

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Editor de mapas mentales online con HTML5 y Javascript

Realizado por

José Luis Molina Soria

Dirigido por

Juan Antonio Falgueras Cano

Departamento

Lenguajes y Ciencias de la Computación

Málaga, Noviembre de 2013

UNIVERSIDAD DE MÁLAGA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente/a D^o/D^a. _____

Secretario/a D^o/D^a. _____

Vocal D^o/D^a. _____

para juzgar el proyecto Fin de Carrera titulado: _____

del alumno/a D^o/D^a : _____

dirigido por D^o/D^a : _____ ,

y, en su caso, dirigido académicamente por D^o/D^a : _____

ACORDÓ POR _____ OTORGAR LA CALIFICACIÓN
DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL
TRIBUNAL, LA PRESENTE DILIGENCIA

Málaga a ____ de _____ de 20__

El/La Presidente/a

El/La Secretario/a

El/La Vocal

Fdo:

Fdo:

Fdo:

Agradecimientos

A mi familia y padres por su paciencia

Tabla de contenidos

| | | |
|----------|--|-----------|
| 1 | Introducción | 6 |
| 1.1 | Motivación | 6 |
| 1.2 | Objetivos | 8 |
| 1.3 | Organización de la memoria | 9 |
| 2 | Mapas mentales | 11 |
| 2.1 | ¿Qué es? | 11 |
| 2.2 | Aplicaciones y beneficios. | 13 |
| 2.3 | Partes de un mapa mental. | 14 |
| 2.4 | Elaboración. | 16 |
| 3 | Análisis y Diseño | 18 |
| 3.1 | Metodología de análisis (UML-WAE) | 18 |
| 3.2 | Casos de uso. | 22 |
| 3.3 | Diagramas de Clase | 24 |
| 4 | Implementación. | 25 |
| 4.1 | Metodología y etapas del desarrollo. | 25 |
| 5 | Herramientas utilizadas | 30 |
| 5.1 | uglify | 30 |
| 5.2 | jsHint | 30 |
| 5.3 | KineticJS | 30 |
| 5.4 | NodeJS | 30 |
| 5.5 | GruntJs | 34 |
| 5.6 | Github | 38 |

| | |
|----------------------------|-----------|
| TABLA DE CONTENIDOS | 3 |
| 5.7 JSDoc | 41 |
| 5.8 Mocha | 42 |
| 6 Manual de usuario | 47 |
| 7 Conclusiones | 49 |
| Bibliografía | 50 |

Índice de figuras

| | | |
|-----|--|----|
| 1.1 | Esquema general de la especificación HTML5 a diciembre de 2011 | 7 |
| 2.1 | Mapa mental de FreeMind | 12 |
| 2.2 | Partes de un mapa mental | 14 |
| 2.3 | Partes de un mapa mental considerando su contenido | 15 |
| 2.4 | Mapa mental de elaboración de mapas mentales | 16 |
| 3.1 | Esquema de diagramas | 19 |
| 3.2 | Estereotipos | 20 |
| 3.3 | Valores etiquetados | 21 |
| 3.4 | Estereotipos | 21 |
| 3.5 | Casos de uso | 23 |
| 5.1 | Logo NodeJS | 31 |
| 5.2 | Logo GruntJS | 35 |
| 5.3 | Mascota de Github | 39 |
| 5.4 | Ejemplo de código fuente documentado con JSDoc | 41 |
| 5.5 | Página generada por JSDoc | 42 |
| 5.6 | Mocha | 43 |
| 5.7 | Resultado de ejecutar mocha -R spec *.js | 44 |
| 5.8 | Resultado de ejecutar mocha -R spec array1-test.js | 44 |

Introducción

1.1. Motivación

Desde hace ya unos años, estamos viviendo una revolución en el desarrollo web, que a provocado un cambio en nuestro estilo de vida, la forma de comunicarnos, en los flujos de información e incluso en nuestras relaciones diarias. HTML¹ y JavaScript son una parte importante de esta revolución, y es por ello, que decidí dar el paso y crear una aplicación que funcionará única y exclusivamente en el navegador (sin necesidad de un servidor).

Estamos viendo como día a día aplicaciones que han sido por antonomasia nativas (editores de texto, hojas de cálculo, aplicaciones de gestión, juegos ...), están siendo implementadas con tecnologías web con gran éxito. Los editores de mapas mentales también han dado ese paso existen aplicaciones como text2mindmap², MindMeister³, etc., tan versátiles o más que las propias aplicaciones nativas.

Hace ya tiempo que vio la luz los primeros borradores de HTML5⁴ (ver figura 1.1) pero su implantación está siendo lenta, no sólo por parte de la comunidad de desarrolladores y diseñadores, sino también por parte de los navegadores.

¹Hypertext Markup Language

²<http://www.text2mindmap.com/>

³<http://www.mindmeister.com/es>

⁴El primer borrador de HTML5 fue publicado en 2008

HTML5 ha tomado en cuenta los defectos de su versión anterior⁵ y mejorar otras características como:

HTML5

Taxonomy & Status (December 2011)

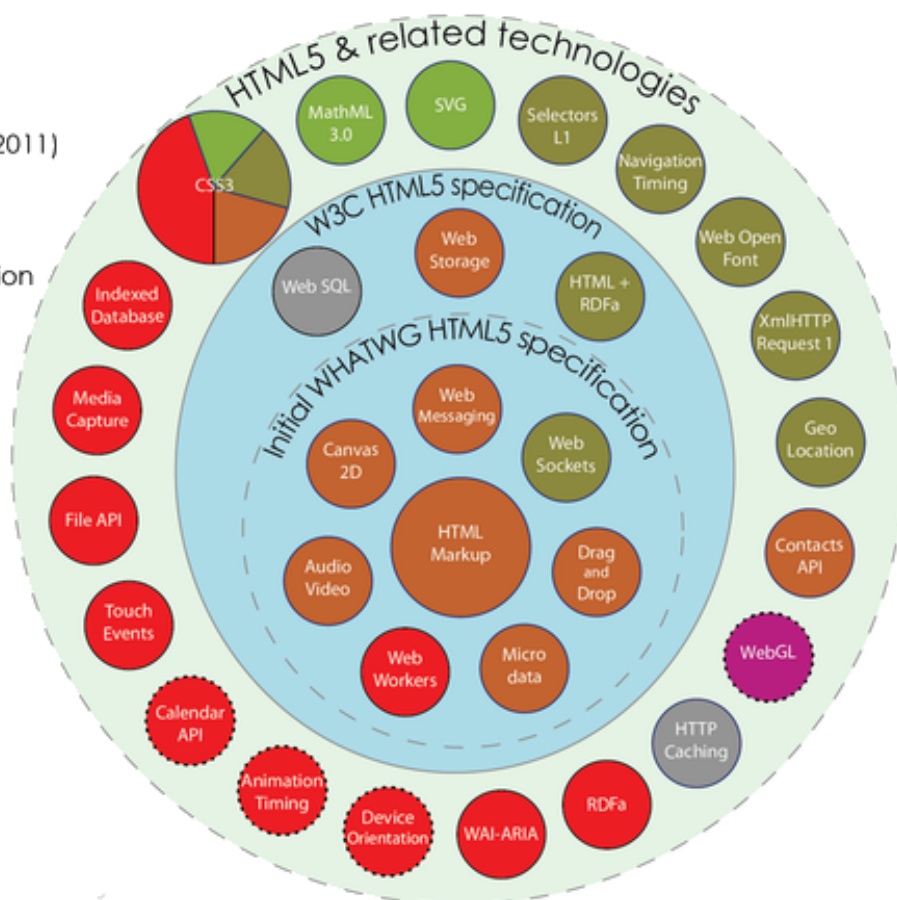
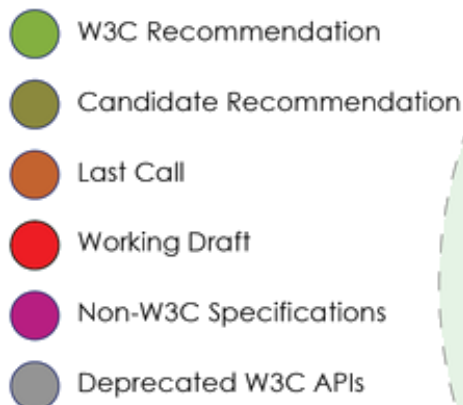


Figura 1.1: Esquema general de la especificación HTML5 a diciembre de 2011

- **Nuevas etiquetas estructurales.** Se ha incorporado un nuevo conjunto de etiquetas pensadas para definir mejor la estructura de una web, entre las más importantes están las de encabezado, barra de navegación, secciones y pie.
- **Manejo de imágenes.** En la versión anterior podía incorporar imágenes ahora, además podemos modificarlas e interactuar con ellas. También disponemos un una etiqueta y un API completo para manejo de canvas⁶.
- **Etiquetas de vídeo y audio.** Sin incluir flash ni aplicaciones externas podemos incorporar un reproductor de vídeo y/o audio.
- **Mejora en la semántica web.** HTML5 incluye elementos que permiten dar

⁵HTML 4.01

⁶En principio, sólo en 2D.

información de la página web a los buscadores para obtener resultados adaptados a las necesidades del usuario.

- **Soporte móvil/tabletas.** Mejoras en las hojas de estilos, nuevos manejadores para evento touch, etc.
- **Acceso a ficheros.** Incorpora un API para lectura/escritura de ficheros.

La motivación no puede ser otra que profundizar en las características de HTML5 y aprender de esta tecnología.

1.2. Objetivos

El principal objetivo de este proyecto es la creación de un editor de mapas mentales online. El editor, frontend, debe ejecutarse completamente en el cliente. Para ello, vamos a utilizar como lienzo de dibujos el canvas de HTML5 y Javascript como lenguaje de desarrollo.

El usuario podrá navegar por el diagrama con los cursores partiendo desde la idea central. Interactuará con el diagrama de forma que, dependiendo del nodo en el que se encuentre y la acción que realice podrá insertar, modificar, anotar, plegar, etc...

Esta fuerte interacción, provoca que dentro de los objetivos del proyecto, se encuentre la elaboración de una extensa librería JavaScript, bien estructurada y testeada.

En todo momento, y en pos de una aplicación lo más estándar posible, se seguirá las especificaciones de la World Wide Web Consortium⁷ (W3C) y la especificación

Como objetivo principal está pues, la universalidad, independencia de sistemas y la inmediatez de uso, sin instalación, siempre actualizada, e incluso la posibilidad de uso en forma local con cualquier navegador actual que sigue el estándar HTML5. Entre las posibles plataformas de uso se tratará de incluir las plataformas táctiles, especialmente los tablets.

⁷Web oficial de la W3C <http://www.w3.org/>

1.3. Organización de la memoria

El presente documento esta dividido en los siguiente capítulos.

1. Introducción.

Descripción general del proyecto y exposición de la estructura de la memoria.

2. Mapas mentales.

Apartado para descubrir que es, historia y usos de los mapas mentales.

3. Diseño e implementación.

En este capítulo se muestra la metodología, diseño, estudio e implementaciones para llevar a cabo la realización del proyecto.

4. Herramientas utilizadas.

Descripción de todas y cada una de las herramientas utilizadas en el desarrollo del proyecto.

5. Manual de usuarios.

Manual para el usuario final.

6. Conclusiones.

Capítulo para la conclusiones finales e impresiones.

Mapas mentales

2.1. ¿Qué es?

Los mapas mentales son un método efectivamente sencillo de asimilar y memorizar información a través de la representación visual de la información.

Por naturaleza, nuestro cerebro tiene un potencial ilimitado y que, en muchas ocasiones es desaprovechado o difícil de interpretar. Tenemos dos hemisferios el izquierdo y el derecho, el racional y el creativo, ambos funcionan de forma separada. Los mapas mentales consiguen relacionar ambos hemisferios (racional y creativo) y lograr que funcionen conjuntamente.

Toda persona tiene una forma natural de elaborar sus propias ideas, mediante pensamiento irradiante¹. El pensamiento irradiante refleja mediante la asociación de ideas nuestros pensamientos y conocimientos sobre una materia concreta. A esta forma de pensamiento podemos acceder mediante los mapas mentales, que irradian y asocian ideas a partir de un concepto central.

¹que irradia

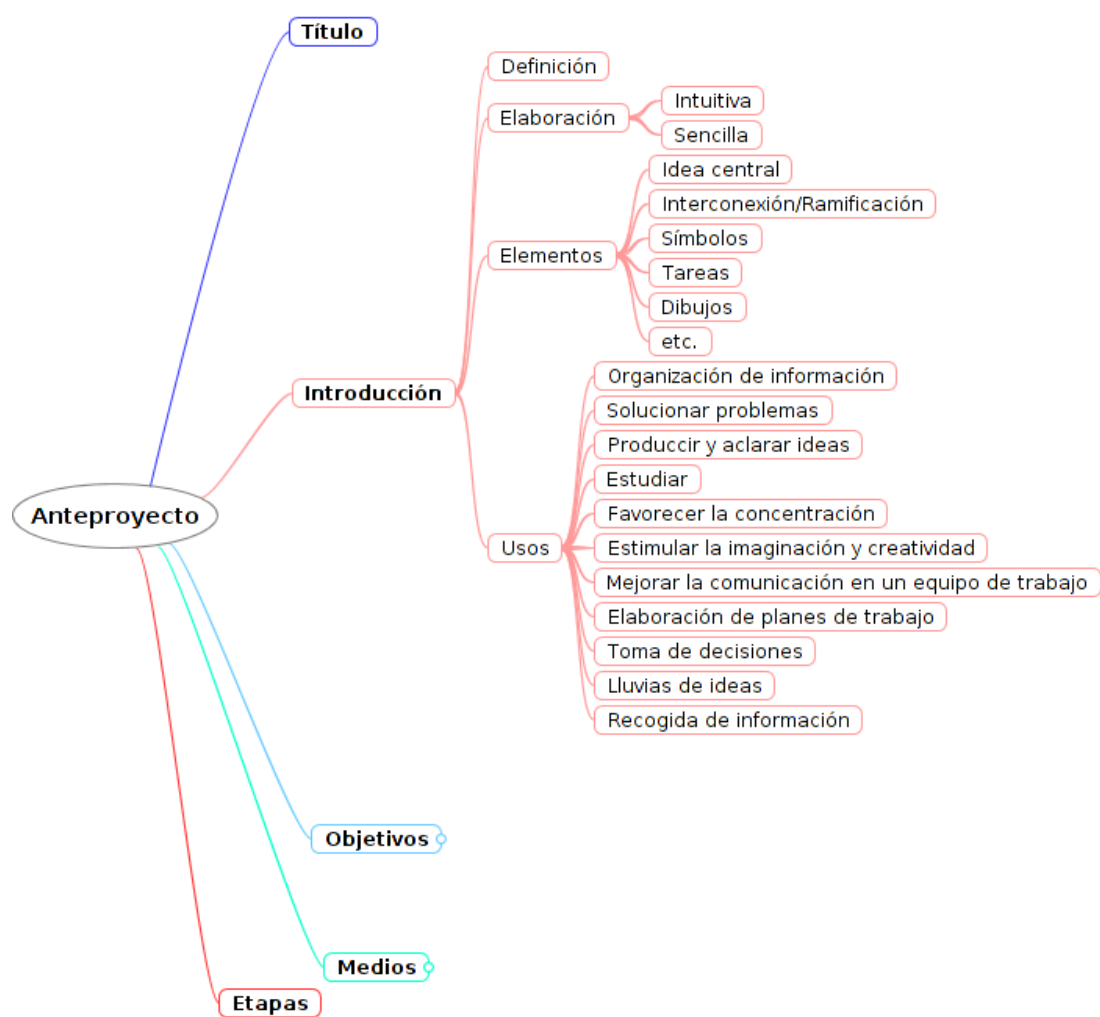


Figura 2.1: Mapa mental de FreeMind

2.2. Aplicaciones y beneficios.

Los campos de aplicación y los beneficios de los mapas mentales son muchos y muy diversos. Entre los más destacados tenemos:

- Estimular la memoria, imaginación y creatividad.
- Organizar información.
- Concentrarnos en la resolución de un problema.
- Tomar notas y apuntes.
- Producir y aclarar ideas o conceptos.
- Visualizar escenarios complejos.
- Consolidar procesos de estudios y aprendizaje.
- Favorecer la concentración.
- Proyectos. Organizar el proyecto y priorizar el plan de trabajo.
- Mejorar la comunicación en un equipo de trabajo.
- Preparar y dirigir una reunión.
- Toma de decisiones.
- Lluvias de ideas.
- Recogida de información.
- Expresar ideas complejas y difíciles de redactar.
- Diseñar el contenido de un escrito o informe.
- Preparar una presentación en público.
- Elaboración de sitios webs.

2.3. Partes de un mapa mental.

2.3.1. Según su estructura.

Un mapa mental tiene las siguientes estructuras esenciales (figura 2.2):

1. Idea central.
2. Aristas. Establece una asociación de ideas.
3. Nodo. Ideas secundarias o asociada a otra idea.

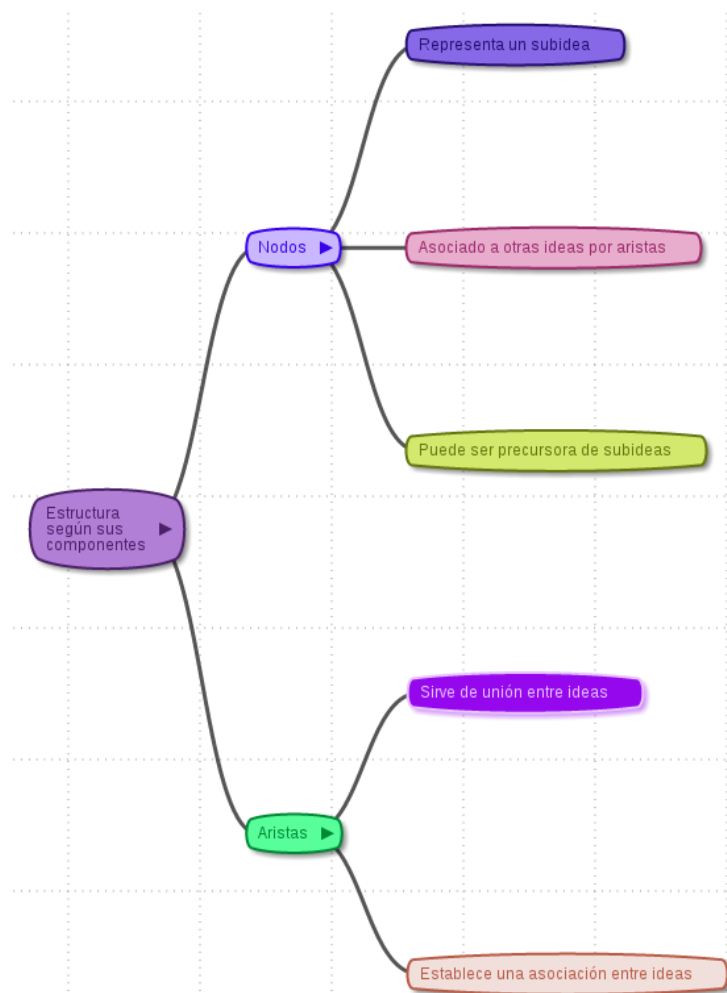


Figura 2.2: Partes de un mapa mental

2.3.2. Por contenido.

Un mapa mental podemos estructurarlo según su contenido (figura 2.3).

1. Idea central
2. Los temas principales del asunto irradian de la imagen central como ramas.
3. Cada rama contiene una imagen o una palabra clave asociada.
4. Las ramas forman una estructura nodal conectada.

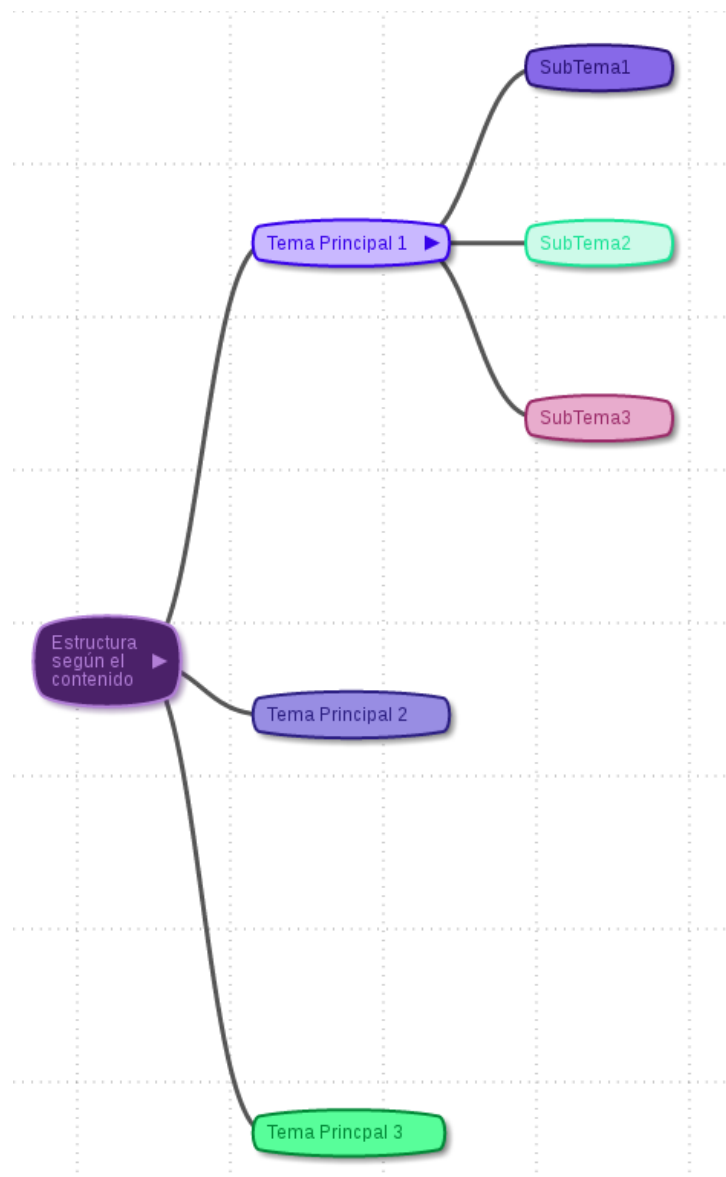


Figura 2.3: Partes de un mapa mental considerando su contenido

2.4. Elaboración.

La elaboración de un mapa mental es un gesto sencillo y casi intuitivo, sólo necesitamos partir de una idea central, de la cual vamos ramificando asociando o interconectando símbolos, palabras, tareas o dibujos.

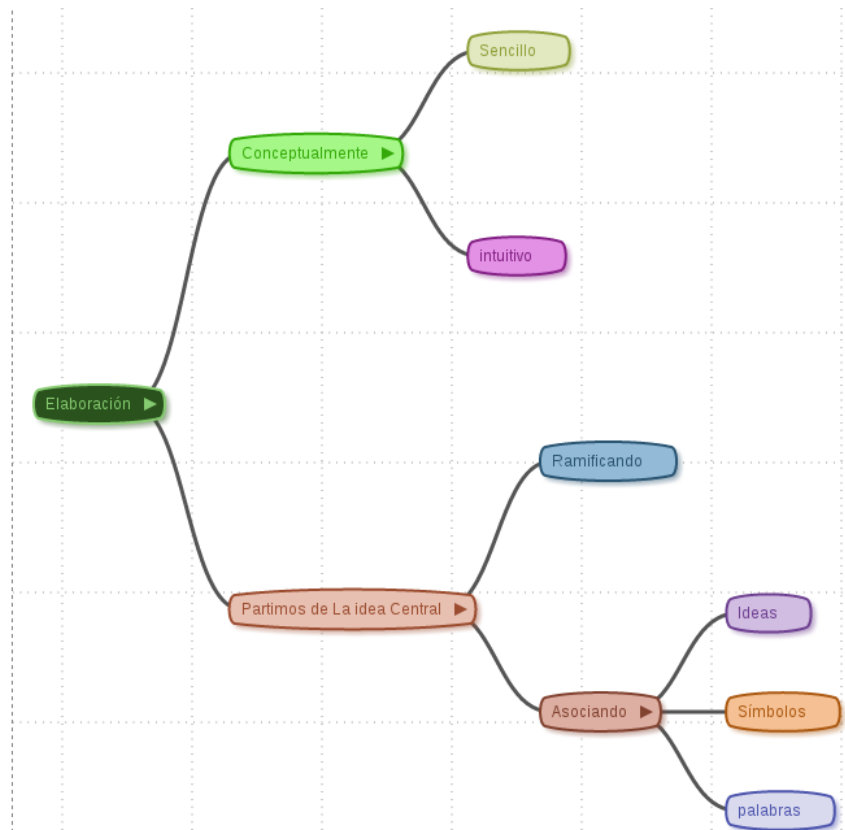


Figura 2.4: Mapa mental de elaboración de mapas mentales

En definitiva, se trata de un diagrama radial que permite a una persona, o grupo de ellas, plasmar su percepción sobre un tema, o idea, mediante la asociación de conceptos palabras y/o imágenes.

Análisis y Diseño

3.1. Metodología de análisis (UML-WAE)

Como metodología de análisis utilizaré el estándar de facto UML.

3.1.1. UML

UML o Lenguaje de Modelado Unificado (Unified Modeling Language) fue creado para unificar las distintas técnicas que existían. Se trata de un lenguaje, principalmente, visual que nos permite visualizar, especificar, construir y documentar un sistema. Como lenguaje de modelado nos da las herramientas para especificar los métodos y/o procesos del sistema.

UML consta principalmente de dos puntos de vista. Un punto de vista estructural o estático en el cual se plasman estructuras estáticas compuesta por objetos, atributos, operaciones y relaciones. El otro punto de vista es dinámico o de comportamiento que permite establecer y/o especificar las colaboraciones entre las distintas partes del sistema. Este último punto de vista, además nos proporciona estados internos de nuestro sistema.

Diagramas

La metodología UML nos dota de un conjunto de diagramas para modelar.

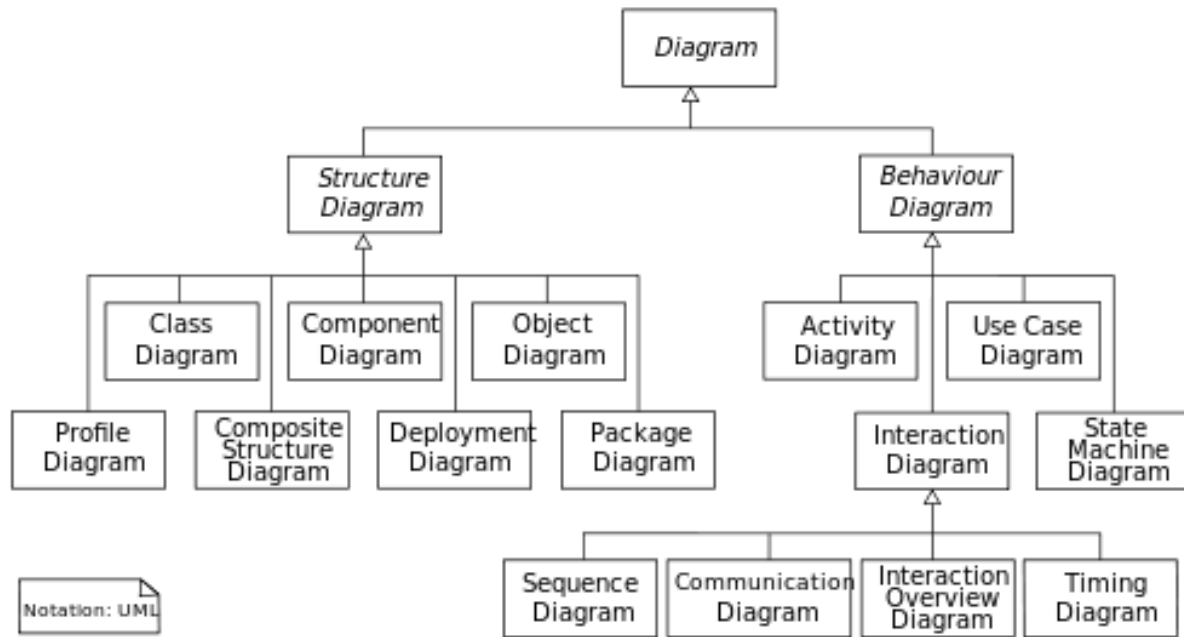


Figura 3.1: Esquema de diagramas

El conjunto de diagramas se divide en dos grandes grupos:

- Diagramas de estructura. Se corresponde con el punto de vista estático. Entre los diagramas más destacados están:
 - Diagrama de clases
 - Diagrama de componentes
 - Diagrama de paquete
- Diagramas de comportamiento. Se corresponde con el punto de vista dinámico. Entre los más utilizados están:
 - Diagrama de casos de usos
 - Diagrama de actividad
 - Diagrama de secuencia

UML 2 es extensible. Los mecanismo para personalizar y extender el UML son los perfiles y los estereotipos. UML-WAE utiliza los dos.

UML-WAE

Hoy en día, la mayoría de las aplicaciones se desarrollan entorno a la web, es decir, son aquellas aplicaciones que en su arquitectura y elementos principales están el navegador y el protocolo HTTP. Fue Jim Conallen, en 1988, quien definió una extensión del estándar UML para aplicaciones webs. Esta extensión pasó a llamarse WAE (Web Application Extension) y se trata de la convención o extensión más aceptada por lo que podríamos a hablar de un estándar dentro de las distintas extensiones webs de UML.

En un principio las aplicaciones webs se basaban en un conjunto bien definidos de tecnologías. Con un servidor web, un cliente web capaz de comunicarse, vía protocolo HTTP, y de renderizar HTML. Con el tiempo se fueron sumando otras tecnologías como CSS, JavaScripts, XML, Ajax, Comet, WebGL, etc . . . Es fácil comprobar, que el desarrollo de una aplicación web, en principio sencilla se ha complicado con el paso del tiempo. De hecho, el propio HTML en principio sencillo y liviano se ha vuelto amplio y pesado. En el siguiente diagrama, podemos ver de forma esquemática un compendio del conjunto de tecnologías que se incorporan o que están en proceso de incorporación al HTML5.

Es evidente que tal conjunto de tecnologías y especificaciones llevan al desarrollador web a un sobre esfuerzo para estar al día. Pero también abre un sin fin de posibilidades que, ya hoy en día, podemos observar en la Internet.

WAE lo que nos ofrece, a los desarrolladores, es la posibilidad de modelar elementos como HTML, CSS, JavaScript, páginas de servidor, etc . . . Para ello, hizo uso de los mecanismos de extensión del UML:

- Estereotipo: permite extender el vocabulario del UML, con el fin de crear nuevo elementos del modelo ya existente. Se representa de la entre «**estereotipo**».

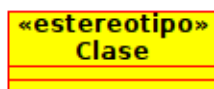


Figura 3.2: Estereotipos

- Valores etiquetados: se trata de un conjunto de pares clave-valor que puede ir asociado a un elemento del modelo. Por ejemplo, memoria="4Gb", nucleos=2. En la versión 1.3, los "valores etiquetados" quedaron obsoletos. A partir de UML2.0 se descartaron los "valores etiquetados", en favor de los atributos. Por lo que, hoy en día se utilizan atributos para estos menesteres.

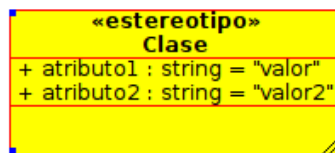


Figura 3.3: Valores etiquetados

- Restricciones: se trata de una etiqueta. Su misión realizar asertos y/o especificar restricciones a una relación entre objetos del modelo. Por ejemplo, A incluye B.

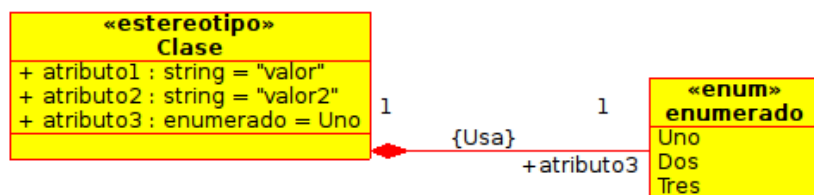


Figura 3.4: Estereotipos

Principales extensiones definidas por UML-WAE

Las principales extensiones que podemos encontrar dentro de la extensión UML-WAE son:

- << Server page >>: representará una instancia de páginas de servidor. Estas páginas se caracterizan por tener scripts y/o código que se ejecutan en el servidor. (.php, .asp, .jsp).
- << Client page >>: entidad que representa una una instancia de página que se renderiza en el cliente. Normalmente HTML y JavaScript.
- << Form >>: representa a la entidad de formulario de un documento HTML. Se trata de una colección de campos de formularios HTML.

- « FrameSet »: representa el conjunto de marcos o frames que tiene una página.
- « JavaScript »: entidad que se refiere a una clase o entidad JavaScript.
- « Script Library »: componente de estereotipo Script Library representa una librería de funciones y/o rutinas que pueden ser incluidas en una página web.
- « Web Page »: componente de estereotipo Web page.
- « Submit »: una asociación de estereotipo submit son asociaciones de formularios HTML a otras páginas de clientes o servidor.
- « Build »: asociación utilizada para representar la salida de una página de servidor en una instancia de página de cliente.
- « Redirect »: sirve para especificar redirecciones realizadas por cualquier página.
- « Link »: representa un ancho o enlace de una página de cliente a otra.

3.2. Casos de uso.

En la figura 3.5 podemos ver de forma general el diagrama de casos de usos.



Figura 3.5: Casos de uso

- 3.2.1. Nuevo mapa mental.
 - 3.2.2. Insertar idea.
 - 3.2.3. Borrar idea.
 - 3.2.4. Editar idea.
 - 3.2.5. Zoom.
 - 3.2.6. Navegar.
 - 3.2.7. Mover idea.
 - 3.2.8. Mover mapa mental.
 - 3.2.9. Salvar/cargar mapa mental.
 - 3.2.10. Hacer/deshacer acciones.
 - 3.2.11. Plegado/desplegado de ramas.
- ### 3.3. Diagramas de Clase

Implementación.

4.1. Metodología y etapas del desarrollo.

4.1.1. Metodología de desarrollo ágil.

En 2001, de un reunión celebrada en EEUU por 17 expertos en la industria del software nace el término “ágil” aplicado al desarrollo de software. El propósito de estos expertos era la elaboración de un manifiesto y principios que permiten a los equipos de desarrollar software rápidamente y responder a los cambios que surjan a lo largo del proyecto. The Agile Alliance, organización surgida de esta reunión, se dedica a promover los conceptos relacionados con el desarrollo ágil y cómo punto de partida tiene un manifiesto con los siguientes 4 puntos:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Quizás estemos ante una de las metodologías de desarrollo más importantes del momento. Se trata de un modelo desarrollo iterativo e incremental donde en cada iteración se elabora una nueva versión para usuario final.

El modelo de desarrollo Ágil tiene como principal objetivo la satisfacción del cliente y la elaboración de un software de calidad. Para ello, involucra al usuario en todas las etapas del desarrollo, aportando ideas y realizando pruebas de los productos de cada iteración. El usuario consigue así un software adaptado a sus necesidades, quedando completamente satisfecho del producto final. Con esta estrecha colaboración entre usuario final y el equipo de desarrollo se busca aunar esfuerzos en pos de un objetivo común.

En cada ciclo se pretende minimizar los riesgos. Es por ello que, para cada iteración se incorpora un conjunto reducidos de funcionalidades. Buscando, no sólo minimizar el riesgo intrínseco al desarrollo, sino que los ciclos de desarrollo sean cortos y se dinamice el proceso productivo. A este respecto, y según los principios de la metodología Ágil, es preferible una versión incompleta a una con errores.

Otro aspecto importante, es que la solución y los requerimientos evolucionan de forma continua. Provocando en ocasiones cambios profundos en los diseños preliminares, algo inconcebible en las metodologías clásicas. La refactorización de código se convierte en algo habitual y deseable, si ello nos lleva a una mejor solución.

Un ciclo de desarrollo en la metodología Ágil consta de la siguientes fases:

- Planificación.
- Análisis de requerimientos.
- Diseño.
- Codificación.
- Revisión.
- Documentación.

La metodología Ágil se adapta muy bien al desarrollo web. Por esto, y por las características que presenta este paradigma, el proyecto seguirá este modelo de desarrollo.

4.1.2. Etapas del desarrollo.

Viendo la dependencia entre librerías a implementar, lo más apropiado, es seguir un diseño ascendente (bottom-up). Siempre que el estadio anterior haya sido verificado y comprobado su completud, se podrá afrontar con éxito la siguiente etapa. Dicho de otra forma, cada etapa es dependiente de la etapa inmediatamente anterior. Siempre siguiendo la metodología ágil se ha decidido afrontar el proyecto en dos fases:

Primera fase: se encargará de llevar a buen término la implementación de las librerías JavaScripts necesarias para la aplicación. En cada ciclo tendremos que realizar una planificación, análisis de riesgos, implementación, pruebas unitarias y documentación de cada librería. La primera fase constará pues, de seis ciclos bien definidos. El orden de los ciclos es el que sigue:

- **Librería base con soporte para herencia.** Esta librería debe tener toda la funcionalidad básica (bindings, currying, etc) y debe estar muy optimizada ya que el perfecto funcionamiento de la aplicación dependerá en buena medida de ella.
- **Librería para manejo de árboles n-arios.**
- **Librería para el manejo de ficheros.** Será la encargada de manejar ficheros, a partir de ella, realizaremos las clases de exportación e importación de mapas mentales de la aplicación.
- **Librería gráfica.** Ciñéndonos al contexto 2D, necesitamos un wrapper sobre la librerías propias del canvas. Esta librería, nos debe permitir pintar, cada uno de los elementos de nuestro árbol. Además de configurar, atributos visuales tales como color del trazo, relleno, etc. No se descarta el uso de alguna librería estándar. Para ello, se realizará una pruebas de concepto sobre ellas.
- **Librería para el manejo de eventos del canvas.** El canvas debe reaccionar tanto al teclado, ratón y touch. El canvas viene desprovisto de eventos sobre los elementos pintados en él y es aquí donde entra en juego esta librería.
- Por último, las librerías propias del mapa y **prototipo** o primera versión.

Segunda fase: una vez implementadas todas las librerías necesarias y una primera versión (inoperativa), nos encontramos en disposición de ir elaborando la aplicación. En esta fase, desde mi punto de vista, el modelo iterativo incremental se adapta a la perfección al proceso de refinamiento de la aplicación. Cada iteración con llevará la implementación de la funcionalidad, en la que nos enfoquemos, y su posterior evaluación y prueba.

Herramientas utilizadas

PENDIENTE DE TERMINAR LA INTRODUCCIÓN DE HERRAMIENTAS UTILIZAS

5.1. uglify

5.2. jsHint

5.3. KineticJS

5.4. NodeJS

Basado en la máquina virtual JavaScript V8 de Google, NodeJS¹ a supuesto una revolución en el mundo de la programación JavaScript, dando un salto de gigante desde el lado del cliente al servidor. Este enorme evolución, y de manos de V8, ha provocado la creación de un entorno de programación completo, en el cual se aglutina desde un REPL² para pruebas y depuración interactiva hasta un gestor de paquetes y librerías NPM³ (Node Packaged Modules).

¹La web oficial de NodeJS es nodejs.org

²Patrón Read-Eval-Print Loop

³La web oficial de NPM es npmjs.org



Figura 5.1: Logo NodeJS

NodeJS nos permite crear aplicaciones de red escalables, alcanzando un alto rendimiento utilizando entrada/salida no bloqueante y un bucle de eventos en una sola hebra. Es decir, que NodeJS se programa sobre un sólo hijo de ejecución y en el caso de que necesite operaciones de entrada/salida, creará una segunda hebra para evitar su bloqueo. En teoría NodeJS puede mantener tantas conexiones simultaneas abiertas como descriptores de fichero soporte el sistema operativo (en UNIX aproximadamente 65.000), en la realidad son bastantes menos (se calcula que entre 20.000 y 25.000).

Como ya se ha mencionado, y debido a que su arquitectura es usar un único hilo, sólo puede usar una CPU. Es el principal inconveniente que presenta la arquitectura de NodeJS.

Sus principales objetivos son:

- Escribir aplicaciones eficientes en entrada y salida con un lenguaje dinámico.
- Soporte a miles de conexiones.
- Evitar las complicaciones de la programación paralela (Concurrencia vs paralelismo).
- Aplicaciones basadas en eventos y callbacks.

5.4.1. Instalación de NodeJS

Existen varias formas de instalar NodeJS, por ejemplo, utilizando los repositorios del sistema operativo o instaladores. En mi caso, he utilizado la compilación del código fuente que esta alojado en GitHub⁴.

Lo primero que tenemos que hacer es clonar el proyecto.

```
$ git clone git://github.com/joyent/node.git
$ cd node
```

Una vez tengamos la copia del código fuente realizaremos un checkout de una versión estable.

```
$ git branch vXXXX Nombre
$ git checkout Nombre
```

⁴Repositorio de NodeJS <https://github.com/joyent/node>

Ahora, ya estamos en disposición de compilar el fuente de la versión estable.

```
$ ./configure --prefix=/usr/local
$ sudo make install
```

5.4.2. Instalación del NPM

Como ya se ha comentado antes NPM⁵ es el gestor de paquetes de NodeJS. En la versiones actuales ya viene instalado, pero eso no fue siempre así. También se puede optar por instalarse de sin NodeJS. Para ello, ejecutaremos el siguiente comando:

```
$ curl https://npmjs.org/install.sh | sh
```

5.4.3. Uso básico de NPM

Iniciar un proyecto nuevo

A continuación se muestra la secuencia de comandos necesaria para crear un proyecto.

```
$ mkdir hola
$ cd hola
$ npm init
npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sane defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg> --save' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (hola)
version: (0.0.0)
git repository:
author:
license: (BSD-2-Clause)
About to write to /tmp/hola/package.json:

{
  "name": "hola",
  "version": "0.0.0",
  "description": "Hola mundo",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "hola",
    "mundo"
  ],
  "author": "",
  "license": "BSD-2-Clause"
}

Is this ok? (yes)
```

⁵Node Packaged Module (NPM) web oficial npmjs.org

El comando **npm init** comenzará a realizarnos sobre los datos del proyecto como nombre, versión, etc. Una vez terminado, tendremos nuestro fichero de configuración (package.json) preparado.

Buscar paquetes y obtener información

El primer comando nos permite buscar paquetes interesantes o útiles a nuestro proyecto, y el segundo, para obtener una descripción más exhaustiva del mismo.

```
$ npm search <palabra>:  
$ npm info <paquete>
```

Instalación de paquetes

Existen varias formas para instalar un paquete y/o librería.

De forma global ⁶ para que lo puedan utilizar todas las librerías del sistema.

```
$ npm install <paquete> -g
```

De forma local⁷, es decir, sólo se podrá utilizar el proyecto actual.

```
$ npm install <package name>
```

También existen dos modificadores muy interesantes *-save* para que se incluya (en el fichero package.json) la librería o paquete como dependencia del proyecto. Y el otro modificador es *-save-dev* para que la dependencia sea de desarrollo. Así quedaría un fichero package.json después de haber incluido un paquete (colors) como dependencia y otro (grunt-cli) como dependencia de desarrollo.

```
1 {  
2   "name": "hola",  
3   "version": "0.0.0",  
4   "description": "Hola mundo",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "keywords": [  
10    "hola",  
11    "mundo"  
12  ],  
13   "author": "",  
14   "license": "BSD-2-Clause",  
15   "dependencies": {  
16     "colors": "~0.6.2"  
17  },  
18   "devDependencies": {  
19     "grunt-cli": "~0.1.9"  
20  }  
21 }
```

⁶Con el modificador -g

⁷Sin el modificador -g

Desinstalación de paquetes

Las instrucciones son la misma salvo por que el comando *install* se sustituye por *uninstall*.

5.5. GruntJs

Se trata de una aplicación Node que está empaquetada y disponible en NPM. GruntJS es una herramienta versátil para la automatización de tareas mediante Javascript, evitándonos dentro de lo posible la realización de tareas repetitivas. Con un simple archivo de configuración nos permite realizar tareas tan diversas como minificar código, lanzar la suite de tests, etc.

5.5.1. Características

- **Acceso a archivos:** No tenemos que preocuparnos del acceso a archivos, sólo tratarlos.
- **Automatización de tareas y conjunto de tareas:** Podemos automatizar pequeñas tareas o mediante un conjunto de ellas automatizar tareas más complejas como la comprensión de una librería Javascript.
- **Fácil instalación:** Esta en NPM, la instalación es simplemente un `npm install`.
- **Plugins comunitarios:** Existe un gran comunidad detrás creando plugins, que podemos utilizar utilizando NPM.
- **Multi-plataforma:** Al ser una librería Node nos permite utilizarlo en cualquier plataforma que soporte Node.

5.5.2. Instalación

La instalación de GruntJS no tiene complicación, ya que, al tratarse de una aplicación Node y estar publicado en NPM sólo necesitamos como prerequisite tener instalado Node y NPM.

Lo primero es instalar el cliente de forma global con el comando:

```
$ npm install grunt-cli -g
```

Y una vez instalado el cliente, en nuestro proyecto debemos ejecutar:

```
$ npm install grunt --save-dev
```

Ya tenemos agregado GruntJS a nuestro proyecto. Con los `--save-dev` le indicamos al NPM que lo añada a las dependencias del proyecto para desarrollo. Así incluirá las líneas pertinentes en nuestro fichero `package.json`.



Figura 5.2: Logo GruntJS

```
1 {  
2   "name": "nombre",  
3   "version": "0.0.1",  
4   "dependencies": {  
5  
6   },  
7   "devDependencies": {  
8     "grunt": "~0.4.1"  
9   }  
10 }
```

5.5.3. Creando el Gruntfile

En el fichero `Gruntfile.js` será donde definamos las tareas que deseamos en nuestro proyecto. El esquema de fichero es:

```
1 module.exports = function(grunt) {  
2  
3   grunt.registerTask('default', 'Tarea Hola Mundo', function() {  
4     grunt.log.write('Hola Mundo!').ok();  
5   });  
6  
7 };
```

Como se puede observar se trata de un modulo Node, que será llamado por grunt cuando lo ejecutemos. En el ejemplo, le hemos registrado una tarea por defecto que imprime "Hola Mundo!". Ahora sólo tenemos que ejecutar el comando `grunt` para ver el resultado de nuestra tarea.

GruntJS tiene un conjunto básico de plugins, nombrados `grunt-contrib-XXXX`, empaquetados en NPM y que podemos instalar fácilmente.

5.5.4. Gruntfile.js de MindMapJS

El fichero de configuración de GruntJS utilizado para el proyecto es :

```

1 module.exports = function(grunt) {
2   var config = {
3     pkg: grunt.file.readJSON('package.json'),
4
5     concat: {
6       options: {
7         separator: ';',
8       },
9       source: {
10        src: ['src/*.js'],
11        dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
12      }
13    },
14
15    replace: {
16      dev: {
17        options: {
18          variables: {
19            version: '<%= pkg.version %>',
20            date: '<%= grunt.template.today("yyyy-mm-dd") %>'
21          },
22          prefix: '@@'
23        },
24
25        files: [{
26          src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.js'],
27          dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.js'
28        }]
29      },
30      prod: {
31        options: {
32          variables: {
33            version: '<%= pkg.version %>'
34          },
35          prefix: '@@'
36        },
37        files: [{
38          src: ['dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'],
39          dest: 'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js'
40        }]
41      }
42    },
43
44    uglify: {
45      options: {
46        banner: '/*! <%= pkg.name %> v<%= pkg.version %> <%= grunt.template.today("yyyy-mm-dd") %> Por JosÃ© Luis Molina Soria */\n'
47      },
48      build: {
49        files: {
50          'dist/<%= pkg.name %>-v<%= pkg.version %>.min.js': 'dist/<%= pkg.name %>-v<%= pkg
51            .version %>.js'
52        }
53      },
54    },
55
56    clean: {
57      build: ['dist/*']
58    },
59
60    jshint: {
61      options: {
62        laxbreak: true,
63        curly: true,
64        eqnull: true,
65        eqeqeq: true,
66        undef: true,
67        browser: true,
68        // immed: true,
69        latedef: true,
70        newcap: true,
71        noarg: true,
72        sub: true,
73        boss: true,
74        globals: {
75          console: true,

```

```

75     window : true,
76     module: true,
77     MM : true,
78     Kinetic : true,
79     require : true,
80     ActiveXObject : true,
81     FileReader : true,
82     DOMParser : true,
83     Blob : true,
84     alert : true
85   },
86   },
87   all : ['src/*.js']
88 },
89
90 jsdoc : {
91   dist : {
92     src: ['src/*.js'],
93     options: {
94       destination: 'docs/jsdocs/'
95     }
96   }
97 },
98
99 mochaTest: {
100   test: {
101     options: {
102       reporter: 'spec',
103       require: 'should'
104     },
105     src: ['test/**/*.js']
106   }
107 }
108
109 };
110
111 grunt.initConfig(config);
112
113 // Load plugins
114 grunt.loadNpmTasks('grunt-contrib-concat');
115 grunt.loadNpmTasks('grunt-replace');
116 grunt.loadNpmTasks('grunt-contrib-uglify');
117 grunt.loadNpmTasks('grunt-contrib-clean');
118 grunt.loadNpmTasks('grunt-contrib-jshint');
119 grunt.loadNpmTasks('grunt-jsdoc');
120 grunt.loadNpmTasks('grunt-mocha-test');
121
122 // Tasks
123 grunt.registerTask('dev', ['clean', 'concat:source', 'replace:dev']);
124 grunt.registerTask('full', ['clean', 'concat:source', 'replace:dev', 'uglify', '
    replace:prod']);
125 grunt.registerTask('test', ['mochaTest']);
126 grunt.registerTask('hint', ['jshint']);
127 grunt.registerTask('jsdoc', ['jsdoc']); // no funca :( utilizar el script "jsdoc.sh"
128 };

```

Como se puede comprobar se han incorporados distintos plugins:

- **grunt-contrib-concat:** permite concatenar un conjunto de ficheros en nuestro caso los ficheros JavaScripts.
- **grunt-replace:** plugins para realizar operaciones de reemplazo dentro de un conjunto de ficheros.
- **grunt-contrib-uglify:** para comprimir y/o minimizar el código JavaScripts.
- **grunt-contrib-clean:** borrar un conjunto de ficheros o el contenido de un directorio.

- **grunt-contrib-jshint:** permite reliazar la verificación y validación de buenas prácticas establecidas en JavaScripts.
- **grunt-jsdoc:** compilar los comentarios JSDocs para generarla documentación HTML del API.
- **grunt-mocha-test:** tarea que lanza la suite de tests unitarios del proyecto.

Con estos plugins se han cubierto todas las necesidades de automatización de tareas del proyecto. Las tareas implementadas son:

- **dev:** que concatena el código fuente y realiza los reemplazo como fechas, versión, etc ...
- **full:** además de realizar las tareas propias de la tarea 'dev', minimiza y realiza los reemplazos de producción.
- **test:** lanza la suite de test
- **hint:** lanza la terea de validación de código JSHint.
- **jsdoc:** genera la documentación del API.

5.6. Github

Todo proyecto que se precie debe estar sustentado con sistema de control de versiones, en nuestro caso ha sido Git⁸. Más concretamente se trata de un sistema distribuido de control de código fuente o SCM⁹ creado por Linus Torvalds, a partir, de su propia experiencia en el desarrollo de los kernels de Linux.

Github¹⁰ es una plataforma online pensada para el desarrollo colaborativo de proyectos, utilizando para ello Git. Github nos permite almacenar de forma pública¹¹ nuestro código fuente, promoviendo el trabajo colaborativo entre profesionales. Así pues, otro profesional ajeno al proyecto puede solicitar cambios sugerir mejoras o reportar bugs.

⁸Web oficial de Git es git-scm.com

⁹SCM (Source Code Management)

¹⁰La web de Github es github.com

¹¹Github permite crear proyectos privados con cuentas de pago

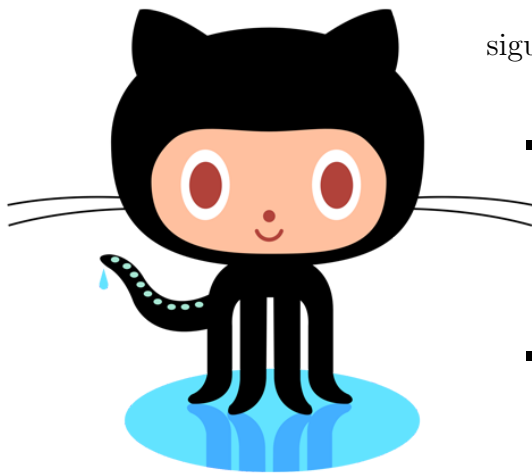


Figura 5.3: Mascota de Github

De las características mas resaltables de Github para el control de versiones, podemos enumerar las siguientes:

- **Wiki para el proyecto**, con el principal propósito de documentar nuestro proyecto Github nos proporciona una Wiki.
- **Gráficas**, tiene un conjunto de gráficas detalladas para determinar el avance del proyecto y el progreso de cada colaborador del proyecto.
- **Página web del proyecto**, para presentar nuestro proyecto y/o repositorio

Como sistemas de colaboración entre programadores tenemos el:

- **Fork**, con un fork podemos clonar un repositorio para realizar cambios que necesitemos, de forma que podamos adaptar el proyecto a nuestras necesidades concretas. Un fork nos permite colaborar con el proyecto original mediante los pull requests.
- **Pull requests**, una vez realizados los cambios, y si lo vemos oportuno, podemos reportar las variaciones al proyecto original mediante un pull request. El pull request pueden ser cambios, mejoras en la funcionalidad, y/o correcciones, que deberá aprobar él/los programadores del proyecto original.

5.6.1. Crear el repositorio

Previo a la creación del repositorio debemos crearnos una cuenta de usuario en Github. una vez realizado, sólo debemos pulsar la opción de "new repository". Ahora, ya tenemos repositorio pero debemos dotarlo de contenido, y para ello, y desde una consola local realizaremos:

- Creamos el directorio del proyecto.

```
$ mkdir ~/proyecto  
$ cd proyecto
```

- Iniciamos el repositorio git

```
$ git init
```

- Creamos el fichero README.md. Se trata de un fichero con formato markdown¹² en el cual hay que introducir un descripción del proyecto. Este fichero se visualizará en la página principal del repositorio.

- Añadimos y confirmamos los cambios.

```
$ git add .  
$ git commit -m 'primer commit'
```

- Cambiamos el remote origin a la ruta de nuestro repositorio.

```
$ git remote add origin https://github.com/usuario/proyecto.git
```

- subimos los cambios al repositorio

```
$ git push origin master
```

5.6.2. Fork/Pull request

Crear un fork de un proyecto utilizando Github es trivial. Tan sólo hay que ir al proyecto en cuestión y pulsar el botón de fork. Github crea una copia del proyecto de forma que si el proyecto original tiene la url <https://github.com/usuarioOriginal/proyecto.git> y la copia tendrá la url <https://github.com/usuario/proyecto.git>. Ahora ya estamos en disposición de trabajar clonando el repositorio:

```
$ git clone https://github.com/usuario/proyecto.git
```

Ya tenemos el repositorio listo para su uso. Si deseamos colaborar con el proyecto original debemos crear una rama¹³, realizar los cambios y subirlos¹⁴ a nuestro fork de Github. Desde Github procede realizar la revisión de los cambios y pulsar sobre la opción de `create a pull request for this comparison`.

¹²<http://es.wikipedia.org/wiki/Markdown>

¹³Operaciones a realizar: branch y checkout

¹⁴Operaciones a realizar: commit y push

5.7. JSDoc

Tan importante como el código es la documentación del mismo, JSDoc¹⁵ es una herramienta inspirada en Javadoc¹⁶ pero pensada para Javascript.

Mediante una conjunto de etiquetas (@class, @function, etc) introducidas como comentarios del código fuente, se generará la documentación en formato HTML¹⁷. Todos los desarrolladores que alguna vez hemos programado en Java y generado documentación de nuestro código, en JavaDoc, estamos familiarizados con el mecanismo de etiquetas, por lo que resulta muy intuitivo la elaboración de la documentación.

En la figura 5.4 se puede ver un ejemplo de uso de las etiquetas en el código fuente. En concreto de una clase PubSub del propio proyecto MindMapJS. Se puede observar claramente, como se usan etiquetas como @author, @versión, @constrcutor, @class, etc ...

```
/**
 * @file pubsub.js Implementación del patrón Publish/Subscribe
 * @author José Luis Molina Soria
 * @version 20130227
 */

if ( typeof module !== 'undefined' ) {
    var MM = require('./MindMapJS.js');
    MM.Class = require('./klass.js');
}

/**
 * @class MM.PubSub
 * @classdesc Implementación del patrón Publish/Subscribe
 * @constructor MM.PubSub
 */
MM.PubSub = MM.Class.extend(/** @lends MM.PubSub.prototype */{

    eventos : {},

    idSus : 1,

    init : function () {
        this.eventos = {};
        this.idSus = 1;
    },

    /**
     * @desc Realiza la notificación a los suscriptores de que se a producido
     * una publicación o evento.
     * @param evento {string} nombre del evento o publicación a notificar
     * @param args {*} argumentos para la función callback
     * @return {boolean} Si el evento no es un nombre válido retorna false en
     * otro caso retorna true
     */
    on : function( evento ) {
        if (!this.eventos[evento]) {
            return false;
        }
    }
});
```

Figura 5.4: Ejemplo de código fuente documentado con JSDoc

En la figura 5.5 tenemos el resultado de compilar el código fuente con JSDoc. El resultado

¹⁵Url del proyecto <https://github.com/jsdoc3/jsdoc>

¹⁶<http://es.wikipedia.org/wiki/Javadoc>

¹⁷Por lo general, se genera HTML pero permite otros formatos como RTF.

es un HTML que podemos retocar y configurar, permitiendo tener una Wiki, vistosa y funcional, de la documentación de nuestro código fuente.

Class: PubSub

MM. PubSub

Implementación del patrón Publish/Subscribe

new PubSub()

Source: [pubsub.js](#), line 12

Methods

desSuscribir(id) → {null|number}

realiza una dessuscripción a un evento o notificación

Parameters:

| Name | Type | Description |
|------|--------|------------------------------|
| id | number | identificador de suscripción |

Source: [pubsub.js](#), line 77

Returns:

null si no se ha podido realizar la dessuscripción

Type
null | number

on(evento, args) → {boolean}

Index

Classes

[Arbol](#)
[Arista](#)
[Borde](#)
[Class](#)
[Globo](#)
[Grid](#)
[FreeMind](#)
[XML](#)
[Mensaje](#)
[NodoSimple](#)
[PubSub](#)
[Rama](#)
[Render](#)
[UndoManager](#)
[ComandoHacerDeshacer](#)

Namespaces

[MM](#)
[DOM](#)
[exportar](#)
[importar](#)
[Properties](#)
[teclado](#)

Figura 5.5: Página generada por JSDoc

5.8. Mocha

Mocha¹⁸ es un framework Javascript para realizar pruebas unitarias. Sus creadores lo definen como:

Mocha is a feature-rich JavaScript test framework running on node.js and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

Y sinceramente, creo que la definición no puede ser más acertada. Permite crear test con relativa facilidad y con una sintaxis clara y concisa. De puedo decir, que es "simple, flexible y divertido"¹⁹.

¹⁸Página oficial de Mocha: <http://visionmedia.github.io/mocha/>

¹⁹En la cabecera de su web podemos leer "mocha simple, flexible, fun"

5.8.1. Características

Entre sus características más destacables están:

- **Soporte para NodeJs.** No sólo soporta el uso con NodeJS sino que esta empaquetado para NPM, por lo que, la instalación y la puesta en marcha resulta muy, muy sencilla. También existen plugins para utilizarlo con GruntJs.
- **Soporte para diferentes navegadores.** Por lo que, podemos probar nuestro interface en el navegador y verificar su correcto funcionamiento.
- **Informes.** Tiene opciones para generar informes en varios formatos dependiendo de las necesidades.
- **Uso de cualquier librería de afirmaciones(Assertions).** Existe principalmente cuatro librerías que pueden ser utilizadas con Mocha Chai, Should, Expect y Better-Assert.
- **Soporte para test síncronos y asíncronos.** Permite abarcar todas las necesidades de nuestro código.

5.8.2. Ejemplo

He aquí un simple ejemplo de uso de mocha.

```
1 var assert = require("assert");
2 describe('Array', function(){
3   describe('#indexOf()', function(){
4     it('debe retorna -1 si el valor no esta presente', function(){
5       assert.equal(-1, [1,2,3].indexOf(5));
6       assert.equal(-1, [1,2,3].indexOf(0));
7     });
8   });
9 });
```



Para empezar, debe realizar importar la librería de afirmaciones (assertions) que vamos a utilizar en el presente ejemplo se ha utilizado assert²⁰.

Figura 5.6: Mocha

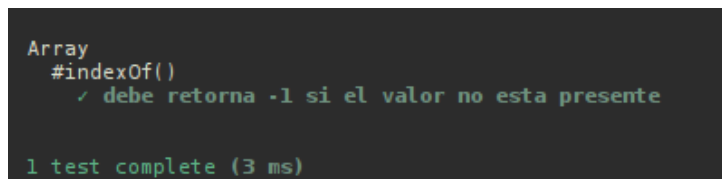
Las líneas 2 y 3 describen a su vez, un módulo y submódulo de ejecución. En nuestro caso el módulo lo hemos llamado .Arrayz el

²⁰En MindMapJS se ha utilizado Should y Chai

submódulo de ejecución `indexOf()`". Conviene ser descriptivos en los nombres de los módulos y submódulos.

La línea 4 describe una prueba unitaria a la que le asociamos una función anónima con las afirmaciones necesaria.

Una vez escrita la prueba unitaria ejecutamos el siguiente comando “`mocha -R *.js`” obtenemos el resultado que podemos observar en la figura 5.7.



```
Array
  #indexOf()
    ✓ debe retorna -1 si el valor no esta presente

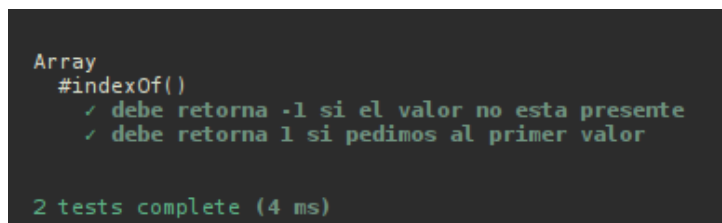
1 test complete (3 ms)
```

Figura 5.7: Resultado de ejecutar `mocha -R spec *.js`

Existe una función especial, que podemos observar en la línea 5 del siguiente código. Esta función especial es “`beforeEach`”, que tiene la misión de ejecutarse antes de cada test unitario. Nuestro ejemplo podemos ver que la hemos utilizado para inicializar variables.

```
1 var assert = require("assert");
2
3 var a;
4
5 beforeEach(function(){
6   a = [1,2,3];
7 });
8
9
10 describe('Array', function(){
11   describe('#indexOf()', function(){
12     it('debe retorna -1 si el valor no esta presente', function(){
13       assert.equal(-1, a.indexOf(5));
14       assert.equal(-1, a.indexOf(0));
15     });
16     it('debe retorna 1 si pedimos al primer valor', function(){
17       assert.equal(0, a.indexOf(1));
18     });
19   });
20 });
```

El resultado de ejecutar nuestro nuevo test es.



```
Array
  #indexOf()
    ✓ debe retorna -1 si el valor no esta presente
    ✓ debe retorna 1 si pedimos al primer valor

2 tests complete (4 ms)
```

Figura 5.8: Resultado de ejecutar `mocha -R spec array1-test.js`

Como se puede deducir, de los ejemplos anteriores, Mocha nos proporciona los mecanismos básicos para poder realizar test unitarios fáciles, rápidos y sobre todo sencillos.

Manual de usuario

Conclusiones



Bibliografía