**Universidad de Alcalá**



**Escuela Politécnica Superior**

**Máster Universitario en Ingeniería del Software para la Web**

Trabajo Fin de Máster

Estudio sobre Angular 2 y superior

Piero Rospigliosi Beltrán

2020

Universidad de Alcalá

Escuela Politécnica Superior

máster universitario en

ingeniería del software para la web

Trabajo Fin de Máster

“Estudio sobre Angular 2 y superior”

**Autor: Piero Rospigliosi Beltrán**

**Director: Carlos Delgado Hita**

Tribunal:

Presidente: ................................................

Vocal 1º: .................................................... Vocal 2º: ....................................................

Calificación: ..................................................

Fecha: ..... de ................................ de .........

En agradecimiento a todos mis compañeros y profesores que siempre estuvieron dispuestos a prestar ayuda.

Índice Resumido

[1. Introducción 1](#_Toc177754577)

[2. Objetivos del Proyecto 5](#_Toc177754578)

[3. Estado del Arte 9](#_Toc177754579)

[4. Estandarización y Legislación en la Web 53](#_Toc177754580)

[5. Evaluación de Accesibilidad Web 129](#_Toc177754581)

[6. Resumen y Conclusión 263](#_Toc177754582)

[7. Bibliografía 273](#_Toc177754583)

[8. Apéndice A. Herramientas de Evaluación y Reparación de Accesibilidad Web i](#_Toc177754584)

[9. Apéndice B. Glosario xxxvii](#_Toc177754585)

[10. Apéndice C. Páginas con Certificados de Accesibilidad Web xlv](#_Toc177754586)

Índice Detallado

[1. Introducción 1](#_Toc177754587)

[2. Objetivos del Proyecto 5](#_Toc177754588)

[3. Estado del Arte 9](#_Toc177754589)

[3.1. Discapacidad 12](#_Toc177754590)

[3.1.1. Discapacidad visual 13](#_Toc177754591)

[3.1.1.1. Ceguera 13](#_Toc177754592)

[3.1.2. Discapacidad auditiva 20](#_Toc177754595)

[3.1.2.1. Sordera 20](#_Toc177754596)

[4. Estandarización y Legislación en la Web 53](#_Toc177754628)

[4.1. Estandarización 56](#_Toc177754629)

[4.1.1. Estándar de iure 57](#_Toc177754630)

[4.1.2. Estándar de facto 58](#_Toc177754632)

[4.1.2.1. AENOR. Asociación Española de Normalización y Certificación 59](#_Toc177754633)

[4.1.2.2. ANSI: Instituto Nacional Americano para Estándares 60](#_Toc177754634)

[5. Evaluación de Accesibilidad Web 129](#_Toc177754683)

[5.1. Herramientas de Evaluación de Accesibilidad Web 132](#_Toc177754684)

[6. Resumen y Conclusión 263](#_Toc177754745)

[6.1. Seguimiento del proyecto 266](#_Toc177754746)

[7. Bibliografía 273](#_Toc177754749)

[8. Apéndice A. Herramientas de Evaluación y Reparación de Accesibilidad Web i](#_Toc177754750)

[9. Apéndice B. Glosario xxxvii](#_Toc177754751)

[10. Apéndice C. Páginas con Certificados de Accesibilidad Web xlv](#_Toc177754752)

Índice de Figuras

**1. INTRODUCCIÓN**

**2. OBJETIVOS DEL PROYECTO**

**3. ESTADO DEL ARTE**

[Figura 1. Clasificación de las discapacidades 12](#_Toc177754914)

[Figura 2. Paletas de colores según deficiencias visuales 16](#_Toc177754915)

[Figura 3. Paleta sin adaptación de colores 17](#_Toc177754916)

[Figura 4. Paleta para alteraciones a colores Azul-Amarillo 18](#_Toc177754917)

**4. ESTANDARIZACIÓN Y LEGISLACIÓN EN LA WEB**

[Figura 22. Estados implicados en el proceso de desarrollo de un estándar ISO 65](#_Toc177754935)

[Figura 23. Estructura de la WAI 72](#_Toc177754936)

**5. EVALUACIÓN DE ACCESIBILIDAD WEB**

[Figura 40. Tipos de Herramientas 134](#_Toc177754953)

[Figura 41. Resultado de WebXACT en http://www.uah.es. Apartado General 143](#_Toc177754954)

**6. RESUMEN Y CONCLUSIÓN**

[Figura 155. Planificación Inicial del Proyecto 266](#_Toc177755068)

**7. BIBLIOGRAFÍA**

**8. APÉNDICE A. HERRAMIENTAS DE EVALUACIÓN Y REPARACIÓN DE ACCESIBILIDAD WEB**

[Figura 160. Herramientas de Evaluación y Reparación de Accesibilidad Web xxxv](#_Toc177755073)

**9. APÉNDICE B. GLOSARIO**

**10. APÉNDICE C. PÁGINAS CON CERTIFICADOS DE ACCESIBILIDAD WEB**

[Figura 161. Páginas con Certificados de Accesibilidad Web xlvii](#_Toc177755074)

Introducción

Los frameworks como Angular surgen de la evolución natural de la programación hacia formas que nos permitan programar a más alto nivel, dando al programador mayor flexibilidad para la abstracción y el centrarse en los detalles propios de cada aplicación a desarrollar.

Así mismo también, de la necesidad de crear aplicaciones cada vez más robustas, lo cual nos conduce a trabajar con frameworks y lenguajes de programación que nos brinden mayor capacidad para el mantenimiento del código y su reutilización.

Esta evolución y aumento de necesidades no es ajeno al desarrollo web en la parte del cliente, comúnmente denominado front-end. Y se debe a la propia naturaleza del desarrollo web la cual está basada en la arquitectura cliente/servidor. En la cual un usuario (cliente), a través de una interfaz realiza peticiones al servidor. El servidor, al recibir la petición, se encarga de procesar la información y devolver el recurso solicitado al cliente, quien se encargará de presentar la información obtenida en el formato adecuado. Debido a que el número de servidores es limitado, interesa trasladar cada vez más responsabilidades a la capa del cliente, ya que estos serán procesados en el ordenador del cliente aligerando la carga de trabajo del servidor.

El tipo de responsabilidades que se han ido trasladando al cliente tienen que ver desde, el renderizado de la información, validaciones en formularios, seguridad, etc.

En este sentido Angular y Typescript, lenguaje de programación utilizado por Angular ofrecen una serie de ventajas que se serán explicadas en detalle en los siguientes capítulos de este trabajo.

Por su parte Typescript, que es una extensión de Javascript, tiene como particularidad el tipado, la orientación a objetos y su alineamiento con estándares nuevos de Javascript como EcmaScript 6 (ES6).

Angular, nos ofrece controladores, servicios, directivas para organizar el proyecto y características tan importantes como el two-way data binding que nos permite reflejar automáticamente cambios en nuestro modelo en el DOM de nuestra aplicación, tarea que tradicionalmente se hacia de forma manual.

.

Objetivos del Proyecto

Uno de los objetivos de este documento es hacer un repaso de la historia y características de este framework y de los componentes que lo que integran, partiendo desde los más básicos como Javascript, Typescript, EcmaScript y la relación entre ellos.

Mediante el repaso detallado de sus características se pretende explicar como cada una de ellas nos facilita el desarrollo de código limpio, robusto y mantenible.

También explicaremos la evolución de Angular, y la aparición de frameworks similares con la misma forma de trabajar.

JAVASCRIPT

Es un lenguaje de programación que permite crear acciones en las páginas web. Es un lenguaje de programación nativo de los navegadores quienes son los encargados de interpretar estos códigos.

Se define como orientado a objetos, basado en prototipos, débilmente tipado, dinámico, imperativo e interpretado.

ECMASCRIPT

Es el estándar que define como debe ser el lenguaje Javascript. Dicho de otra manera es la especificación que deben seguir los creadores de software para crear intérpretes para Javascript.

Javascript siendo un lenguaje interpretado que ejecuta las instrucciones directamente sin una previa compilación , cuenta con multitud de intérpretes como como los propios navegadores web o servidores como NodeJS.

Para establecer las reglas de juego que deben cumplir dichos intérpretes, ECMA Intenational publica el estándar ECMAScript.

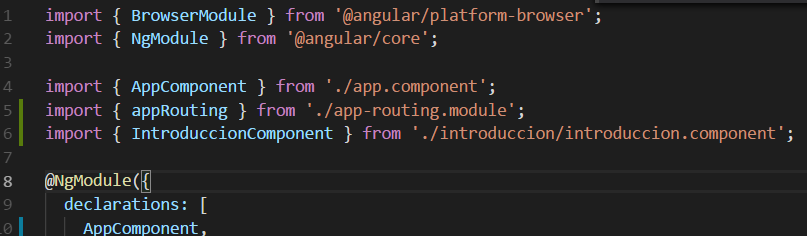
ECMAScript ha ido evolucionado con el tiempo y es en el año 2015 donde se lanza ECMAScript 6, siendo el estándar que utiliza TypeScript que se explicará en el siguiente apartado.

ECMAScript 6 introduce cambios importantes como:

-El uso de clases, que nos permite utilizar estructuras predefinidas como molde para crear objetos. Estos nos permiten declarar atributos y métodos, que en Javascript clásico equivaldrían a una variable a una función. La aparición de las clases nos brinda un valor importantísimo porque nos permite reutilizar con mayor facilidad nuestras estructura de datos.

IMAGEN CLASES

-El concepto de módulos que nos permite definir archivos que contienen directivas de importación y exportación a nivel superior.



-Declaraciones de variables, const y let. La primera nos permite crear constantes, y la segunda variables locales en nuestros bloques. Esto también nos facilita crear aplicaciones más robustas, ya que podemos usar estas declaraciones para asegurar el alcance de estas y si se puede modificar o no.

IMAGEN const let

-Otras características que nos permiten diseñar mejor nuestra lógica de negocio como el poder tener valores por defecto en nuestras funciones, el uso de lambdas o poder utilizar el paradigma de la programación funcional. Un ejemplo de estos son las funciones de orden superior, map, filter y reduce

.map: El resultado nos devuelve una colección de elementos que son el resultado de la expresión aplicada al atributo o elemento creado.

Por ejemplo: [1,2,3].map(n => n+1) . En este ejemplo se tomaría cada uno de los valores, que vendrían a ser la n y se sumaría 1, de esta forma el resultado esperado sería [2,3,4]

. filter: El resultado tendrá la misma forma que el argumento de entada, sin embargo solo se quedará con los elementos que cumpa la condición dada.

Por ejemplo: [1,2,3,4].filter(n=> n < 3) . Este caso le estamos diciendo que solo se quede con los valores menores que 3, por lo que el resultado esperado sería [1,2]

.reduce: Nos permite aplicar una función contra un acumular y cada valor de la matriz (de izquierda a derecha) para reducirlo a un solo valor.

Por ejemplo [1,2,3,4].reduce((acumulador, valorActual) => acumulador + valorActual). En este caso cada valor se sumará al actual valor acumulado hasta dicho momento. Actuando en este ejemplo como una sumataria donde el resultado sería 10.

Siendo los pasos:

0+1 => 1

1+2 => 3

3+3=> 6

6+4=> 10

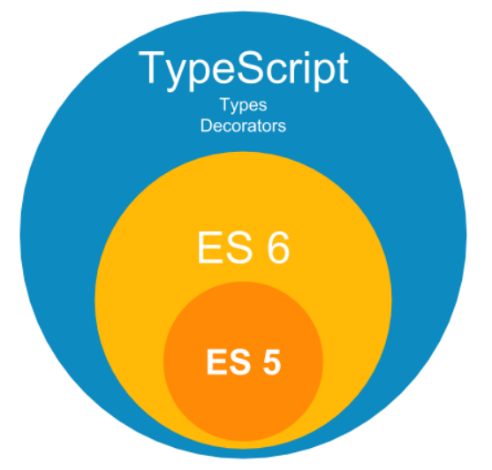
Como vemos cada nuevo valor se suma al anterior valor acumulado en dicho momento.

Este tipo de funciones nos permiten abstraernos de cómo se hace, y centrarnos en lo que queremos hacer para definir nuestra lógica de negocio de manera más clara y a su vez crear funciones más reutilizables y mantenibles.

IMAGEN EJEMPLO PROGRAMACION FUNCIONAL

## Typescript

De ECMAScript6 surge Typescript, que es un superset del primero. Esto quiere decir que incluye todas las funcionalidades de ES6 pero añade funcionalidades extra.



Entonces, ¿Qué funcionalidades extras ofrece con respecto ES6, para que la comunidad de Angular prefería utilizarlo en lugar del estándar?

Las características extras mas importantes con respecto a ES6, son:

-Es un lenguaje compilado: Lo cual es una ventaja porque nos permite encontrar errores en tiempo de compilación antes de abordar la ejecución. El compilador de Typescript transforma el código a código Javascript tradicional el cual puede interpretado por todos los navegadores.

-Variables tipadas: Nos permite aportar mayor robustez al código, porque nos permite encontrar en tiempo de compilación el paso de argumentos o asignaciones a variables que no corresponde su tipo.

Por ejemplo:

Definimos un array de tipo number

numeros:Array<number>

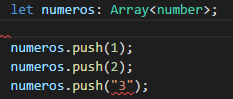
Luego añadimos dos elementos:

numeros.push(1);

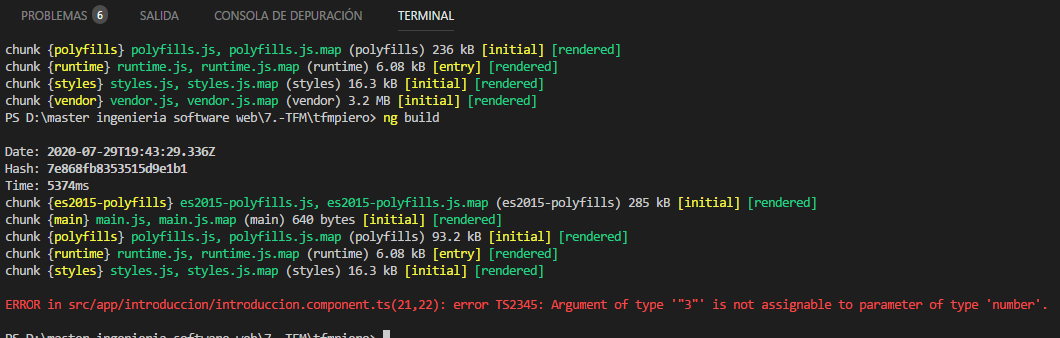
numeros.push(2);

Hasta el momento tenemos al array [1,2]. Ahora intentamos añadir un número, pero en forma de string.

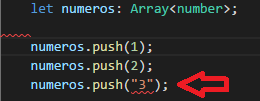
numeros.push(“3”);



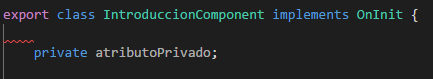
Esto en javascript clásico, ES5, no daría ningún problema. Sin embargo en Typescript veríamos un error en tiempo de compilación.



Esto nos ofrece una gran ventaja, porque nos evitamos encontrar el error en tiempo de ejecución. Incluso si utilizamos un editor adecuado, este mismo nos señalará el error al instante, sin necesidad de compilar el código.

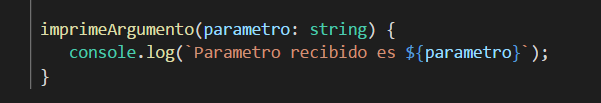


-Modificadores de acceso público, private y protected: Los cuales nos permiten definir el alcance de un miembro de una clase. Nuevamente, esto nos permite detectar errores en tiempos de compilación haciendo que nuestra aplicación sea más robusta.

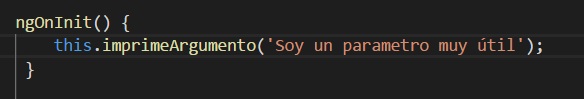


-Decoradores: Nos permite implementar el patrón decorador, el cual nos permite añadir funcionalidad extra de forma dinámica, lo cual nos permite evitar tener que crear nuevas clases que heredan de nuestra clase a expandi..

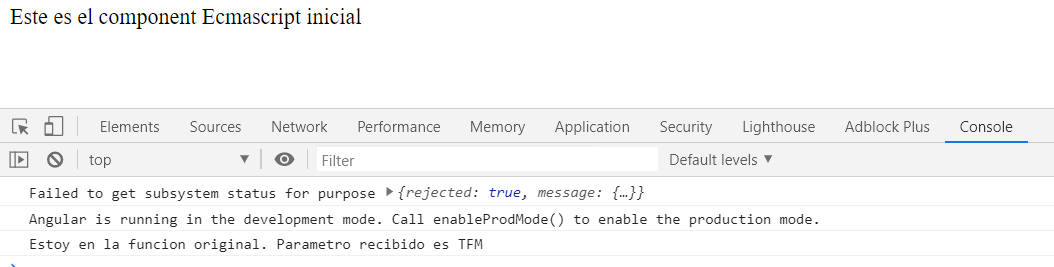
Por ejemplo imaginemos que tenemos la clase llamada ECMAScript la cual tiene una función que nos imprime el argumento pasado por parámetro.



Llamamos a la función



El resultado de esta función tal como se espera es la impresión en consola del parámetro recibido.



Sin embargo, ahora se requiere que cuando se llame a ciertas funciones que consideramos importantes, se imprima información relativa a su ejecución, como la clase de donde se invoca y los argumentos recibidos.

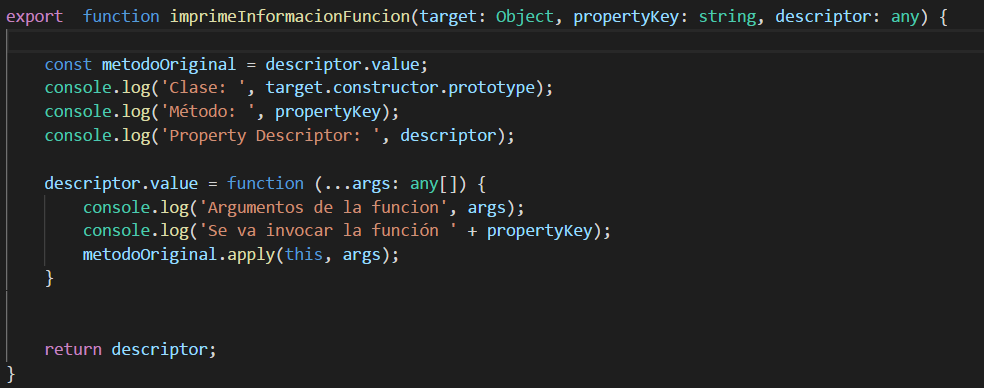
Podríamos modificarlas funciones deseadas y añadir esta funcionalidad o recurrir a la herencia y sobrescribir el método imprimeArgumento mediante nueva una nueva clase. Sin embargo, el principio de responsabilidad única (Single Responsability Principle, SRP) de SOLID numerito y explicación de solidº , nos indica que una clase o módulo debe tener uno y una solo responsabilidad.

Si nuestra función tiene un objetivo, que es imprimir argumentos, cualquier otra acción secundaria nos dificultará la legibilidad, mantenimiento y la escalabidad de nuestra función y nuestra aplicación.

Por ello vamos aprovechar la ventaja que nos ofrecen los decoradores y esconder todo el código secundario nuevo para que no ensucie nuestra función. De esta forma, el siguiente desarrollador a cambiar la funcionalidad principal, solo tendrá que preocuparse de eso, olvidándose de las acciones secundarias teniendo así nuestro código mayor legibilidad.

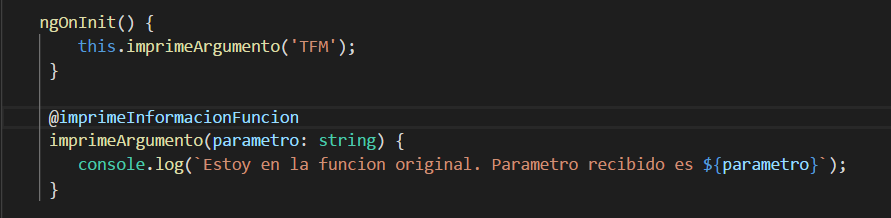
La manera en que funciona el decorador es que primero se entrará por la función del decorador y podemos realizar acciones con los metadatos recibidos de la función decorada.

Se crea del decorador el cual como podemos observar tiene una forma predefinida en los argumentos que nos facilita Typescript.

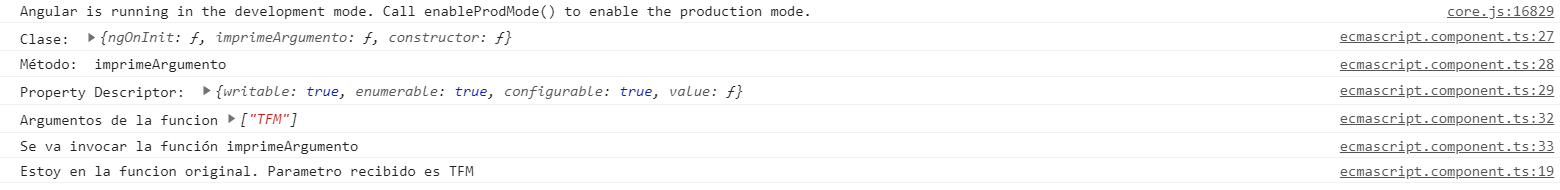


Vemos que imprimimos en consola la información de los metadatos de la función. Y luego invocamos a la función original para que se continue haciendo la funcionalidad principal.

Ahora podemos decorar nuestro método con nuestro nuevo decorador.



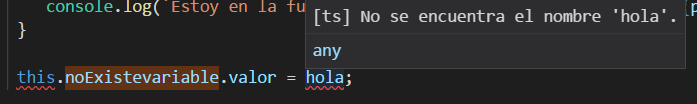
Vemos el resultado en la consola, y efectivamente ha impreso los metadatos para luego invocar la función original.



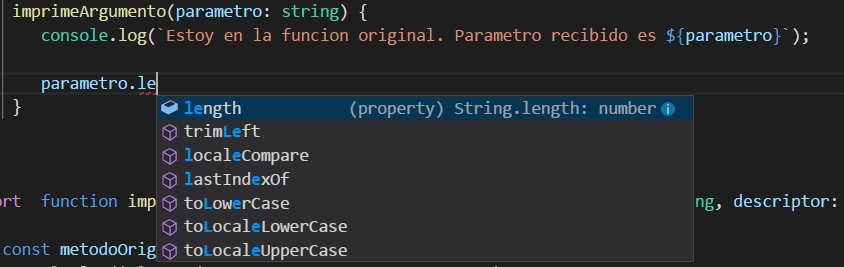
De esta forma hemos añadido código nuevo y secundario de forma elegante y sin ensuciar nuestra funcionalidad principal.

-Editores para trabajar con TypeScript. Es una experiencia más agradable y eficiente trabajar con un editor como Visual Studio Code, que nos permite tener autocompletado o detección errores al momento de programar.

Por ejemplo aquí vemos como nos indica que no existe la variable hola.



Y aquí nos sugiere mediante un autocompletar las posibles funciones a utilizar para la variable string parámetro



-El apoyo de grandes empresas y proyectos. Empresas como Microsoft, el creador de TypesScript. O Google, quien mantiene el framework Angular, el cual es objeto de estudio en este trabajo.

CARACTERISTICAS

<https://github.com/Microsoft/TypeScript/blob/730f18955dc17068be33691f0fb0e0285ebbf9f5/doc/spec.md#1>

### Discapacidad visual

Las deficiencias visuales más comunes…

#### Ceguera

La ceguera implica…

#### Baja visión

Se refiere a aquellas discapacidades que disminuyen la calidad de la visión, sin imposibilitarla…

#### Falta de sensibilidad a los colores

Se traduce en una falta de respuesta a ciertos colores…

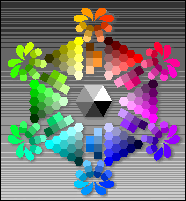
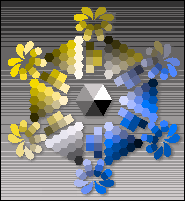
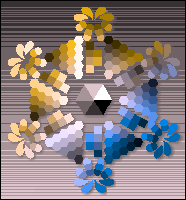
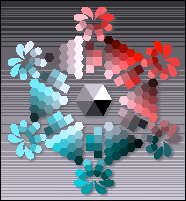
   

Figura 2. Paletas de colores según deficiencias visuales

…

****

Esos tres botones se verían por una persona con defecto de visión del color de la siguiente manera.

****

La lección para el diseñador de interfaces es sencilla. No codificar ninguna conducta importante únicamente mediante colores.

Evaluación de Accesibilidad Web

**Descripción experimental, cuando sea necesario, descripción del diseño, resultados, etc.**

Este capítulo denominado Evaluación de Accesibilidad Web es el más práctico de los vistos hasta ahora debido a que se aplicarán todos los conocimientos teóricos adquiridos para realizar una evaluación exhaustiva de una serie de plataformas Web…

Resumen y Conclusión

**Resumen del trabajo en un máximo de cien (100) palabras.**

En este último capítulo se realizará un balance de los resultados obtenidos después de la realización del Proyecto Fin de Carrera…

## Presupuesto

**Presupuesto (en su caso), que incluya: ejecución material (materiales y mano de obra), gastos generales y beneficio industrial, honorarios de dirección y redacción (tarifas del Colegio, en su caso), coste de ejecución por contrata y presupuesto total.**

... De esta forma, lo que hacemos es estimar la duración del proyecto en torno a 9 meses con una jornada de 6 horas diarias, exceptuando los fines de semana. Así entonces, obtenemos un total de 1080 horas invertidas en este Proyecto Fin de Carrera…

## Conclusiones y Futuras Líneas de Trabajo

**Conclusiones y futuro trabajo.**

Después de dar por finalizado este Proyecto Fin de Carrera “Accesibilidad en la Web: Normas y Aplicación”, hacemos balance sobre los conocimientos adquiridos y las impresiones obtenidas a lo largo del mismo…

Bibliografía

**Bibliografía, donde se indicará el conjunto de las referencias utilizadas como citas y otros materiales de consulta, siempre que, a lo largo del trabajo, se afirma algo que no se demuestra, conteniendo cada una los siguientes datos:**

**- Autor/es.**

**- Título del artículo, libro, monografía,...**

**- Editorial o nombre de la revista y editorial.**

**- Número de la revista, volumen y páginas consultadas.**

**- Año de publicación.**

**- Asimismo, en este apartado se reseñarán los distintos catálogos utilizados.**

1. Abascal, J., Valero, P. (2001), Curso Introducción a la Interacción Persona-Ordenador: El libro electrónico, AIPO, <http://griho.udl.es/ipo/libroe.html>
2. Asociación Española de Normalización y Certificación (1986), Normas y Publicaciones <http://www.aenor.es/>
3. …

Apéndice A. Herramientas de Evaluación y Reparación de Accesibilidad Web

**En un Apéndice se adjunta información extra, no contenida en puntos anteiores que de más facilidad a la comprensión del tema en cuestión.**

En este apartado se ofrece una guía de referencia de las herramientas de evaluación y/o reparación de accesibilidad Web existentes en la actualidad…

Apéndice B. Glosario

**Un glosario es catálogo de palabras con definición o explicación de cada una de ellas.**

El objetivo de este glosario es facilitar el acceso a una definición de los principales términos que se mencionan a lo largo de este libro.

A

Accesible. El contenido es accesible cuando puede ser usado por alguien con discapacidad.

AENOR. Asociación Española de NORmalización y Certificación, es una entidad dedicada al desarrollo de la normalización y la certificación (N+C) en todos los sectores industriales y de servicios.

AIPO. Asociación Interacción Persona-Ordenador

ANSI. American National Standards Institute es una organización sin ánimo de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

Applet. Componente de software que corre en el contexto de otro programa, por ejemplo un navegador Web.

ASME. American Society of Mechanical Engineers es una asociación profesional, que además ha generado un código de construcción, inspección y pruebas para equipos.

ASQ. American Society for Quality, primeramente conocido como ASQC es una comunidad global basada en el conocimiento formada por expertos de control de calidad.

ASTM. American Society for Testing and Materials, es una organización de estándares voluntaria internacional que desarrolla y produce estándares técnicos para materiales, productos, sistemas y servicios.

AU. Agentes de Usuario, es decir, aplicaciones de usuario.

**B**

BITV. Barrierefreie InformaTionstechnik Verordnung. Tecnología de la Información Libre de Barreras. Estándar alemán cuyo contenido se basa en las directrices 1.0 de W3C-WAI.

BSI. British Standards Institution.

...