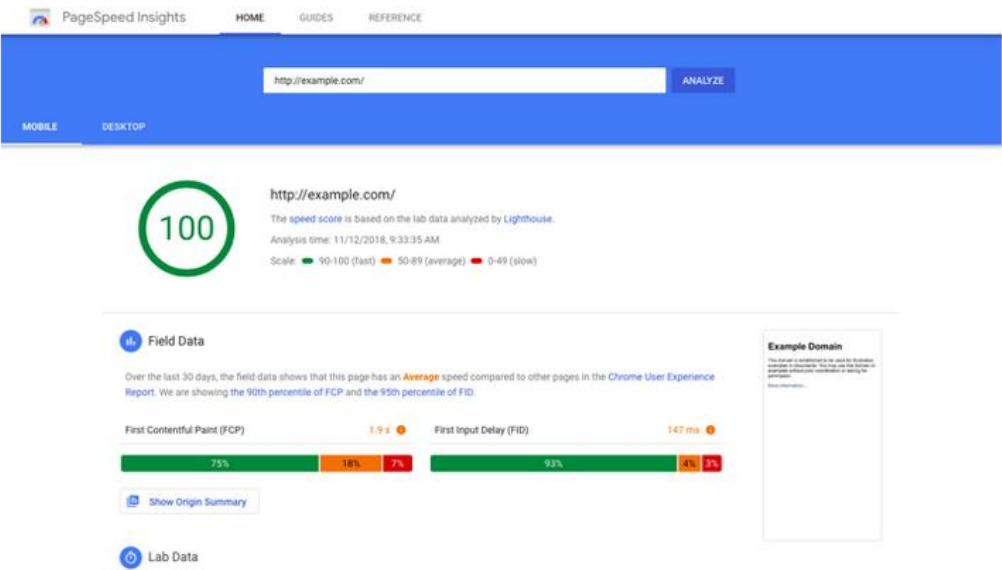
6.2. OPTIMIZACIÓN DE INTERFACES

6.2.3. Herramientas online de comprobación de velocidad

Goole Page Speed Insights

Además de ofrecer herramientas para automatizar el análisis, Google Developers nos ofrece esta herramienta online que nos analiza cualquier página y nos indica los puntos de mejora.

Analyze and optimize your website with PageSpeed tools



Analyze with PageSpeed Insights

Get your PageSpeed score and use PageSpeed suggestions to make your web site faster through our online tool.

[Run Insights](#) [Read docs](#)



Speed up with the PageSpeed Modules

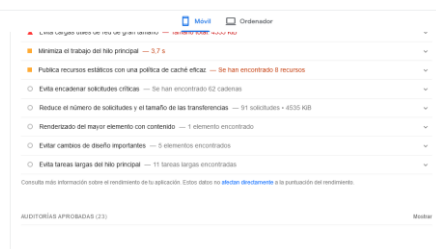
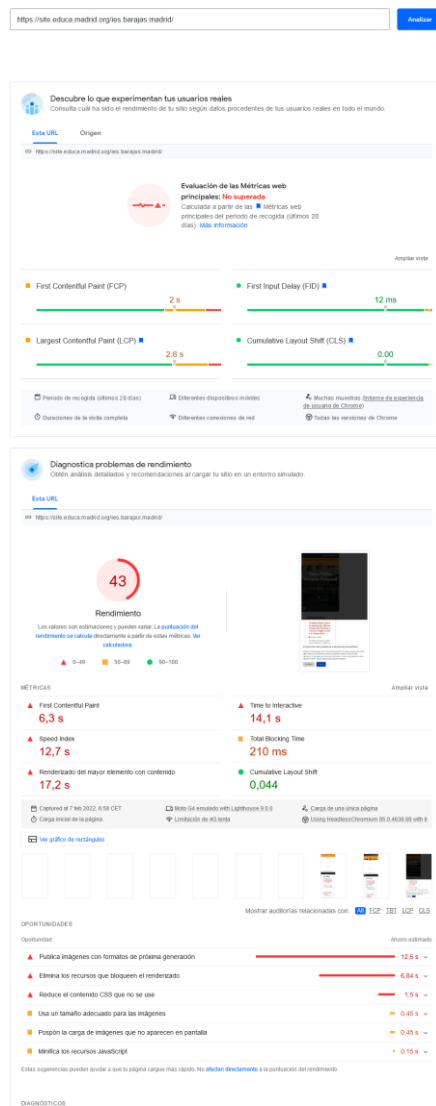
Run the open source PageSpeed Modules on your Apache or Nginx server to automatically rewrite and optimize resources on your web site.

[Learn more about the PageSpeed Modules](#)

Impleme

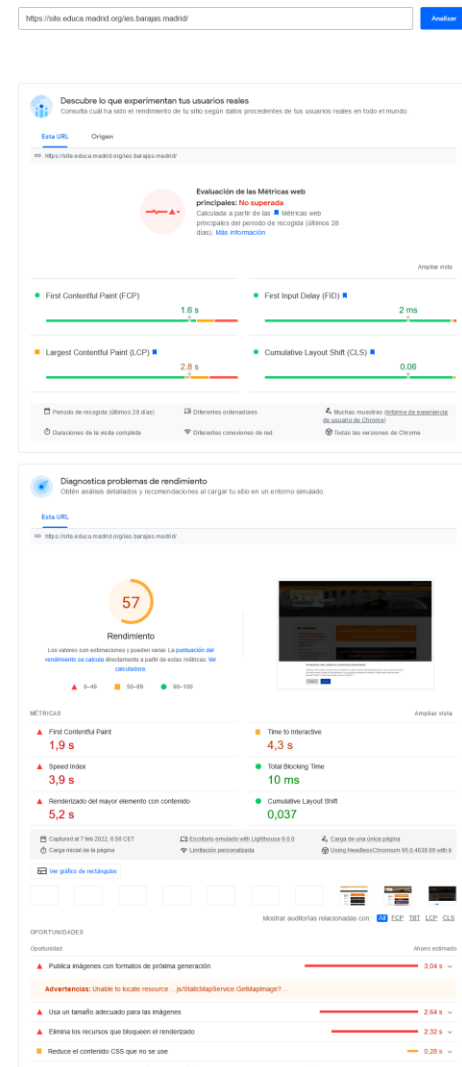
6.2. OPTIM

6.2.3. Herra Goole Page



la Usabilidad

RFACES comprobación de



Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.3. Herramientas online de comprobación de velocidad

Pingdom Tools

<http://tools.pingdom.com/>

En su modo gratuito basta con indicar la URL y el test procede a llevar a cabo una petición contabilizando todas las peticiones derivadas, tiempo de carga, etc. Y ofreciendo una nota final de rendimiento. También se muestra un desglose de cada elemento que debe descargarse y el tiempo que ha necesitado.

Impleme

6.2. OPTIM

6.2.3. Herrar

Pingdom Tool

solarwinds

pingdom

Product

Pricing

Resources

Support

Tools

URL

www.example.com

Test from

Europe - United Kingdom - London

START TEST

The internet is fragile. Be the first to know when your site is in danger.

START YOUR FREE 14-DAY TRIAL

Web

Nobody Likes a Slow Website

We built this Website Speed Test to help you analyze your website load speed.

The test is designed to help make your site faster by identifying what about a webpage is fast, slow, too big, and so on.

We have tried to make it useful both for experts and novices alike. In short, we wanted it to be an easy-to-use tool built to help webmasters and web developers everywhere optimize their website performance.

About Pingdom

Pingdom offers cost-effective and reliable uptime and performance monitoring for your website.

We use more than 70 global polling locations to test and verify our customers' websites 24/7, all year long.

With Pingdom you can monitor your websites' uptime, performance, and interactions for a better end-user-experience.

Your customers will thank you.

State Colors

The following colors are used in the chart bars to indicate the different stages of a request.

<div></div> DNS	Web browser is looking up DNS information
<div></div> SSL	Web browser is performing an SSL handshake
<div></div> Connect	Web browser is connecting to the server
<div></div> Send	Web browser is sending data to the server
<div></div> Wait	Web browser is waiting for data from the server
<div></div> Receive	Web browser is receiving data from the server
<div></div> Blocked	Web browser is not ready to send

Content Types

The following icons are used to indicate different content types.

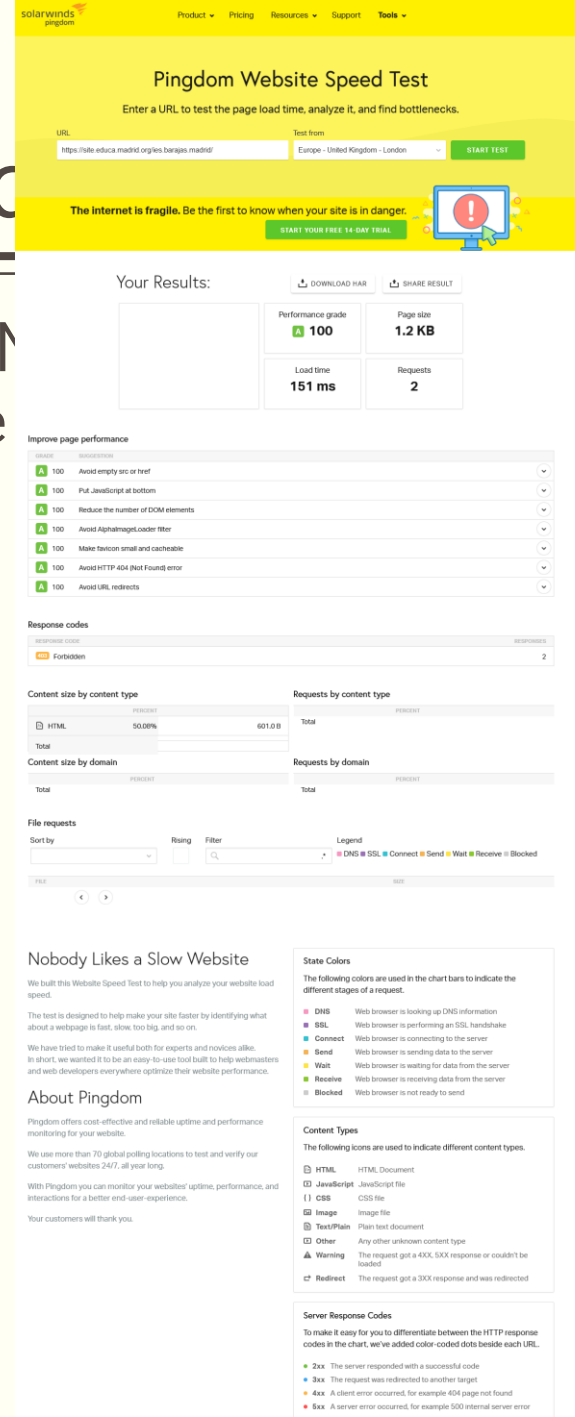
<div></div> HTML	HTML Document
<div></div> JavaScript	JavaScript file
<div></div> CSS	CSS file
<div></div> Image	Image file
<div></div> Text/Plain	Plain text document
<div></div> Other	Any other unknown content type
<div></div> Warning	The request got a 4XX, 5XX response or couldn't be loaded
<div></div> Redirect	The request got a 3XX response and was redirected

Implementación de la Web

6.2. OPTIMIZACIÓN DE IMAGENES

6.2.3. Herramientas online

Pingdom Tools



Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN DE INTERFACES

6.2.3. Herramientas online de comprobación de velocidad

Web Page Analyzer

<http://www.websiteoptimization.com/services/analyze>

Tras mostrar todo lo que ha necesitado descargar hace un análisis en distintas áreas: HTML, ficheros JS, etc y nos aporta recomendaciones al respecto


[home](#) > [services](#) > analyze

Web Page Analyzer - 0.98 - from Website Optimization

Free Website Performance Tool and Web Page Speed Analysis

Try our free web site speed test to improve website performance. Enter a URL below to calculate page size, composition, and download time. The script calculates the size of individual elements and sums up each type of web page component. Based on these page characteristics the script then offers advice on how to improve page load time. The script incorporates the latest best practices from [Website Optimization Secrets](#), web page size guidelines and trends, and web site optimization techniques into its recommendations.

Enter URL to diagnose:

Help Speed Up the Web

Tired of waiting for slow web sites? Support this free website optimization tool by linking to this page, or [WebPageAnalyzer.com](#). Help spread the word about this free website speed test by linking back to the analyzer with the example HTML below.

```
<a href="http://www.webpageanalyzer.com">Web Page Analyzer</a> - Free web page analysis tool calculates page size, composition, and download time. Gives speed recommendations based on best practices for usability, HCI, and website optimization.
```

Consider optimizing your site - with our [Website Optimization Secrets](#) book, [contacting us](#) about our optimization services, or our [Speed Tweak](#) tutorials.

Related Website Optimization Services

Try our targeted services to optimize your web site's ROI.

- [Speed Optimization and Analysis Service](#) now includes [Web Graphics Optimization](#)
- [Search Engine Marketing](#)
- [Website Development Services](#) - Includes Professional Redesigns

Version History

See the [version history](#) for all revisions.

[About the Book](#)
[About the Author](#)
[Buy @Amazon US](#)
[Buy @Amazon UK](#)
[Book News](#)
[Table of Contents](#)


Book News

[A Professional Web Designer's Thoughts on Websites, SEO...](#)

[Top 100 SEO Books by Techie Mania](#)

[The Third Road](#)

[New Polish and Simplified Chinese Translations of Website Optimization Book](#)

[Website Optimization Voted Best 14 Books of 2009](#)

[Kate Trgovac Reviews Website Optimization Secrets Book](#)

[Google Releases Page Speed](#)

[VoiceAmerica to Interview Author Andrew King](#)

[more book news »](#)

News Channels

[Aggregate \(home\)](#)

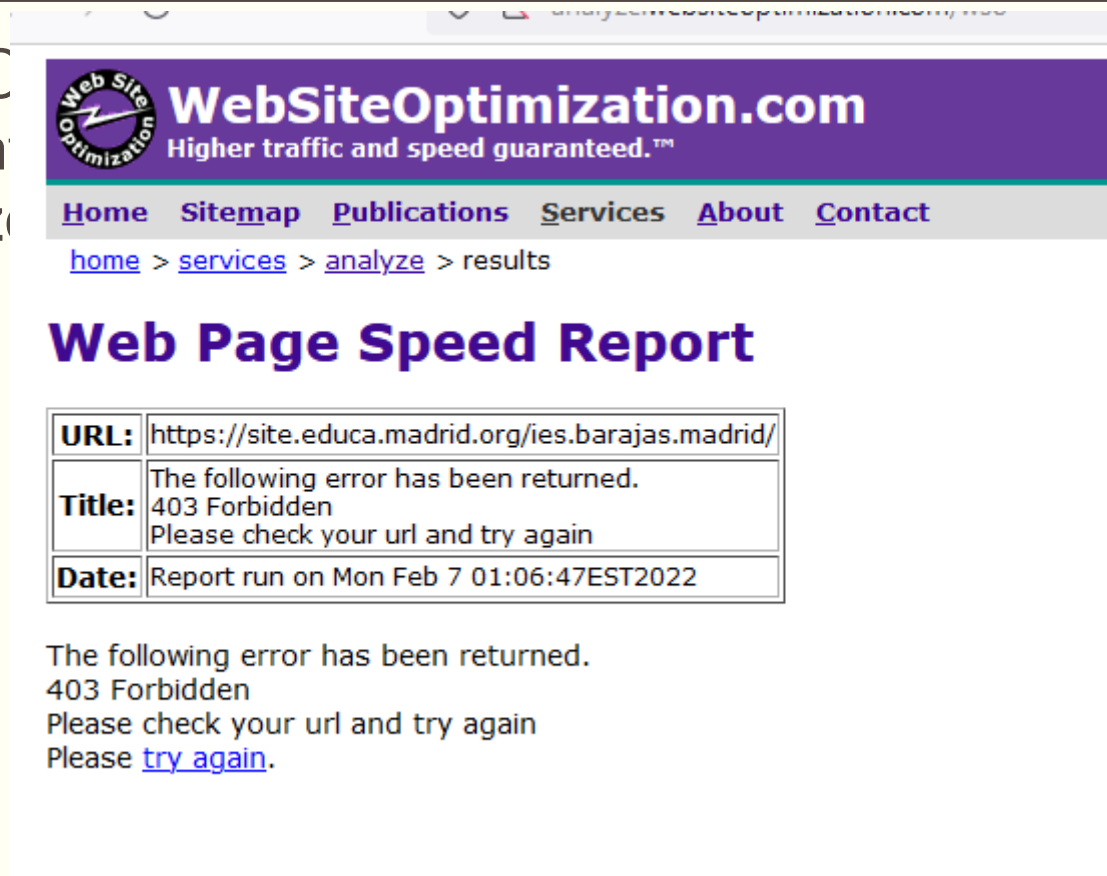
[Bandwidth Report](#)

Implementación de la Usabilidad en la Web

6.2. OPTIMIZACIÓN

6.2.3. Herramientas

Web Page Analyzer



The screenshot shows the WebsiteOptimization.com interface. The header is purple with the site's logo and name. A navigation bar contains links to Home, Sitemap, Publications, Services, About, and Contact. Below this is a breadcrumb trail: home > services > analyze > results. The main heading is 'Web Page Speed Report'. A table displays the analysis details, including the URL, the error message (403 Forbidden), and the report date. Below the table, the error message is repeated in plain text with a 'try again' link.

URL:	https://site.educa.madrid.org/ies.barajas.madrid/
Title:	The following error has been returned. 403 Forbidden Please check your url and try again
Date:	Report run on Mon Feb 7 01:06:47EST2022

The following error has been returned.
403 Forbidden
Please check your url and try again
Please [try again](#).

Implementación de la Usabilidad en la Web

6.3. TESTING

Cuando se desarrolla una aplicación Web con distintas páginas, comprobar que todo funciona correctamente resulta una tarea necesaria, pero muy tediosa, más aún cuando hay que hacer la cada vez que se hace un cambio. Por suerte contamos con herramientas como Selenium.

Aplicación de ejemplo

Para mostrar el uso de selenium vamos a utilizar una sencilla aplicación desarrollada totalmente en HTML5, css y el framework AngularJS. Se trata de un simple gestor de contactos

Implementación de la Usabilidad en la Web

6.3. TESTING

Aplicación de ejemplo

La cabecera con los botones y el pie con los enlaces se mantienen constante en todas las pantallas. Al pulsar sobre la lista de contactos vamos a una página donde se muestran los contactos: cada uno es un registro con un nombre y un teléfono.

Los registros se almacenan de forma temporal en un array de JavaScript; obviamente podríamos haber utilizado otras formas de almacenamiento que provee HTML5 o incluso apoyarnos en un backend, pero para el propósito de comprobar el testeo es suficiente: a Selenium no le importa cómo este desarrollada la aplicación, ya que simplemente se dedicará a emular el comportamiento de un usuario. Por ejemplo, pulsando los botones Detalle, Modificación y Borrado:

Implementación de la Usabilidad en la Web

6.3. TESTING

Aplicación de ejemplo

El botón detalle (Detail) nos lleva simplemente a una página donde se muestra uno de los contactos con todos los campos disponibles, además, repetir los botones de modificar y eliminar para tenerlos siempre disponibles:

Id es un valor que se crea automáticamente y que no se puede modificar. Es crucial ya que sirve para poder distinguir un contacto de otro. Si pulsamos en el botón de modificar podremos cambiar los valores de ese registro.

Por otro lado, eliminar no tiene una pantalla aparte, simplemente se elimina directamente el elemento y se vuelve al listado. En el caso de querer añadir un nuevo contacto, se muestra un formulario como el siguiente para, a continuación, acabar también en el listado:

Implementación de la Usabilidad en la Web

6.3. TESTING

Aplicación de ejemplo

Todos estos pasos los podemos testear con herramientas como Selenium, tanto a través de un plugin de Firefox como a través de la programación.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Esta herramienta Open Source nos permite crear toda clase de test para comprobar el correcto funcionamiento de aplicaciones Web simulando las acciones de un usuario. Selenium nos permite desarrollar el testeo de distintas maneras y también podremos llevar a cabo el testing a través de herramientas de programación o de plugins de navegador

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

La forma más directa de crear testeos automatizados con Selenium es a través de su addon para Firefox, el cual podemos instalar fácilmente a través del repositorio oficial de addons. Una vez hecho esto, veremos este pequeño icono en el mismo



Inicio > Extensiones > Selenium IDE



Selenium IDE

Ofrecido por: seleniumhq.org

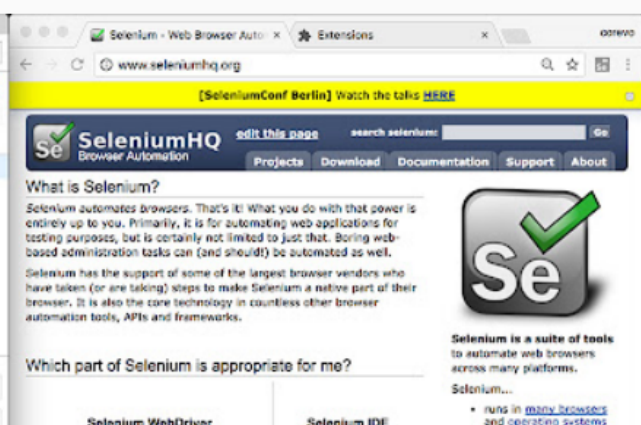
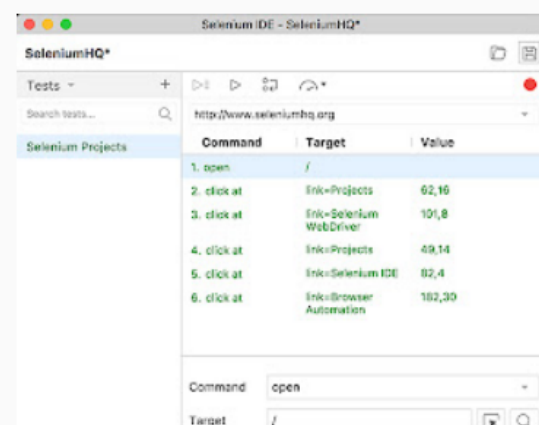
★★★★★ 227

Herramientas para desarrolladores

500.000+ usuarios

Añadir a Chrome

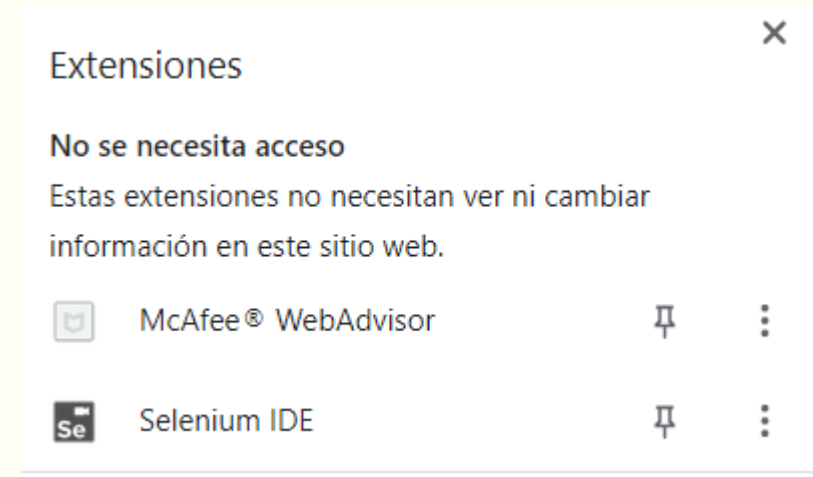
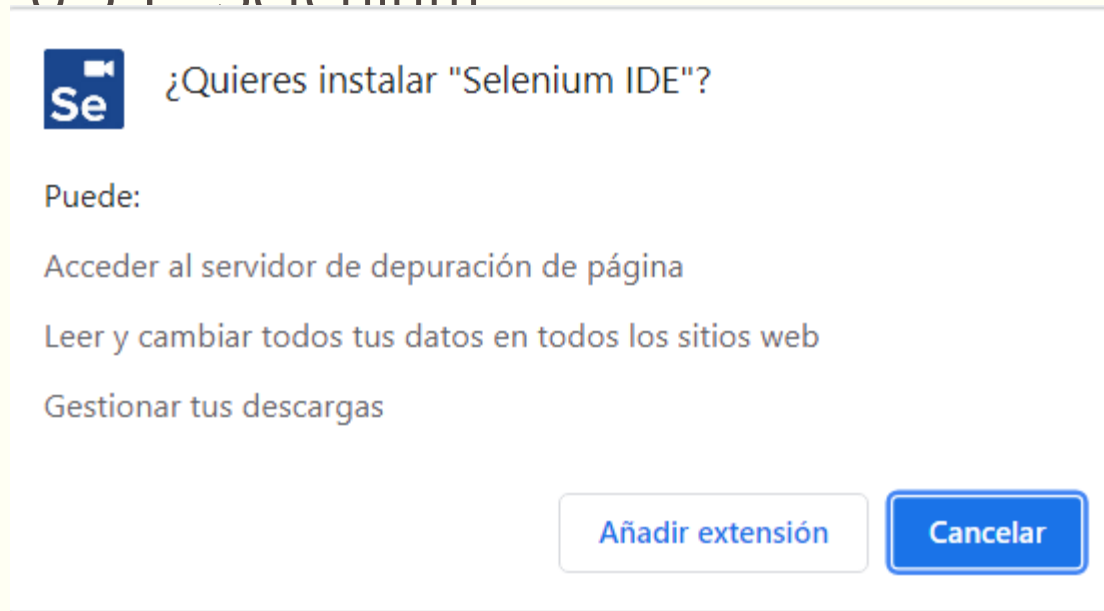
- Descripción general
- Prácticas de privacidad
- Reseñas
- Ayuda
- Relacionados

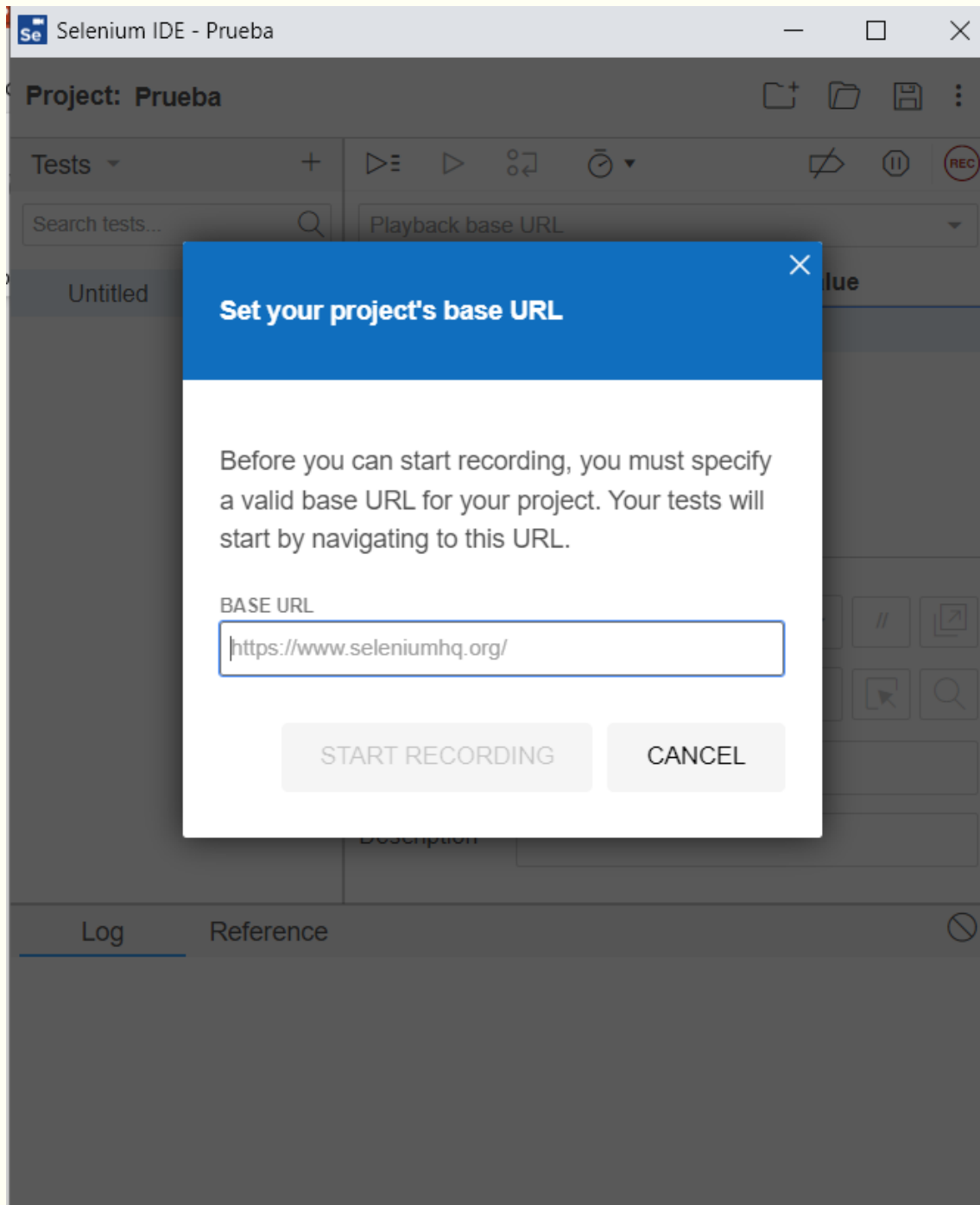


Implementación de la Usabilidad en la Web

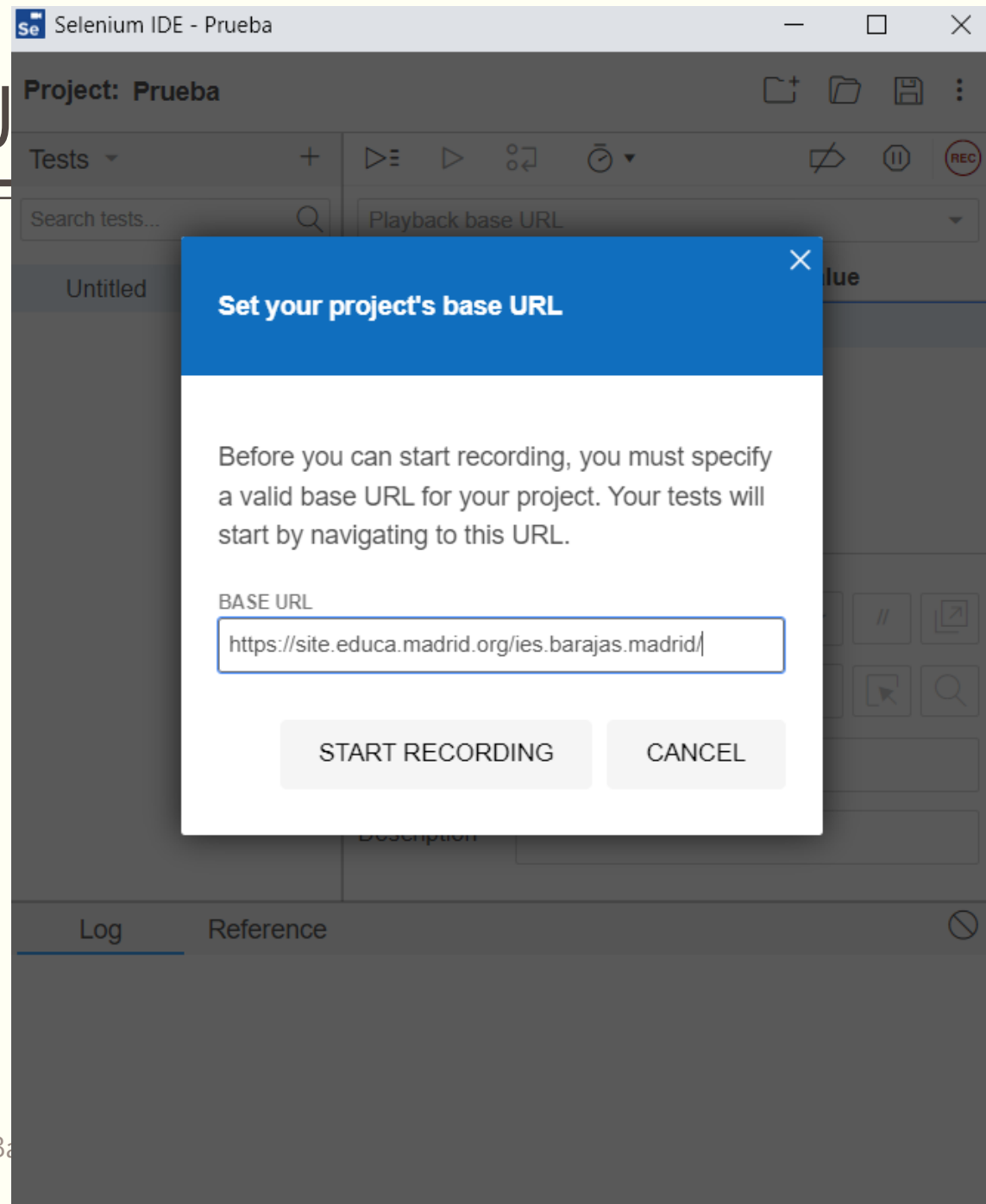
6.3. TESTING

6.3.1 Selenium





U



IES Ba

Selenium IDE - Prueba*

Project: Prueba*

Tests +

Search tests...

https://site.educa.madrid.org

	Command	Target	Value
1	open	/ies.barajas.madrid/	
2	set window size	1536x832	
3	click	id=archives-dropdown-3	
4	select	id=archives-dropdown-3	label=regex enero 2022

Command

Target

Value

Description

Log Reference

Name your new test

Please provide a name for your new test.

TEST NAME

Primerd

You can change it at any time by clicking the : icon next to its name in the tests panel.

OK

LATER

Implementa

la Web

6.3. TESTING

6.3.1. Selenium Seleniun-Addo

Selenium IDE - Prueba*

Project: Prueba*

Tests + [Run] [Repeat] [Clock] [Filter] [REC]

Search tests... Run current test Ctrl+R

	Command	Target	Value
✓ Primero*			
2	✓ set window size	1536x832	
3	✓ click	id=archives-d ropdown-3	
4	✓ select	id=archives-d ropdown-3	label=regex p:enero 2022 \s+\(1\)

Command [] [//] []

Target [] [] []

Value []

Description []

Log Reference

Running 'Primero' 12:18:42

1. open on /ies.barajas.madrid/ OK 12:18:43

2. setWindowSize on 1536x832 OK 12:18:43

3. click on id=archives-dropdown-3 OK 12:18:43

4. select on id=archives-dropdown-3 with value label=regex:enero 2022\s+\(1\) OK 12:18:45

'Primero' completed successfully 12:18:46

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

Al pulsarlo se abrirá la herramienta con la cual ya podremos hacer tests. En realidad es tan sencillo como pulsar el botón de grabación, utilizar la web que deseemos testear, y todas las acciones quedarán grabadas en Selenium. De esa manera, podremos reproducir los test siempre que queramos

En este addon se gestionan dos tipos de elementos, los cuales se pueden crear, guardar, eliminar, exportar, etc:

1. Test-Case: es una prueba o conjunto de acciones concretas.
2. Test-Suite: es un conjunto de casos o Test-Case.
3. Por defecto se crea un Test-Suite con un Test-Case al que le podremos añadir más casos

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

Crear un test

Primero, debemos llevar manualmente el navegador a la página o aplicación que queramos testear. Una vez iniciado el addon podemos pulsar el botón rojo de grabación o bien ir a

Menú > Action > Record

Tras eso, no tenemos más que utilizar la web como si fuéramos un usuario.

Podemos agrupar todas las acciones en el mismo test o bien hacer pequeños test deteniendo la grabación y creando nuevos Test-Case. Conforme vayamos usando la página, veremos como el addon va registrando todas las acciones que luego podría reproducir:

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

Crear un test

Primero, debemos llevar manualmente el navegador a la página o aplicación que queramos testear. Una vez iniciado el addon podemos pulsar el botón rojo de grabación o bien ir a

Menú > Action > Record

Tras eso, no tenemos más que utilizar la web como si fuéramos un usuario.

Podemos agrupar todas las acciones en el mismo test o bien hacer pequeños test deteniendo la grabación y creando nuevos Test-Case. Conforme vayamos usando la página, veremos como el addon va registrando todas las acciones que luego podría reproducir:

Implementa

la Web

6.3. TESTING

6.3.1. Selenium Seleniun-Addo

Selenium IDE - Prueba*

Project: Prueba*

Tests + [Run] [Repeat] [Clock] [Filter] [REC]

Search tests... Run current test Ctrl+R

	Command	Target	Value
✓ Primero*			
2	✓ set window size	1536x832	
3	✓ click	id=archives-d ropdown-3	
4	✓ select	id=archives-d ropdown-3	label=regex p:enero 2022 \s+\(1\)

Command [] [//] []

Target [] [] []

Value []

Description []

Log Reference

Running 'Primero' 12:18:42

1. open on /ies.barajas.madrid/ OK 12:18:43

2. setWindowSize on 1536x832 OK 12:18:43

3. click on id=archives-dropdown-3 OK 12:18:43

4. select on id=archives-dropdown-3 with value label=regex:enero 2022\s+\(1\) OK 12:18:45

'Primero' completed successfully 12:18:46

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

Crear un test

Una vez registradas las acciones, ya podemos pulsar el botón play para reproducirlas. Esta es una de las partes más vistosas de Selenium ya que nos mostrará cómo efectivamente toma el control del navegador y reproduce esas acciones a toda velocidad. Al final nos indicará si hay algún error o algún comportamiento inesperado o, por el contrario, el color verde que significa que todo se ha reproducido correctamente. Si alguno de los pasos falla, indica de forma individual en el listado de acciones.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.1. Selenium

Seleniun-Addon

Selenium API

Una de las grandes aportaciones de Selenium es que dispone de APIs para desarrollar tests en distintos lenguajes de programación: Java, C#, Python, JavaScript, etc. Una vez desarrollado el test Selenium pondrá en ejecución un navegador y simulará las operaciones que les hayamos indicado. En caso de no recibir alguna respuesta esperada el test falla, de tal manera que podemos detectar errores.

Implementa

la Web

6.3. TESTING

6.3.1. Selenium Seleniun-Addor Selenium API

Selenium IDE - Prueba*

Project: Prueba*

Tests +

Search tests...

https://site.educa.madrid.org

	Command	Target	Value
3	click	id=archives-d ropdown-3	
4	select	id=archives-d ropdown-3	label=regex p:enero 2022 \\s+\\(1\\)

Command

Target

Value

Description

Log Reference

4. select on id=archives-droppaown-3 with value label=regexp:enero 2022\\s+\\(1\\) OK 12:18:45

'Primero' completed successfully 12:18:46

Running 'Primero' 12:24:15

1. open on /ies.barajas.madrid/ OK 12:24:17

2. setWindowSize on 1536x832 OK 12:24:17

3. click on id=archives-dropdown-3 Undetermined 12:24:17
Aborting...

'Primero' was aborted 12:24:38

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Selenium es una gran herramienta con la que podemos crear test e2e (end-to-end) de manera muy ágil en cualquier ordenador que tenga el Firefox o Chrome instalado. Pero generalmente los test se hacen mucho más específicos, se programan y se integran dentro de builds o procesos de integración continua

Podemos desarrollar tests e2e (end to end) para Selenium con los lenguajes más populares, y en este caso mostraremos como se llevan a cabo desde uno de los lenguajes esenciales de la Web: JavaScript. Además haremos uso de la infraestructura que hemos utilizado a lo largo de todo este libro.



Protractor

end to end testing for Angular

[View on GitHub](#)[Follow @ProtractorTest](#)

Protractor is an end-to-end test framework for Angular and AngularJS applications. Protractor runs tests against your application running in a real browser, interacting with it as a user would.

Test Like a User

Protractor is built on top of WebDriverJS, which uses native events and browser-specific drivers to interact with your application as a user would.

For Angular Apps

Protractor supports Angular-specific locator strategies, which allows you to test Angular-specific elements without any setup effort on your part.

Automatic Waiting

You no longer need to add waits and sleeps to your test. Protractor can automatically execute the next step in your test the moment the webpage finishes pending tasks, so you don't have to worry about waiting for your test and webpage to sync.

Setup

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

La herramienta que nos permite organizar rápidamente los tests es Protractor. Este es un proyecto ideado para llevar a cabo los test e2e de aplicaciones basadas en el framework Angular.js, aunque lo podemos usar perfectamente en cualquier aplicación web, ya que al ser precisamente e2e no nos importan los lenguajes y librerías que se utilicen, sino el uso de la web en sí.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Protractor se ejecuta, como muchas otras herramientas Web, sobre Node.js y puede instalarse rápidamente a través de su gestor de paquetes:

```
npm install -g protractor
```

Esto instalará de forma global en el sistema el propio Protractor y el webdriver de selenium que se encargará de ejecutar los propios test. Antes de hacer nada debemos actualizar el webdriver:

```
webdriver-manager update
```


Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

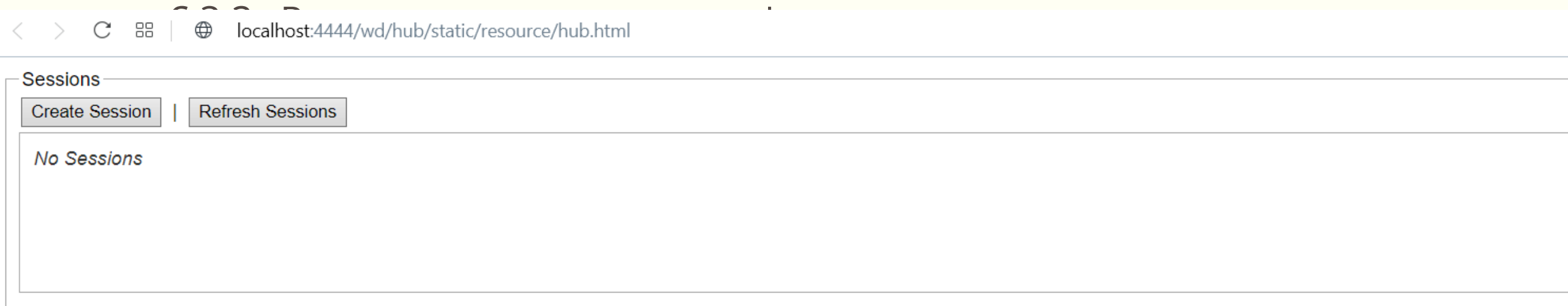
Y una vez hecho lo dejamos en marcha como servicio para que Protractor le ordene aquello que debe testear:

`webdriver-manager start`

Ese comando iniciará un servidor de selenium en el puerto 4444 cuyo estado se puede comprobar en <http://localhost:4444/wd/hub>.

Implementación de la Usabilidad en la Web

6.3. TESTING



Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Todo esto se puede organizar en un proyecto con su fichero `package.json` y con una serie de scripts predefinidos. En el ejemplo en que se basa este apartado todo ese trabajo ya está hecho.

Protractor necesita un fichero de configuración donde indicamos algunas opciones esenciales, como ubicación de los ficheros de testeo, el o los navegadores a utilizar, etc. Por defecto, basta con crear un fichero llamado `protractor-config.js`, aunque puede ser cualquier otro siempre que al ejecutar Protractor se indique que fichero de configuración se requiere.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Un ejemplo minimo seria:

```
exports.config = {  
    seleniumAddress: 'http://localhost: 4444/wd/hub',  
    specs: ['test-spec.js'];  
};
```

Donde specs es un array con todos los ficheros de testeo. Con esa configuración los test se lanzarán utilizando Chrome.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Si queremos utilizar Firefox e indicar un directorio llamado e2e donde se encuentran todos los tests el aspecto es el siguiente:

```
exports.config = {  
    allScriptsTimeout: 11000,  
    specs: [  
        "e2c/* spec.js" ],  
    capabilities: [  
        'browserName': 'Firefox'],  
    chromeOnly: true,  
    baseUrl: 'http://localhost:8080/',  
    framework: 'jasmine',  
    jasmineNodeOpts: {  
        defaultTimeout Interval: 30000  
    }  
};
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Como se puede ver también se establece una dirección o url base para la aplicación:

```
baseUrl: 'http://localhost:8080/'
```

Todos los tests se lanzarán a esa dirección. Puede cambiarse esa dirección con cualquier otra, donde el test se esté ejecutando. Es una variable que utilizaremos en todos los tests. Para ejecutarlos test basta con hacer:

```
protractor protractor-conf.js
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

En cuanto al framework utilizado para los tests, protractor nos permite configurar el que queramos y en este caso optamos por el popular Jasmine.

GET STARTED

DOCS

SUPPORT

RELEASES

GITHUB

FAST

Low overhead, jasmine-core has no external dependencies.

BATTERIES INCLUDED

Comes out of the box with everything you need to test your code.

NODE AND BROWSER

Run your browser tests and Node.js tests with the same framework.

```
describe("A suite is just a function", function() {  
  var a;  
  
  it("and so is a spec", function() {  
    a = true;  
  
    expect(a).toBe(true);  
  });  
});
```

Sample Code

Jasmine is a behavior-driven development framework for testing JavaScript code. It does not depend on any other JavaScript frameworks. It does not require a DOM. And it has a clean, obvious syntax so that you can easily write tests.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Jasmine, es un framework de testeo BBDD cuya sintaxis es muy expresiva, ya que hace que la redacción de tests se asemejen precisamente a historias. El aspecto general que tiene un test con Jasmine es el siguiente:

```
describe('Comprobación de la web de Garceta', function()  
it('La página de inicio se carga correctamente', function t)  
browser.get('http://www.editorialgarceta.es');  
expect browser.getTitle().toEqual('Editorial Garceta ");  
});});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

describe: nos sirve para agrupar las funcionalidades que queremos verificar bajo una misma estructura.

it: contiene el test concreto que se va a realizar.

expect: es el resultado que esperamos obtener. Podemos tener más de uno dentro del bloque it.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Preparación del entorno

Antes de eso podemos tener operaciones para descargar la página o seleccionar determinados elementos. A continuación vamos a ver cómo se testearía cada uno de los escenarios de la aplicación.

Encada uno se pretende mostrar distintas maneras de seleccionar, acceder y manipular los elementos de la página web.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test index

Se trata de comprobar la página índice. La comprobación básicamente consiste en solicitar la url inicial y comprobar que efectivamente se ha cargado. ¿Cómo...? En este caso se verifica el título de la página.

```
describe('Contact App index page', function() {  
  it('should have this title', function()  
    browser.get('/');  
  expect(browser.getTitle().toEqual('Contact App'));});});
```

Puede pensarse cualquier otra forma de testeo, por supuesto, como, por ejemplo, comprobar un contenido clave dentro del cuerpo de la pagina.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test index

Se trata de comprobar la página índice. La comprobación básicamente consiste en solicitar la url inicial y comprobar que efectivamente se ha cargado. ¿Cómo...? En este caso se verifica el título de la página.

```
describe('Contact App index page', function() {
```

```
  it('should have this title', function()
```

```
    browser.get('/');
```

```
    expect(browser.getTitle().toEqual('Contact App'));});});
```

Puede pensarse cualquier otra forma de testeo, por supuesto, como, por ejemplo, comprobar un contenido clave dentro del cuerpo de la pagina.

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test home

Se trata de comprobar la página índice. Antes de cada bloque it se ejecutará el código beforeEach. Siempre lo utilizaremos para cargar la página que queremos testear. Para cada test la pagina comenzará desde una carga limpia.

```
describe('Contact App home page', function() {  
  it('beforeEach (function () {  
    browser.get('#/home');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test home

Comprobamos si se carga la página home a través de la url que se queda en el navegador, del contenido de un título h2 y de un

```
 párrafo: it'should load home page', function() {  
  expect (browser.getTitle).toEqual('Contact App');  
  expect browser.getCurrentUrl().toEqual (browser.baseUrl +  
  '#/home');  
  expect (element by.css('main section > h2')).getText().toEqual  
  ('Contacts');  
  expect (element by.css('main section p')).getText().toEqual('Welcome  
  to contacts app');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test home

Comprobamos si se carga la página home desde la página index, haciendo clic en el enlace home y comprobando otra vez el título de la página y la url de la misma:

```
It('should load home page from index', function() {  
  browser.get('/');  
  element by.css('a[href="/home"]').click();  
  expect (browser.getTitle().toEqual('Contact App'));  
  expect (browser.getCurrentUrl()).toEqual (browser.baseUrl +  
    '#/home');});});
```


Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test about

Se trata de comprobar que la página about o acerca de se carga correctamente:

```
describe ('Contact App about page', function () {  
  beforeEach(function()browser.get('#/about');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test about

Comprobamos si se carga la página about a través de la url que se queda en el navegador y del título de la página:

```
It('should lead about page', function()
```

```
expect (browser.getTitle().toEqual ('Contact App');
```

```
expect (browser.getCurrentUrl().coEqual (browser.baseUrl + '/about');
```

```
expect element (by.id('title')).getText().toEqual('About Contact App');
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test about

Comprobamos si se carga la página about desde la página index, haciendo click en el enlace about y comprobando otra vez el título de la página y la url de la misma:

```
It(' should load about page from index', function()  
  browser.get('/');  
  element (by.css('a[href="#/about"]')).click();  
  expect (browser.getCurrentUrl()) .toEqual (browser.baseUrl +  
    '#/about');  
  expect element (by.id('title')).getText().toEqual('About Contact  
App');});});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test list

Se trata de comprobar la página que lista todos los contactos que tenemos. En definitiva, comprobamos cosas similares a las básicas, además de ver si realmente hay una lista de contactos

```
describe ('Contact Apr list page', function() {  
  beforeEach(function () {  
    browser.get('#/contact/list');  
  });  
});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test list

Comprobamos si se carga la página de la lista a través de la url que se queda en el navegador, del contenido de un título h2 y de un párrafo. Observa como podemos usar selectores al estilo de CSS o como hace jQuery.

```
It('should load contact list page', function()
expect (browser.getTitle()).toEqual('Contact App').
expect browser.getCurrentUrl().toEqual
(browser.baseUrl()/'contact/list');
expect element by.css('section > h2')).getText().toEqual ('Contact
List');
expect elementid by.css('section p')).getText().toEqual ('Contact list'););
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test list

Comprobamos si se carga la página de la lista verificando si existe el título de Contacts y al final si tenemos más de un elemento cuya clase CSS es contact

```
it('should have contacts, one at least', function(){  
  expect(element(by.id('contacts')).isPresent().toBeTruthy():  
  expect(elementby.tagName(n)).getText().toEqual ('Contacts');  
  expect(element.all (by.css('contact')).count().toBeGreaterThan(0)  
});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test list

Comprobamos que al menos se carga el contacto número 1, a través de su id. y de los datos de nombre y teléfono

```
it('should load at least contact number 1', function() {  
  expect (element lby.id('contacti')).isPresent().toBeTruthy();  
  expect (element (by.css('contact1  
.contactname')).getText().toEqual('JonSnow');  
  expect element  
(hy.css('contact1.contactnumber')).getText().toEqual('666332322');
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test list

Comprobamos si se carga la página con la lista haciendo click sobre su enlace, del contenido de un titulo de la página y la url:

```
It(' should load contact list page from index', function ()
```

```
browser.get (browser.baseUrl());
```

```
element (by.css('a href="#/contact/list"]')).click();
```

```
expect (browser.getTitle() .toEqual ('Contact App');
```

```
expect browser.getCurrentUrl().toEqual
```

```
(browser.baseUrl+ '/contact/list');});});
```


Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test detail

Se trata de comprobar la página que muestra el detalle de un registro concreto.

```
describe('Contact App detail page', function() {  
  beforeEach(function() browser.get(' +/contact/detail/1');  
})
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test detail

Comprobamos si se carga la página de un contacto concreto a través de la url que se queda Enel navegador, del contenido de un titulo h2 que va seguido de un elemento section y de una etiqueta h3 que está en algún lugar dentro del elemento section:

```
it('should load contact detail page', function()
expect (browser.getTitle()).toEqua ('Contact App');
expect (browser.getCurrentUrl()).toEqual (browser.baseUrl
+#/contact/detail/1');
expect element (by.css' section > h2')).getText().toEqual('ContactDetail');
expect (element (by.css('section h3')).getText().toContain ('Contact
detail:Jon Snow'
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test detail

Comprobamos si se carga la página de detalle y se comprueban los campos concretos que se deberían cargar

```
It('should load correct details', function () {  
  expect(element.all(by.css('contact')).count().toBe(1);  
  expect(element(by.casi('contact.name')).getText().toEqual('Name:  
JonSnow');  
  expect(element(by.casi('contactid')).getText().toEqual('Id: ');  
  expect(element(by.casi('contactnumber')).getText().toEqual('Number:666332322');  
});});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test new

Se trata de comprobar la página que nos permite introducir nuevos usuarios. Para cada prueba empezaremos por cargar la pagina que crea nuevos registros. Comprobaremos si realmente cargamos esa página y, además, si funciona o no su formulario.

```
describe ('Contact App new contact page'. Function() {  
  beforeEach(function() ;  
  browser.get('#/contact/new');  
});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test new

Comprobamos si se carga la página de nuevos registros a través de la url que se queda en el navegador, del contenido de un titulo h2 y de un párrafo

```
It('should load new contact page', function() {  
  expect(browser.getTitle()).toEqual('Contact App')  
  expect(browser.getCurrentUrl().toEqual(browser.baseUrl +  
    './contact/new'):  
  expect(element(by.css('section > h2')).getText().toEqual('New  
Contact');  
  Expect(element(by.css('section p')).getTexti.toContain('New contact');
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test new

Comprobamos si se carga la página de nuevos registros buscando a ver si hay un formulario y si ese formulario contiene los campos necesarios. Los tests preguntan si esos elementos están presentes.

```
it('should lead a form', function() {  
  expect(element.all(by.tagName('form')).count().toBe(1) :  
  expect (element (by.css('form input name')).is Present().toBe(true);  
  expect element (by.css('form input#number')).is Present().toBe(true);
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test new

Comprobamos ahora mismo si efectivamente el formulario de añadir nuevos registros funciona introduciendo contenidos y pulsando el botón submit para enviarlo. Tras eso la aplicación debería cargar la página con la lista y comprobamos que el nuevo registro ya aparece.

```
It('should write input and submit fom', function()
element(by.model('contact.name')).sendKeys('Frotractor');
element(by.model('contact number')).sendKeys('424242');
element(by.css('form input ivalue="Save new contact"]')).click();
expect(browser.getCurrentUrl().toEqual browser.baseUrl + "/contact/list");
expect(element.all(by.css
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test new

```
It('should write input and submit fom', function()
element (by.model('contact.name').sendKeys ('Protractor');
element (hy.todel('contact number')).sendKeys('424242');
element by.css('form input ivalue="Save new contact"]')).click();
expect (browser.cetCurrentUrl().toEqual browser.baseUrl + "
/contact/list'l:
expect element.all (by.ess
ContainingText('.contactnumber'.1424242')).count().tobe (1);
expect (element.all by.casContainingTexti'.contactname',Protractor')).
Court l)). Tobe 11);:
```


Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test update

Se trata de comprobar la pagina más compleja, que es la de actualización. Una vez más tenemos que verificar, por un lado, que las páginas tienen el contenido esperado y que, además, la funcionalidad se cumple.

```
describe('Contact App update page', function() {  
  beforeEach(function()  
    browser.get('#/contact/update/i');
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test update

Comprobamos si se carga la página de modificación a través de la url que se queda en el navegador, del contenido de un titulo h2 y de un párrafo

```
it(' could load contact update page', function () {  
  expect browser.getTitle().toEqual ('Contact App');  
  expect browser.getCurrentUrl().toEqual (browser.baseUrl  
+ '#/contact/update/1');  
  expect (element (by.css('section >  
h2')).getText().toEqual('ContactOpdate');  
  expect (element (by.css('section p')).getText().tecontain contact  
update');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test update

Comprobamos si se carga la página de modificación de registro buscando a ver si hay un formulario y si ese formulario contiene los campos necesarios con el contenido concreto de ese registro que vamos a modificar ya cargado. Los tests preguntan si esos elementos están presentes.

```
it('should load a form', function() {  
  expect(element.all(by.tagName('form')).count().toBe(1);  
  expect(element(by.css('form  
input name [type="text"]')).is Present().toBe(true);  
  expect(element(by.model('contact.name')).getAttribute('value'))  
    .toEqual('JonSnow');  
  expect(element(by.model('contact.number')).getAttribute('value'))  
    .toEqual('666332322');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test update

Comprobamos a continuación si efectivamente el formulario modifica el registro, introduciendo cambios en las cajas de texto y pulsando el botón submit para enviarlo. Tras eso la aplicación debería cargar la página con la lista y comprobamos que el registro modificado tiene los nuevos valores:

```
it('should write input, submit form and update correctly', function() {  
  element(by.model('contact.name')).clear().sendKeys('Jon Targaryen');  
  element(by.model('contact.number')).clear().sendKeys('657');  
  element(by.css('form input[value="Update Contact"]').click();  
  expect(browser.getCurrentUrl().to Equal (browser.baseUrl - "/contact/list");  
  expect(element(by.css('Containing Text' contact.contactname, 'JonTargaryen')).is  
  Present().toBe(true);expect(element(by.css('ContainingText('#contactl  
  .contactnumber', '667')).is Present().toBe(true);
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test delete

Por último, comprobamos que la funcionalidad de eliminar sea efectiva:

```
describe('Contact App delete page', function()  
beforeEach(function() {  
browser.get('/contact/list');});
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test delete

La funcionalidad de eliminar funciona desde dos puntos, ahora comprobaremos su funcionalidad desde la página de la lista de elementos. Hacemos un clic sobre el enlace de eliminar un registro concreto, para luego verificar que ese elemento no aparece en la lista.

```
it'should delete a contact with one click', function()
expect (browser.getTitle()) .toEqual ("Contact App:
expect browser.getCurrentUrl().toEqual (browser.baseUrl -* 1/contact/list'li
expect element (by.cssContainingtext t'contact 1 .contactname', 'JonSnow')).is
present (1. to Be (true);
expect lelement (by.css ContainingText('contactl .contactnumber'.666332322')).is
Present().toBe(true):
element (by.css('href="/contact/delete/1" )}).click();
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test delete

```
expect (browser.getCurrentUrl()).toEqual (browser.baseUrl + '#/contact/list');  
expect (element (by.cssContainingText('contactl.contactname', 'Jon Snow')). is  
present().toBefalse);  
expect (element (by.cssContainingText("#contacti  
.contactnumber",666332322'])).isPresent) 1. toBe (false);11:
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test delete

Ahora hacemos la misma comprobación pero desde la página de detalle. En teoría debería funcionar exactamente igual. Pulsamos en el enlace de eliminar y luego verificamos que efectivamente no aparece en la lista porque se elimina.

```
it'should delete from detail page', function() {  
  element(by.css('a[href="#/contact/detail/2"]').click();  
  expect(browser.getCurrentUrl().toEqual(browser.baseUrl + '#/contact/detail/2');  
  expect(element(by.css('section h3')).getText().toContain('Contact detail:Merlin');  
  expect(element(by.css('.contactid')).getText().toEqual('Id: 2');  
  element(by.css('a[href="#/contact/delete/2"]').click();  
  expect(browser.getCurrentUrl().toEqual(browser.baseUrl + '#/contact/list');  
  expect(element(by.cssContainingText('#contact2.contactname', 'Merlin')).isPresent()  
    .toBe(false);  
  expect(element  
    (by.cssContainingText('#contact2.contactnumber', '777000777')).isPresent()).toBe(false);1)
```


Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.2. Protractor: tests programados

Test delete

Ahora hacemos la misma comprobación pero desde la página de detalle. En teoría debería funcionar exactamente igual. Pulsamos en el enlace de eliminar y luego verificamos que efectivamente no aparece en la lista porque se elimina.

```
it'should delete from detail page', function() {  
  element(by.css('a[href="#/contact/detail/2"]').click();  
  expect(browser.getCurrentUrl().toEqual(browser.baseUrl + '#/contact/detail/2');  
  expect(element(by.css('section h3')).getText().toContain('Contact detail:Merlin');  
  expect(element(by.css('.contactid')).getText().toEqual('Id: 2');  
  element(by.css('a[href="#/contact/delete/2"]').click();  
  expect(browser.getCurrentUrl().toEqual(browser.baseUrl + '#/contact/list');  
  expect(element(by.cssContainingText('#contact2.contactname', 'Merlin')).isPresent()  
    .toBe(false);  
  expect(element  
    (by.cssContainingText('#contact2.contactnumber', '777000777')).isPresent()).toBe(false);1)
```

Implementación de la Usabilidad en la Web

6.3. TESTING

6.3.3. Verificación de la usabilidad en diferentes navegadores y tecnologías

BrowserShots

<http://browsershots.org/>

Esta es una herramienta online que es capaz de sacar un pantallazo para toda una gama de distintos navegadores. Nosotros no nos tenemos que preocupar más que poner una URL y podremos ver el resultado para todos los navegadores imaginables.

Una vez enviada la dirección, el resultado no es inmediato y tendremos que esperar aproximadamente 2 o 3 minutos, pero luego este es un ejemplo de lo que podemos obtener:



Scale your E-Commerce

Anticipate Heavy Traffic

Easy load testing solution for Magento, Shopify, Woocommerce and many e-commerce solutions

gatling.io

[OPEN](#)[Browser Compatibility Test](#)[US Stock Filings](#)[HK Stock Filings](#)

Enter URL Here:

[Submit](#)

Linux



Windows



Mac



BSD



No active screenshot factories. Please try again later.



What is Browsershots?

Browsershots makes screenshots of your web design in different operating systems and browsers. It is a free open-source online web application providing developers a convenient way to test their website's browser compatibility in one place. When you submit your web address, it will be added to the job queue. A number of distributed computers will open your website in their browser. Then they will make screenshots and upload them to our central dedicated server for your review.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

Las características de las aplicaciones han evolucionado desde la llegada de la web y, en especial, desde el uso masivo de los smartphones. Incluso más allá de los ordenadores convencionales, toda clase de dispositivos se están incorporando en el día a día, desde electrodomésticos o vehículos hasta ropa, relojes, etc. Por no hablar del IoT o Internet of Things, un concepto que supone integrar en las redes todo lo que nos rodea.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

Tengan la forma que tengan, lo cierto es que como mínimo en cada bolsillo se lleva un ordenador con aplicaciones funcionando. Los usuarios tienen todas las facilidades para instalar nuevas aplicaciones y éstas se actualizan periódicamente. Este difiere enormemente del escenario tradicional de los programas, donde se desarrollaba una aplicación en base a los requerimientos de un cliente, se instalaba en ordenadores fijos y era usada por ellos.

Hoy en día se desarrollan aplicaciones que se usan de forma masiva y los usuarios, de toda clase de perfiles, demandan funcionalidades, cambios y reportan un feedback continuo. Eso obliga a los creadores a aplicar metodologías de desarrollo ágiles para poder responder a esas demandas. Además, se precisan de sistemas que permitan hacer cambios en los programas asegurando que estos siguen funcionando, aplicando:

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

- Validaciones
- Optimizaciones
- Test unitarios
- Publicación
- Etc.

Todos esos pasos deben ejecutarse una y otra vez y eso es en lo que consiste el concepto de integración continua: un proceso que siguen los desarrolladores de software en el que todos los cambios que lleva a cabo el equipo de programación se fusionan, se testean y se publica. El objetivo es, precisamente, desarrollar un proyecto de forma ágil con especial atención en que, además de que todo funcione, se deben prevenir problemas de integración.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

Aparte de integrar el código, también se integran todas las herramientas en un proceso que va mucho más allá de una simple compilación: además de compilar, hay que testear, optimizar, y esto obliga a utilizar: control de versiones, ejecución de tareas, frameworks de testing, navegadores, etc.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

Travis es una herramienta PaaS de integración continua que, por tanto, se encarga de integrar y testear todo el código en la nube. Para ello se descarga el código desde el repositorio de software y a partir de ahí lleva a cabo todo el proceso en un maquina virtual creada en el momento. Tiene soporte para los lenguajes de programación principales: Java, C#, PHP, JavaScript, etc.

El alta es muy simple. Se puede hacer un alta directa o bien entrar a través de la cuenta de GitHub. Esto último es lo recomendable, ya que Travis comprueba los repositorios y nos permite elegir a cuáles aplicar el proceso de integración:

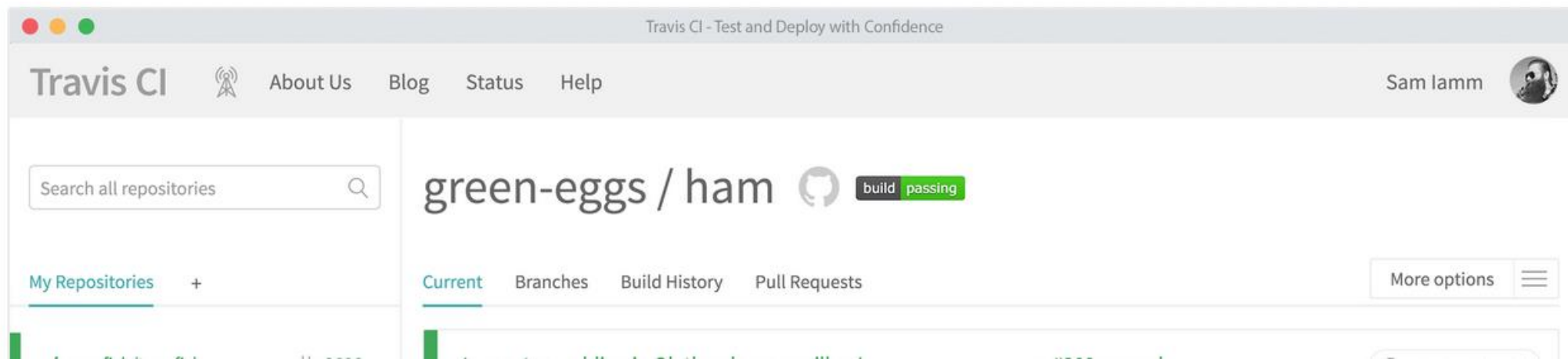
Since June 15th, 2021, the building on [travis-ci.org](#) is ceased. Please use [travis-ci.com](#) from now on.

Test and Deploy with Confidence

Easily sync your projects with Travis CI and you'll be testing your code in minutes!



Sign Up



Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

El fichero .travis.yml

El único requisito que debe cumplir nuestro proyecto para ser compatible con Travis es que disponga de un fichero en formato yaml llamado `travis.yml`. Este fichero contiene la configuración necesaria para que Travis sepa qué herramientas de compilación debe descargar.

Por cada lenguaje la web de Travis nos guía para saber la configuración que debemos indicar. Para proyectos web de fronten donde los lenguajes que se utilizan son básicamente HTML, CSS y JavaScript, lo habitual es utilizar Node.js como plataforma. Este es el fichero `travis.yml` utilizado en un proyecto web con Angular 1.5

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

El fichero .travis.yml

language: nodes

node_js:

- "6"

branches :

only:

- master

before_script:

- npm install -g webdriver-manager

- npm install -g jasmine

- npm install -g protractor

- npm install phantoms

- npm run post-install

- npm run selenium &

- npm run start &

- sleep 3

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

El fichero .travis.yml

Como se puede observar, primero se indica el tipo de plataforma y su versión. A continuación la rama de build y seguidamente, hay que asegurarse de que se instalan ciertas herramientas generales en el sistema: en este caso se trata de Selenium, Jasmine y Protractor.

El proceso de testear e2e con Protractor requiere de ciertas acciones adicionales. Todas ellas están definidas en scripts del fichero `package.json`: que se actualice Selenium (post-install en el `package.json` del proyecto), que se ponga en marcha el servidor de Selenium, que se ponga en marcha en el servidor web local y, por último dormimos, 3 segundos para que los procesos anteriores tengan tiempo a estar iniciados.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

El fichero `.travis.yml`

Tras todo lo que ejecuta `before_script`. Travis pone en marcha los test, y para eso se limita a hacer:

`Npm run test`

Que es otro script definido en `package.json`, precisamente el encargado de lanzar los test de Protractor

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

Navegadores headless

En este caso el proyecto tiene un build que consiste en las pruebas unitarias con Protractor, como las mostradas antes. Solo hay un pequeño cambio. Travis no utiliza un entorno gráfico y las pruebas e2e de Protractor deben hacerse con un navegador.

Para este tipo de situaciones, en las que se necesita hacer tests sin tener que usar Firefox o Chrome, se pueden utilizar navegadores headless: son navegadores con el mismo engine o interprete que un navegador convencional, pero que no tienen interfaz gráfica. En este proyecto lo que se utiliza es Phantoms.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

Proceso de integración

Una vez el proyecto está listo se puede ver o reconfigurar en su pantalla de Travis:
Su funcionamiento no puede ser más sencillo:

1. El desarrollador comienza haciendo un push al repositorio de GitHub
2. Travis detecta el push y pone en marcha una máquina virtual para el proceso de build.
3. En ese proceso, podemos ver como Travis se descarga el software que necesita según el tipo de proyecto declarado en travis.yml.
4. Se descarga el software del repositorio y el software adicional que precisa este proyecto.
5. Lanza los procesos de sistema adicionales declarados en travis.yml.
6. Ejecuta los test y nos da el resultado.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

Proceso de integración

En el momento que el proceso de build está en marcha, podemos seguirlo a través de la web del proyecto. Esta es la parte más espectacular de Travis, cuando vemos como pone en marcha todo el proceso para compilar y testear nuestro proyecto de forma remota. Cuando hay un build en proceso se muestra así:

El build viene coloreado según el éxito del resultado. Este es el aspecto de un proceso de build que ha concluido con errores:

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.1. Travis

Proceso de integración

En el caso de que el proceso concluya sin errores y con todos los test unitarios pasados correctamente veremos lo siguiente:

La información que se refleja en cada build es muy esencial: rama del repositorio, último mensaje de commit, resultado, tiempo de build, usuario que hizo el commit, id del push y fechade build

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Jenkins es una herramienta de integración continua open-source que como tal controla el proceso de compilación, testeo y, en definitiva, integración de proyectos de software. Lo podemos utilizar de varias maneras:

- Se puede utilizar remotamente como servicio PaaS.
- Se puede descargar y ejecutar localmente en el equipo de desarrollo..
- Se puede instalar en la red local una o más instancias.

Pese a ser una herramienta desarrollada en Java nos permite actuar sobre gran variedad de lenguajes. Uno de sus puntos fuertes, sin duda, es su popularidad y la disponibilidad de plugins que nos permiten llevar a cabo builds tan específicos como necesitemos,

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins



Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Instalación y puesta en marcha

La instalación no puede ser más simple. Existen variantes específicas para distintas plataformas: Windows, Linux, Docker, etc. Pero se puede usar la versión genérica en cualquiera. Basta con descargar un fichero comprimido con extensión war (Web Archive). Jenkins es, por tanto, una aplicación web Java que se puede ejecutar simplemente haciendo:

```
java -jar jenkins.war
```

Ese comando inicia un servidor web en el puerto 8080. Este es un puerto que puede producir conflictos con otro software como Tomcat; salvo que exista ese tipo de conflictos, Jenkins se iniciará sin problemas.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Instalación y puesta en marcha

En la primera instalación Jenkins genera una contraseña de administrador y la muestra por consola, como se puede apreciar. En el navegador se indica `http://localhost:8080` y se utiliza esa contraseña

Una vez usada procederemos a la puesta en marcha de Jenkins seleccionando plugins. De forma predeterminada se pueden elegir los plugins más comunes o bien seleccionarlos uno a uno.

[Back to Dashboard](#)[Manage Jenkins](#)[Update Center](#)

Filter:

Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	Ant Plugin Adds Apache Ant support to Jenkins	1.7		Uninstall
<input checked="" type="checkbox"/>	Apache HttpComponents Client 4.x API Plugin Bundles Apache HttpComponents Client 4.x and allows it to be used by Jenkins plugins.	4.5.3-2.0		Uninstall
<input checked="" type="checkbox"/>	Authentication Tokens API Plugin This plugin provides an API for converting credentials into authentication tokens in Jenkins.	1.3		Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.2		Uninstall
<input checked="" type="checkbox"/>	Branch API Plugin This plugin provides an API for multiple branch based projects.	2.0.18		Uninstall
<input checked="" type="checkbox"/>	Build Timeout This plugin allows builds to be automatically terminated after the specified amount of time has elapsed	1.19		Uninstall

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Instalación y puesta en marcha

Por ejemplo, se puede activar el soporte para integrar GitHub y Bitbucket y los builds de Microsoft y Node.js. El hecho de usar Jenkins no significa abandonar otras herramientas, precisamente destaca por su capacidad para integrar muchas tecnologías ya existentes. Una vez instalados los plugins, Jenkins solicitará un nombre y contraseña definitiva para el acceso de administrador y todo estará listo.

- [Crear nueva Tarea](#)
- [Actividad](#)
- [Historia de ejecuciones](#)
- [Administrar Jenkins](#)

Trabajos en la cola		
No hay tareas pendientes		
Estado de los nodos		
#	Estado	
1	Disponible	
2	Disponible	

Administrar Jenkins



[Configurar el Sistema](#)

Configurar variables globales y rutas.



[Actualizar configuración desde el disco duro.](#)

Descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Útil cuando se modifican ficheros de configuración directamente en el disco duro.



[Administrar Plugins](#)

Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.



[Información del sistema](#)

Muestra información del entorno que puedan ayudar a la solución de problemas.



[Registro del Sistema](#)

El log del sistema captura la salidad de la clase `java.util.Logging` en todo lo relacionado con Jenkins.



[Estadísticas de Carga](#)

Comprobar la utilización de los recursos y comprobar si es necesario añadir nuevos nodos para la ejecución de tareas.



[Interfaz de linea de comandos \(CLI\) de Jenkins](#)

Accede y administra Jenkins desde la consola, o desde scripts



[Consola de scripts](#)

Ejecutar script para la administración, diagnóstico y solución de problemas.



[Administrar Nodos](#)

Añadir, borrar, gestionar y monitorizar los nodos sobre los que Jenkins ejecuta tareas.



[About Jenkins](#)

See the version and license information



[Preparar Jenkins para apagar el contenedor](#)

Detener la ejecución de nuevas tareas para que el sistema pueda apagarse de manera segura.



Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Instalación y puesta en marcha

Una vez dentro del panel, podemos observar el estado de las tareas, gestionar Jenkins o empezar a crear tareas. Si falta alguna herramienta, o no aparecen disponibles para Jenkins programas como Maven o Git, o simplemente se necesita ajustar alguna opción, podemos seleccionar Administrar Jenkins y desde ahí llevar a cabo ajustes en:

- Global Tool Configuration.
- Configurar el sistema

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Tareas de build

Si pulsamos en nueva tarea podremos elegir varios tipos de build.

Estos son algunos de ellos:

- **Estilo libre:** es el más común y nos permite hacer uso de otros sistemas como ant, maven, scripts junto con repositorios de software.
- **Basado en Maven:** podemos aprovechar la existencia de un fichero pom.xml para que Jenkins lo utilice para hacer el build.
- **Pipelines:** de manera similar a la que hace Gulp, Jenkins puede encadenar un proceso con otro.
- **Proyectos multi-configuración:** en el caso de que un proyecto maneje distintos entornos es precisa esta tarea.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Proyectos Maven

Maven es una herramienta de gestión de proyectos, ampliamente usada en proyectos Java, y está soportada por Jenkins de serie.

Para crear una tarea que compile proyectos Maven locales, lo único de lo que hay que preocuparse es de indicar la ruta donde se encuentra el fichero pom.xml en la configuración de la tarea.

Una vez hecho, se puede ejecutar la tarea desde Jenkins y este se encargará de todo basándose en la configuración que se encuentra en pom.xml.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Proyectos de estilo libre

Todos los builds son personalizables, pero, sin duda, al elegir esta opción es cuando se abren más las opciones de personalización. Si elegimos esta opción podemos configurar lo siguiente:

- Opciones generales del proyecto.
- Repositorio de código fuente de origen: Git, Mercurial Subversion, etc.
- Opciones o disparadores de inicio del proceso de build, que pueden ser por ejemplo los push a un repositorio Git.
- Opciones para el entorno de ejecución.
- Ejecutar operaciones.
- Ejecutar operaciones posteriores.

Estas dos últimas nos permiten añadir virtualmente cualquier tarea.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Proyectos de estilo libre

Estas dos últimas nos permiten añadir virtualmente cualquier tarea. En la ejecución de tareas se pueden ir agregando las que necesitemos, incluso con condicionales. Dado que muchas de ellas consisten en scripting u otras herramientas de building para varios entornos, las posibilidades son infinitas.

Implementación de la Usabilidad en la Web

6.4. INTEGRACIÓN CONTINUA

6.4.2. Jenkins

Ejecución de builds

Pueden hacerse periódicamente, por cambios en repositorio, etc. Aunque en cualquier momento se puede lanzar un build desde el panel de administración. En todo momento se puede seguir su evolución y en el resultado final, sea correcto o no, comprobar que ha pasado en la consola. Así es como se muestran los builds de una tarea concreta. La última de ellas con éxito:

 [Back to Dashboard](#)

 [Status](#)

 [Changes](#)

 [Workspace](#)

 [Build Now](#)

 [Delete Project](#)

 [Configure](#)

 [Rename](#)


Project Test

 [Workspace](#)

 [Recent Changes](#)



Build History

[trend](#) 

x



#2

Jul 19, 2018 12:06 AM



#1

Jul 18, 2018 10:56 PM



[RSS for all](#)



[RSS for failures](#)

Permalinks

- [Last build \(#2\), 33 sec ago](#)
- [Last stable build \(#1\), 1 hr 10 min ago](#)
- [Last successful build \(#1\), 1 hr 10 min ago](#)
- [Last failed build \(#2\), 33 sec ago](#)
- [Last unsuccessful build \(#2\), 33 sec ago](#)
- [Last completed build \(#2\), 33 sec ago](#)