Universidad Carlos III de Madrid

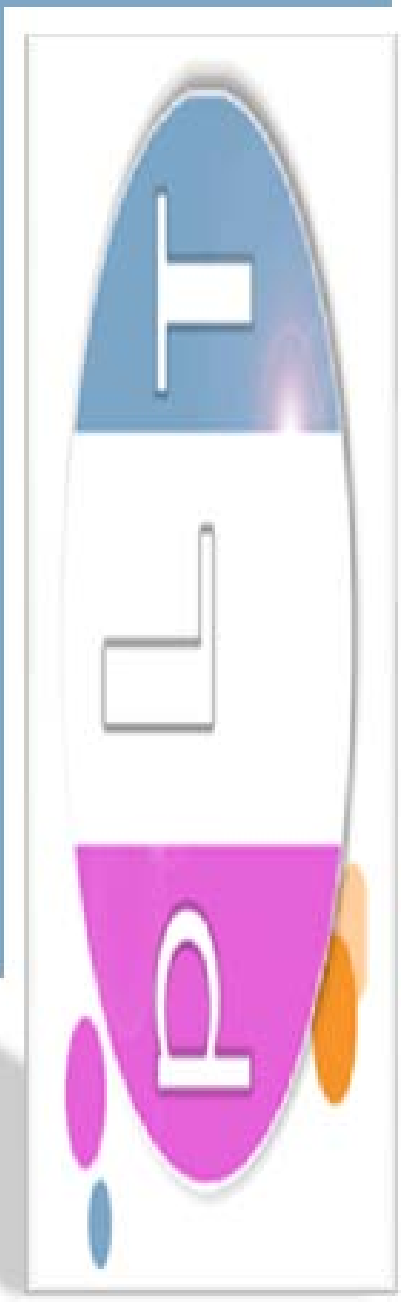
Escuela Politécnica Superior

Ingeniería Técnica

en Informática de Gestión

Memetracker-Web:

Desarrollo de una interfaz web para un sistema de monotorización de la blogosfera política.



Autor: Ainhoa Sánchez de Pablo Moreno

Tutor: Cesar de Pablo

INDICE

1. INTRODUCCIÓN **5**

1.1. Contexto **5**

1.2. Objetivos **6**

2. PROYECTOS RELACIONADOS TOMADOS COMO REFERENCIA **7**

2.1. Funcionalidad **7**

2.2. Semejanzas con nuestro proyecto **8**

2.3. Diferencias con nuestro proyecto **9**

3.TECNOLOGÍAS **11**

3.1. Arquitectura y diseño de sistemas web **11**

3.2. Accesibilidad **16**

3.2.1. Preguntas frecuentes sobre las “pautas de accesibilidad al contenido en la web 1.0” 16

3.2.2. Motivos del diseño accesible 18

3.2.3. Pautas de Accesibilidad al Contenido de la Web. 20

3.3. Optimización CSS y Validación de código W3C **23**

3.4. Elección de Resolución **24**

3.5. Elección de Navegadores **26**

3.6. HTML **30**

3.7. CSS **34**

3.8. JSP **36**

3.9. JDBC **39**

4. METODOLOGÍA DEL DESARROLLO **42**

4.1. Análisis y Diseño **43**

4.1.1.- Entorno de trabajo 44

4.1.2.- Tecnologías y herramientas utilizadas 45

4.1.3.- Crawler 45

4.1.4.- Base de Datos 46

4.1.4.1.- Modelo de Entidad Relación de la BD..... 47

4.1.4.2.- Grafo relacional 49

4.1.4.3.- Diccionario de Términos..... 50

4.1.4.4.- Conexión a BD..... 55

4.1.4.5.- Objetos 56

4.1.4.6.- Consultas a BD 59

4.1.5. Diseño y funcionalidad de las páginas desarrolladas 66

4.1.5.1 Diseño inicial de las páginas con plantillas estáticas (HTML). 66

4.1.5.2. Diseño final, funcionalidades implementadas y herramientas utilizadas para la visualización de las páginas. 71

5. FUNCIONALIDADES..... 80

5.1 Sistema de Paginación:..... **80**

5.2. Sistema de Votación:..... **83**

5.3. Nube de Tags:..... **83**

5.4. Introducir Comentarios:..... **84**

6. PRUEBAS 86

6.1.- Pruebas para la paginación:..... **86**

6.2.- Pruebas en las votaciones:..... **88**

6.3.- Pruebas para la introducción de comentarios:..... **89**

6.4.- Pruebas para enlazar las páginas:..... **91**

6.5.- Si no recupera datos que muestre mensaje informativo:..... **91**

6.6.- Comprobar la correcta visualización teniendo en cuenta las longitudes máximas y mínimas de los campos... **92**

7. SERVICIOS REST 93

7.1. Presentando REST..... **93**

7.2. Los 4 principios de REST..... **93**

7.3. REST utiliza los métodos HTTP de manera explícita..... **94**

7.4. REST no mantiene estado..... **97**

7.5. Servicios con estado vs. sin estado..... 98

7.6. Responsabilidad del servidor..... **99**
7.7. Responsabilidades del cliente de la aplicación..... 100
7.8. REST expone URIs con forma de directorios..... 101
7.9. REST transfiere XML, JSON, o ambos..... 102
7.10. Conclusión..... 104
7.11. Utilidad de REST en futuras funcionalidades..... 104
8. PRESUPUESTO..... 107
8.1. Método de los Puntos de función..... 107
8.1.1. ¿Qué son los puntos de funcnión?..... 108
8.1.2. Procedimiento de estimación de los puntos de función..... 109
8.1.2.1. Obtener información del Sistema..... 109
8.1.2.2. Identificar los componentes del Sistema..... 110
8.1.2.3 Calcular número de elementos y su complejidad..... 112
8.1.2.4. Obtener los PF sin Ajustar (PFSA)..... 113
8.1.2.5. Obtener los PF Ajustados (PFA)..... 114
8.1.2.6. Cálculo del esfuerzo..... 114
8.1.2.7. Cálculo de la duración del proyecto..... 114
8.1.2.8. Cálculo del presupuesto del proyecto..... 115
9. LÍNEAS FUTURAS..... 116
9.1 Arquitectura lógica..... 116
9.2. Control de versiones:..... 116
9.3. Accesibilidad:..... 117
10. CONCLUSIONES..... 118
11. BIBLIOGRAFÍA..... 119
12. ANEXO..... 122

1. INTRODUCCIÓN

1.1. Contexto

Podemos denominar a la Blogosfera como un sistema virtual que contiene páginas Web en forma de blogs, y que se organizan según un tema o perfil de interés. De esta manera estas páginas representan la información que se desea mostrar y la Blogosfera el lugar donde mostrarla. La conexión entre estos blogs es un fenómeno social, que crea tendencias, opiniones y una serie de reacciones tratadas como un ente colectivo.

Últimamente el mercado, cada vez más, empieza a poner su ojo en Internet. Permitiendo a empresas pequeñas y jóvenes convertirse en los nuevos sucesores de la prensa del futuro.

La Blogosfera tiene un comportamiento similar al que la naturaleza realiza en su evolución, como es la selección natural, permitiendo quedarse de entre la información recibida de varios sitios con la que más nos interese según nuestros objetivos.

En cuanto al contexto tecnológico, podemos hablar desde los primeros tiempos de la computación cliente-servidor, en la que cada aplicación tenía su propio programa cliente que servía como interfaz de usuario que tenía que ser instalado por separado en cada ordenador personal de cada usuario. El cliente realizaba peticiones a otro programa -el servidor- que le daba respuesta. Una mejora en el servidor, como parte de la aplicación, requería normalmente una mejora de los clientes instalados en cada ordenador personal, añadiendo un coste de soporte técnico y disminuyendo la productividad.

A diferencia de lo anterior, las aplicaciones Web generan dinámicamente una serie de páginas en un formato estándar, como HTML o XHTML, que soportan por los navegadores Web comunes. Se utilizan lenguajes interpretados en el lado del cliente, tales como JavaScript, para añadir elementos dinámicos a la interfaz de usuario. Generalmente cada página Web en particular se envía al cliente como un documento estático, pero la secuencia de páginas ofrece al usuario una experiencia interactiva. Durante la sesión, el navegador Web interpreta y muestra en pantalla las páginas, actuando como cliente para cualquier aplicación Web.

Las interfaces Web tienen ciertas limitaciones en las funcionalidades que se ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio como dibujar en la pantalla o arrastrar-y-soltar que no están soportadas por las tecnologías Web estándar. Los desarrolladores Web generalmente utilizan lenguajes interpretados o script en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez (lo que suele resultar molesto a los usuarios). Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con tecnologías en el lado del servidor, como por ejemplo PHP. Como ejemplo, AJAX, es una técnica de desarrollo Web que usa una combinación de varias tecnologías.

1.2. Objetivos

Agregar y organizar en un sitio la información política y el debate político que se produce en Internet. Para ello se pretende organizar información que se encuentra distribuida y dispersa en diferentes medios electrónicos y blogs. La organización hace uso de tecnologías semánticas para organizar la información por las facetas más importantes como son personas, partidos y temas.

Organizar esta información de forma que esté relacionada entre sí, teniendo en cuenta su relevancia, su actualidad y que permita una profundización adaptada a los intereses del usuario. Creando un diseño que sea vistoso y fácil de seguir para el usuario.

Investigar métodos adecuados para la gestión y acceso de la información generada por comunidades en tiempo real, así como las técnicas de visualización y navegación por espacios de información complejos y en constante evolución, creando páginas accesibles y amigables para todo tipo de usuarios y PC.

Por último está el aspecto social que pretende que los usuarios puedan valorar la información, a través de votaciones y comentarios sobre los elementos del debate.

2. PROYECTOS RELACIONADOS TOMADOS COMO REFERENCIA

2.1. FUNCIONALIDAD

En primer lugar cabe destacar que éste es un proyecto que se ha realizado entre varias personas, y cada una de ellas se ha dedicado a desarrollar y elaborar una parte concreta para que con la aportación de cada uno se consiguiese un proyecto sólido y consolidado.

Relación de personas que han participado con sus correspondientes aportaciones al proyecto:

- Guillermo García Cubero - crawler
- Álvaro Rojas Bravo - base de datos
- Ignacio Nieto Vidauzarraga - detección de tópicos/seguimiento de noticias
- Alvaro Segura Cuervo - herramientas para la documentación de personas y partidos
- José Vicente Sevillano Martín - modelo de predicccion de fuentes RSS - crawler v2
- **Ainhoa Sanchez de Pablo Moreno - interfaz web**
- César de Pablo Sánchez - coordinación

Por lo tanto a la hora de tomar como referencia otros proyectos, voy a centrarme exclusivamente en la parte que hace relevancia a la interfaz web. También cabe destacar que el proyecto que aquí se está detallando es un prototipo de interfaz web, y las referencias tomadas son ya interfaces definitivas y en uso.

DIGG

Se trata de una página web en inglés que permite a los usuarios enviar, votar y jerarquizar las informaciones que crean en sus webs o encuentran en Internet. Es una de las webs precursoras en la valoración social de la información en la Red. Para introducir información hay que estar registrado.

MENÉAME

Es un web que te permite enviar una historia que será revisada por todos y será promovida, o no, a la página principal. Cuando un usuario envía una noticia ésta queda en la *cola de pendientes* hasta que reúne los votos suficientes para ser promovida a la página principal.

NUESTRO PROTOTIPO

Es una web cuya temática es mantener un debate político, intenta mantener toda su información relacionada, es decir, la información que contienen cada uno de los bloques de una página está relacionada entre sí. Intenta dar profundidad a la información, podemos sacar mucha más información sobre algo particular que nos interese simplemente seleccionando aquel punto que elijamos. Contiene cuatro páginas principales para presentar la información, la Página de Inicio que nos presenta de forma generalizada la información, la Página de Personas, la Página de Temas y la Página de Partidos, presentan información específica de cada elemento seleccionado, facilitando también la navegabilidad y el acceso a toda información relacionada con este elemento.

2.2. SEMEJANZAS CON NUESTRO PROYECTO

Digg:

Digamos que Digg fue la referencia para menéame, en cuento a estética podemos decir que estructuralmente es similar a menéame, por lo en líneas posteriores describiremos esta semejanza.

Es accesible y está validada con los estándares.

En común con nuestro prototipo y a diferencia de menéame no usa el concepto del Karma, le da a todos los usuarios la misma importancia.

Menéame:

Se permiten votos anónimos, es decir, no es necesario registrarse como usuario para poder realizar el voto.

Ambas tienen en común que son accesibles y que están validadas con los estándares.

La presentación es bastante similar, se tomo como referencia para el prototipo. Tenemos un bloque principal a la izquierda que contiene las noticias más relevantes y otro bloque a la derecha que contiene nube de tags y otros temas relacionados.

El bloque de la noticia se divide en cabecera, titular de la noticia, con el logo del usuario que la introducido y al posicionarnos sobre el obtenemos más información al igual que ocurre en menéame. Posee una cabecera y logo común para todas las páginas con funcionalidades de búsqueda.

En cuanto a la funcionalidad podemos decir que las semejanzas con nuestro prototipo se basan en la valoración social de la información, es decir, se facilita opciones de

votación, introducción de comentarios y según esta información se valora la representación de la información, manteniendo el debate activo.

2.3. DIFERENCIAS CON NUESTRO PROYECTO

Digg:

No permite votos anónimos.

Menéame:

Es necesario registrarse para enviar historias y agregar comentarios, nuestro proyecto de momento es un prototipo de interfaz, por lo que intenta simplemente plasmar la visualización del contenido, la accesibilidad y usabilidad de ésta y las primeras funcionalidades básicas que queremos obtener, de momento permitimos introducir comentarios con un alias que no es necesario q este registrado.

Diferencias comunes a los dos proyectos tomados como referencia:

En Digg y Menéame son los usuarios los que envían las noticias y URLs. En nuestro proyecto esta información es actualizada automáticamente.

Tienen muchas más funcionalidades que nuestro proyecto, ya que son web ya rodadas y con experiencia. Por ejemplo, en menéame cuando pinchamos con el ratón sobre el número de votos de la noticia nos dirige a otra página con una serie de etiquetas que permiten ver los comentarios, votos…etc. En nuestro prototipo como de momento somos varias personas trabajando a la vez y no están todas las funcionalidades de cada uno implementadas nos vemos obligados a dejar estas consultas y no visualizar este tipo de información.

Una de las diferencias más destacadas es que nuestra información está relacionada entre los diferentes bloques, y además podemos obtener la información en profundidad ya que si queremos saber más de un tema basta con pinchar sobre este y seguiremos teniendo más información a cerca de él y de su información relacionada. Mientras que en los prototipos relacionados cada noticia es independiente del resto de bloques.

Nuestro proyecto permite agrupar las noticias en relación a una persona, partido, o tema determinado. Al igual que se puede obtener un grupo de noticias y las personas, partidos o temas que están relacionados con estas.

Otra diferencia es que se centra más en mostrar información relacionada con una temática concreta como es la política.

En nuestro prototipo se ha intentado mejorar la paleta de colores y el diseño de la interfaz. Se han dejado los bloques mucho más diferenciados de manera que sea fácil

distinguir las funcionalidades en cada momento. Se mantiene la estructura en todas las páginas para facilitar la compresión en su uso al usuario.

En el prototipo hemos querido diferenciar cada página con diferentes colores para que el usuario sepa identificar en cada momento en que página se encuentra. En cambio en las referencias que hemos tomado mantiene un mismo color para todas sus páginas.

En nuestro prototipo no se le da valor al orden de las noticas, partidos o personas relacionadas por el número de votos. Saldrán primero aquellas que tengan una fecha más actual o que el administrador considere que son más relevantes. En cambio en menéame se tiene en cuenta tanto los votos como el karma de los usuarios que han votado. Resumiendo, en nuestro caso queremos destacar las noticias más actuales o que hemos recogido en las últimas fechas, en cambio menéame le da más importancia a aquellas que tienen más impacto social, a través de las votaciones e integridad de los usuarios. En digg ocurre lo mismo a diferencia del término de Karma, que como se comento en las semejanzas con nuestro prototipo no existe.

3.TECNOLOGÍAS

3.1. Arquitectura y diseño de sistemas web

Las aplicaciones web se han convertido en pocos años en complejos sistemas con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, dando servicio a procesos de negocio de considerable envergadura y estableciéndose sobre ellas requisitos estrictos de accesibilidad y respuesta. Esto ha exigido reflexiones sobre la mejor arquitectura y las técnicas de diseño más adecuadas.

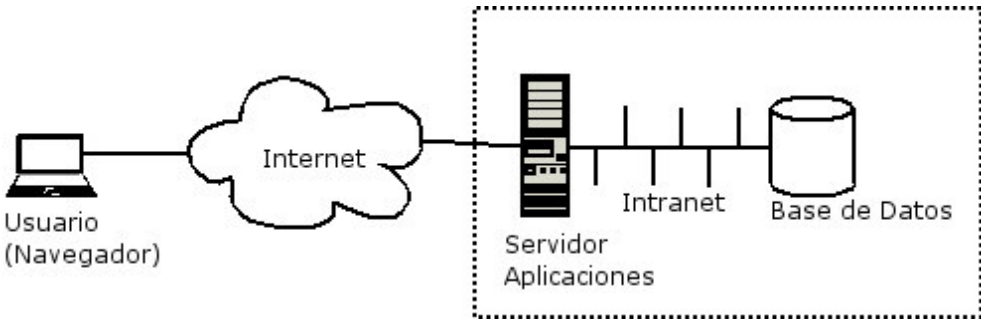
En los últimos años, la rápida expansión de Internet y del uso de intranets corporativas ha supuesto una transformación en las necesidades de información de las organizaciones. En particular esto afecta a la necesidad de que:

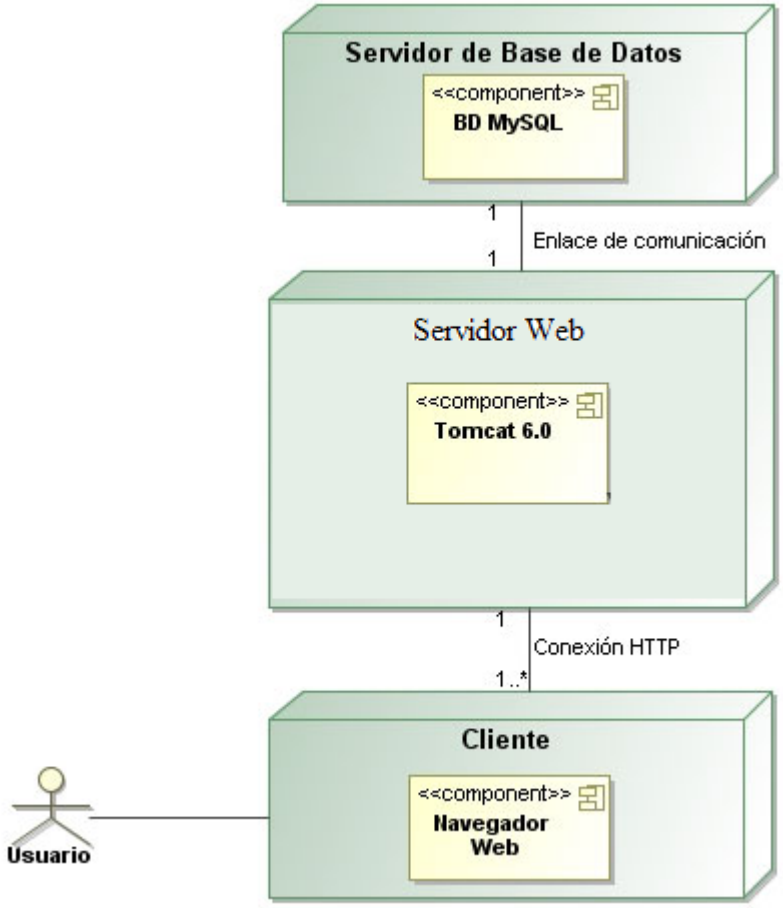
1. La información sea accesible desde cualquier lugar dentro de la organización e incluso desde el exterior.

2. Esta información sea compartida entre todas las partes interesadas, de manera que todas tengan acceso a la información completa (o a aquella parte que les corresponda según su función) en cada momento.

Estas necesidades han provocado un movimiento creciente de cambio de las aplicaciones tradicionales de escritorio hacia las aplicaciones web, que por su idiosincrasia, cumplen a la perfección con las necesidades mencionadas anteriormente. Por tanto, los sitios web tradicionales que se limitaban ha mostrar información se han convertido en aplicaciones capaces de una interacción más o menos sofisticada con el usuario. Inevitablemente, esto ha provocado un aumento progresivo de la complejidad de estos sistemas y, por ende, la necesidad de buscar opciones de diseño nuevas que permitan dar con la arquitectura óptima que facilite la construcción de los mismos.

El usuario interacciona con las aplicaciones web a través del navegador. Como consecuencia de la actividad del usuario, se envían peticiones al servidor, donde se aloja la aplicación y que normalmente hace uso de una base de datos que almacena toda la información relacionada con la misma. El servidor procesa la petición y devuelve la respuesta al navegador que la presenta al usuario. Por tanto, el sistema se distribuye en tres componentes: el navegador, que presenta la interfaz al usuario; la aplicación, que se encarga de realizar las operaciones necesarias según las acciones llevadas a cabo por éste y la base de datos, donde la información relacionada con la aplicación se hace persistente. Esta distribución se conoce como el modelo o arquitectura de tres capas.





Podemos dividir la aplicación en tres áreas o niveles:

1. **Nivel de presentación**: es el encargado de generar la interfaz de usuario en función de las acciones llevadas a cabo por el mismo.

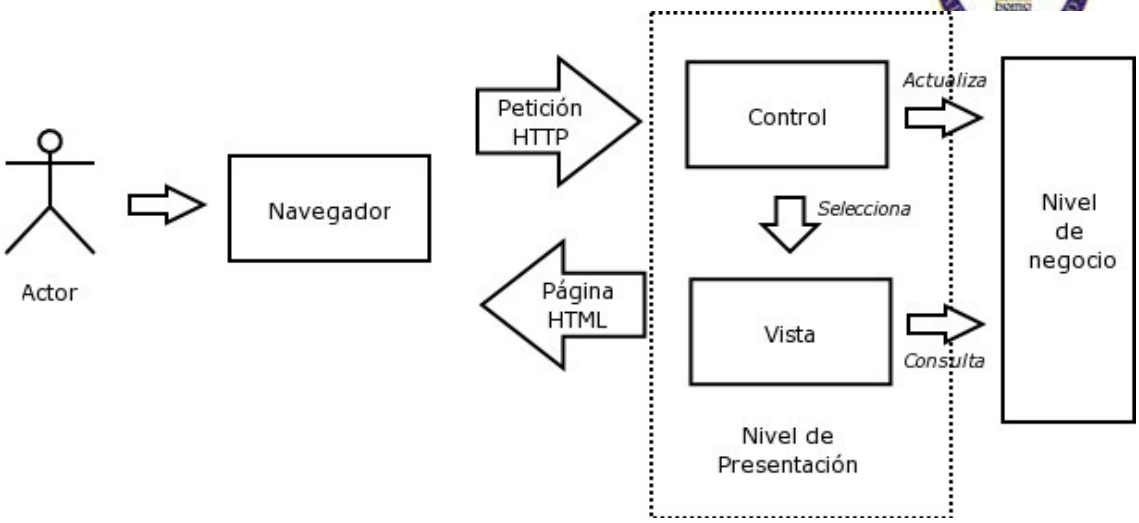
2. **Nivel de negocio**: contiene toda la lógica que modela los procesos de negocio y es donde se realiza todo el procesamiento necesario para atender a las peticiones del usuario.

3. **Nivel de administración de datos**: encargado de hacer persistente toda la información, suministra y almacena información para el nivel de negocio.

Los dos primeros y una parte del tercero (el código encargado de las actualizaciones y consultas), suelen estar en el servidor mientras que la parte restante del tercer nivel se sitúa en la base de datos (notar que, debido al uso de procedimientos almacenados en la base de datos, una parte del segundo nivel también puede encontrarse en la misma). Teniendo en cuenta estas características en la arquitectura de los sistemas web, a continuación veremos algunos patrones de diseño de aplicación básica que pueden facilitar un diseño apropiado para este tipo de sistemas.

Uno de los patrones que ha demostrado ser fundamental a la hora de diseñar aplicaciones web es el **Modelo-Vista-Control (MVC)**. Este patrón propone la separación en distintos componentes de la interfaz de usuario (vistas), el modelo de negocio y la lógica de control. Una vista es una "fotografía" del modelo (o una parte del mismo) en un determinado momento. Un control recibe un evento disparado por el usuario a través de la interfaz, accede al modelo de manera adecuada a la acción realizada, y presenta en una nueva vista el resultado de dicha acción. Por su parte, el modelo consiste en el conjunto de objetos que modelan los procesos de negocio que se realizan a través del sistema.

En una aplicación web, las vistas serían las páginas HTML que el usuario visualiza en el navegador. A través de estas páginas el usuario interactúa con la aplicación, enviando eventos al servidor a través de peticiones HTTP. En el servidor se encuentra el código de control para estos eventos, que en función del evento concreto actúa sobre el modelo convenientemente. Los resultados de la acción se devuelven al usuario en forma de página HTML mediante la respuesta HTTP.

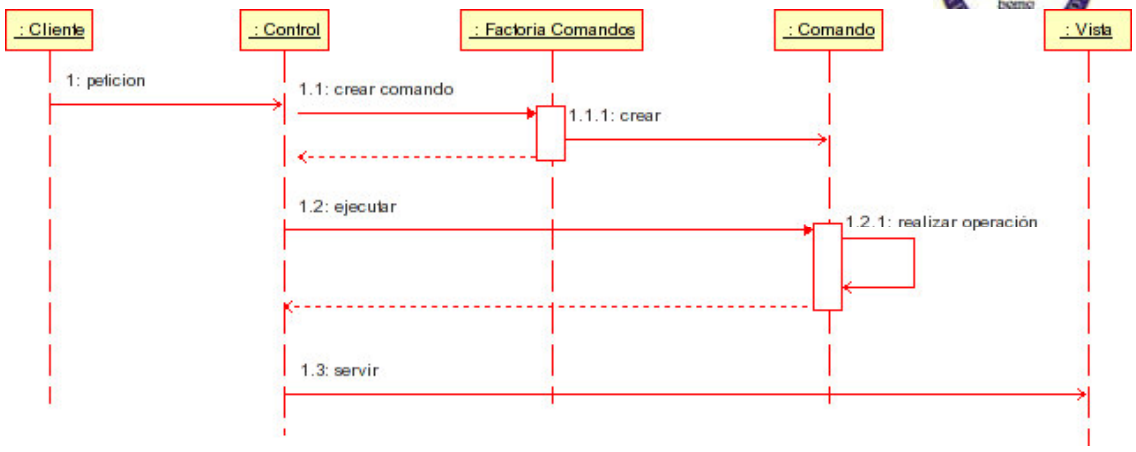


La clave está en la separación entre vista y modelo. El modelo suele ser más estable a lo largo del tiempo y menos sujeto a variaciones mientras que las vistas puede cambiar con frecuencia, ya sea por cambio del medio de presentación (por ejemplo HTML a WAP o a PDF) o por necesidades de usabilidad de la interfaz o simple renovación de la estética de la aplicación. Con esta clara separación las vistas pueden cambiar sin afectar al modelo y viceversa. Los controladores son los encargados de hacer de puente entreambos, determinando el flujo de salida de la aplicación (qué se ve en cada momento).

Si tomamos como referencia la plataforma J2EE, las vistas podrían ser JSPs (o plantillas Velocity o documentos XML tratados con XSLT, ...etc) los controladores serían servlets y

el modelo podría implementarse utilizando EJBs u objetos Java normales en combinación con frameworks de persistencia como Hibernate o JDO.

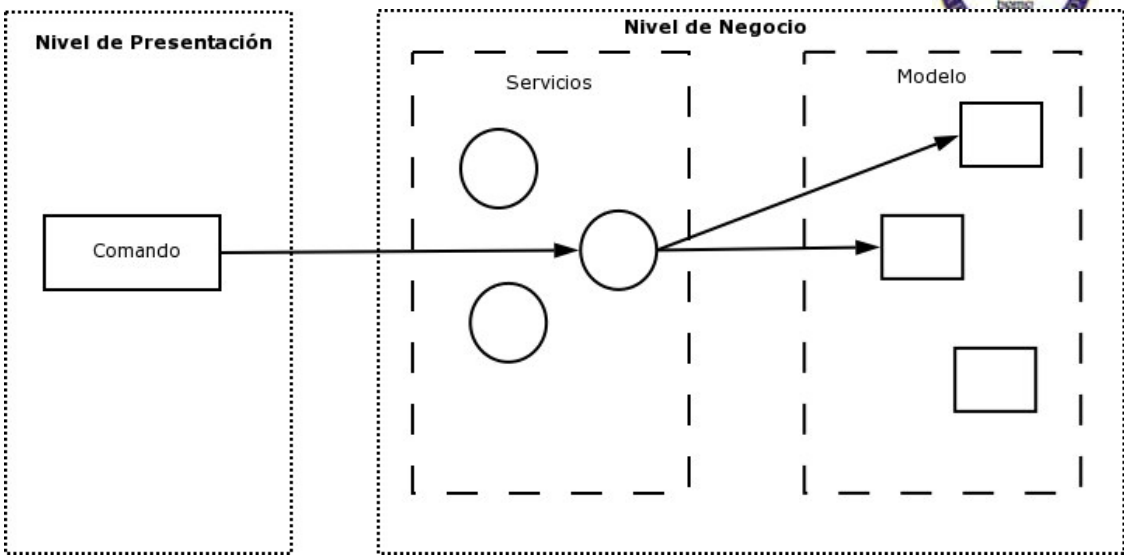
A la hora de utilizar el MVC en aplicaciones web es conveniente utilizar un único servlet como controlador para toda la aplicación. Este control gestiona todas las peticiones, incluyendo invocaciones a servicios de seguridad, gestión de excepciones, selección de la siguiente vista, etc. Esto se conoce como el patrón **Front Controller** (controlador frontal o fachada). El poder centralizar en un solo punto servicios como la gestión de conexiones a base de datos, comprobaciones de seguridad o gestión de errores favorecen que la aplicación sea mucho más robusta y aísla de todos estos aspectos al resto de componentes. Si aplicamos esto junto con el patrón **Command** podemos seguir lo que se conoce como **estrategia Comando y Controlador**. En este caso el componente Control (entendido dentro del marco MVC y no como el servlet controlador de este caso concreto) está formado por el servlet que recibe las peticiones y por un conjunto de clases implementadas a través del patrón **Command** en las que delega las tareas a llevar a cabo según la acción invocada. Estos comandos seleccionan la siguiente vista en función de los resultados de su procesamiento y el servlet controlador sirve esta vista al cliente.



La utilización de esta estrategia tiene varias ventajas: permite cambiar fácilmente el procesamiento para una acción determinada sustituyendo el comando que la implementa por otro, permite reutilizar comandos y favorece el encadenamiento de dos o más comandos para la implementación de tareas complejas.

Actualmente existen varios frameworks de desarrollo de aplicaciones web basados en el patrón MVC como son Struts, Spring, WebWork o Maverick entre otros. El de mayor difusión en la actualidad es Struts. Este framework (al igual que muchos otros) utiliza la estrategia que hemos visto.

Si recordamos los tres niveles en los que dividimos una aplicación web (presentación, negocio, administración de datos), lo que hemos dicho hasta ahora afecta fundamentalmente al nivel de presentación y, en algunos casos, al nivel de negocio. Respecto a este último no hay ninguna "receta" para asegurar un buen diseño. La aplicación de patrones de diseño conocidos y el respeto a los principios de encapsulación de información y distribución de responsabilidad (que cada objeto haga solo aquello que le es propio) es la mejor manera de conseguir un diseño apropiado. Un principio que suele resultar de bastante utilidad es agrupar en servicios las operaciones de negocio relacionadas. Por ejemplo, en una aplicación de comercio virtual, podríamos tener un servicio para la gestión de usuarios, otro para la tramitación de pedidos, etc. De esta manera, si aplicamos la estrategia vista anteriormente, los comandos invocarían uno o más métodos de estos servicios, que serían los que accederían a los objetos que constituyen el modelo.



Por último hablaremos brevemente del nivel de administración de datos. Este último nivel es proporcionado por el framework de persistencia utilizado junto con la base de datos propiamente dicha. Actualmente existen diversos frameworks de persistencia de datos (Entity beans, Hibernate, JDO, OJB, etc.) que realizan automáticamente el mapeo

entre objetos de la aplicación y tablas en la base de datos relacional o incluso podríamos optar por utilizar JDBC.

Cada uno tiene unas características y funcionalidad concretas que obligarán a adaptar el diseño de manera apropiada. Solo apuntar que normalmente, en una aplicación web una petición HTTP equivale a una transacción. Es decir, si mientras se sirve la petición ocurre un error todo lo hecho a raíz de esa petición debe deshacerse. Por tanto, en función del modelo de persistencia, habrá que actuar de manera que los cambios a la base de datos no se hagan definitivos hasta que la petición se haya completado. La mejor forma de gestionar esto es en el servlet controlador, pues al ser el que finaliza la

petición enviando la respuesta al cliente, es el mejor lugar para estar seguros de que todo ha ido bien y hacer definitivos los cambios en la base de datos.

Con esto completamos el breve análisis de la arquitectura de los sistemas web modernos, donde hemos dado apuntes para un diseño lo más correcto posible de los mismos, sobre todo en lo referente al nivel o capa de presentación.

3.2. Accesibilidad

3.2.1. Preguntas frecuentes sobre las “pautas de accesibilidad al contenido en la web 1.0”

1. ¿Qué son las Pautas de Accesibilidad al Contenido en la Web?

Son una serie de premisas, en concreto catorce, que se deben tener en cuenta para desarrollar un sitio web accesible por todo tipo de usuarios, navegadores y otros recursos basados en la voz. Cada una de estas pautas está asociada a uno o más puntos de verificación que informan como lograr el objetivo de la pauta a través de la tecnología empleada.

2. ¿Qué son las "prioridades", los "niveles de adecuación" y cómo se usan los logotipos?

Las prioridades se asignan a los puntos de verificación para indicarle al desarrollador cuáles de estos puntos son más necesarios de satisfacer para que los usuarios tengan más capacidad de acceso a la información de un sitio.
Existen 3 prioridades, de más prioritaria a menos serían: la prioridad 1, la prioridad 2 y la prioridad 3.

Los niveles de adecuación, como su nombre indica, es el nivel de adaptación de la página para poder ser accesible por los diferentes usuarios.

1.- El nivel de adecuación "A", incluye los puntos de verificación de prioridad uno.
2.- El nivel de adecuación "AA", incluye la prioridad 1 y 2.
3.- El nivel de adecuación "AAA", incluye las 3 prioridades.

Los logotipos se usan para identificar qué páginas siguen estas pautas y en qué nivel de adecuación se encuentran.

3. ¿Para quién están escritas estas pautas?

Para todas aquellas personas que estén interesadas en facilitar el acceso a la información de personas con discapacidad o con medios diferentes a los que comúnmente son usados.

4. ¿Cómo afectan estas pautas a la manejabilidad y apariencia de los sitios para los usuarios sin discapacidad?

En realidad no se ven afectados de forma negativa, la manejabilidad y apariencia para ellos puede ser la misma que en el caso de que no fuesen accesibles. Pero para ellos también presentan las páginas accesibles varias ventajas ya que son páginas más flexibles, dando la opción de operar con ellas de diferentes modos (con ratón, teclado, voz...etc.), además de ofrecerles páginas más inteligibles y útiles si no soportan tecnologías o navegadores específicos.
Se crean sitios más fácilmente navegables y ofrece la posibilidad de acceder a ellos a través de diversos dispositivos.

5. ¿Es más costoso hacer un sitio accesible?

No supone un incremento significativo del coste del desarrollo, puede incluso reducir el coste del mantenimiento y actualización. Los sitios accesibles pueden llegar a resultar más rentables puesto que permiten el acceso a más tipo de usuarios y de recursos.

6. ¿Hay herramientas que puedan ayudarme? ¿Puedo comprobar la accesibilidad de mi sitio?

   Sí que hay herramientas, por ejemplo cito una de ellas: “Bobby”, que es un revisor de accesibilidad y ejecuta un test automático “on-line” de muchos de los puntos de verificación. Aunque siempre se debe contar con una revisión manual, puesto que no se puede ejecutar una revisión automática completa.

   Sí existen páginas que facilitando el código, enlace o archivo, nos permiten comprobar el nivel de adecuación de nuestra página.

3.2.2. Motivos del diseño accesible

Las pautas se enmarcan en dos motivos generales: asegurar una transformación correcta y hacer el contenido comprensible y navegable.

Asegurar una transformación correcta

Siguiendo estas pautas, los desarrolladores de contenidos pueden crear páginas que se transformen de manera correcta. Este tipo de páginas siguen siendo accesibles a pesar de cualquiera de las limitaciones descritas en la introducción, incluyendo las discapacidades físicas, sensoriales y cognitivas, las restricciones debidas al trabajo y las barreras tecnológicas. He aquí algunas claves para el diseño de páginas que se transforman correctamente:

- Separe el contenido de la estructura y de la presentación.
- Proporcione textos (incluidos equivalentes textuales). Los textos pueden ser interpretados por la inmensa mayoría de los mecanismos de navegación y accesibles a la inmensa mayoría de usuarios.
- Cree documentos que funcionen incluso si el usuario no puede verlos y/u oírlos. Proporcione información que sirva al mismopropósito y función tanto en audio como en vídeo para que se disponga de un canal sensorial alternativo. Esto no significa crear una versión pregrabada de audio de todo el sitio para hacerlo accesible a los usuarios ciegos. Los usuarios ciegos pueden usar lectores de pantalla para interpretar toda la información textual de una página.
- Cree documentos que no sólo funcionen con un tipo determinado de hardware. Las páginas deben poder ser usadas por personas que no dispongan de ratón, con

pantallas pequeñas, de baja resolución, en blanco y negro, sin pantallas, o sólo con salida de voz o texto, etc.

Hacer comprensible y navegable el contenido

Los desarrolladores de contenidos deben hacer que el contenido sea comprensible y navegable. Esto incluye no sólo la utilización de un lenguaje claro y simple, sino también proporcionar mecanismos comprensibles para navegar por y entre páginas. El proporcionar herramientas de navegación e información orientativa en las páginas maximizará la accesibilidad y la utilidad. No todos los usuarios pueden utilizar las pistas visuales tales como mapas, barras de desplazamiento, marcos contiguos o gráficas que guían a los usuarios videntes de navegadores de sobremesa. Los usuarios pierden también información del contexto cuando sólo pueden visualizar una parte de la página, tanto porque acceden a la página palabra por palabra (con sintetizadores de voz o dispositivos braille), o de sección en sección (pantallas pequeñas o magnificadores de pantalla). Sin una información orientativa, es posible que los usuarios no comprendan tablas, listas o menús muy largos, etc.
El motivo para hacer el contenido comprensible y navegable se recoge en las pautas 12 a 14.

Cómo se organizan las pautas

Son catorce pautas o principios generales de diseño accesible. Cada pauta incluye:
• Número de la pauta.
• Exposición de la pauta.
• El fundamento que sustenta la pauta y algunos grupos de usuarios que se benefician de ella.
• Una lista de definiciones de los puntos de verificación. Las definiciones de los puntos de verificación explican cómo se aplica la pauta en situaciones típicas de desarrollo de contenidos. Cada definición de punto de verificación incluye:
• Número del punto de verificación.
• Explicación del punto de verificación.
• La prioridad del punto de verificación. Los puntos de verificación de Prioridad 1 están resaltados a través del uso de hojas de estilo.
• Notas informativas opcionales, ejemplos aclaratorios y referencias cruzadas a pautas o puntos de verificación relacionados.

Cada punto de verificación pretende ser lo suficientemente específico, como para que cualquiera que revise una página o sitio pueda comprobar que dicho punto ha sido satisfecho.

Prioridades

Cada punto de verificación tiene un nivel de prioridad asignado por el Grupo de Trabajo y fundamentado en su impacto en la accesibilidad.

[Prioridad 1]

Un desarrollador de contenidos de páginas Web tiene que satisfacer este punto de verificación. De otra forma, uno o más grupos de usuarios encontrarán imposible acceder a la información del documento. Satisfacer este punto de verificación es un requerimiento básico para que algunos grupos puedan usar los documentos Web.

[Prioridad 2]

Un desarrollador de contenidos de páginas Web debe satisfacer este punto de verificación. De otra forma, uno o más grupos encontrarán dificultades en el acceso a la información del documento. Satisfacer este punto de verificación eliminará importantes barreras de acceso a los documentos Web.

[Prioridad 3]

Un desarrollador de contenidos de páginas Web puede satisfacer este punto de verificación. De otra forma, uno o más grupos de usuarios encontrarán alguna dificultad para acceder a la información del documento. Satisfaciendo este punto de verificación mejorará la accesibilidad de los documentos Web.

Algunos puntos de verificación tienen especificado un nivel de prioridad que puede variar bajo ciertas condiciones.

Adecuación

Los niveles de adecuación, como su nombre indica, es el nivel de adaptación de la página para poder ser accesible por los diferentes usuarios.

1.- El nivel de adecuación "A", incluye los puntos de verificación de prioridad uno.
2.- El nivel de adecuación "AA", incluye la prioridad 1 y 2.
3.- El nivel de adecuación "AAA", incluye las 3 prioridades.

3.2.3. Pautas de Accesibilidad al Contenido de la Web.

Pauta 1

Proporcionar alternativas equivalentes a los contenidos que puedan presentar alguna dificultad para aquellos usuarios que presenten algún tipo de discapacidad visual, auditiva u otro tipo de dificultad para el acceso al contenido.

Por ejemplo, proporcionar a los contenidos no textuales algún equivalente textual y viceversa. De esta manera los usuarios que presenten alguna discapacidad visual, tendrán traductores que a través del sonido les interpretarán con el texto facilitado la información no textual. Y en el caso contrario, usuarios con dificultad para la lectura o

personas analfabetas podrán a través del contenido no textual, por ejemplo una imagen, entender fácilmente el contenido textual.

Pauta 2: No se base sólo en el color

Esta pauta lo que nos quiere señalar es que si empleamos los colores con algún tipo de funcionalidad debemos buscar otro medio alternativo para representarla, puesto que existirán usuarios que presenten dificultades visuales, ya sea por carecer visión o por no tener una buena percepción de los colores, además de usuarios que empleen pantallas en blanco y negro o utilicen otros medios que no puedan representarlas.

Pauta 3. Utilice marcadores y hojas de estilo y hágalo apropiadamente

Los marcadores deben usarse de forma correcta para que cierto tipo de usuarios con software especializado puedan entender la organización y navegación de la página.
Por ejemplo, para las personas con discapacidad visual, existen software especializados que se encargan de leerles la página, si a los marcadores no se le da el uso correcto, este tipo de software se ve incapaz de interpretar la página, por lo que el usuario en cuestión no podría acceder a la información.
El uso de hojas de estilo en cascada facilita a los autores Web la revisión de la accesibilidad y usabilidad del sitio que están creando.

Pauta 4. Identifique el idioma usado

"Use marcadores que faciliten la pronunciación o interpretación de texto abreviado o extranjero".

Al especificar el idioma usado, permitimos al software especializado que pueda cambiar automáticamente al nuevo lenguaje, haciendo más accesible la información a los usuarios. Se debe añadir la expansión de las abreviaturas y los acrónimos.

Pauta 5. Cree tablas que se transformen correctamente

Evitar el uso de tablas para maquetar páginas, o para otra finalidad que no sea la de tabular. Por ello nos debemos asegurar de asignarles los marcadores correctos para permitir esta funcionalidad.

Pauta 6. Asegúrese de que las páginas que incorporan nuevas tecnologías se transformen correctamente

Debemos asegurarnos si empleamos tecnologías actuales de que la página seguirá siendo accesible cuando estas no estén soportadas o se desactive su uso. Por ejemplo, si usamos Javascript, y un usuario tiene en su navegador desactivada esta opción, debería poder acceder a la información sin ningún tipo de problemas.

Pauta 7. Asegure al usuario el control sobre los cambios de los contenidos tempo-dependientes

Debemos asegurarnos de que aquellos elementos que presenten movimiento permitan ser detenidos, ya que existen cierto tipo de usuarios y de lectores de pantalla que esta cualidad les puede dar problemas para acceder a la información

Pauta 8. Asegure la accesibilidad directa de las interfaces de usuario incrustadas

Si existe un objeto incrustado en la página que tiene una interfaz independiente a la que usa la página en cuestión, debemos asegurarnos de que esta sea también accesible y sino facilitar soluciones alternativas.

Pauta 9. Diseñe para la independencia del dispositivo

Usar las características o propiedades adecuadas para la activación de los diferentes elementos de la página a través de diferentes dispositivos de entrada.

Pauta 10: Utilice soluciones provisionales

Existen tecnologías y medios antiguos que se espera que desaparezcan pero mientras tanto es necesario utilizar estas soluciones provisionales para que cualquier usuario que todavía posea estos medios pueda acceder a la información.

Pauta 11. Utilice las tecnologías y pautas W3C

Se recomienda el uso de estas tecnologías porque poseen características accesibles ya incorporadas, además de poseer revisiones para que se tengan en cuenta los temas de accesibilidad en la fase de diseño y siguen unos estándares que son más fáciles de seguir por los diferentes recursos. Esto no impide que usemos otras tecnologías pero siempre debemos proporcionar páginas equivalentes accesibles.

Pauta 12. Proporcione información de contexto y orientación

Agrupar e informar sobre los elementos de la página de manera que sea fácil su seguimiento.

Pauta 13. Proporcione mecanismos claros de navegación.

Proporcionar ayudas de navegación fácilmente comprensibles por el usuario, para permitirle encontrar fácilmente la información que está buscando.

Pauta 14. Asegúrese de que los documentos sean claros y simples.

Asegurarse de que toda la información que presenta la página puede ser entendida por cualquier tipo de usuario.

3.3. OPTIMIZACIÓN CSS Y VALIDACIÓN DE CÓDIGO W3C

La utilización de CSS (hojas de estilo en cascada) es una de las maneras más efectivas para optimizar y posicionar un sitio web. Influye en el funcionamiento integral de la web, abreviando y limpiando el código HTML de una página. En un comienzo, esta técnica se utilizaba para regular estilos y comportamientos. Pero actualmente su desarrollo ha avanzado al punto de que permite diseñar completamente un sitio web casi sin la necesidad de trabajar sobre el código HTML.

Los resultados que se obtienen con la implementación de esta tecnología superan otras alternativas en cuanto a posicionamiento en buscadores, velocidad de carga de la página en el navegador, ahorro de espacio en el servidor, accesibilidad, compatibilidad con más navegadores, entre otras, ya que se utiliza un código limpio y breve, sintácticamente correcto.

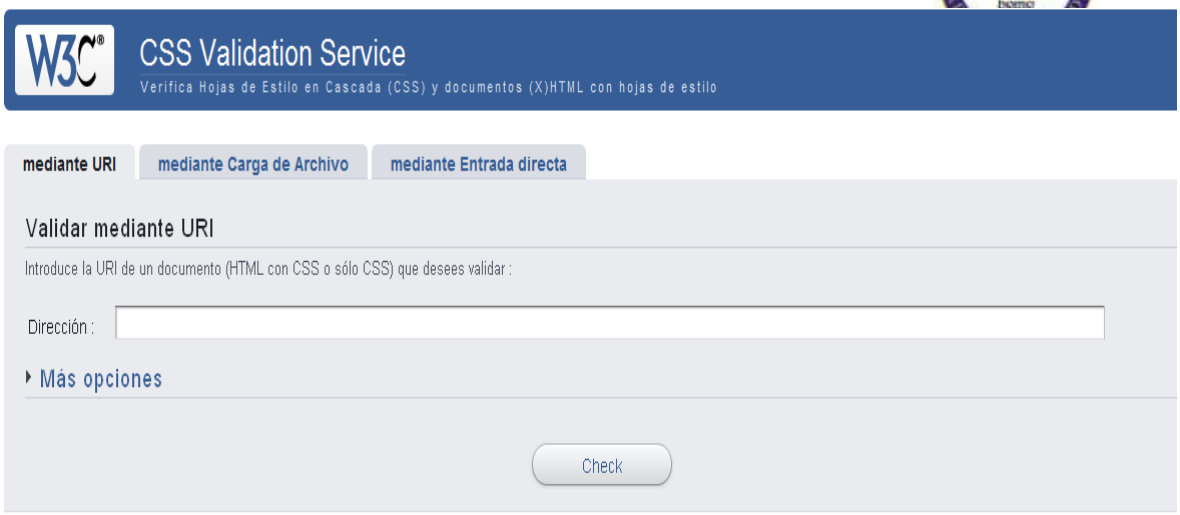
Existe una entidad mundial que certifica los sitios correctamente optimizados mediante un validador online. La W3C (The World Wide Web Consortium) es la reguladora de los estándares web a nivel mundial y acredita la calidad de un sitiio web.

El servicio de optimización consiste en analizar su sitio actual, re-diseñarlo completamente reescribiendo el código HTML y utilizando hojas de estilo en cascada para la maquetación de las páginas y validar el código en la W3C para obtener el certificado de XHTML y CSS acreditados.

Cómo validar una página web mediante un programa de validación XHTML

Después de crear un página web, utilizaremos el servicio de validación (http://validator.w3.org) para comprobar si el código HTML cumple con el estándar del W3C para su tipo de documento. Para utilizar el servicio de validación, seguiremos los siguientes pasos:

- Iniciamos un navegador e introducimos la dirección http://validator.w3.org



- Si queremos validar un página que aún no hayamos publicado en el servidor web, hacemos clic en Examinar dentro de Validate by file upload. Esto nos permite cargar un archivo de nuestra máquina local.

- Hacemos clic en Check.

Una vez hayamos completado estos pasos el explorador web nos enviará la página web que hayamos introducido o seleccionado al sitio web del w3c. El programa de validación del sitio examinará el documento XHTML y enviará al explorador web una página de resultados.

Debemos de tener en cuenta que el documento HTML o XHTML es válido (es decir, cumple el estándar) sólo cuando no hay errores. Por lo tanto, si vemos errores, los corregimos en el documento web y repetimos el procedimiento de validación.

El programa de validación no sólo ofrece una lista de errores, también ofrece sugerencias de corrección que tiene que realizar para que el código cumpla el estándar.

Además de validar los documentos web almacenados en los archivos de las unidades local o de red, también podemos validar cualquier página web que ya esté publicada en un servidor web.

3.4. ELECCIÓN DE RESOLUCIÓN

Las aplicaciones encargadas de presentar las páginas web son los navegadores (browsers), entre los que cabe destacar por su importancia y difusión Internet Explorer, Netscape Navigator y Opera.

La interfaz de un navegador, en su modo estándar, presenta al usuario una o más barras superiores (de menús, de dirección, etc.), una barra inferior (la barra de estado) y un espacio principal, la ventana, en el que son presentadas las páginas web.

La forma de esta ventana es siempre rectangular, pero su tamaño varía, dependiendo del monitor y de la tarjeta gráfica. Podemos hablar de dos tamaños de pantalla diferentes:

Tamaño absoluto: es el tamaño "real" de la ventana del monitor, medido generalmente en pulgadas. Depende del monitor.

Resolución o tamaño relativo: viene determinada por el número de pixels que se muestran en la ventana del monitor, siendo el píxel la unidad mínima de información que se puede presentar en pantalla, de forma generalmente rectangular. Depende de la tarjeta gráfica.

El tamaño absoluto se suele expresar en pulgadas de diagonal (1 pulgada = 25,4 mm). El más común en la actualidad es de 17'' en ordenadores de sobremesa, aunque todavía quedan bastantes equipos antiguos con monitores de 15'' y existen en el mercado bastantes de 21''. El tamaño absoluto de los monitores de los equipos portátiles suele ser de 14-15''.

En cuanto a la resolución, los valores más comunes son de 800x600 y de 1024 x768 pixels, aunque quedan todavía usuarios que trabajan por debajo, a 640x480, y por encima, a resoluciones de 1152x864 y 1280x960 pixels.

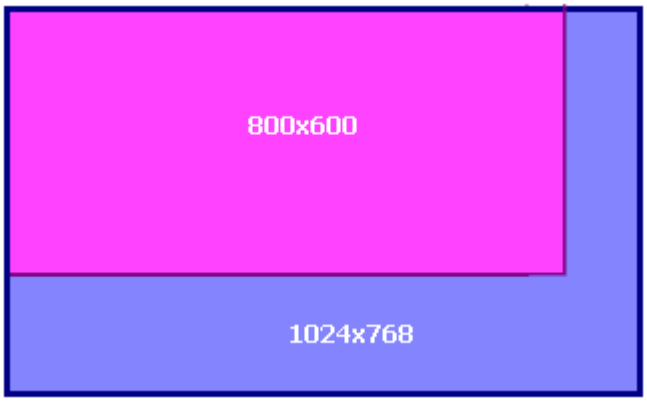
El tamaño absoluto y la resolución deben estar en concordancia para una visualización correcta, siendo valores aceptables los siguientes:

"14"- "15": Resolución máxima apreciable: 800x600
"17": Resolución máxima apreciable: 800x600 ó 1024x768
"21": A partir de 1024 x 768

Las posibles resoluciones de trabajo de un equipo dependen sobre todo de la calidad del monitor y de la tarjeta gráfica del ordenador, y se configuran, en sistemas operativos Windows, bien haciendo click con el botón derecho del ratón sobre el escritorio, seleccionando la opción Propiedades y accediendo a la pestaña Configuración, bien desde Inicio > Panel de control > Pantalla.

La importancia de la resolución de pantalla sobre la forma de ser visualizadas las páginas web en la ventana del navegador es muy importante. A mayor resolución se dispone de más puntos de información para presentar los elementos en pantalla, pero estos puntos son más pequeños, con lo que los elementos de la interfaz (textos, imágenes, objetos de formulario, etc.) se ven más pequeños.

Tamaño de página y resoluciones



Si diseñamos una página para una resolución dada, ocupando toda la ventana del navegador, aquellos usuarios que la visualicen a resoluciones menores no tendrán espacio en pantalla para contener toda la página, por lo que se verán obligados a usar las barras de desplazamiento del navegador. Por el contrario, aquellos usuarios que la visualicen a resoluciones mayores tendrán demasiado espacio en pantalla para tan poca página, por lo que les quedará bastante espacio vacío, sin contenidos.

Para solucionar estas diferencias, lo normal es que se diseñen las páginas web para una resolución base, generalmente la más usada en la actualidad (800x600), y se construyan internamente mediante tablas o capas de tamaños relativos, con anchuras definidas en %, con lo que se consigue que al ser visualizadas en monitores de más resolución se "abran", ocupando todo el espacio de pantalla disponible.

Las pegas que tiene este sistema son que no da soporte a los usuarios de con menores resoluciones y que, en el caso de resoluciones mayores, el diseño de la página puede verse seriamente afectado al modificar sus elementos las dimensiones originales.

Otra posibilidad es maquetar toda la página dentro de un contenedor padre (una tabla o capa) y asignar a éste una alineación centrada, con lo que la página quedará en el centro de la pantalla si se usa una resolución mayor que la de diseño.

3.5. ELECCIÓN DE NAVEGADORES

Un **navegador**, **navegador red** o **navegador web** es un programa que permite visualizar la información que contiene una página web.

El navegador interpreta el código, HTML generalmente, en el que está escrita la página web y lo presenta en pantalla permitiendo al usuario interactuar con su contenido y navegar hacia otros lugares de la red mediante enlaces o hipervínculos.

La funcionalidad básica de un navegador web es permitir la visualización de documentos de texto, posiblemente con recursos multimedia incrustados. Los documentos pueden estar ubicados en la computadora en donde está el usuario, pero

tambien pueden estar en cualquier otro dispositivo que esté conectado a la computadora del usuario o a través de Internet, y que tenga los recursos necesarios para la transmisión de los documentos.

Actualmente el navegador más utilizado en el mundo es Internet Explorer en su versión 7, algunas empresas indican que esta ventaja se debe a que viene integrado con Windows, detrás de éste está el navegador de Mozilla Firefox, el cual se está popularizando cada vez más. Firefox es un competidor serio al producto de Microsoft que ya alcanza una quinta parte de la cuota total. Luego le sigue la versión 6 de Internet Explorer, con una cuota de poco menos de 19,21%, Safari con más del 8% es otro navegador en rápida progresión. Existen también los navegadores, Netscape Navigator, Opera y Chrome los cuales tienen un uso de menos del 2% en el mercado.

ANÁLISIS DE LOS 3 EXPLORADORES MÁS USADOS: INTERNET EXPLORER, FIREFOX Y SAFARI

Mozilla Firefox:

Por la sencillez, estabilidad y el gran número de posibilidades que ofrece Mozilla Firefox se convierte en el mejor navegador que existe. Hay gran variedad de skins para variar su apariencia y es posible añadirle "applets" para configurar distintos estilos. El uso de pestañas sin dudas constituye una gran innovación, que luego fue copiada en versiones posteriores de otros exploradores de Internet.

Lo bueno también radica en que se trata de un navegador de código abierto (es decir que su código de fuente es liberado o disponible libremente). Por ser multiplataforma puede soportar distintos sistemas operativos lo que permite adaptar su funcionamiento a la plataforma en la que es ejecutado.

Este premiado navegador de Mozilla es anunciado como el más rápido y el más seguro que permite una completa personalización para adaptarlo a nuestros gustos cuando exploramos la Web, aunque en ocasiones poco estable y con un consumo de recursos medianamente elevado.

Internet Explorer

Es el navegador de Internet más utilizado de la actualidad. Sin embargo no siempre es elegido como el mejor ya que se le reconocen carencias frente a otras opciones de exploradores Web. No respeta los estándares de la web por lo que fuerza a los

diseñadores de páginas a crear dos versiones de sus sitios: una para todos los navegadores y otra para Explorer.

Este explorador gratuito de Microsoft está integrado a Windows y a medida que nuevas versiones fueron apareciendo, fue modernizándose e incorporando funciones

innovadoras de sus competidores (como la tecnología RSS, o la exploración con pestañas rápidas, etc.). En cuanto al uso de pestañas debemos resaltar que es posible mostrar las miniaturas de las páginas de las pestañas que se encuentren abiertas y también es posible crear grupos de pestañas de manera fácil para abrir todo el conjunto cada vez que lo queramos. Si bien se anuncian mejoras en cuanto a la protección de datos no está catalogado como uno de los más seguros para el resguardo de información.

Presenta una interfaz renovada y mejorada posibilitando maximizar el área donde vemos la página web. Incorporó las búsquedas de Internet a su barra de herramientas y de esta forma evita la acumulación de barras. Presenta la posibilidad de hacer zoom en un área determinada.

Safari

Este navegador de Apple, denominado Safari crece cada día más en el mercado. Si bien está lejos de ocupar el primer lugar en lo que a exploradores web respecta, posee muchas características rescatables que lo hacen digno de estar incluido en la lista de los mejores navegadores.

Mediante la tecnología llamada RSS se puede reunir en canales de noticias los titulares y resúmenes de prensa que proporcionan las agencias de noticias, los blogs y las comunidades de Internet. Esto permite agrupar y ver más rápidamente los canales de noticias sin publicidad y de forma sencilla. Otra de las ventajas promocionadas es la posibilidad de guardar las páginas web y enviarlas por e-mail aunque hayan desaparecido de Internet. Es ideal para guardar páginas web de corta vida como informes bancarios o artículos estadísticos.

También mediante Safari podemos navegar con completa seguridad ya que protege tus datos privados sin posibilidad de filtrar información confidencial. También los controles parentales para la navegación de los niños es muy efectiva ya que se puede especificar a

qué sitios pueden acceder sus hijos en la Barra de Favoritos. Al activar los controles ellos sólo podrán ver las páginas web especialmente permitidas.

Este es un navegador de dos caras: por un lado, en los sistemas operativos Mac es un muy buen navegador, mientras que la versión de Windows consume una cantidad

excesiva de memoria RAM. Su principal ventaja es ser uno de los más rápidos del mercado.

A la hora de decidir por qué explorador decantarnos para visualizar nuestro prototipo de interfaz, hemos seleccionado los dos más usados por los usuarios. Para poder adaptarnos a estos dos exploradores ha sido necesario analizar detalladamente sus similitudes y diferencias para que nuestro prototipo se visualice correctamente en ambos.

SEMEJANZAS Y DIFERENCIAS ENTRE INTERNET EXPLORER Y FIREFOX:

Es hora de analizar a dos de los navegadores Web más famosos y utilizados por los usuarios, que en honor a la verdad poseen muchas más diferencias que semejanzas entre ambos.

Si de diferencias hablamos tanto la descarga de archivos como la gestión de añadiduras son dos de las principales carencias de Internet Explorer. Por el contrario Firefox en estos dos aspectos es considerablemente superior, contando con características como búsqueda y agregación de add-ons sin la necesidad de abrir otra pantalla, controlando en todo momento las opciones de activación o desactivación.

Otra diferencia a considerar es la opción de personalizar de manera mucho más completa el navegador según nuestros gustos y necesidades que nos ofrece FireFox, como el soporte para temas y/o cambio de Skins de diferentes diseños. En cuanto a desarrollo, Internet Explorer ha introducido una nueva tecnología llamada ACTIVITIES, que nos entrega la opción de acceder de manera rápida y directa a buscadores de direcciones, servicios de mapas en línea, traductores, etc. Todo en una sola pantalla.

En seguridad tanto Internet Explorer como Firefox cuentan con interesantes herramientas, aunque los complementos de FireFox son muchos más útiles y eficaces en el manejo de archivos de dudosa procedencia o posiblemente infectados.

Internet Explorer no queriendo quedarse atrás construyo una aplicación Anti-phishing la cual entrega un completo informe sobre si el dominio y la dirección URL corresponden a la que aparentemente el sitio nos señala. Otra función muy atractiva y funcional es el denominado Cross Site Scrpting que bloquea de manera automática los ataques de diversos Scripts, a todo esto hay que agregar un excelente protector Anti-Malware y un

desarrollado sistema de controles Parentales para bridar una navegación mucho mas segura y protegida en todo ámbito de cosas.

Como dijimos anteriormente Mozilla FireFox cuenta con supremacía a la hora de descargar archivos, y gestionar marcadores, pudiendo organizar y detener cada uno de

los ficheros de manera más limpia. Sobre la gestión de marcadores no hay mucho que decir, ya que FireFox fue uno de los precursores en el manejo de esta útil aplicación que nos permite acceso a toda la información que deseemos en un solo lugar y de manera ordenada.

Un factor muy relevante y digno de un análisis profundo es el rendimiento y la velocidad (tanto de navegación como descarga) que nos proporcionan Internet Explorer y Mozilla FireFox.

Factor que lamentablemente para Microsoft es uno de los serios defectos e inconvenientes a la hora de navegar con Internet Explorer, retraso en la transferencia de datos, o lentitud exagerada (dependiendo del ancho de banda disponible) a la hora de descargar algún fichero o archivo de considerable peso. No así con FireFox, de hecho y justamente este factor ha sido la bandera de lucha para ganar adeptos que se aburren de las deficiencias de Internet Explorer, gozando con notable ventaja Mozilla FireFox es mucho más rápido (entre un 30 y 40%) que su competidor, otorgando una navegación mucho más rápida y eficiente, sobre todo a la hora de descargar información de la Web, dando prioridad y acelerando los procesos que lo requieran, convirtiéndose mucho más funcional y practico de lo que nosotros esperamos. En todo caso y para ser justos hay

muchas variables que influyen a la hora de una gestión mejorada, que son ajenas a los navegadores, como por ejemplo, las características del PC (Procesador, memoria RAM), y más importante aun la conexión a internet que tengamos (Banda Ancha, ADSL, Modem, etc.), estos en conjunto harán que el trabajar y utilizar la Web sea mucho más placentero y funcional.

Ahora si de similitudes y características semejantes hablamos, tendríamos que mencionar la evidente similitud en apariencia con que cuenta Internet Explorer en su última versión y Mozilla Firefox (también en su última versión). En cuanto a funciones y aplicaciones, cabe mencionar la navegación por pestañas, característica que los dos poseen, igual gestor de RSS, el automático bloqueo de ventanas emergentes (pop-ups), y algunas características de seguridad como la detección de información y archivos poco confiables.

Concluyendo tanto Firefox como Internet Explorer han evolucionado de manera considerable en cada una de sus nuevas versiones y en la medida de las capacidades de cada uno de sus desarrolladores, pese a esto, siempre habrá tanto opiniones a favor, como en contra respecto del funcionamiento de uno y otro. Por lo tanto para la realización de este prototipo de interfaz se han tenido en cuenta las características de ambos exploradores, adaptando nuestros recursos para que dicha interfaz se visualice correctamente en estas dos herramientas.

3.6. HTML

HTML ("Lenguaje para marcado de hipertexto"). Se trata de un lenguaje para estructurar documentos a partir de texto en World Wide Web. HTML consiste de varios componentes vitales, incluyendo *elementos* y sus *atributos*, *tipos de data*, y la *declaración de tipo de documento*.

Elementos

Los elementos son la estructura básica de HTML. Los elementos tienen dos propiedades básicas: atributos y contenido. Cada atributo y contenido tiene ciertas restricciones para que se considere válido al documento HTML. Un elemento generalmente tiene una etiqueta de inicio (p.ej. <nombre-de-elemento>) y una etiqueta de cierre (p.ej. </nombre-de-elemento>). Los atributos del elemento están contenidos en la etiqueta de inicio y el contenido está ubicado entre las dos etiquetas (p.ej. <nombre-de-elemento atributo="valor">Contenido</nombre-de-elemento>). Algunos elementos,

tales como
, no tienen contenido ni llevan una etiqueta de cierre. Debajo se listan varios tipos de elementos de marcado usados en HTML.

Estructura general de una línea de código en el lenguaje de etiquetas HTML

El marcado **estructural** describe el propósito del texto. El marcado estructural no define cómo se verá el elemento, pero la mayoría de los navegadores web han estandarizado el formato de los elementos. Un formato específico puede ser aplicado al texto por medio de hojas de estilo en cascada.

El marcado **presentacional** describe la apariencia del texto, sin importar su función. Por ejemplo, negrita indica que los navegadores web visuales deben mostrar el texto en **negrita**, pero no indica qué deben hacer los navegadores web que muestran el contenido de otra manera (por ejemplo, los que leen el texto en voz alta). En el caso de negrita e itálica, existen elementos que se ven de la misma manera pero tienen una naturaleza más semántica: enfásis fuerte y énfasis. Es fácil ver cómo un lector de pantalla debería interpretar estos dos elementos. Sin embargo, son equivalentes a sus correspondientes elementos presentacionales: un lector de pantalla no debería decir más fuerte el nombre de un libro, aunque éste esté en *itálicas* en una pantalla. La mayoría del marcado

presentacional ha sido desechada con HTML 4.0, en favor de Hojas de estilo en cascada.

El marcado **hipertextual** se utiliza para enlazar partes del documento con otros documentos o con otras partes del mismo documento. Para crear un enlace es necesario

utilizar la etiqueta de ancla <a> junto con el atributo href, que establecerá la dirección URL a la que apunta el enlace.

Atributos

La mayoría de los atributos de un elemento son pares nombre-valor, separados por un signo de igual "=" y escritos en la etiqueta de comienzo de un elemento, después del nombre de éste. El valor puede estar rodeado por comillas dobles o simples, aunque ciertos tipos de valores pueden estar sin comillas en HTML (pero no en XHTML). De todas maneras, dejar los valores sin comillas es considerado poco seguro. En contraste con los pares nombre-elemento, hay algunos atributos que afectan al elemento simplemente por su presencia (tal como el atributo ismap para el elemento img).

Códigos HTML básicos

- <html>: define el inicio del documento HTML, le indica al navegador que lo que viene a continuación debe ser interpretado como código HTML.
- <head>: define la cabecera del documento HTML, esta cabecera suele contener información sobre el documento que no se muestra directamente al usuario. Como por ejemplo el título de la ventana del navegador. Dentro de la cabecera <head> podemos encontrar:
    - <title>: define el título de la página. Por lo general, el título aparece en la barra de título encima de la ventana
    - <link>: para vincular el sitio a hojas de estilo o iconos.
    - <style>: para colocar el estilo interno de la página, ya sea usando CSS, JavaScript u otros lenguajes similares. No es necesario colocarlo si se va a vincular a un archivo externo usando la etiqueta <link>.
- <body>: define el contenido principal o cuerpo del documento. Esta es la parte del documento html que se muestra en el navegador; dentro de esta etiqueta pueden definirse propiedades comunes a toda la página, como color de fondo y márgenes. Dentro del cuerpo <body> podemos encontrar numerosas etiquetas. A continuación se indican algunas a modo de ejemplo:
    - <h1>, <h2>, <h3>, <h4>, <h5>, <h6>: encabezados o títulos del documento con diferente relevancia.
    - <table>: define una tabla
    - <tr>: fila de una tabla
    - <td>: celda de datos de una tabla
    - <a>: Hipervínculo o enlace, dentro o fuera del sitio web. Debe definirse el parámetro de pasada por medio del atributo *href*.
    - : área de la página

- <img>: imagen. Requiere del atributo *src*, que indica la ruta en la que se encuentra la imagen.
- : Etiquetas para listas.
- <i>: texto en cursiva
- <u>: texto subrayado

- La mayoría de etiquetas deben cerrarse como se abren, pero con una barra ("/") .

El lenguaje HTML puede ser creado y editado con cualquier editor de textos básico, como puede ser Gedit en Linux, el Bloc de Notas de Windows, o cualquier otro editor que admita texto sin formato como GNU Emacs, Microsoft Wordpad, TextPad, Vim, Notepad++, etc.

Existen además, otros programas para la realización de sitios Web o edición de código HTML, como por ejemplo Microsoft FrontPage, el cual tiene un formato básico parecido al resto de los programas de Office. También existe el famoso software de

Macromedia (llamado Dreamweaver, siendo uno de los más utilizados en el ámbito de diseño y programación Web. Estos programas se les conoce como editores WYSIWYG o What You See Is What You Get (en español: “lo que ves es lo que obtienes”). Esto significa que son editores en los cuales se ve el resultado de lo que se está editando en tiempo real a medida que se va desarrollando el documento. Ahora bien, esto no significa una manera distinta de realizar sitios web, sino que una forma un tanto más simple ya que estos programas, además de tener la opción de trabajar con la vista preliminar, tiene su propia sección HTML la cual va generando todo el código a medida que se va trabajando.

Combinar estos dos métodos resulta muy interesante, ya que de alguna manera se ayudan entre sí. Por ejemplo; si se edita todo en HTML y de pronto se olvida algún código o etiqueta, simplemente me dirijo al editor visual o WYSIWYG y se continúa ahí la edición, o viceversa, ya que hay casos en que sale más rápido y fácil escribir directamente el código de alguna característica que queramos adherirle al sitio, que buscar la opción en el programa mismo.

Existe otro tipo de editores HTML llamados WYSIWYM (Lo que ves es lo que quieres decir) que dan más importancia al contenido y al significado que a la apariencia visual. Entre los objetivos que tienen estos editores es la separación del contenido y la presentación, fundamental en el diseño Web.

HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma en la que debe aparecer en su navegador

el texto, así como también las imágenes y los demás elementos, en la pantalla del ordenador.

El diseño en HTML aparte de cumplir con las especificaciones propias del lenguaje debe respetar unos criterios de accesibilidad web, siguiendo unas pautas, o las normativas y leyes vigentes en los países donde se regule dicho concepto. Se encuentra disponible y desarrollado por el W3C a través de las Pautas de Accesibilidad al Contenido Web 1.0 WCAG.

3.7. CSS

Las **hojas de Estilo en cascada** (*Cascading Style Sheets*), CSS es un lenguaje artificial usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El WWWC (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

Asimismo, al proporcionar un nivel de abstracción, las CSS incorporadas en la cabecera de un documento mejoran el código del mismo, el tiempo de respuesta y la velocidad de

presentación de la página, siendo posible declarar una regla de estilo una sola vez y hacer que las condiciones de presentación se apliquen a todos los elementos de ese tipo.

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento. Esto es así porque fue concebido para otros usos, distinto a los actuales, mucho más amplios.

Para solucionar estos problemas los diseñadores han utilizado técnicas tales como la utilización de tablas imágenes transparentes para ajustarlas, utilización de etiquetas que no son estándares del HTML y otras. Estas "trampas" han causado a menudo problemas en las páginas a la hora de su visualización en distintas plataformas.

Además, los diseñadores se han visto frustrados por la dificultad con la que, aun utilizando estos trucos, se encontraban a la hora de maquetar las páginas, ya que muchos de ellos venían maquetando páginas sobre el papel, donde el control sobre la forma del documento es absoluto.

Finalmente, otro antecedente que ha hecho necesario el desarrollo de esta tecnología consiste en que las páginas web tienen mezclado en su código HTML el contenido del documento con las etiquetas necesarias para darle forma. Esto tiene sus inconvenientes ya que la lectura del código HTML se hace pesada y difícil a la hora de buscar errores o depurar las páginas. Aunque, desde el punto de vista de la riqueza de la información y la utilidad de las páginas a la hora de almacenar su contenido, es un gran problema que

estos textos estén mezclados con etiquetas incrustadas para dar forma a estos: se degrada su utilidad.

CSS proporciona tres caminos diferentes para aplicar las reglas de estilo a una página Web:

1. **Una hoja de estilo externa**, que es una hoja de estilo que está almacenada en un archivo diferente al archivo donde se almacena el código HTML de la página Web. Esta es la manera de programar más potente, porque separa completamente las reglas de formateo para la página HTML de la estructura básica de la página.
2. **Una hoja de estilo interna**, que es una hoja de estilo que está incrustada dentro de un documento HTML. (Va a la derecha dentro del elemento <head>). De esta manera se obtiene el beneficio de separar la información del estilo, del código HTML propiamente dicho. Se puede optar por copiar la hoja de estilo incrustada de una página a otra, (esta posibilidad es difícil de ejecutar si se desea para guardar las copias sincronizadas). En general, la única vez que se usa una hoja de estilo interna, es cuando se quiere proporcionar alguna característica a una página Web en un simple fichero, por ejemplo, si se está enviando algo a la página web.
3. **Un estilo en línea**, que es un método para insertar el lenguaje de estilo de página, directamente, dentro de una etiqueta HTML. Esta manera de proceder no es excesivamente adecuada. El incrustar la descripción del formateo dentro del documento de la página Web, a nivel de código se convierte en una tarea larga, tediosa y poco elegante de resolver el problema de la programación de la página. Este modo de trabajo se podría usar de manera ocasional si se pretende aplicar

   un formateo con prisa, al vuelo. No es todo lo claro, o estructurado, que debería ser, pero funciona.

Los beneficios de usar CSS son dobles. Por un lado, evitamos hacer a los archivos demasiado pesados (excluyendo el largo código requerido para las tablas anidadas y el añadido de características gráficas), y definimos el "estilo visual" de un sitio entero sin necesidad de hacerlo etiqueta por etiqueta, para cada una de las páginas. Por otro, trabajamos con estándares, y separamos hasta cierto punto la estructura de la presentación, logrando una manera más nítida de trabajar, y lo que es más: en un sencillo documento CSS. Vale decir, que cualquier cambio hecho a un estilo CSS, se reflejará en todos los elementos que sean referidos a éste, automáticamente, con sólo editar un sencillo documento CSS.

Las ventajas de utilizar CSS son:

1.- Control centralizado de la presentación de un sitio web completo con lo que se agiliza de forma considerable la actualización del mismo.

2.- Los Navegadores permiten a los usuarios especificar su propia hoja de estilo local que será aplicada a un sitio web, con lo que aumenta considerablemente

la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo para aumentar el tamaño del texto o remarcar más los enlaces.

3.- Una página puede disponer de diferentes hojas de estilo según el dispositivo que la muestre o incluso a elección del usuario. Por ejemplo, para ser impresa, mostrada en un dispositivo móvil, o ser "leída" por un sintetizador de voz.

4.- El documento HTML en sí mismo es más claro de entender y se consigue reducir considerablemente su tamaño (siempre y cuando no se utilice estilo en línea).

3.8. JSP

La tecnología JavaServer Pages permite a los diseñadores y desarrolladores Web desarrollar rápidamente y mantener fácilmente páginas web dinámicas y ricas en información que aprovechen los sistemas empresariales existentes. La tecnología JSP, que forma parte de la familia JavaTM, permite un rápido desarrollo de aplicaciones web independientes de cualquier plataforma. La tecnología JavaServer Pages separa el interface de usuario de la generación de contenidos, permitiendo a los diseñadores cambiar el diseño de las páginas sin alterar el contenido dinámico.

Cuando la interacción con el usuario se complica, surge la necesidad de utilizar los componentes Java específicamente diseñados para resolver la lógica del negocio al mismo tiempo que es configurada fácilmente a través de páginas externas HTML.

Este tipo de componentes son los llamados Enterprise JavaBeans y las páginas HMTL desde las que dichos objetos son ejecutados se llaman páginas JSP. Las páginas JSP son páginas web que permiten integrar objetos Java (JavaBeans). Cuando un cliente pide este tipo de página, el servidor ejecuta los comandos Java y JavaBeans y distribuye la página resultante al cliente.

Este mecanismo hace del software de DEISTER SOFTWARE un sistema altamente configurable, permitiendo integrar JavaBeans en la web que cumplan los procesos específicos de negocio de cada compañía. JavaServer Pages es la plataforma Java para la construcción de aplicaciones con contenido dinámico web tal como HTML, DHTML, XHTML y páginas XML. El uso de la tecnología JSP ofrece las siguientes ventajas:

- **Separación entre contenidos dinámico y estático:** El modelo JavaServer Pages le permite capturar la lógica de la aplicación en componentes JavaBeans stándar

y reutilizables en los que puede definir la presentación utilizando etiquetas especiales JSP y pequeñas secciones de código Java conocido como scriptlets.

- **Soporte para programación dinámica (scripting):** Las Java Server Pages permiten incluir líneas de programación dinámica ejecutadas en el servidor cuando la página es pedida.
- **Una vez escritas, funcionan en cualquier lugar:** Las Javaserver Pages son una extensión de los Java servlets, un estándar 100% Java de JavaSoft. Las Java Server Pages heredan todas las ventajas de la plataforma Java incluyendo ?Escríbalo una vez y hágalo funcionar en cualquier lugar? o funcionalidad multi-plataforma. Debido a que JSP es un estándar abierto, puede implementar aplicaciones JSP en cualquier plataforma que se adhiera a su especificación.
- **Rendimiento avanzado y escalable:** Las aplicaciones JSP disfrutan de la misma escalabilidad y rendimiento que los Java servlets ya que se trata de una extensión de la arquitectura Java servlet.
- **Gran calidad de soporte y documentación:** La tecnología JSP es un estándar Java que permite a los usuarios y desarrolladores ordenar toda la documentación en este estándar.

TAG LIBRARIES

Las librerías de etiquetas JSP definen una funcionalidad modular que puede ser reutilizada por cualquier página JSP. Reducen la necesidad de integrar grandes cantidades de código Java en las páginas JSP trasladando las funciones de las etiquetas a clases de implementación de etiquetas. Con ello facilitan la creación de páginas JSP tanto para el editor de páginas web como para las herramientas que muestran la función encapsulada por la librería del autor.

Las librerías de etiquetas incrementan la productividad estimulando una división del trabajo entre los desarrolladores y los usuarios. Son creadas por un desarrollador experto en el acceso a datos y otros servicios. Son utilizadas por el autor de páginas web para poder centrarse en el diseño del interface de usuario.

La tecnología **WebStudio** de DEISTER SOFTWARE utiliza ampliamente estas librerías, predefiniendo todos los componentes GUI necesarios para crear formularios complejos en la web. Esto permite a los desarrolladores construir fácilmente formularios, deshaciéndose del problema de la complejidad HMTL/Java.

Arquitectura

JSP puede considerarse como una manera alternativa, y simplificada, de construir servlets. Es por ello que una página JSP puede hacer todo lo que un servlet puede hacer, y viceversa. Cada versión de la especificación de JSP está fuertemente vinculada a una versión en particular de la especificación de servlets.

El funcionamiento general de la tecnología JSP es que el Servidor de Aplicaciones interpreta el código contenido en la página JSP para construir el código Java del servlet

a generar. Este servlet será el que genere el documento (típicamente HTML) que se presentará en la pantalla del Navegador del usuario.

JSP -> Servidor Aplicaciones (Servlets) -> Cliente (Navegador)

Es posible enriquecer el lenguaje de etiquetas utilizado por JSP. Para ello debemos extender la capa de alto nivel JSP mediante la implementación de Librerías de Etiquetas (Tags Libraries). Un ejemplo de estas librerías son las propocionadas por Sun bajo la denominación de JSTL o las distribuidas por Apache junto con el Framework de Struts.

TagLibs -> JSP -> Servidor Aplicaciones (Servlets) ->Cliente

El rendimiento de una página JSP es el mismo que tendría el servidor equivalente, ya que el código es compilado como cualquier otra clase Java. A su vez, la máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

La principal ventaja de **JSP** frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que la aplicación termine siendo ejecutada en otra.

Los servlets y Java Server Pages (JSPs) son dos métodos de creación de páginas web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, ASP o los CGIs, programas que generan páginas web en el servidor. Sin embargo, se diferencian de ellos en otras cosas.

Para empezar, los JSPs y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada servlet (o JSP, a partir de ahora lo usaremos de forma indistinta) se ejecuta en su propia hebra, es decir, en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo (cargar programa + intérprete). Su persistencia le permite también hacer una serie de cosas de forma más eficiente: conexión a bases de datos y manejo de sesiones, por ejemplo.

Los JSPs son en realidad servlets: un JSP se compila a un programa en Java la primera vez que se invoca, y del programa en Java se crea una clase que se empieza a ejecutar en el servidor como un servlet. La principal diferencia entre los servlets y los JSPs es el enfoque de la programación: un JSP es una página Web con etiquetas especiales y código Java incrustado, mientras que un servlet es un programa Java puro que recibe peticiones y genera a partir de ellas una página web.

3.9. JDBC

Java Database Connectivity, más conocida por sus siglas **JDBC**, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos a las que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Establecer la conexión

Una vez registrado el controlador con el DriverManager, se debe especificar la fuente de datos a la que se desea acceder. En JDBC, una fuente de datos se especifica por medio de un URL con el prefijo de protocolo **jdbc:** , la sintaxis y la estructura del protocolo es la siguiente:

jdbc:{subprotocolo}:{subnombre}

El {subprotocolo} expresa el tipo de controlador, normalmente es el nombre del sistema de base de datos, como db2, oracle o mysql.

El contenido y la sintaxis de {subnombre} dependen del {subprotocolo}, pero en general indican el nombre y la ubicación de la fuente de datos.

El formato general para conectarse a MySQL es:

jdbc:mysql://[servidor][:puerto]/[base_de_datos][?param1=valor1][param2=valor2]...

Una vez que se ha determinado el URL, se puede establecer una conexión a una base de datos.

El objeto **Connection** es el principal objeto utilizado para proporcionar un vínculo entre las bases de datos y una aplicación en Java. Connection proporciona métodos para manejar el procesamiento de transacciones, para crear objetos y ejecutar instrucciones SQL, y para crear objetos para la ejecución de procedimientos almacenados.

Se puede emplear tanto el objeto Driver como el objeto DriverManager para crear un objeto Connection. Se utiliza el método **connect()** para el objeto Driver, y el método **getConnection()** para el objeto DriverManager.

El objeto Connection proporciona una conexión estática a la base de datos. Esto significa que hasta que se llame en forma explícita a su método **close()** para cerrar la conexión o se destruya el objeto Connection, la conexión a la base de datos permanecerá activa.

La manera más usual de establecer una conexión a una base de datos es invocando el método getConnection() de la clase DriverManager. A menudo, las bases de datos están protegidas con nombres de usuario (login) y contraseñas (password) para restringir el acceso a las mismas. El método getConnection() permite que el nombre de usuario y la contraseña se pasen también como parámetros.

Connection conn = DriverManager.getConnection(url,login,password);
Creación de sentencias

El objeto Connection permite establecer una conexión a una base de datos. Para ejecutar instrucciones SQL y procesar los resultados de las mismas, se debe hacer uso de un objeto **Statement**.

Los objetos Statement envían comandos SQL a la base de datos, y pueden ser de cualquiera de los tipos siguientes:

- Un comando de definición de datos como CREATE TABLE o CREATE INDEX.

- Un comando de manipulación de datos como INSERT, DELETE o UPDATE.
- Un sentencia SELECT para consulta de datos.

Un comando de manipulación de datos devuelve un contador con el número de filas (registros) afectados, o modificados, mientras una instrucción SELECT devuelve un

conjunto de registros denominado conjunto de resultados (*result set*). La interfaz Statement no tiene un constructor , sin embargo, podemos obtener un objeto Statement al invocar el método **createStatement()** de un objeto Connection.

```
conn = DriverManager.getConnection(url,login,pasword);
Statement stmt = conn.createStatement();
```

Una vez creado el objeto Statement, se puede emplear para enviar consultas a la base de datos usando los métodos execute(), executeUpdate() o executeQuery(). La elección del método depende del tipo de consulta que se va a enviar al servidor de bases de datos:

Método	Descripción
execute( )	Se usa principalmente cuando una sentencia SQL regresa varios conjuntos de resultados. Esto ocurre principalmente cuando se está haciendo uso de procedimientos almacenados.
executeUpdate( )	Este método se utiliza con instrucciones SQL de manipulación de datos tales como INSERT, DELETE o UPDATE.
executeQuery( )	Se usa en las instrucciones del tipo SELECT.

Es recomendable que se cierren los objetos Connection y Statement que se hayan creado cuando ya no se necesiten. Lo que sucede es que cuando en una aplicación en Java se están usando recursos externos, como es el caso del acceso a bases de datos con el API JDBC, el recolector de basura de Java (*garbage collector*) no tiene manera de conocer cuál es el estado de esos recursos, y por lo tanto, no es capaz de liberarlos en el caso de que ya no sean útiles. Lo que ocurre en estos casos es que se pueden quedar almacenados en memoria grandes cantidades de recursos relacionados con la aplicación de bases de datos que se está ejecutando. Es por esto que se recomienda que se cierren de manera explícita los objetos Connection y Statement.

De manera similar a Connection, la interfaz Statement tiene un método **close()** que permite cerrar de manera explícita un objeto Statement. Al cerrar un objeto Statement se liberan los recursos que están en uso tanto en la aplicación Java como en el servidor de bases de datos.

```
Statement stmt = conn.createStatement();
....
stmt.close();
```

Ejecución de consultas

Cuando se ejecutan sentencias SELECT usando el método executeQuery(), se obtiene como respuesta un conjunto de resultados, que en Java es representado por un objeto **ResultSet**.

```
Statement stmt = conn.createStatement();
ResultSet res  = stmt.executeQuery("SELECT * FROM ejemplo");
```

La información del conjunto de resultados se puede obtener usando el método **next()** y los diversos métodos **getXXX()** del objeto ResultSet. El método next() permite moverse fila por fila a través del ResultSet, mientras que los diversos métodos getXXX() permiten accesar a los datos de una fila en particular.

Los métodos getXXX() toman como argumento el índice o nombre de una columna, y regresan un valor con el tipo de datos especificado en el método. Así por ejemplo, getString() regresará una cadena, getBoolean() regresará un booleano y getInt() regresará un entero. Cabe mencionar que estos métodos deben tener una correspondencia con los tipos de datos que se tienen en el ResultSet, y que son a las vez los tipos de datos provenientes de la consulta SELECT en la base de datos, sin embargo, si únicamente se desean mostrar los datos se puede usar getString() sin importar el tipo de dato de la columna.

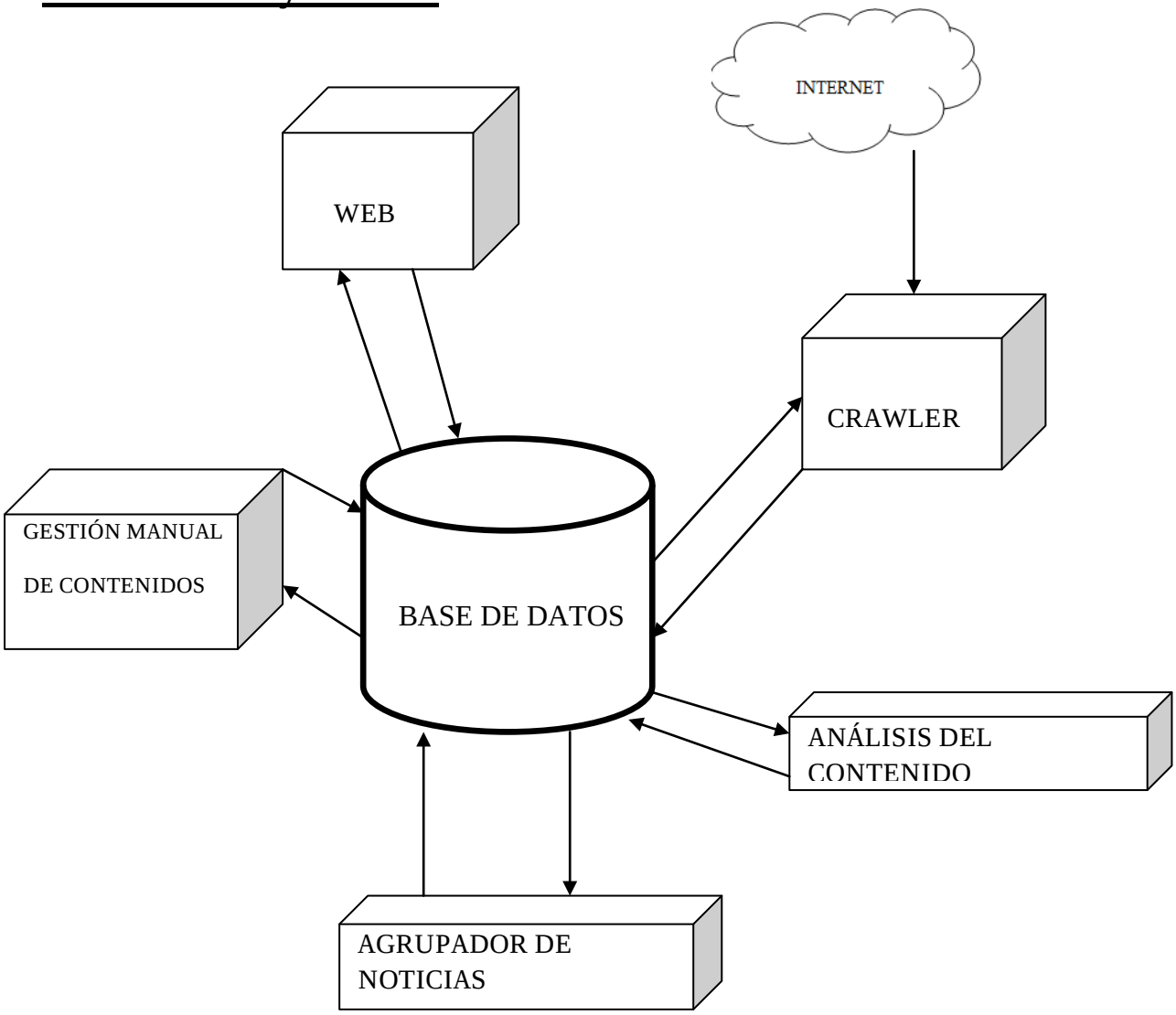
Por otra parte, si en estos métodos se utiliza la versión que toma el índice de la columna, se debe considerar que los índices empiezan a partir de 1, y no en 0 (cero) como en los arreglos, los vectores, y algunas otras estructuras de datos de Java.

Existe un objeto ResultSetMetaData que proporciona varios métodos para obtener información sobre los datos que están dentro de un objeto ResultSet. Estos métodos permiten entre otras cosas obtener de manera dinámica el número de columnas en el conjunto de resultados, así como el nombre y el tipo de cada columna.

```
ResultSet res  = stmt.executeQuery("SELECT * FROM ejemplo");
ResultSetMetaData metadata = res.getMetaData();
```

4. METODOLOGÍA DEL DESARROLLO

4.1. Análisis y Diseño



Vamos a explicar brevemente el funcionamiento y objetivos globales del proyecto conjunto.

En primer lugar es necesario recaudar toda la información sobre todos los medios posibles. Consiste en una base de datos muy voluminosa generada como resultado de la indexación de partes significativas de los documentos que han sido analizados previamente en Internet. Los motores de búsqueda suelen recoger documentos en formato *HTML* y otros tipos de recursos, como noticias. La tarea es realizada por un programa denominado **CRAWLER** (*robot o spider*) que recorre la red de forma automática explorando los servidores a nivel mundial, o en el ámbito de especialización del buscador (geográfico, idiomático o temático). La recuperación se realiza gracias a un sistema de gestión de base de datos que permite distintos tipos de consulta y a la ordenación de los resultados por relevancia, en función a la estrategia de consulta. Los

motores de búsqueda son más exhaustivos que los índices en cuanto al volumen de páginas referenciadas, pero son mucho menos precisos que los índices, al no ser su contenido objeto de indexación humana.

Un crawler recupera un documento y recursivamente todos los documentos con los que mantiene vínculos, indexa la información de acuerdo a un criterio predefinido. Los criterios son: el título del documento, los meta datos, el número de veces que se repite una palabra en un documento, algoritmos para valorar la relevancia del documento, etc. y el peso de cada criterio varía de acuerdo al motor de búsqueda. La información se almacena en una base de datos, la cual puede ser consultada por los usuarios de Internet para recuperar la información deseada. Para mantener actualizada la base de datos, los *crawlers* vuelven a visitar los sitios para verificar que las páginas registradas se mantengan activas, de no ser así (cuando se mueven a otro sitio o desaparecen) las eliminan de la base de datos.

Toda esa información es almacenada bajo los esquemas que hemos presentado en nuestra base de datos. El crawler se encargará de ir actualizando esta base de datos cada cierto tiempo, según se haya configurado.

A continuación será necesario analizar toda la información que ha sido almacenada, y tratarla de forma coherente y correcta para facilitar su posterior visualización.

El siguiente paso sería agrupar las noticias de tal forma que se agrupen aquellas que están relacionadas entre sí o traten de un mismo tema. Este proceso se denomina **reconocimiento de nombres de entidades**, *Named entity recognition* (NER), es una subtarea de la recuperación de información que busca localizar y clasificar elementos atómicos en texto sobre categorías predefinidas como nombres de personas, organizaciones, localizaciones, expresiones de horas, cantidades, valores monetarios, porcentajes, en nuestro caso buscará por los nombres de las entidades que hemos diseñado.

Existe otro proceso que se debe gestionar de forma manual, será asignado a un equipo de personas aún por determinar que se encarguen de introducir, tratar, o actualizar la base de datos con todo aquella información que no se haya podido conseguir a través de los medios automáticos de los que disponemos.

El último paso es al que se dedica este proyecto, la interfaz web, que mostrará de toda aquella información que hemos estado tratando, aquellos aspectos que nos interesan que el usuario manipule.

4.1.1.- Entorno de trabajo

Se ha utilizado un sistemas operativos, Windows Vista, aunque se podría haber utilizado cualquier sistema operativo para el desarrollo de los servicios, puesto que todas las tecnologías utilizadas son multiplataforma y todo lo implementado es código abierto desarrollado en Java.

Para el servidor *web* se ha utilizado Apache Tomcat 6.0, que también nos ofrece la posibilidad de poder desplegar Servicios Web. Utiliza la máquina virtual de Java.

En las bases de datos se ha utilizado un servidor MySQL para los datos relacionales y accede a ellos mediante Xquery.

Como plataforma de desarrollo para la implementación se ha elegido Java en su versión JDK 6 de la plataforma Java 2 *Standar Edition*, de Sun Microsystems, que también incluye una máquina virtual Java JRE. Como entorno de programación se ha utilizado Eclipse, versión 3.2.

Para el desarrollo de la memoria se han utilizado conjuntamente Open Office de Sun Microsystems y Office 2007 de Microsoft.

4.1.2.- Tecnologías y herramientas utilizadas

JDK 6 SE: Versión de desarrollo de Java, que además proporciona el entorno de ejecución. Utilizado para desarrollar el software de la implementación, los Servicios Web, y ejecutar las herramientas necesarias.

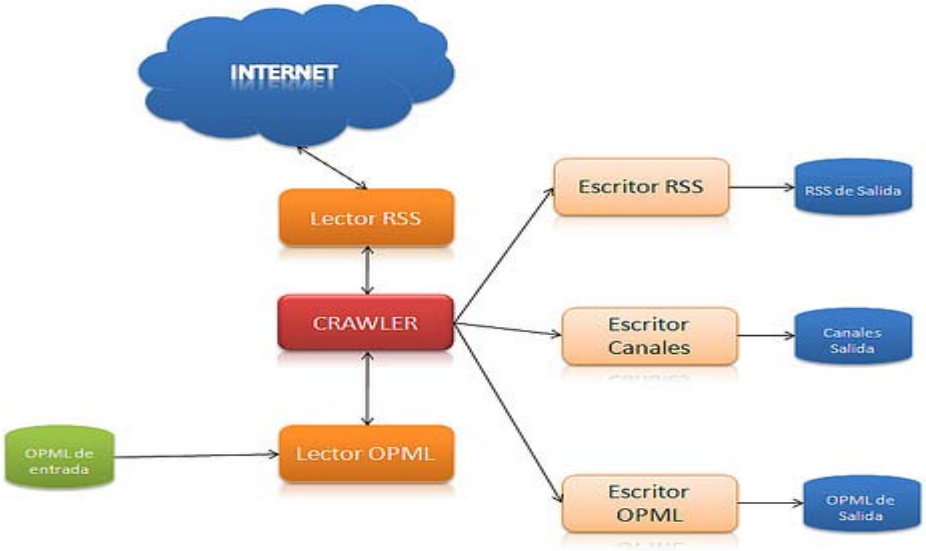
Apache Tomcat 6: Servidor de aplicaciones que permitirá ejecutar los Servicios Web necesarios y la aplicación *web*.
MySQL Server 6: Base de datos relacional utilizada para almacenar losdatos personales de usuario.

En un futuro será necesario:

J2EE: Java 2 *Enterprise Edition*, plataforma de desarrollo sobre la que trabaja el *framework* Struts.

Struts 1.3: Herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC [9]. Utilizada para crear la aplicación *web* con la que se conectarán los usuario.

4.1.3.- CRAWLER



Esta parte ha sido diseñada por Guillermo García Cubero, pero es necesario hacer una pequeña mención de ella para poder comprender mejor el funcionamiento de la recogida de la información que se presentará en nuestro prototipo de interfaz.

Sistema capaz de obtener explorar la red en busca de una información determinada para su descarga y procesamiento posterior.

Se basa en la utilización de formatos estándares para la entrada (OPML, lista de enlaces) y para la recuperación de información (RSS, ATOM, en todas susvariantes) dando como salida un fichero RSS 2.0 con la información obtenida del proceso.

Además, de forma adicional, el sistema da como salida la lista de enlaces original, en formato OPML, con la fecha de ultima pasada del Crawler actualizada, aumentando la eficiencia del proceso en pasadas posteriores (evitamos la redundancia, si una entrada la bajó en un proceso de "Rastreo" anterior no la volverá a bajar). También nos devuelve información sobre los Canales que visitamos, en este caso en un formato basado en XML propio.

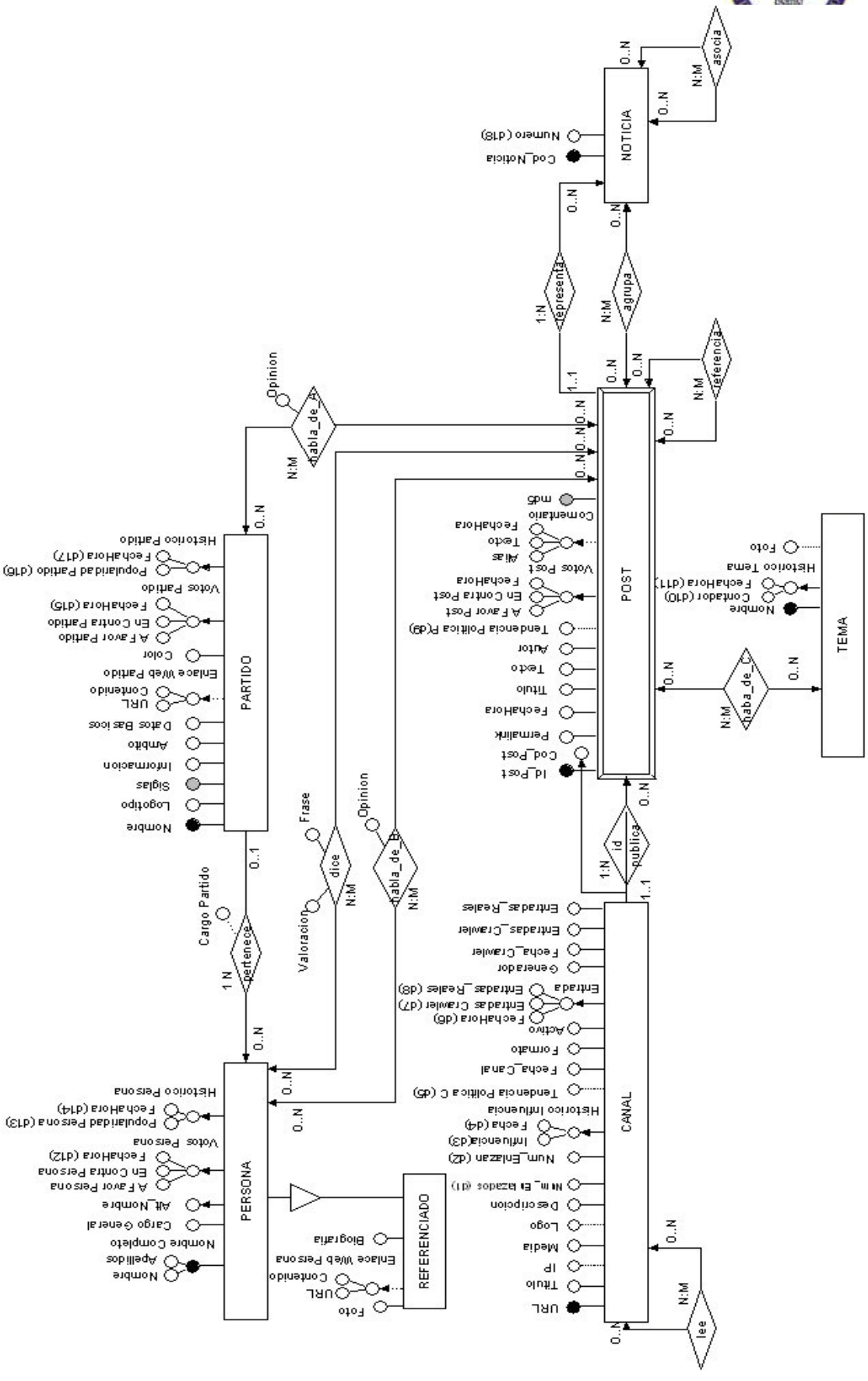
4.1.4.- BASE DE DATOS

El modelo relacional de base de datos es el más utilizado actualmente ya que se ajusta a problemas reales y administra datos dinámicamente. La idea principal de este modelo es el uso de relaciones entre tablas. Se pueden considerar de forma lógica como conjuntos de datos llamados tuplas, aunque a nivel físico las tablas pueden tener una estructura completamente distinta. Un punto fuerte del modelo relacional es la sencillez de su estructura lógica. Pero detrás de esa simple estructura hay un fundamento teórico importante del que carecen los BBDD de la primera generación, lo que constituye otro

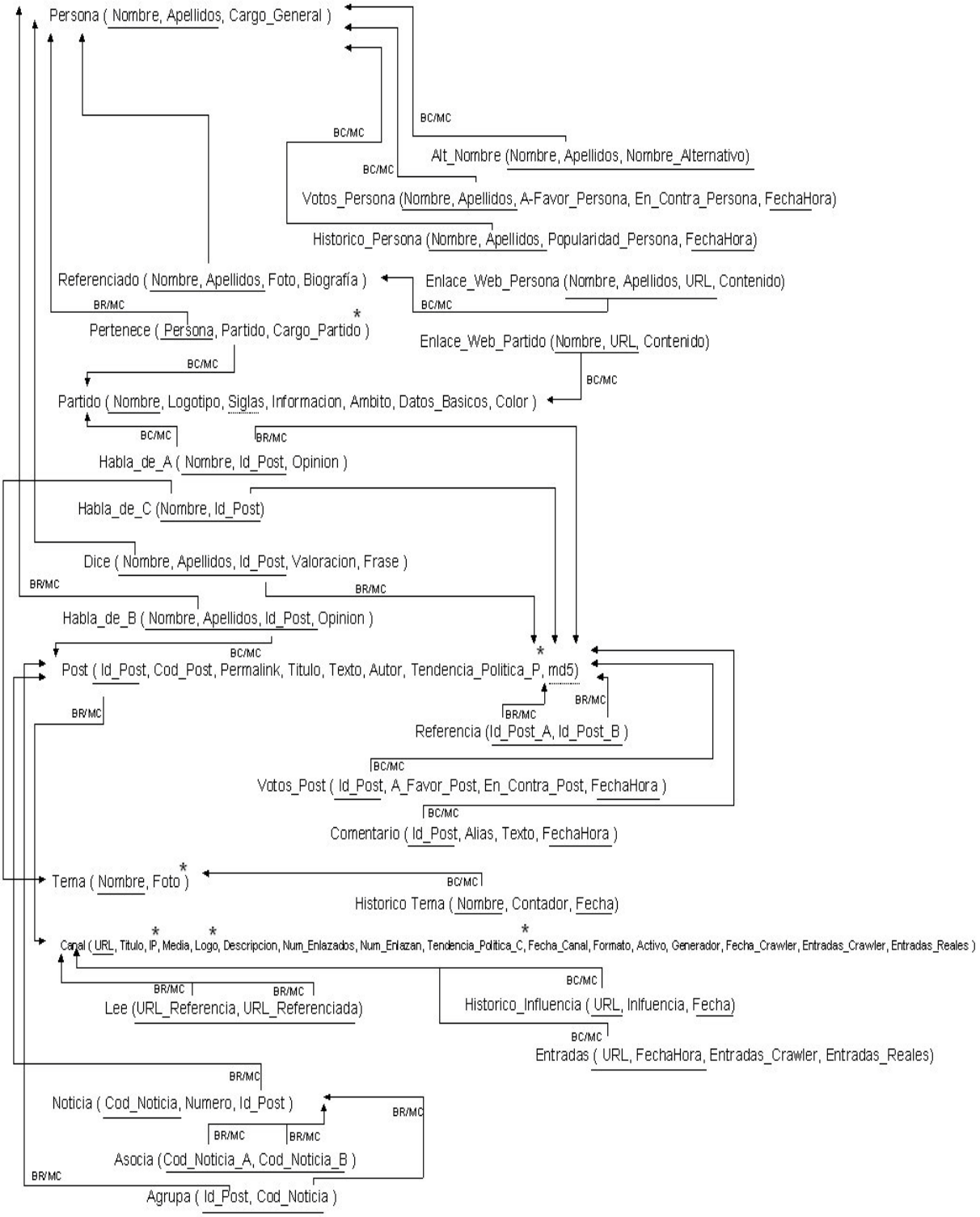
punto a su favor. Para poder acceder a las BBDD y hacer consultas se utilizan dos herramientas matemáticas: el álgebra relacional y cálculo relacional. Ambos utilizan operadores lógicos para la manipulación de los datos. El lenguaje *SQL, lenguaje declarativo,* implementado en los principales sistemas de gestión de las bases de datos, es el encargado de hacer la manipulación de los datos más transparente de cara al usuario.

A continuación en la siguiente página pasamos a detallar el modelo entidad relación correspondiente con nuestra base de datos y a partir del cuál se ha desarrollado todo el entramado de consultas para poder mostrar toda la información relevante en nuestro prototipo de interfaz web.

4.1.4.1.- Modelo de Entidad Relación de la BD


Modelo E.R PolitikTracker v.1.8
PERSONA
Nombre
Apellidos
Nombre Completo
Cargo General
Alt_Nombre
A Favor Persona
En Contra Persona
FechaHora (d12)
Votos Persona
Popularidad Persona (d13)
FechaHora (d14)
Historico Persona
REFERENCIADO
Foto
URL
Contenido
Enlace Web Persona
Biografia
PARTIDO
Nombre
Logotipo
Siglas
Informacion
Ambito
Datos Basicos
URL
Contenido
Enlace Web Partido
Color
A Favor Partido
En Contra Partido
FechaHora (d15)
Votos Partido
Popularidad Partido (d16)
FechaHora (d17)
Historico Partido
pertenece
Cargo Partido
1:N
0..1
0..N
dice
Frase
Valoracion
N:M
habla_de_A
Opinion
N:M
habla_de_B
Opinion
N:M
POST
Id_Post
Cod_Post
Permalink
FechaHora
Titulo
Texto
Autor
Tendencia Politica P (d9)
A Favor Post
En Contra Post
FechaHora
Votos Post
Alias
Texto
FechaHora
Comentario
md5
publica
1:N
1..1
0..N
CANAL
URL
Titulo
IP
Media
Logo
Descripcion
Num_Enlazados (d1)
Num_Enlazan (d2)
Influencia(d3)
Fecha (d4)
Historico Influencia
Tendencia Politica C (d5)
Fecha_Canal
Formato
Activo
FechaHora (d6)
Entradas Crawler (d7)
Entrada
Entradas_Reales (d8)
Generador
Fecha_Crawler
Entradas_Crawler
Entradas_Reales
lee
N:M
0..N
habla_de_C
N:M
TEMA
Nombre
Contador (d10)
FechaHora (d11)
Historico Tema
Foto
referencia
N:M
agrupa
N:M
representa
1:N
1..1
NOTICIA
Cod_Noticia
Numero (d18)
asocia
N:M

4.1.4.2.- Grafo relacional



4.1.4.3.- DICCIONARIO DE TÉRMINOS

ENTIDADES

CANAL

URL: Dirección URL del canal.
Titulo: Nombre del titulo del canal.
IP: Dirección IP del canal.
Media: Si es “true” indica que es un periódico, “false” indica que se trata de un blog.
Logo: Logotipo que identifica el canal.
Descripción: Breve descripción del canal semejante a una biografía en persona.
Num_Enlazado (**d1**): Guarda el número de enlaces a otros canales.
Num_Enlazan (**d2**): Contiene el número de canales que enlazan con este canal.
Histórico Influencia: Atributo compuesto que indica la influencia de la fuente en una fecha concreta.
Influencia (**d3**): Atributo derivado. Se calcula con Nom_Enlazado y Nom_Enlazan
Fecha (**d4**): Atributo derivado que se calcula a partir de la fecha del sistema.
Tendencia Política C (**d5**): Atributo derivado que indica la afinidad del canal con una determinada fuerza política. Se calculará según la tendencia de los post que genere.
Fecha_Canal: Fecha y Hora de la primera entrada que aporto ese canal.
Formato: Formato de datos utilizado por el canal.
Activo: Indica si permitimos que dicho canal aporte nuevos post a la B.D.
Entradas: Atributo compuesto que indica la última vez agregamos entradas de ese canal.
FechaHora (**d6**): Atributo derivado que indica la fecha de la actualización.
Entradas_Crawler(**d7**): Atributo derivado que guarda las entradas del crawler.
Entradas_Reales(**d8**): Atributo derivado que guarda las entradas reales.
Generador: Almacena información sobre el programa que generó el fichero RSS.
Fecha_Crawler: Fecha y Hora de la ultima introducción de datos en ese canal.
Entradas_Crawler: Número de entradas bajadas en la ultima pasada del crawler.
Entradas_Reales: Número final de entradas introducidas en la base de datos para la ultima pasada del crawler.

POST

Id_Post: Identificador numérico que lista las filas.
Cod_Post: URL del canal al que pertenece.
Permalink: Enlace permanente al post.
FechaHora: Fecha y Hora en la que fue publicado el post.
Titulo: Titulo del post.
Texto: Texto que resume la noticia.
Autor: Autor de dicho post.
Tendencia Política P (**d9**): Atributo derivado que define la posición política que defiende dicho post. Este atributo se extraerá automáticamente del texto.
Votos Post: Atributo compuesto que refleja un histórico de los votos que los usuarios del sistema han dado a un post en una fecha concreta.
A Favor Post: Indica el número de votos positivos para el post.

En Contra Post: Indica el número de votos negativos para el post.
FechaHora: Guarda la fecha y la hora.
Comentarios: Atributo compuesto que guarda los comentarios de los usuarios del sistema respecto a un post en una fecha concreta.
Alias: Nombre del usuario que hace el comentario
Texto: Refleja el comentario vertido por el usuario.
FechaHora: Indica la fecha y la hora en la que se hizo el comentario.
md5: Resultado de calcular el algoritmo con los valores “Titulo” y “Permalink”. Este valor será único para cada fila.

TEMA

Nombre: Nombre identificador de dicho tema.
Histórico Tema: Atributo compuesto en el que guardaremos una relación del número de post que contiene cada tema en una fecha (pensado para reflejar la nube de tags)
Contador (**d10**): Atributo derivado que contiene el número de post de un determinado tema. Se calcula al sumar todos los post que hablan de un tema concreto.
Fecha (**d11**): Atributo derivado que guarda la fecha. Se calcula a partir de la fecha del sistema.
Foto: Atributo opcional que guarda una foto representativa del tema tratado.

PERSONA

Nombre Completo: Atributo compuesto que define el nombre completo de la persona.
Nombre: Nombre relativo a la persona
Apellidos: Apellidos relativos a la persona.
Cargo General: Cargo que ocupa dicha persona dentro de la esfera social.
Alt_Nombre: Nombres alternativos con los que se conoce a la persona en cuestión.
Votos Persona: Atributo compuesto que refleja un histórico de los votos que los usuarios del sistema han dado a una persona en una fecha concreta.
A Favor Persona: Indica el número de votos positivos para la persona.
En Contra Persona: Indica el número de votos negativos para la persona.
FechaHora (**d12**): Atributo derivado que guarda la fecha y la hora.
Histórico Persona: Este atributo compuesto guarda una relación de la popularidad en una determinada fecha. Se compone de dos campos (fecha y Popularidad)
Popularidad Persona (**d13**): Atributo derivado que indica la popularidad la persona. Esta tendencia puede ser al alza, a la baja o mantenerse igual .Se calcula sabiendo el número de post que hablan de dicha persona en un periodo de tiempo.
FechaHora (**d14**): Atributo derivado que guarda la fecha. Se calcula a partir de la fecha del sistema.

REFERENCIADO

Foto: Imagen de la persona referenciada.
Enlace Web: Atributo compuesto que aporta direcciones Web de la persona indicada.
URL: Indica la dirección URL del recurso

Contenido: Especifica que encontraremos en la dirección URL.

Biografía: Breve texto que comenta a grandes rasgos la vida de dicha persona.

PARTIDO

Nombre: Nombre del partido político.
Logotipo: Imagen con el logotipo identificador del partido político.
Siglas: Siglas que identifican al partido político.
Información: Breve descripción del partido semejante a una biografía en persona.
Ámbito: Ámbito político que define mejor la posición del partido político.
Datos Básicos: Otros datos de relevancia sobre el partido (numero de socios, sede…)
Enlace Web: Enlace a la pagina oficial del partido político.
Color: Color con el que identificaremos al partido dentro del sistema.
Votos Partido: Atributo compuesto que refleja un histórico de los votos que los usuarios del sistema han dado a un partido en una fecha concreta.
A Favor Partido: Indica el número de votos positivos para la persona.
En Contra Partido: Indica el número de votos negativos para el partido.
FechaHora (**d15**): Atributo derivado que guarda la fecha del sistema.
Histórico Partido: Este atributo compuesto guarda una relación de la popularidad en una determinada fecha. Se compone de dos campos (fecha y Popularidad)
Popularidad Partido (**d16**): Atributo derivado que indica la popularidad del partido. Esta tendencia puede ser al alza, a la baja o mantenerse igual .Se calcula sabiendo el número de post que hablan de dicho partido en un periodo de tiempo.
FechaHora (**d17**): Atributo derivado que guarda la fecha del sistema.

NOTICIA

Cod_Noticia: Atributo con el que identificamos todas las noticias del sistema.
Número (**d18**): Atributo derivado que refleja el número de post que forman parte de la noticia. Se calcula sumando todos los post de la noticia.

RELACIONES ENTRE ENTIDADES Y ATRIBUTOS QUE CONTIENEN

PERSONA – PARTIDO (pertenece)

Esta relación establece que personas están relacionadas con que partidos.
Una persona puede pertenecer o no a un partido (0..1)
Un partido esta compuesto desde por ninguna a muchas personas (0..N)
Cargo Partido: Cargo que ocupa dicha persona en el partido político al que pertenece.

PERSONA – POST (habla de B)

Esta relación agrupa la información que en los medios de comunicación tradicionales se reposta sobre la persona.
Un post puede hablar desde de ninguna persona hasta de muchas personas. (0..N)

De una persona se puede hablar desde en ningun post hasta en muchos posts (0..N)
Opinión (**d23**): Refleja la opinión expresada en un post (positiva, neutral o negativa) en el que se hable de una persona. Será un atributo derivado que se calcula automáticamente al analizar el texto.

PERSONA – POST (dice)

Esta relación organiza la información en forma de posts disponibles de la persona con carácter más subjetivo.
Un post puede decir cosas de desde ninguna persona hasta de muchas. (0..N)
De una persona se pueden decir cosas desde en ningun post hasta en muchos. (0..N)
Valoración (**d24**): Atributo derivado que indica la valoración del texto. Este atributo se calcula de forma automática al analizar el texto.
Frase (**d25**): Texto que recoge una frase citada en un post sobre una persona. Este será un atributo derivado, ya que se elegirá automáticamente sacando la frase más significativa del post.

PARTIDO – POST (habla de A)

Indica el hecho de que en un post se hable de un partido político.
Un post puede habar desde ninguno a muchos partidos políticos. (0..N)
De un partido político pueden hablar desde ninguno a muchos post. (0..N)
Opinión (**d26**): Refleja la opinión expresada en un post (positiva o negativa) en el que se hable de un partido. Será un atributo derivado que se calcula automáticamente al analizar el texto.

TEMA – POST (habla de C)

Existe una relación entre el post y el tema del que habla.
Un post puede tratar desde ningún tema* a muchos temas. (1..N)
Un tema puede ser hablado desde en ningún post hasta en muchos post
* El caso de que un post no hable de ningun tema se debe a que puede que ese tema no este incluido en la base de datos.

CANAL – CANAL (lee)

Esta relación hace referencia al hecho de que un blog tiene una lista de blogs que consulta (blogrolls).
Un blog puede consultar desde ningún blog a muchos blogs (0..N)
Un blog puede ser consultado por ninguno o por muchos blogs (0..N)

CANAL – POST (publica)

Representa la acción de que un canal (blog o periódico digital) haga público un post. Este último será la entidad débil y canal será la entidad fuerte, ya que existe una dependencia en identificación.
Un canal puede publicar desde ninguno a muchos post (0..N)
Un post es publicado por un único canal. (1..1)

POST – POST (referencia)

Esta relación indica que un post puede tomar como referencia otros post, es decir, que aporta como datos un enlace a otro post anterior.
Un post puede referenciar desde ninguno a muchos post (0..N)
Un post puede estar referenciado desde ninguno a muchos post (0..N)

POST – NOTICIA (agrupa)

En muchos casos la información que aporten los post será redundante. Con la entidad noticia lo que intentamos es agrupar todos los post similares, con los que después crearemos una lista resumida con los post mas significativos de cada noticia.
Una noticia agrupa desde ninguno a muchos post (0..N)
Un post esta agrupado solamente en una noticia (1..1)

POST – NOTICIA (representa)

Como las noticias agrupan muchos post, queremos saber que post es el que mejor representa la noticia.
Podríamos representarlo poniendo un atributo booleano en la relación “agrupa” e indicar así si ese post es el que representa la noticia, pero no aseguraríamos que ese post fuera único.
Un post puede ser el más representativo desde ninguna a muchas noticias. (0..N)
Una noticia esta representada por un único post. (1..1)

NOTICIA – NOTICIA (asocia)

Las noticias pueden estar asociadas con otras, con esto obtendremos una línea en el tiempo de como la noticia a ido evolucionando desde sus orígenes.
Una noticia se asocia con ninguna o con muchas noticias (0..N)
Una noticia es asociada con ninguna o con muchas noticias (0..N

4.1.4.4.- Conexión a BD

ContextListener

ContextListener es un oyente del contexto servlet al que se llamará cuando arranque y cuando se pare la aplicación PFC. Esto se configura en el descriptor de despliegue de la aplicación web.xml. Cuando se arranca la aplicación PFC se crea un ejemplar de la clase Politiktracker y se almacena como un atributo de contexto; cuando se cierra la aplicación, el objeto Politiktracker se recupera del atributo de contexto y se cierra la conexión con la base de datos. Entre tanto, las JSP recuperan el objeto Politiktracker del atributo de contexto siempre que necesitan acceder a la base de datos.

Propósito de la clase ContextListener:

El propósito de la clase JavaContextListener es crear un ejemplar de Politiktracker y almacenarlo como un atributo del contexto servlet identificado como: "BaseDatos". Esto permitirá a las páginas JSP recuperar el ejemplar de Politiktracker y acceder a cualquiera de las tablas que se han definido.

Métodos de la clase ContextListener

- **contextInitialized**

  Se llama a contextInitialized siempre que la aplicación web arranca.

  Primero se recupera el contexto servlet desde el evento de contexto servlet:

  ```
  ServletContext servletContext = servletContextEvent.getServletContext ();
  ```

  Luego se crea un ejemplar de Politiktracker y se graba en el contexto servlet identificado como: "BaseDatos". Si este proceso lanza una excepción se guarda un log:

  ```
  try {
     Politiktracker politiktracker = new Politiktracker ();
     servletContext.setAttribute ("BaseDatos", politiktracker);
  } catch (Exception e)  { servletContext.log ("No se pudo crear el atributo BaseDatos: " + e.getMessage());
  }
  ```

- **contextDestroyed**

  Se llama a contextDestroyed cuando se cierra la aplicación web.

Primero se recupera el contexto servlet desde el evento ServletContexto:

```
ServletContext servletContext = servletContextEvent.getServletContext ();
```

Luego se recupera el objeto Politiktracker desde el contexto y se cierra la base de datos:

```
Politiktracker Politiktracker = (Politiktracker) servletContext.getAttribute ("BaseDatos");

Politiktracker.close ();
```

Luego se elimina el atributo del contexto servlet:

```
servletContext.removeAttribute ("BaseDatos");
```

4.1.4.5.- Objetos

Para facilitar el tratamiento de la información, a la hora de recoger, mostrar, actualizar…etc. Se han creado las siguientes clases objeto, de manera que tienen una función lógica de agrupar la información para su posterior utilización.

InformacionPagPrincipal

Este objeto se ha creado con la finalidad de almacenar toda la información referente al bloque principal de las páginas, que coincide con la información relevante para mostrar las noticias. Si nos fijamos en el esquema de entidad relación, podemos observar que agrupa la información relevante de las relaciones entre las tablas de NOTICIA, POST, CANAL Y TEMA.

POST – NOTICIA (agrupa)
En muchos casos la información que aporten los post será redundante. Con la entidad noticia lo que intentamos es agrupar todos los post similares, con los que después crearemos una lista resumida con los post mas significativos de cada noticia.
Una noticia agrupa desde ninguno a muchos post (0..N)
Un post esta agrupado solamente en una noticia (1..1)

CANAL – POST (publica)
Representa la acción de que un canal (blog o periódico digital) haga público un post. Este último será la entidad débil y canal será la entidad fuerte, ya que existe una dependencia en identificación.
Un canal puede publicar desde ninguno a muchos post (0..N)
Un post es publicado por un único canal. (1..1)

TEMA – POST (habla de C)
Existe una relación entre el post y el tema del que habla.

Un post puede tratar desde ningún tema* a muchos temas. (1..N)
Un tema puede ser hablado desde en ningún post hasta en muchos post
* El caso de que un post no hable de ningun tema se debe a que puede que ese tema no este incluido en la base de datos.

A continuación se detalla la información que puede llegar a almacenar:

1.- Lista de Post

2.- Lista de Canales

3.- Lista de Temas

Más adelante pasaremos a detallar cada uno de los objetos que contienen estas listas.

Post

Almacena toda la información relevante de la tabla POST:

- id_Post = Identificador numérico que lista las filas.
- cod_Post = Url del canal al que pertenece.
- Permalink = Enlace permanente al post.
- Titulo = Título del post.
- Texto = Texto que resume la noticia.
- Autor = Autor de dicho post.
- fechaHora
- tendencia_Politica_P = Atributo derivado que define la posición política que defiende dicho post. Este atributo se extraerá automáticamente del texto.
- md5 = Resultado de calcular el algoritmo con los valores “Titulo” y “Permalink”. Este valor será único para cada fila.
- numComentarios = número de comentarios que posee.
- numAfavor = número de votos a favor.

Canal

Almacena toda la información referente a la tabla CANAL:

- url = dirección url del canal.
- titulo= nombre del título del canal.
- ip = dirección ip del canal.
- logo = logotipo que identifica al canal.

- descripción = breve descripción del canal, semejante a una biografía en persona.
- num_enlazados = guarda el número de enlaces a otros canales.
- num_enlazan = contiene el número de enlaces que enlazan con este canal.

- tendencia_politica_c = atributo derivado que indica la afinidad del canal con una determinada fuerza política. Se calculará según la tendencia de los post que genere.
- fecha_canal = Fecha y hora de la primera entrada que aportó ese canal.
- numEntradas = Número de entradas a ese canal.

Tema

Almacena la información referente a la tabla TEMA.

- nombre = nombre identificador de dicho tema.
- foto = atributo opcional que guarda una foto representativa del tema tratado.
- numApariciones = atributo que se va a utilizar para elaborar la nube de tags.

Partido

Almacena la información relevante para la tabla PARTIDO.

- nombre = nombre del partido político.
- logotipo = imagen con el logotipo identificador del partido político.
- siglas = siglas que identifican al partido político.
- información = breve descripción del partido semejante a una biografía en persona.
- ámbito = ámbito político que define mejor la posición del partido.
- datos_basicos = otros datos de relevancia sobre el partido (número de socios, sede,…)
- color = color con el que identificaremos al partido dentro del sistema.
- votosAfavor = indica el número de votos positivos para el partido.
- votosEnContra = indica el número de votos negativos para el partido.

Persona

Gestiona la información relevante de la tabla PERSONA y de sus relaciones con la tabla PARTIDO Y REFERENCIADO.

- nombre = nombre relativo a la persona.
- apellidos = apellidos relativos a la persona.
- foto = imagen de la persona referenciada.
- biografia = breve texto que comenta a grandes rasgos la vida de dicha persona.
- votosAfavor = votos a favor de la persona.
- votosEnContra = votos en contra de la persona.
- partido = partido político de la persona.

- cargoGeneral = cargo que ocupa la persona dentro de la esfera social.
- cargoPartido = cargo que ejerce la persona en el partido.

4.1.4.6.- Consultas a BD

El objetivo principal de las consultas es obtener toda aquella información que queremos mostrar en nuestras páginas. Todas las consultas a base de datos se encuentran ubicadas en la clase “Politiktracker”.

Consultas de la Página Principal

La funcionalidad primordial de las consultas para esta página es obtener las noticias de actualidad sobre el debate político y toda la información que se relacione con estas, es decir, las personas que estén implicadas o relacionadas con la noticia, temas que aparecen en las noticias y los partidos que estén relacionadas con estas.

Consulta 1:

```
public InformacionPagPrincipal getpPrincipal (int desde, int hasta) {…}
```

Consulta de las noticias actuales, mostrará las últimas noticias de las que se dan todos los datos de su post asociado. El número de noticas que se recuperará será el que delimite el intervalo que se facilita por parámetro. Cabe destacar que este intervalo se usa para controlar el número de registros que queremos mostrar, de esta manera y con estas variables controlaremos la paginación de registros, para no ser repetitivos este inciso sobre dichas variables será válido para todas las funciones que usen estos parámetros.

Esta función devolverá null si se ha producido error en la conexión, ejecutando la query o no ha encontrado ningún registro que cumpla con la query. Devolverá un objeto de la clase InformacionPagPrincipal, que contiene toda la información de los registros encontrados.

Se utiliza para recuperar la información que deberemos mostrar en el bloque principal, debajo del gráfico.

Consulta 2:

```
public ArrayList<Persona> getPersonasPprincipal (int desde, int hasta, String fecha) {…}
```

Muestra las últimas personas (Nombre, Apellidos, contador de numero de noticias, Votos a favor, votos en contra, Foto (puede ser NULL) y Biografía (puede ser NULL) que han aparecido en las ultimas noticias desde la fecha que se le pasa por parámetro a la función. Recupera el número de personas que se le indica en el intervalo que se le pasa por parámetro.

Si no se ha producido ningún problema en la ejecución de la query, la consulta devolverá un array vacío cuando no se haya encontrado ningún registro que

cumpla la query o en caso contrario uno que contendrá tantos objetos persona como los registros encontrados dentro de dicho intervalo.

Se utiliza para recuperar los datos que mostraremos en el bloque superior del bloque derecho de la página principal (derecha del gráfico).

Consulta 3:

```
public ArrayList<Tema> getconsultaParaNubeDeTags (String fecha) {…}
```

Obtiene un número de temas a partir de la fecha que le indicamos por parámetro.

Devolverá un array vacion cuando no se haya encontrado ningún registro que cumpla con la query. En caso contraro devolverá un array con todos los registros en contrados, nombre del tema, más el número de veces que se repite, además del permalink para saber donde enlaza.

Se utiliza para recuperar la información necesaria para construir la nube de tags.

Consulta 4:

```
public ArrayList<Partido> getPartidosPprincipal (int desde, int hasta,String fecha) {…}
```

Obtiene el intervalo de partidos que se le pasa por parámetro a partir de la fecha que le indicamos.

Devolverá null si se ha producido error en la conexión o ejecutando la query. Devolverá un array vacío cuando no se haya encontrado ningún registro que cumpla la query. Devolverá un array con todos los registros encontrados en caso contrario.

Esta consulta obtiene la información necesaria que se mostrará en el bloque derecho de la página principal, más concretamente en el bloque inferior de éste.

Consultas de la página de políticos:

La funcionalidad de las consultas para esta página se basa en obtener toda la información relevante para la persona que se esté tratando en cuestión. Se obtendrá información sobre la biografía y datos personales de la persona, se obtendrán las noticias, partidos, personas y temas que estén relacionados con la persona que se está tratando en cuestión.

Consulta 5:

```
public Persona getPoliticoPpoliticos (String nombre, String apellidos) {…}
```

Muestra los atributos referenciados (si los tiene, sino NULL) para la persona con los datos correspondientes a los que se le pasan por parámetro.

Devolverá null si se ha producido error en la conexión o ejecutando la query.

Devolverá null cuando no encuentre ningun registro.
Si no se ha producido ningún error devolverá la información referente a una persona.

Al seleccionar una persona de interés del bloque superior derecho de cualquier página, se ejecuta esta query para recoger la información referente, por ejemplo, al político en cuestión para enlazar con la página de políticos y mostrar los datos recogidos.

Consulta 6:

```
public Persona getPoliticoPartidoPpoliticos(String nombre, String apellidos){}
```

Muestra los atributos del partido al que pertenece la persona que se le pasa por parámetro.

Devolverá null si se ha producido error en la conexión o ejecutando la query.
Si no se ha producido ningún error devolveraá la información del partido, si existe.

Consulta utilizada en la página de políticos para mostrar el partido al que pertenece.
Consulta 7:

```
public InformacionPagPrincipal getNoticiasPpersonas (int desde, int
hasta,String nombre,String apellidos,String fecha) {…}
```

Muestra el intervalo de las últimas noticias en las que se habla de una persona a partir de una fecha dada. El intervalo, datos de la persona y la fecha se pasan por parámetro.

Esta consulta es utilizada en la página de políticos para obtener el bloque inferior izquierdo, referente a las noticias relacionadas con la persona en cuestión.

Consulta 8:

```
public ArrayList<Tema> getconsultaParaNubeDeTagsPagPoliticos (String nombre,
String apellidos,String fecha) {…}
```

Consulta de temas recientes relacionados con la persona de la que se está hablando en la página de políticos.

Esta consulta se utiliza para el bloque intermedio del bloque derecho de la página, y gracias a ella seremos capaces de construir la nube de tags que enlazará con la página de temas.

Consulta 9:

```
public ArrayList<String> getIdPostsPpersonas (String nombre,String
apellidos,String fecha) {…}
```

Esta consulta es necesaria para obtener los identificadores de los post en los que se habla de la persona que le pasamos por parámetro a esta función. Además le pasamos por parámetro la fecha a partir de la cuál queremos obtener dicha información.

Es una consulta intermedia para poder obtener la información que buscamos y recogeremos en la siguiente consulta.

Consulta 10:

```
public ArrayList<Persona> getPersonasPpersonas (int post,String
nombre, String apellidos) {…}
```

Consulta de personas relacionadas con la persona que se le pasa por parámetro a la función.

Por cada identificador del post sacado en la anterior consulta, sabemos que personas además de la persona que le facilitamos a la función se habla, obtendrá todas aquellas que o bien tengan el nombre o bien tenga los apellidos distintos o ambos casos.

Con estas dos últimas consultas seremos capaces de mostrar la información referente a la parte superior del bloque derecho de la página de políticos.

Consultas de la página de partidos:

La funcionalidad es similar a las consultas de la página de temas, con la diferencia de que en este caso toda la información que se recupera y muestra es referente a los partidos. Con estas consultas queremos dar al usuario un conocimiento sobre el partido en cuestión, sobre en qué actividades anda metido, gracias a las noticias en las que aparece, y sobre que personas participan en él o están relacionadas de alguna manera con él.

Consulta 11:

```
public InformacionPagPrincipal getNoticiasPpartido (int desde, int
hasta, String nombre, String fecha) {…}
```

Consulta de noticias relacionadas con el partido, muestras las noticias que marca el intervalo que se le pasa a la función, sobre la persona que se le facilita a partir de una fecha dada.

Función necesaria para obtener la información que se mostrará en el bloque inferior izquierdo de la página de partidos.

Consulta 12:

```
public ArrayList<Tema> getconsultaParaNubeDeTagsPagPartidos (String
nombre,String fecha) {…}
```

Consulta de temas recientes relacionados con el partido. Obtiene los últimos temas desde una fecha dada de los que hablan los post relacionados con un partido de nombre dado. Gracias a esta consulta podremos construir la nube de tags de la página del partido.

Se mostrará en la parte intermedia del bloque derecho de dicha página, manteniendo así el mismo formato que en el resto de páginas.

Consulta 13:

```
public Partido getInformacionSobrePartido (String nombre) {}
```

Recoge toda la información de un partido determinado por el campo que se pasa por parámetro a la función.

Devolverá NULL tanto si se ha producido algún error de conexión, ejecutando la query o cuando no se haya encontrado ningún registro que cumpla los requisitos.

Devolverá una lista con todos los registros encontrados en caso contrario.

Esta información se mostrará en la parte superior del bloque izquierdo.

Consulta 14:

```
public ArrayList<String> getIdPostsPpartidos (String nombre, String
fecha) {…}
```

Consulta intermedia que obtiene la información referente a los últimos identificadores de post en los que se habla del partido que se le pasa por parámetro.

Consulta 15:

```
public ArrayList<Persona> getPersonasPpartidos (int post) {}
```

Al igual que ocurría en la página de personas al obtener las personas relacionadas, esta consulta necesita la información de la anterior para a partir de los identificadores de post obtenidos, obtener la información de las personas relacionadas.

Consulta 16:

```
public ArrayList<Partido> getPartidosPpartidos (int post,String
nombre) {…}
```

Consulta de partidos relacionados con el partido, por cada identificador de post obtenido, sabemos que partidos además del partido pasado por parámetro se habla.

Esta información se muestra en la parte inferior del bloque derecho.

Consultas de la página de temas:

La funcionalidad de las consultas para esta página se basa en la extracción de información de la base de datos sobre un tema en particular, de manera que mostremos el logo, imagen o fotografía si tiene alguna asociada del tema en cuestión, además de obtener todas las noticias que estén relacionadas con este tema, además de recuperar las

personas y partidos que estén relacionados con éste y con las noticias que se han obtenido a partir de éste.

Consulta 17:

```
public Tema getInformacionSobreTema (String nombre) {…}
```

Obtiene toda la información sobre el tema que se le pasa por parámetro. Actualmente solo se dispone del campo foto.

Esta información será mostrada a través del jsp pagina de temas en la parte superior del bloque izquierdo.

Consulta 18:

```
public InformacionPagPrincipal getNoticiasPtema (int desde, int hasta,String
nombre,String fecha) {…}
```

Obtiene las últimas noticias en las que se habla de un tema específico que se pasa por parámetro, a partir de una fecha determinada que también se facilita.

Esta información se mostrará en la parte inferior del bloque izquierdo manteniendo la estructura que hasta ahora seguimos para todas las páginas.

Consulta 19:

```
public ArrayList<Persona> getPersonasPtema (int desde, int hasta,String
nombre,String fecha) {…}
```

Obtiene el intervalo indicado de post y las personas de las que se habla a partir de una fecha determinada relacionados con el tema específico que se le facilita por parámetro a la función.

Se muestra en la parte superior del bloque derecho de la página de temas.

Consulta 20:

```
public ArrayList<Partido> getPartidosPtema (String nombre,String fecha) {…}
```

Obtiene la información relative a los partidos que están relacionados con el tema que se le pasa por parámetro a partir de una fecha determinada.

Esta información se muestra a continuación de la información que se muestra en la consulta anterior, ya que en la página de temas no existe el bloque intermedio del bloque derecho puesto que no se ha definido ninguna nube de tags.

Consultas, altas y modificaciones que se pueden producir en cualquiera de las páginas:

Consulta 21:

```
    public int insertarComentario( String comentario,String id_Post,String
alias,String fechaHoraActual) {…}
```

Esta función da de alta un nuevo comentario para el identificador del post que se haya seleccionado.

Devuelve 0 en el caso de que se haya producido algún error.

Esta función se utiliza cuando por ejemplo, pinchamos sobre el icono de comentarios de una noticia, se abre una nueva ventana y nos da la posibilidad de introducir un nuevo comentario, donde esta función se encargará de darlo de alta en la base de datos, además de almacenar también el alias de quien lo ha escrito.

Consulta 22:

```
public int modificarNumAfavorPost (int votosActuales,String idPost) {…}
```

Modifica el número de votos a favor de la noticia en la que se haya pinchado el icono, se le pasa por parámetro el número actual de votos y se encarga de incrementar en uno este campo para el post dado.

Devuelve 0 en el caso que se haya producido algún error.

Se utiliza cuando pinchamos sobre el icono de votos a favor.

Consulta 23:

```
public int modificarNumComentarosPost (int votosActuales,String idPost) {…}
```

Modifica el número de comentarios, se le pasa por parámetro el número actual de comentarios y se encarga de incrementar en uno este campo para el post dado.

Devuelve 0 en el caso que se haya producido algún error.

Se utiliza cuando pinchamos sobre el icono de enviar comentario. Ya que debe actualizar el número de comentarios teniendo en cuenta que estamos añadiendo uno nuevo.

Consulta 24:

```
public int modificarVotosPositivosPersona (int votosActuales,String
nombre,String apellidos) {…}
```

Modifica el número de votos positivos de la persona, se le pasa por parámetro el número actual de votos y se encarga de incrementar en uno este campo para la persona dada através de los campos nombre y apellidos que se pasa por parámetro.

Devuelve 0 en el caso que se haya producido algún error.

Se utiliza cuando pinchamos sobre el icono de votos a favor en el bloque de personas relacionadas que se encuentra en cualquiera de las páginas.

Consulta 25:

```
public int modificarVotosNegativosPersona (int votosActuales,String
nombre,String apellidos) {…}
```

Igual que para los votos positivos de persona con la diferencia que el campo afectado esta vez es el de votos negativos de ésta.

Consulta 26:

```
public int modificarVotosPositivosPartido (int votosActuales,String nombre)
{…}
```

Igual que para los votos positivos de persona con la diferencia que el campo afectado esta vez es el de votos positivos del partido.

Consulta 27:

```
public int modificarVotosNegativosPartido (int votosActuales,String nombre)
{…}
```

Igual que para los votos negativos de persona con la diferencia que el campo afectado esta vez es el de votos negativos del partido. Se produce cuando se vota negativamente sobre un partido.

4.1.5. DISEÑO Y FUNCIONALIDAD DE LAS PÁGINAS DESARROLLADAS

4.1.5.1 DISEÑO INICIAL DE LAS PÁGINAS CON PLANTILLAS ESTÁTICAS (HTML).

Página principal

En la página principal los usuarios accederán a una visión panorámica y sintética de la información más relevante y reciente que se actualizará de forma dinámica. Entre las funcionalidades disponibles en esta primera página tendremos:

- Visualización de un gráfico con la popularidad de los candidatos principales a lo largo de las últimas semanas. Los candidatos principales serán determinados por

los administradores del sistema. (Apartado sin implementar por pertenecer a otro proyecto)

- Se mostrará una lista con las personas de interés. Esta lista mostrará una foto del candidato y permitirá enlazar con uno de los ejes o vista que se pretende desarrollar (Página de personas relacionadas), cuyas funcionalidades se describen con más detalle en líneas posteriores.
- En otro módulo de información mostraremos una nube de temas (al estilo de las nubes de tags) que permitirá acceder a la página específica de un tema. La selección de los temas será semiautomática, es decir, habrá un conjunto de temas seleccionados por el administrador y otros que se generarán dinámicamente a

  partir del contenido de las entradas. La nube de temas mostrará gráficamente la importancia del tema mediante el uso de tipos de letra más grandes y con colores

- más destacados. Cada uno de estos enlaces nos guiará a otro de los ejes que hemos considerado, Página de Temas que se describe más adelante.
- Además existirá un nuevo módulo que mostrará un listado con los partidos políticos más relevantes, decisión que se le atribuirá al administrador. Permitirá

  acceder a las páginas dedicadas en el sistema a cada uno de los partidos y que corresponden con el tercer eje que hemos tenido en cuenta para acercarnos a la información.

- Además dispondremos de funcionalidades que faciliten la navegación, que hagan de nuestro prototipo de interfaz un prototipo accesible para la mayoría de personas y recursos.

Páginas de Políticos

- Permitirá ver una foto del político y una breve biografía.

- Se representará aquellas noticias en las que tenga algo que ver el político o persona de interés en cuestión. La información, se presentará de forma sintética y agrupada respecto a temas, polaridad y aportaciones de estas opiniones. Del mismo modo se tendrá en cuenta la relevancia o influencia de las fuentes que emiten la información y las opiniones para confeccionar el orden y la representatividad.

- En otro bloque de información presentaremos una nube de temas, semejante a la de la página principal pero que mostrará los temas más relevantes alrededor del político.

- Del mismo modo se podrá acceder a la información de otras personas relacionadas con este político así como a la información de su partido.

Pagina de partidos

- Presentará información semejante a las anteriores pero en este caso para los partidos. La página debe incluir información sobre el partido, siglas, ámbito y datos básicos sobre el partido, semejantes a los de los candidatos.

- Del mismo modo será posible disponer de una lista de los candidatos que facilite la navegación.

- La información en el bloque principal contendrá las noticias que estén relacionadas con ese partido.

Página de temas

El objetivo de esta página es organizar la información más reciente sobre un tema de debate que tratará de ilustrarse mediante una fotografía, a ser posible reciente. Esta fotografía se puede elegir automáticamente o ser asignada por los administradores.

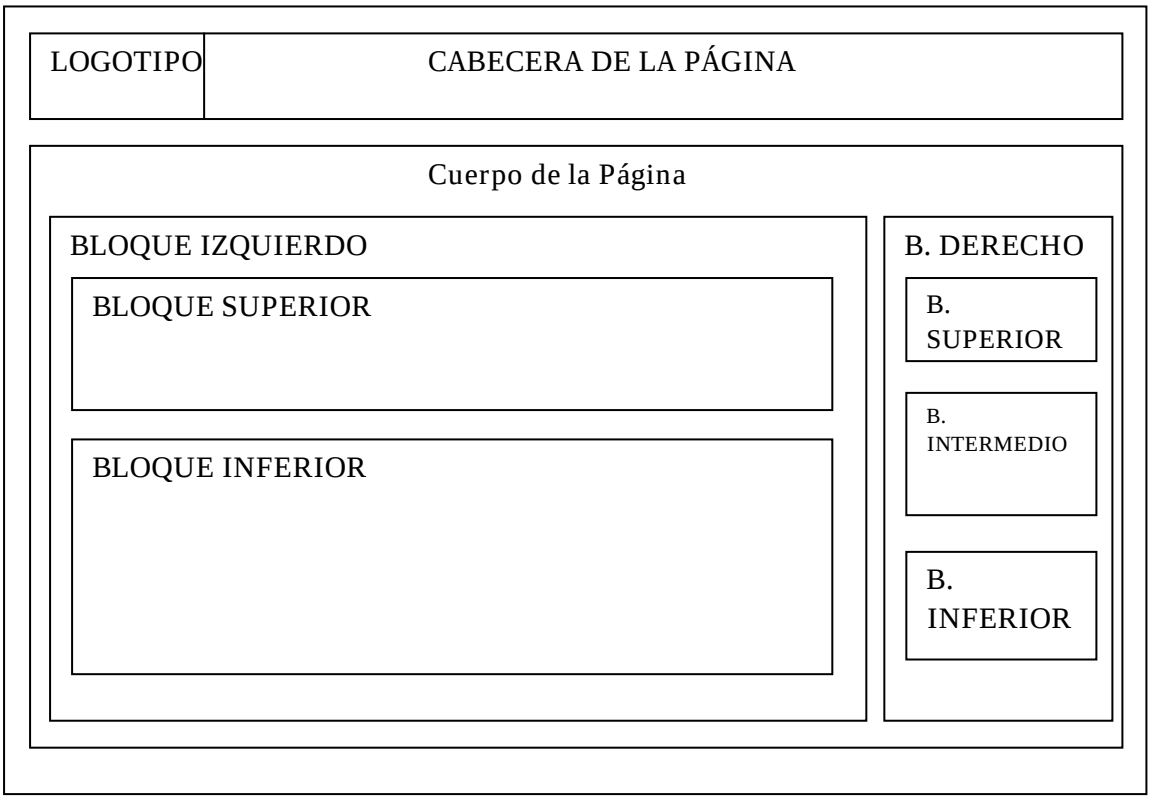
- Se mostrará en un bloque de información aquellas noticias que estén relacionadas con el tema.

- Además se dispondrá de opciones de navegación hacia las páginas de los políticos y de sus partidos y de la página principal.

- Además se dispondrá de opciones de navegación hacia las páginas de los políticos y de sus partidos.

4.1.5.2. DISEÑO FINAL, FUNCIONALIDADES IMPLEMENTADAS Y HERRAMIENTAS UTLIZADAS PARA LA VISUALIZACIÓN DE LAS PÁGINAS.

A continuación vamos a presentar la distribución que se ha elegido para el diseño de todas las páginas, con la excepción de la página de temas que carece de nube de tags (el bloque intermedio del bloque derecho). Para desarrollar este diseño se han utilizado jsp, hojas de estilo (CSS) y la ayuda de javascript. Cada una de las páginas contiene la asociación a una hoja de estilos y al javascript. Se han tenido en cuenta los estándares para hacer nuestras páginas accesibles y cada una de ellas está validada, cumpliendo así con éstos objetivos. En cada página aparecen las etiquetas de validación.



La cabecera de la página va a contener la funcionalidad de búsqueda, este prototipo no incluye su implementación que está reservada para otro proyecto. A excepción de la página principal, también incluye un enlace hacia está para que desde cualquiera del resto de páginas se pueda acceder al inicio.

El cuerpo de la página va a incluir toda la información que queremos mostrar en nuestra página, se divide en dos grandes bloques:

- El bloque principal (bloque izquierdo): mostrará la información más importante en cada momento.
- El bloque derecho: muestra la información relacionada con el bloque principal. Se divide en tres bloques:

  1.- Boque superior: incluye la información si existe, de las personas de interés que están relacionadas con la información que se presenta en el bloque principal, a excepción de la página principal normalmente se relaciona con la información que contiene el bloque superior del bloque izquierdo.
  2.- Bloque intermedio: nube de tags de los temas relacionados.

3.- Bloque inferior: partidos relacionados, si existen, con la información del bloque izquierdo.

CONSTRUCCIÓN DE LA PÁGINA ATRAVÉS DE LA HOJA DE ESTILO (CSS)

Nuestro objetivo desde un principio ha sido crear un código que pueda ser reutilizable, gracias a este concepto nuestro trabajo se ha visto facilitado, puesto que creando la estructura para la primera página que desarrollamos, hemos obtenido también el resto de páginas.

Cabe destacar que nuestra página está desarrollada totalmente por bloques, es decir, el elemento html que hemos utilizado son los “div”, esto nos ha permitido reutilizar la mayor parte de nuestros bloques, y dejar atrás los anticuados y laboriosos “table” que se usaban antes.

A continuación vamos a explicar más detalladamente cómo se ha logrado obtener esta estructuración para cada una de las páginas gracias a la ayuda de “estiloPprincipal.css”, la hoja de estilos (CSS) que hemos creado.

Es importante a la hora de definir los estilos de nuestra CSS, tener en cuenta que no todos los usuarios van a acceder con las mismas resoluciones de PC. Esto implica que según con la resolución que posea el PC que intenta acceder a nuestra página, ésta puede visualizarse de forma incorrecta, para que esto no ocurra, nuestra hoja de estilos tiene en cuenta todas sus dimensiones a través de porcentajes, de esta manera conseguimos que nuestra página se adapte de la mejor forma posible a cada resolución.

Definición de capas:

Capa 1: corresponde a la etiqueta “body” de html y será la capa sobre la que se posicionarán el resto de capas, por este motivo en nuestra CSS hemos creado el siguiente estilo:

```
body,html { margin:0;padding:0;height:100%;width:100%;background-
color:white;}
```

Este estilo le proporcionará a la capa 1 el máximo tamaño de pantalla, sin dejar ningún tipo de margen y proporcionando un fondo blanco, sobre esta capa irán el resto de capas por lo que este fondo nunca llegará a verse. Con tan solo la definición de un estilo en la página CSS podemos darle forma a la etiqueta body de cada una de las páginas, si

quisiéramos darle alguna particularidad a algún estilo común basta con poner a la etiqueta en cuestión un atributo “id” o “class” y utilizar la propiedad en cascada que nos proporciona la hoja de estilos.

Capa Cabecera: Esta capa corresponde con la parte superior de la página y se mantendrá para todas las páginas. El estilo que le corresponde es "#capaCabecera {}”. Sobre ella se encuentran otras dos capas que gracias a la flotabilidad de estas, podemos definir su profundidad y colocarlas encima. Estas dos capas son:

1.- La **capa** que contendrá la imagen del **logotipo** de nuestra página. Denominada en la hoja de estilos como #logo{…}.

2.- La **capa** que contendrá las capas de **búsqueda** (“#capaContienebuscar {…}) y que contendrá a excepción de la página principal la **capa** con el **enlace** a esta última (capaContieneEnlacePagPrincipalPPartido{…}).

Capa Cuerpo: corresponde al bloque cuerpo ya mencionado anteriormente, el estilo que define esta capa en la hoja de estilos se denomina “#capaContenido {…}“. Es uno de los estilos comunes a todas las páginas. Sobre esta capa se van a posicionar las distintas capas para presentar toda la información obtenida de bd. Como se comentó a la hora de estructurar la página vamos a ver qué estilos corresponden a cada uno de los bloques con todas sus peculiaridades:

- El bloque izquierdo: su estilo para la página principal es: “.bloqueIzq{}”, se diferencia del resto de páginas en su ancho, siendo este un poco más estrecho. Cada página tiene un class de estilo distinto para este bloque puesto que hemos querido diferenciar el color de cada página. Los estilos que corresponden al resto de páginas para este bloque son:“.bloqueIzqPtemas”,“.bloqueIzqPpoliticos” ,”. bloqueIzqPpartidos”. Esta capa contiene dos capas:

    1. La capa del bloque superior: que corresponde al estilo “.graficoPP” es común para todas las páginas. Se adapta al ancho definido por la capa que lo contenga.
    2. La capa del bloque inferior: se corresponde con el estilo “.bloquePrincipal” que también es común para todas las páginas. El bloque principal contiene a su vez una serie de capas que no vamos a ir nombrando una a una por no hacer demasiado extenso este documento, ya que solo queremos determinar que estilos son los que logran conseguir la estructuración principal de la página.

- El bloque derecho: “.bloqueMenuDcho” para la página principal y para el resto: “.bloqueMenuDchoPpartidos”, “.bloqueMenuDchoPtemas”, “.bloqueMenuDchoPpoliticos”, se crea un estilo para cada página puesto que sus dimensiones y color de fondo son diferentes.

  1.- Bloque superior: corresponde con el estilo “.bloquePoliticos”, es un único estilo común para poder ser usado en las 4 páginas, la capa que contiene a este estilo será la encargada de determinar las diferencias entre páginas.

  2.- Bloque intermedio: al bloque intermedio le porporciona sus características el estilo “.nubeTagsBNube” y para poder crear su contenido y de esta manera la nube son necesarios los siguientes estilos:

```
#nubeTagsBNube .tag1{ font-size:13px; color:#800000;}
#nubeTagsBNube .tag2{ font-size:17px; color:#ff6020;}
#nubeTagsBNube .tag3{ font-size:19px; color:#ff4020;}
#nubeTagsBNube .tag4{ font-size:21px; color:#a00000;}
```

  Los cuales dan color y tamaño a los enlaces de la nube, para resaltar aquellos con más importancia utilizamos colores y tamaño de letra más llamativo, en este caso de mayor a menor importancia sería del tag4 al tag1.

```
#nubeTagsBNube a{ text-decoration:none; }
#nubeTagsBNube a:hover{ text-decoration:underline; }
```

  Estos los usamos para que aparezca subrayado el enlace cuando nos posicionamos sobre él.

  3.- Bloque inferior: al igual que ocurre para el bloque superior el estilo del inferior denominado “.bloquePartidos” es común para todas las páginas.

MOSTRAR INFORMACIÓN USUARIO AL PASAR SOBRE EL LOGO DEL CANAL

Gracias a las hojas de estilo (CSS), podemos mostrar información que debido al reducido espacio del que disponíamos va a aparecer y desaparecer según la necesitemos o no.

En el bloque inferior izquierdo, cuando mostramos la información principal existe un logotipo que corresponde al logotipo del canal de donde se ha obtenido la información, donde al posicionar el ratón sobre el podremos obtener información relevante del usuario del canal. Como es un lugar donde se agolpa mucha información, decidimos

utilizar está técnica para mostrar y ocultar la capa que contiene esa información. La capa se muestra cuando posicionamos el ratón sobre la imagen y se oculta al salir de ella.

<%= canal.getLogo()%>" alt="Logo del Canal" border="0" onmouseout="ocultar('informacionUsuario<%=numRegistroPag%>')" onmouseover="mostrar('informacionUsuario<%=numRegistroPag%>')"/>

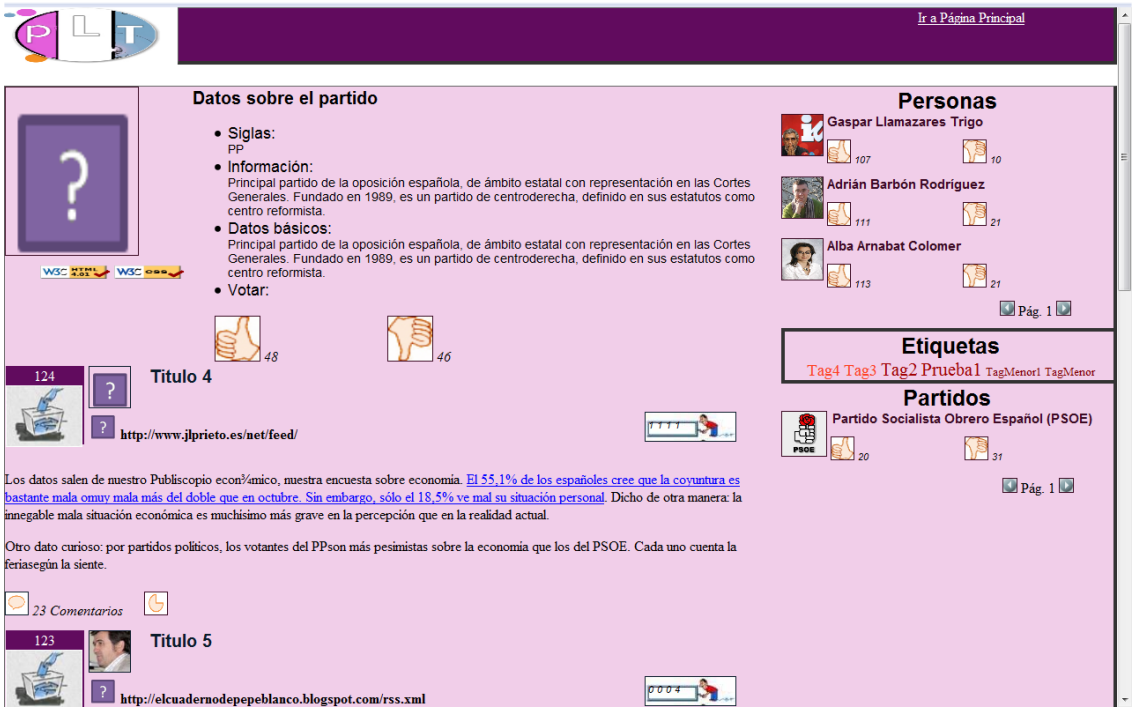
A continuación mostramos las funciones javascript que hemos desarrollado para este objetivo:

```
//esta función muestra la capa que se le pasa por parámetro
function mostrar(nombreCapa){
      document.getElementById(nombreCapa).style.visibility="visible";
}

//Oculta la capa que se le pasa por parametro
function ocultar(nombreCapa){
document.getElementById(nombreCapa).style.visibility="hidden";

}
```

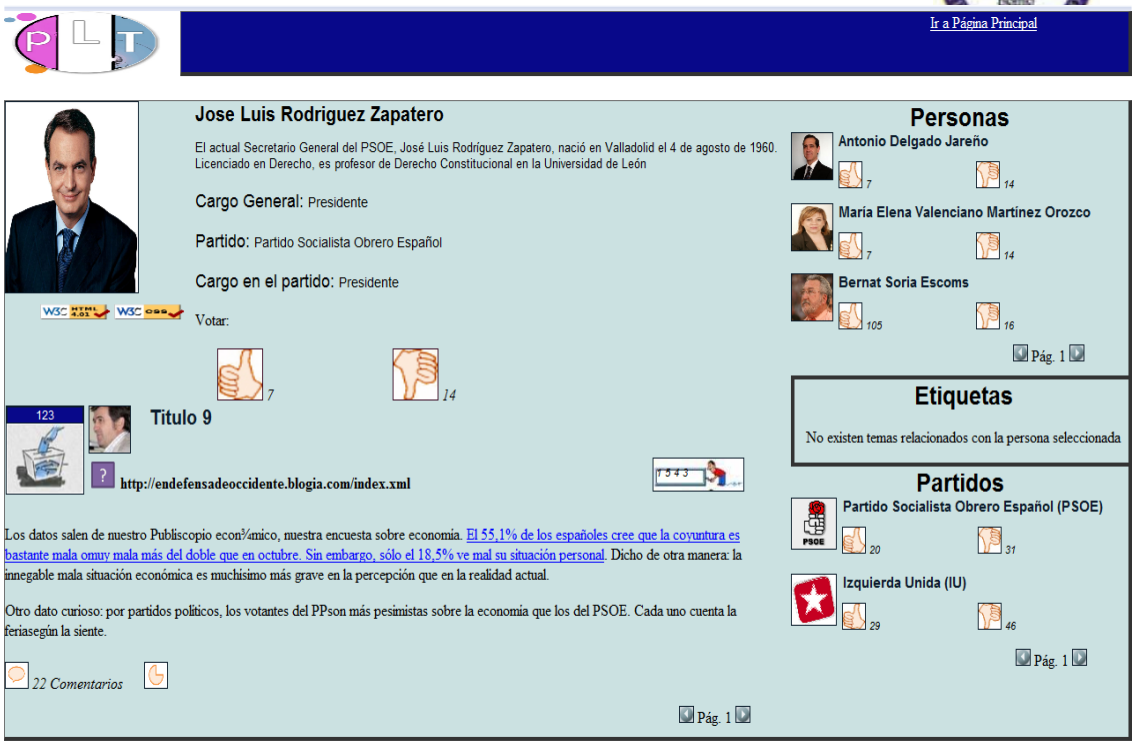
A continuación se muestra el resultado definitivo de las páginas de nuestro prototipo:

Página Principal:

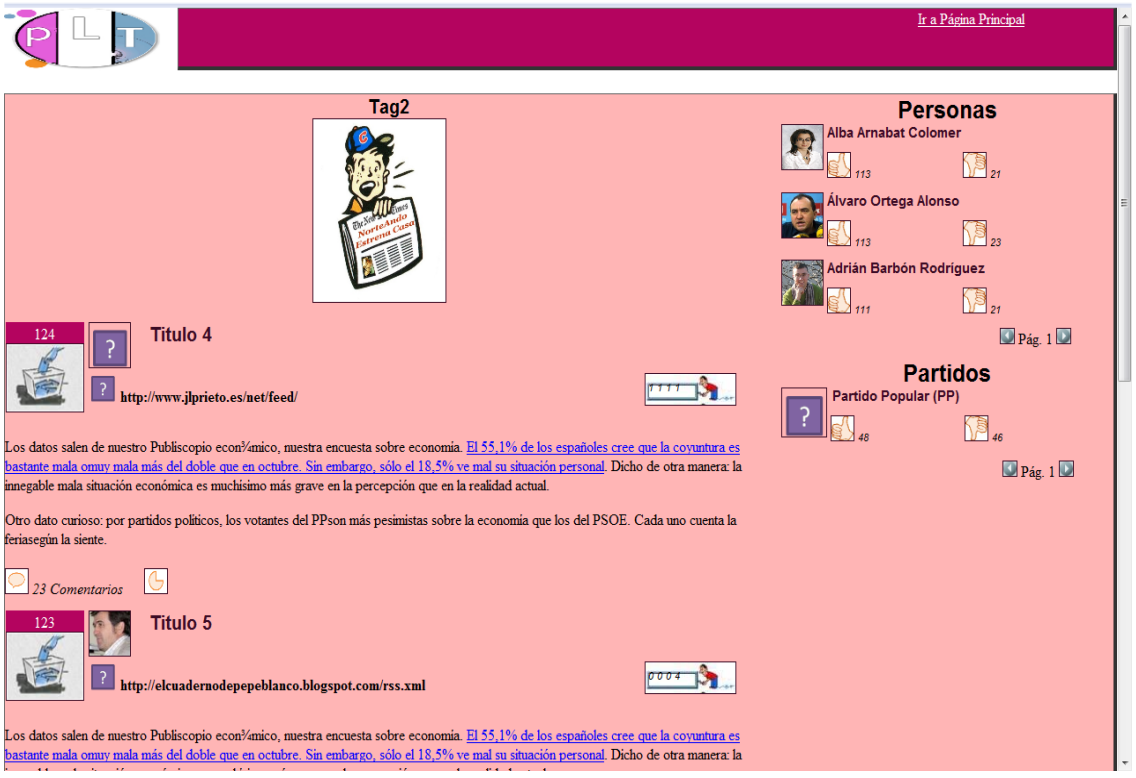
Página de Partidos:



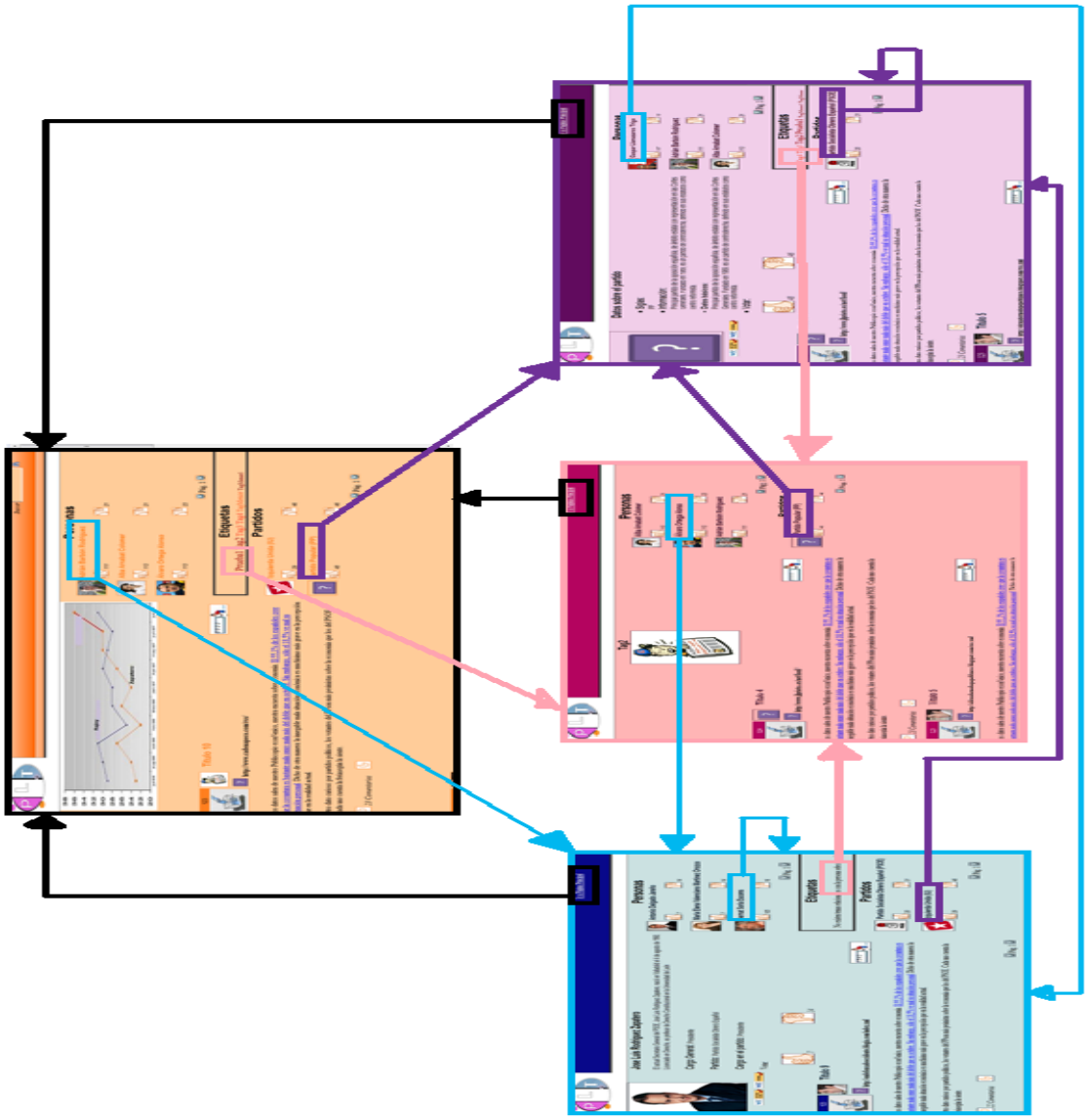
Página de Personas Relacionadas:



Página de Temas:



MAPA DE NAVEGACIÓN



Notas:

Indica la navegación hacia la Página de Partidos.

Indica la navegación hacia la página de Persona Relacionada.

Indica la navegación hacia la Página de Temas.

Indica la navegación hacia la Página

Principal.

5. FUNCIONALIDADES

5.1 SISTEMA DE PAGINACIÓN:

Ha sido necesario crear un sistema de paginación debido al volumen de información que se quiere representar y al espacio del que disponemos. Los bloques que poseen este sistema son el bloque inferior izquierdo, donde se representará la información principal, el bloque superior derecho, donde representamos las personas relacionadas y el bloque inferior derecho donde se representan los partidos relacionados.

Cada uno de los bloques mencionados anteriormente contiene un grupo de variables para el control de la paginación, a continuación, vamos a explicar en qué consiste este sistema. Utilizaremos como ejemplo el grupo de variables referente al bloque inferior izquierdo, para el resto de bloques se utiliza el mismo mecanismo, con la única diferencia que su grupo de variables poseen identificadores distintos según el bloque del que se trate.

"Rowsperpage" = esta variable va a determinar el número de registros que queremos mostrar por página, para el bloque de noticias se ha decidido mostrar 5 noticias por página, para el bloque de personas relacionadas 3 registros al igual que para el de partidos relacionados. Al almacenar este dato en ésta variable nos permite en cualquier momento si cambiamos de opinión en cuanto a la representación de esta información, modificar su valor sin necesidad de tocar ninguna línea de código más y representar tantos registros como deseemos. De esta manera obtenemos un código totalmente reutilizable y accesible, y puesto que de momento es un prototipo y está sujeto a cambios facilitamos la tarea posterior de desarrollo.

"Size" = almacenará el número de registros recuperados, tras realizar la consulta a la base de datos, como máximo será el valor de la variable "Rowsperpage", eso querrá decir que ha recuperado el total de registros de una página. Si su valor se comprende entre 1 y "Rowsperpage" indicará que estamos en la última página y mostrará únicamente los registros recuperados. Si su valor es 0, indica que no se han recuperado registros por lo tanto no hay más páginas.

"CurrentPage" : almacena el valor de la página actual.
"CurrentRow": guarda el valor del primer registro de la página actual.

Estas dos últimas variables son cruciales, ya que gracias a ellas podremos realizar todos los cálculos necesarios para la paginación. Por este motivo estos dos valores deben ser enviados cada vez que actualicemos la página, para poder calcular en cada momento la paginación correspondiente. La forma de enviarlos es como atributos en la llamada a la

página, de esta manera y gracias a las jsp que nos permiten utilizar código java podemos recuperar estos valores a través del request.

"Previous": contiene el primer registro de la página anterior.
"Next": primer registro de la página siguiente.
"PreviousPage": almacena en número de página anterior.
"NextPage": posee el número de página siguiente.
"NumPagQVamosRecuperando": ahora mismo nuestro prototipo está pensado para paginar de una en una, por eso el valor de esta variable es igual a 1, pero al tener esta variable dejamos abierta la posibilidad de cambiar esta opción y paginar a nuestro antojo, el código ya está preparado para ello, solo sería necesario cambiar este valor a nuestro gusto.
"UltimaPag": esta es una variable que nos indica verdadero o falso, según si hemos llegado a la última página o no, nos facilita el control para la paginación.

Vamos a ver la funcionalidad y actuación de estas variables para los diferentes casos que se nos pueden presentar.

En la descripción y explicación de las consultas en puntos anteriores, hemos observado que en la mayoría se pasaban por parámetro dos variables que determinaban el intervalo de registros que se querían recuperar. Estas variables se calculan teniendo en cuenta la paginación y de la siguiente manera:

```
int desde=currentRow; //registro a partir del cuál debe recuperar en
la base de datos.
int hasta=Rowsperpage*numPagQVamosRecuperando; //numero de registros
que se desea recuperar
```

El punto anterior es común y se realiza para todas las casuísticas que a continuación se describen:

1.- Se llama por primera vez a la página, por lo tanto el "currentRow" todavía no ha sido calculado, y su valor será nulo, gracias a esta información sabemos que es la primera vez que accedemos a la página e inicializaremos su valor a 0 y el del "currentPage" a uno, una vez inicializados estos valores ya podremos calcular el resto.

2.- Si detectamos que es la última página, esto ocurre cuando el objeto en cuestión que intentamos recuperar viene vacío, debemos trucar los valores de las variables para volver a obtener la página anterior que es la última que contiene información.

```
ultimaPag=true;
desde=desde-Rowsperpage;
currentRow=currentRow-(2*Rowsperpage);
currentPage=currentPage-2;
```

Se vuelve a realizar la consulta con los nuevos valores del intervalo. De esta manera siempre que pulsemos a página siguiente mostrará la última página que contenga información.

En este caso en particular, se han trucado los valores para poder realizar la consulta y obtener la información, por ello después es necesario volver a dejar estas variables con sus valores verdaderos. Gracias a la variable "ultimaPag" identificamos que debemos restablecer los valores de la siguiente manera:

```
currentPage=currentPage+1;
currentRow=currentRow+Rowsperpage;
if (currentPage > 2){
            previous=previous+Rowsperpage;
            previousPage=previousPage+1;

}else{
      previous=0;
      previousPage=1;
}
```

3.- El cálculo del resto de variables a partir de "currentRow" y "currentPage" se realiza de la siguiente manera:

```
if (currentRow==0 ||  currentPage==1 ){ //Para la primera página
      previous =0;
      next=Rowsperpage;
      previousPage=1;
      nextPage=2;
}else{ // Para el resto de páginas
      previous = (currentPage - 2) * Rowsperpage;
      next=currentRow+Rowsperpage;
      if ( currentPage > 2){
            previousPage = currentPage - 1;
      } else {
            previousPage = 1;
      }
      nextPage=currentPage+1;
}
```

4.- La paginación viene determinada en cada bloque por una imagen de flecha hacia a la izquierda (retrocede en la paginación), el número de página actual y una imagen de flecha hacia la derecha que se encarga de avanzar en la paginación.

Cada imagen lleva asociada su llamada para realizar su función:

Retroceder: llamada a la página en cuestión pasando como atributos de la llamada las variables necesarias para la paginación, en este caso el "currentRow" tomará el valor de la variable "previous" y el "currentPage" tomara el valor de la variable "previousPage".

Avanzar: contiene la llamada a la página en cuestión pasándole como atributos el “currentRow” con el valor de la variable “next” y el “currentPage” con el valor de la variable “nextPage”.

5.2. SISTEMA DE VOTACIÓN:

Existen tres tipos de iconos para realizar la votación:

Votar a la noticia: 

Votos en contra: 

Votos a favor: 

Aunque el mecanismo para realizarla es el mismo para estos iconos, la única diferencia será el campo donde se asigne el resultado. Asi que vamos a explicar por ejemplo la votación cuando pulsamos el icono de votos a favor:

1.- Cuando pulsalmos sobre el icono se ejecuta la función onclick de éste:

onClick="sumar1('<%=votosActualesPositivos%>','votoPositivoPersona<%=numRegistroPag%>','paginaPrincipal.jsp?currentRowPersonas=<%=currentRowPersonas%>&currentPagePersonas=<%=currentPagePersonas%>&currentRow=<%=currentRow%>&currentPage=<%=currentPage%>&currentRowPartidos=<%=currentRowPartidos%>&currentPagePartidos=<%=currentPagePartidos%>&');"

Se le pasan como parámetros a la función, los votos actuales, el tipo de voto que se va a realizar y la página que queremos actualizar para mostrar el nuevo número de votos, esta página contiene una serie de atributos referentes a la paginación para mantener y mostrar la misma paginación que antes de votar. El objetivo de esta función es incrementar en uno el número de votos y actualizar la página.

2.- Cuando se hace la llamada a la página, esta se encarga de analizar si se ha modificado el número de votos, de que tipo se ha modificado y sobre que registro y con esta información llama a la función de modificación correspondiente.

5.3. NUBE DE TAGS:

Para poder elaborar la nube de tags son necesarios los siguientes pasos:

1.- Consultar a la base de datos para obtener una lista de objetos “Tema”. Cada página tiene su consulta específica para obtener esta información tal y como se ha explicado en el apartado de consultas.

2.- Existen dos posibilidades, que esta lista venga vacía, en cuyo caso este bloque mostrará el siguiente mensaje: “No existen temas relacionados con …” o por el contrario que traiga información, en cuyo caso haremos los cálculos necesarios para construir la nube de Tags.

3.- La lista contiene por cada elemento el número de veces que aparece, con esta información podemos calcular cuál es el que aparece el mayor número de veces, este resultado lo almacenaremos en la variable “max”.

4.- Ahora para cada elemento de la lista y gracias a la variable anterior calcularemos el porcentaje que le corresponde de la siguiente manera:

```
porcentaje=(nubeTag.get(i).getnumApariciones()/max)*100;
```

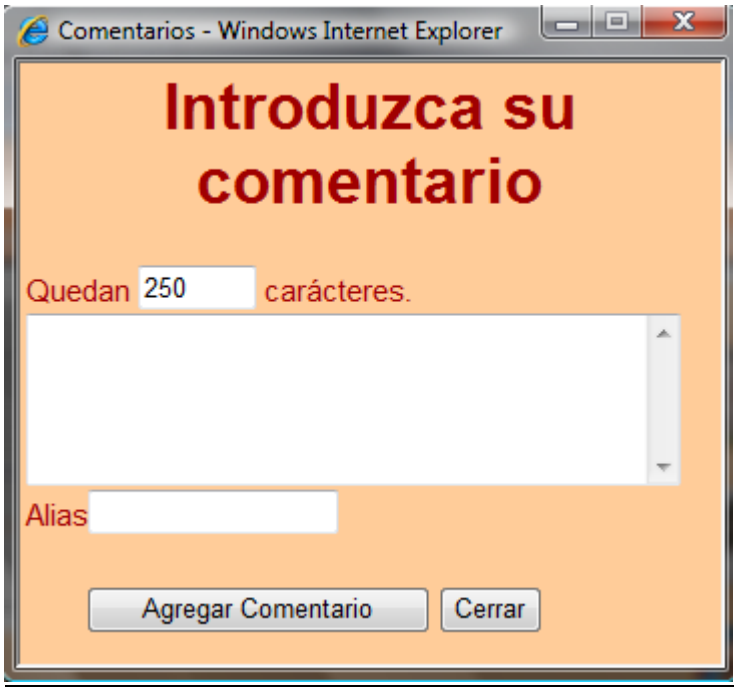
5.- Se han definido 4 estilos para crear la nube de tags, entre ellos varía el color y tamaño. Cuanto mayor sea el porcentaje de apariciones, más querremos destacar su importancia, por lo tanto su tamaño de letra será mayor. Estos 4 estilos serán asignados según al rango de porcentaje que pertenezcan, es decir, si el porcentaje está entre 0 y 25% se le asignará el estilo “tag1”, si se encuentra entre 25% y 50% le corresponderá el estilo “tag2”, entre 50% y 75% el “tag3” y entre 75% y 100% el “tag4”.

Cabe destacar que se ha definido un número de elementos máximos que formen parte de la nube para controlar su tamaño. También es importante saber que cada elemento de la nube corresponde a un enlace a la página de temas y como esta necesita información para representarse se ha guardado en sesión el nombre del tema para cada uno de los elementos, de esta forma la página de temas recuperará de sesión esta información y podrá obtener a través de la consulta correspondiente toda la información que necesite para representarse.

5.4. INTRODUCIR COMENTARIOS:

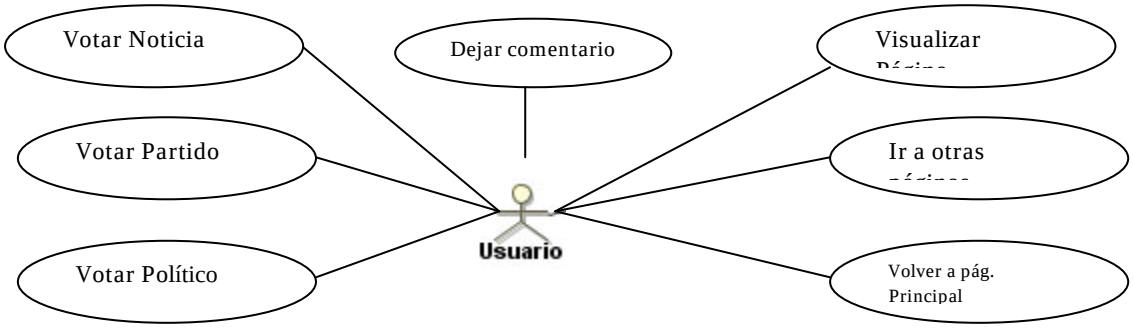
En todas las páginas existe un icono de comentarios (), al pinchar sobre este se abre la ventana de la figura y nos permite introducir el comentario que deseemos y nuestro alias. Al pulsar el botón “Agregar Comentario”, este comentario será dado de alta en la base de datos y la ventana será cerrada. Si por el contrario pulsamos el botón “Cerrar”, se cerrará la ventana sin realizar ninguna acción de alta.

Hemos realizado una función en javascript que nos permite mostrar al usuario el número de caracteres que en cada momento se le permiten introducir y además le hemos dado un máximo al textarea para asegurarnos que no pueda introducir más caracteres de los determinados, de esta forma controlamos que las características de tamaño del comentario coincidan y no produzcan errores cuando intentemos darlo de alta en la base de datos. También se han creado validaciones para los campos de ésta página, para no permitir introducir campos en blanco (vacíos) y ambos campos son obligatorios para poder dar de alta el comentario.



Cabe destacar que la figura anterior corresponde a la introducción de comentarios para la página principal, cuyo fondo está en correspondencia con ésta. Esta ventana está adaptada para adquirir el fondo correspondiente según la página que la solicite, para conseguir una armonía de colores entre ambas.

Casos de Uso Administración de páginas



6. PRUEBAS

Para visualizar la aplicación se puede hacer bien desde Eclipse o bien desde el explorador. Se ha creado una jsp de inicio que nos dirige a nuestra página principal.

Para arrancarla desde eclipse, basta con posicionarse sobre esta página, pulsar el botón derecho del ratón, seleccionar “Run as” -> “Run on Server”, se abrirá una ventana en la que bastará con pulsar el botón “Finish” y se arrancara nuestra aplicación mostrándonos la página principal de la aplicación.

Otra forma sería arrancar “Tomcat” y copiar la siguiente url, “http://localhost:8080/PFC/jsp/inicio.jsp”, en el explorador.

Cabe destacar que todas las pruebas que se describen a continuación, son necesarias para cada uno de los exploradores en los que deseamos que funcione correctamente nuestra aplicación. En nuestro caso se han utilizado “Mozilla Firefox” y “Internet Explorer”.

6.1.- Pruebas para la paginación:

Cada una de las pruebas que se describen en este punto, es necesario realizarlas para los 3 bloques que presentan paginación:

- Bloque inferior izquierdo o bloque principal. (**B1)**
- Bloque superior derecho (Personas Relacionadas). **(B2)**
- Bloque inferior derecho (Partidos Relacionados). **(B3)**

y en cada una de las páginas:

- Página Principal.
- Página de Temas.
- Página de Personas Relacionadas.
- Página de Partidos.

En teoría con probarlo para una de las páginas sería suficiente puesto que el mecanismo es idéntico para el resto, pero hemos querido asegurarnos que no ha habido ningún error en el desarrollo del código.

Prueba 1:

Comprobación para el correcto funcionamiento del icono de retroceso. Inicialmente se carga la primera página, comprobamos que al pulsar sobre éste se vuelva a cargar la primera página, tal y como se ha cargado la primera vez.

Lo siguiente que deberemos comprobar es que posicionándonos en otra página posterior al pulsar este icono retroceda a la página anterior, para ello deberá visualizar todos los registros recuperados y haber reducido en uno el número de página.

Prueba 2:

Comprobación para el correcto funcionamiento del icono de avance. Inicalmente se carga la primera página, al pulsar sobre éste deberá ir a la página siguiente incrementando en uno el número de página y mostrando los registros que correspondan. Si la siguiente página es la última deberá mostrar o bien el total de registros por página o bien sólo aquellos que haya recuperado, es decir, si el total de registros es 13 y se pagina de 5 en 5 la última página sólo deberá visualizar los últimos 3 registros.

Prueba 3:

Es importante considerar la última página como un caso particular de prueba, puesto que si nos encontramos en la última página y pulsamos siguiente debe volver a mostrar la última página que contenga información, es decir, la misma en la que estamos. Como se trucan ciertas variables para conseguir este objetivo es importante volver a pulsar el icono de avance para ver comprobar que se han vuelto a restablecer las variables y se vuelve de nuevo a recuperar la última página sin producir ningún tipo de error.

A continuación mostramos una tabla con los resultados obtenidos para la Página principal y comprobamos que el funcionamiento es correcto.

Prueba 4:

Otra prueba interesante es comprobar que al paginar uno de los bloques, el resto de bloques mantienen su paginación.

Por ejemplo: Si el bloque 1 (bloque principal) se encuentra en la página 2, y el bloque 2 (personas relacionadas) se encuentra en la página 4, al pulsar sobre cualquiera de los iconos de la paginación del bloque 3 (bloque de partidos relacionados), se actualiza la página pero se mantienen las páginas y la información idéntica en los otros bloques, de manera que solo se actualice la del bloque 3.

Esta prueba sería conveniente realizarla para cada uno de los bloques, comprobando que el estado de los otros se mantiene y sólo se actualiza el bloque sobre el que se está realizando la prueba.

	Reg. Recuperados de la BD.	**Reg. X Pág.**	**Reg. Última Página**
B1	13	5	3
B2	13	3	1
B3	3	2	1

	En última página pulso >	En primera página pulso <	Avance de páginas intermedias.	Retroceso de páginas intermedias.
B1	Vuelve a mostrar la última pág. (nº 3) y 3 registros.	Vuelve a mostrar la primera pág. (nº 1) y 5 registros.	Va incrementando correctamente el número de páginas y recupera correctamente los registros (5 por página)	Va decrementando correctamente el número de pág. Y muestra por página el número total de registros (5 por página).
B2	Muestra la última página nº 5 y con 1 registro.	Muestra la pág. nº 1 y 3 registros.	Incrementa y muestra los registros correctamente	Decrementa y muestra número de página y los registros correctamente
B3	Muestra la última página (la nº 2) con un solo registro.	Muestra la primera página con nº 1 y 2 registros.	Incrementa y muestra los registros correctamente	Decrementa el número de página y muestra los registros correctamente

6.2.- Pruebas en las votaciones:

Prueba 1:

- Bloque inferior izquierdo o bloque principal. (**B1).** Iconos: 

Al pulsar sobre el icono que se muestra anteriormente, debemos comprobar que el número de votos se ha incrementado en uno, y que el único dato que se ha actualizado al recargar la página haya sido este el resto de datos deben permanecer invariables. También es necesario comprobar que en la BD se ha actualizado este campo, puesto que la actualización de la página no vuelve a realizar una consulta, sino que actualiza según los datos que nos ha devuelto nuestra función y posteriormente modifica en la base de datos.

Prueba 2:

Debemos probarla para ambos bloques puesto que actualizan campos de la base de datos distintos.

- Bloque superior derecho (Personas Relacionadas). **(B2)** Iconos: 
- Bloque inferior derecho (Partidos Relacionados). **(B3)** Iconos: 

Al pulsar sobre estos iconos debemos comprobar que el número de votos ha aumentado en uno y que los campos correspondientes a votos en contra en la base de datos se han visto actualizados. El resto de datos de la página deben permanecer invariables, solo se debe observar que se actualiza este dato.

Prueba 3:

- Bloque superior derecho (Personas Relacionadas). **(B2)** Iconos:
- Bloque inferior derecho (Partidos Relacionados). **(B3)** Iconos:

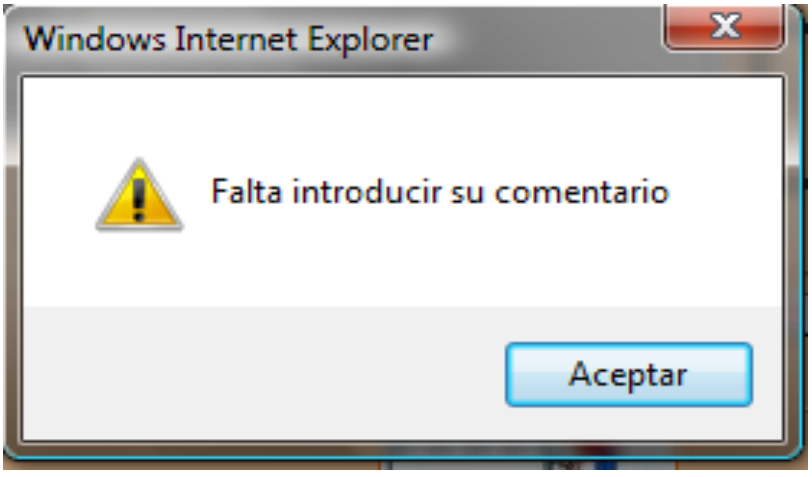
Idem a la prueba 2 con la excepción de que los campos que se deben haber visto modificados en la base de datos son los de votos a favor.

Con probarlo para una de las páginas es suficiente, puesto que estamos reutilizando código, y la funcionalidad y el código es el mismo para el resto de páginas. Siempre se utiliza la misma función que se encuentra en la hoja de javascript, por lo tanto con probarlo una vez sabremos que funciona para el resto de casos.

6.3.- Pruebas para la introducción de comentarios:

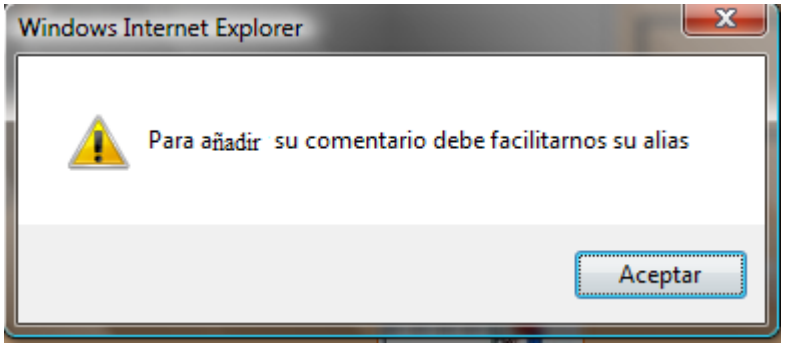
Prueba 1:

Pulsar sobre el icono comentarios, y al abrir la ventana dejar el campo comentarios vacío. Debe aparecer una validación que nos indique que este campo es obligatorio:



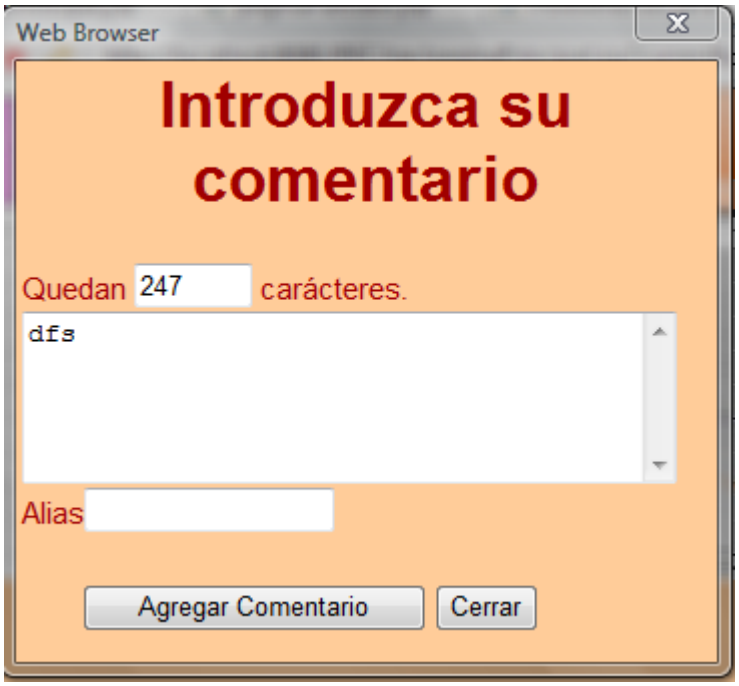
Prueba 2:

Pulsar sobre el icono comentarios, y al abrir la ventana dejar el campo alias vacío. Debe aparecer una validación que nos indique que este campo es obligatorio:



Prueba 3:

Comprobar que según se van introduciendo caracteres en el textarea, el número que se visualiza de caracteres que quedan es correcto. Como podemos observar en la figuara siguiente tenemos 250 caracteres disponibles, al introducir 3 quedan 247… es correcto.



Prueba 4:

Al pulsar el botón cerrar debe cerrar esta ventana, sin producir ningún cambio ni en la base de datos ni en la página sobre la que se ha abierto.

Prueba 5:

Al pulsar sobre el botón “Agregar Comentario” debe cerrar la ventana, actualizar en la página sobre la que se ha abierto el número de comentarios incrementándolo en uno y debemos comprobar que en la BD se ha actualizado el campo correspondiente.

6.4.- Pruebas para enlazar las páginas:

Un caso de prueba para cada una de las posibilidades de, desde una página enlazar con el resto.
1.- Pulsar sobre una de las personas relacionadas y que nos dirija a su página y muestre toda la información correctamente.

2.- Pulsar sobre uno de los temas relacionados y que nos dirija a su página mostrando toda la información correctamente.

3.- Pulsar sobre uno de los partidos relacionados y que cargue su página correctamente.

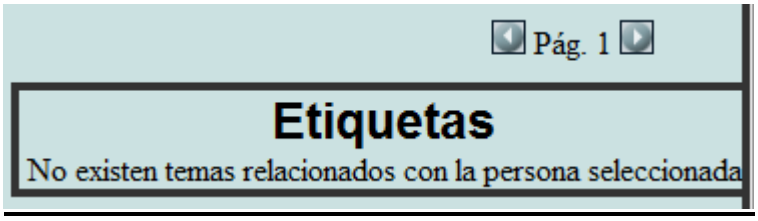
4.- Pulsar sobre el enlace que existe en cualquiera de las páginas a excepción de la página principal, hacia ésta y que se cargue correctamente.

6.5.- Si no recupera datos que muestre mensaje informativo:

Existirá un caso de prueba para cada uno de los bloques que recogen información.
1.- Bloque principal
2.- Bloque de personas relacionadas
3.- Bloque de la nube de tags.
4.- Bloque de partidos relacionados.

Con probarlo para una de las páginas es suficiente, porque el código se reutiliza para el resto de páginas.
Vamos a mostrar dos ejemplos, puesto que sería lo mismo para el resto de bloques:

1.- Cuando en la página de personas relacionadas, no existen temas relacionados con la persona o político en cuestión, se muestra un mensaje informativo sobre ello. A continuación se muestra el fragmento de la página del político en cuestión donde se visualiza dicho mensaje.



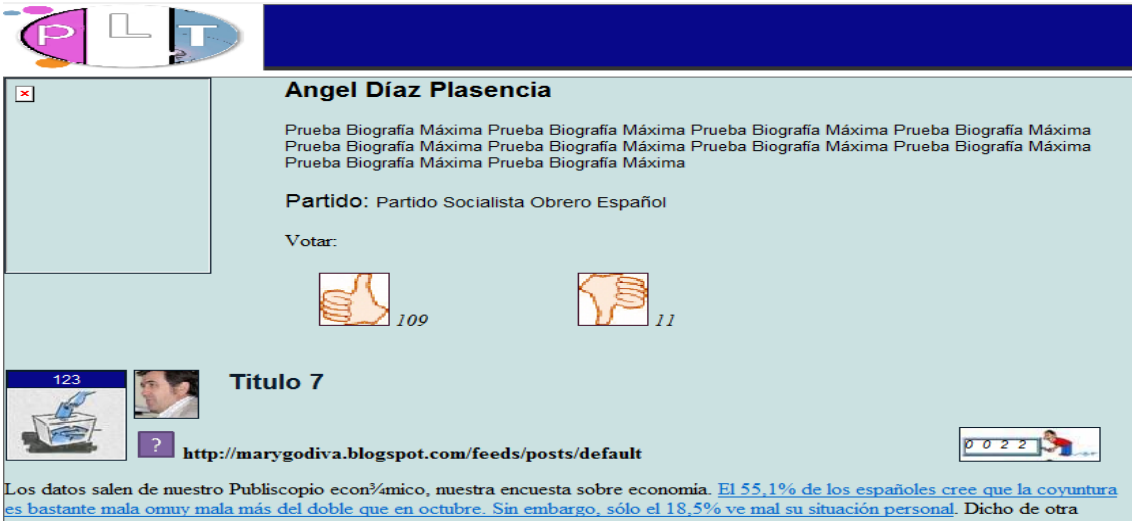
2.- El segundo ejemplo que mostramos a continuación se produce cuando no existen personas relacionadas, en la página de políticos. A continuación se muestra el fragmento de la página donde se visualiza dicha información.

6.6.- Comprobar la correcta visualización teniendo en cuenta las longitudes máximas y mínimas de los campos.

Todos los campos que se visualizan en cualquiera de las páginas tienen en nuestra base de datos una longitud máxima, incluso algunos no son obligatorios o pueden aparecer en blanco, por lo tanto es necesario probar que ocurriría para cada uno de los casos, para comprobar la estética de nuestra página.

Vamos a comentar una de las pruebas que hemos realizado para uno de ellos para que no resulte extremadamente denso y repetitivo el documento.

Por ejemplo: En la página de políticos existe en el bloque superior izquierdo un campo que visualiza la bibliografía de la persona en cuestión, probamos que en el caso de que este campo venga con su longitud máxima la visualización de la página sigue manteniendo su estética y se visualiza correctamente. A continuación mostramos un fragmento de la página donde se puede apreciar esta prueba:



7. SERVICIOS REST

La Transferencia de Estado Representacional (REST - Representational State Transfer) fue ganando amplia adopción en toda la web como una alternativa más simple a SOAP y a los servicios web basados en el Lenguaje de Descripción de Servicios Web (Web Services Descripcion Language - WSDL). Ya varios grandes proveedores de Web 2.0 están migrando a esta tecnología, incluyendo a Yahoo, Google y Facebook, quienes marcaron como obsoletos a sus servicios SOAP y WSDL y pasaron a usar un modelo más fácil de usar, orientado a los recursos.

Veamos los principios de REST para entender más esta tecnología.

7.1. Presentando REST

REST define un set de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios. De hecho, REST logró un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

7.2. Los 4 principios de REST

Una implementación concreta de un servicio web REST sigue cuatro principios de diseño fundamentales:

- utiliza los métodos HTTP de manera explícita
- no mantiene estado
- expone URIs con forma de directorios
- transfiere XML, JavaScript Object Notation (JSON), o ambos

A continuación vamos a ver en detalle estos cuatro principios, y explicaremos porqué son importantes a la hora de diseñar un servicio web REST.

7.3. REST utiliza los métodos HTTP de manera explícita

Una de las características claves de los servicios web REST es el uso explícito de los métodos HTTP, siguiendo el protocolo definido por RFC 2616. Por ejemplo, HTTP GET se define como un método productor de datos, cuyo uso está pensado para que las aplicaciones cliente obtengan recursos, busquen datos de un servidor web, o ejecuten una consulta esperando que el servidor web la realice y devuelva un conjunto de recursos.

REST hace que los desarrolladores usen los métodos HTTP explícitamente de manera que resulte consistente con la definición del protocolo. Este principio de diseño básico establece una asociación uno-a-uno entre las operaciones de crear, leer, actualizar y borrar y los métodos HTTP. De acuerdo a esta asociación:

- se usa POST para crear un recurso en el servidor
- se usa GET para obtener un recurso
- se usa PUT para cambiar el estado de un recurso o actualizarlo
- se usa DELETE para eliminar un recurso

Una falta de diseño poco afortunada que tienen muchas APIs web es el uso de métodos HTTP para otros propósitos. Por ejemplo, la petición del URI en un pedido HTTP GET, en general identifica a un recurso específico. O el string de consulta en el URI incluye un conjunto de parámetros que definen el criterio de búsqueda que usará el servidor para encontrar un conjunto de recursos. Al menos, así como el RFC HTTP/1.1 describe al GET.

Pero hay muchos casos de APIs web poco elegantes que usan el método HTTP GET para ejecutar algo transaccional en el servidor; por ejemplo, agregar registros a una base de datos. En estos casos, no se utiliza adecuadamente el URI de la petición HTTP, o al menos no se usa "a la manera REST". Si el API web utiliza GET para invocar un procedimiento remoto, seguramente se verá algo como esto:

GET /agregarusuario?nombre=Antonio HTTP/1.1

Este no es un diseño muy atractivo porque el método aquí arriba expone una operación que cambia estado sobre un método HTTP GET. Dicho de otra manera, la petición HTTP GET de aquí arriba tiene efectos secundarios. Si se procesa con éxito, el

resultado de la petición es agregar un usuario nuevo (en el ejemplo, Antonio) a la base de datos. El problema es básicamente semántico. Los servidores web están diseñados para responder a las peticiones HTTP GET con la búsqueda de recursos que concuerden con la ruta (o el criterio de búsqueda) en el URI de la petición, y devolver estos resultados o una representación de los mismos en la respuesta, y no añadir un registro a la base de datos. Desde el punto de vista del protocolo, y desde el punto de vista de servidor web compatible con HTTP/1.1, este uso del GET es inconsistente.

Más allá de la semántica, el otro problema con el GET es que al ejecutar eliminaciones, modificaciones o creación de registros en la base de datos, o al cambiar el estado de los recursos de cualquier manera, provoca que las herramientas de caché web y los motores de búsqueda (crawlers) puedan realizar cambios no intencionales en el servidor. Una forma simple de evitar este problema es mover los nombres y valores de los parámetros en la petición del URI a tags XML. Los tags resultantes, una representación en XML de la entidad a crear, pueden ser enviados en el cuerpo de un HTTP POST cuyo URI de petición es el padre de la entidad.

Antes:

```
GET /agregarusuario?nombre=Antonio HTTP/1.1
```

Después:

```
POST /usuarios HTTP/1.1
Host: miservidor
Content-type: application/xml
<usuario>
   <nombre>Antonio</nombre>
</usuario>
```

El método de arriba es un ejemplo de una petición REST: hay un uso correcto de HTTP POST y la inclusión de los datos en el cuerpo de la petición. Al recibir esta petición, la misma puede ser procesada de manera de agregar el recurso contenido en el cuerpo como un subordinado del recurso identificado en el URI de la petición; en este caso el nuevo recurso debería agregarse como hijo de /usuarios. Esta relación de contención entre la nueva entidad y su padre, como se indica en la petición del POST, es análoga a la forma en la que está subordinado un archivo a su directorio. El cliente indica esta

relación entre la entidad y su padre y define el nuevo URI de la entidad en la petición del POST.

Luego, una aplicación cliente puede obtener una representación del recurso usando la nueva URI, sabiendo que al menos lógicamente el recurso se ubica bajo /usuarios

```
GET /usuarios/Antonio HTTP/1.1
Host: miservidor
Accept: application/xml
```

Es explícito el uso del GET de esta manera, ya que el GET se usa solamente para recuperar datos. GET es una operación que no debe tener efectos secundarios, una propiedad también conocida como *idempotencia*.

Se debe hacer un refactor similar de un método web que realice actualizaciones a través del método HTTP GET. El siguiente método GET intenta cambiar la propipedad "nombre" de un recurso. Si bien se puede usar el string de consulta para esta operación, evidentemente no es el uso apropiado y tiende a ser problemático en operaciones más complejas. Ya que nuestro objetivo es hacer uso explícito de los métodos HTTP, un enfoque REST sería enviar un HTTP PUT para actualizar el recurso, en vez de usar HTTP GET.

Antes:

```
GET /actualizarusuario?nombre=Antonio&nuevoNombre=David HTTP/1.1
```

Después:

```
PUT /usuarios/Antonio HTTP/1.1
Host: miservidor
Content'Type: application/xml
<usuario>
   <nombre>Antonio</nombre>
</usuario>
```

Al usar el método PUT para reemplazar al recurso original se logra una interfaz más limpia que es consistente con los principios de REST y con la definición de los métodos

HTTP. La petición PUT que se muestra arriba es explícita en el sentido que apunta al recurso a ser actualiza identificándolo en el URI de la petición, y también transfiere una nueva representación del recurso del cliente hacia el servidor en el cuerpo de la petición PUT, en vez de transferir los atributos del recurso como un conjunto suelo de parámetros (nombre = valor) en el mismo URI de la petición.

El PUT arriba mostrado también tiene el efecto de renombrar al recurso Antonio a David, y al hacerlo cambia el URI a */usuarios/David*. En un servicio web REST, las peticiones siguientes al recurso que apunten a la URI anterior van a generar un error estándar "404 Not Found".

Como un principio de diseño general, ayuda seguir las reglas de REST que aconsejan usar sustantivos en vez de verbos en las URIs. En los servicios web REST, los verbos están claramente definidos por el mismo protocolo: POST, GET, PUT y DELETE. Idealmente, para mantener una interfaz general y para que los clientes puedan ser explícitos en las operaciones que invocan, los servicios web no deberían definir más verbos o procedimientos remotos, como ser */agregarusuario* y */actualizarusuario*. Este principio de diseño también aplica para el cuerpo de la petición HTTP, el cual debe usarse para transferir el estado de un recurso, y no para llevar el nombre de un método remoto a ser invocado.

7.4. REST no mantiene estado

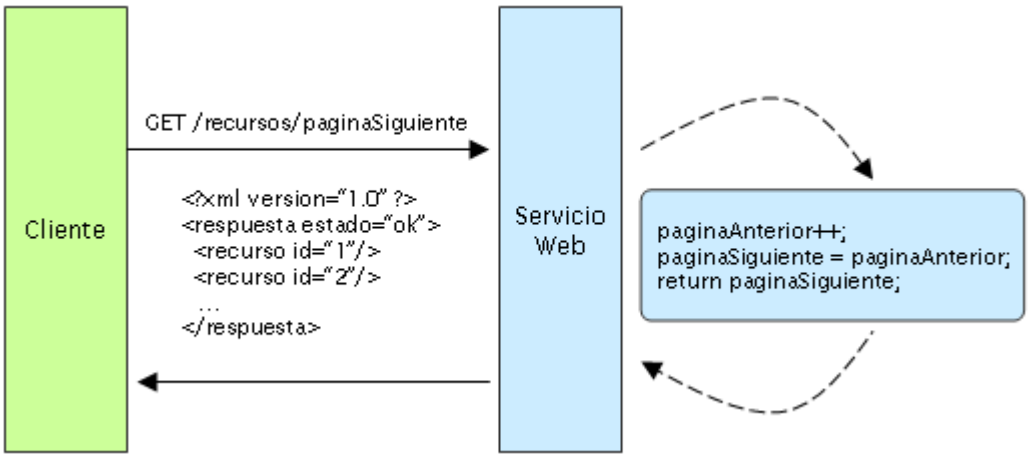
Los servicios web REST necesitan escalar para poder satisfacer una demanda en constante crecimiento. Se usan clusters de servidores con balanceadores de carga y alta disponibilidad, proxies, y gateways de manera de conformar una topología servicial, que permita transferir peticiones de un equipo a otro para disminuir el tiempo total de respuesta de una invocación al servicio web. El uso de servidores intermedios para mejorar la escalabilidad hace necesario que los clientes de servicios web REST envíen peticiones completas e independientes; es decir, se deben enviar peticiones que incluyan todos los datos necesarios para cumplir el pedido, de manera que los componentes en los servidores intermedios puedan redireccionar y gestionar la carga sin mantener el estado localmente entre las peticiones.

Una petición completa e independiente hace que el servidor no tenga que recuperar ninguna información de contexto o estado al procesar la petición. Una aplicación o

cliente de servicio web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta. De esta manera, el no mantener estado mejora el rendimiento de los servicios web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.

7.5. Servicios con estado vs. sin estado

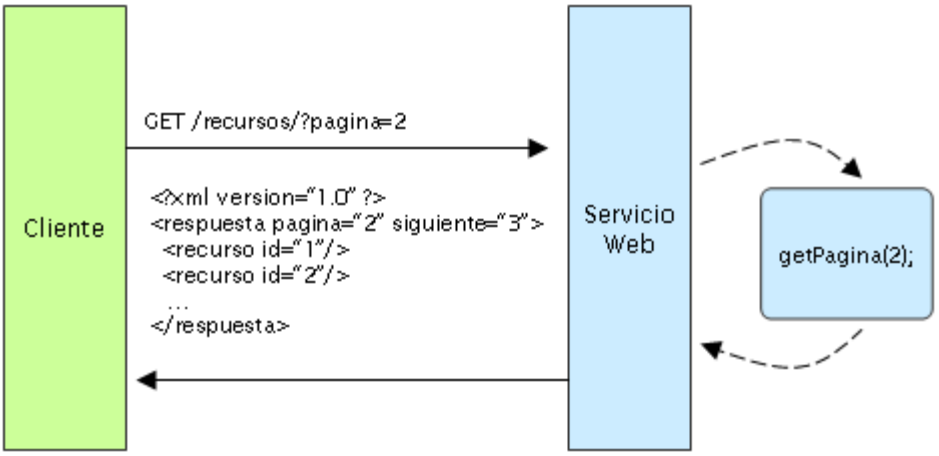
La siguiente ilustración nos muestra un **servicio con estado**, del cual una aplicación realiza peticiones para la página siguiente en un conjunto de resultados múlti-página, asumiendo que el servicio mantiene información sobre la última página que pidió el cliente. En un diseño con estado, el servicio incrementa y almacena en algún lugar una variable *paginaAnterior* para poder responder a las peticiones siguientes.



Los servicios con estado tienden a volverse complicados. En la plataforma Java Enterprise Edition (Java EE), un entorno de servicios con estado necesita bastante análisis y diseño desde el inicio para poder almacenar los datos eficientemente y poder sincronizar la sesión del cliente dentro de un cluster de servidores. En este tipo de ambientes, ocurre un problema que le resulta familiar a los desarrolladores de servlets/JSP y EJB, quienes a menudo tienen que revolver buscando la causa de una *java.io.NotSerializableException* cuando ocurre la replicación de una sesión. Puede ocurrir tanto sea en el contenedor de Servlets al intentar replicar la *HttpSession* o por el contenedor de EJB al replicar un EJB con estado; en todos los casos, es un problema

que puede costar mucho esfuerzo resolver, buscando el objeto que no implementa *Serializable* dentro de un grafo complejo de objetos que constituyen el estado del servidor. Además, la sincronización de sesiones es costosa en procesamiento, lo que impacta negativamente en el rendimiento general del servidor.

Por otro lado, los **servicios sin estado** son mucho más simples de diseñar, escribir y distribuir a través de múltiples servidores. Un servicio sin estado no sólo funciona mejor, sino que además mueve la responsabilidad de mantener el estado al cliente de la aplicación. En un servicio web REST, el servidor es responsable de generar las respuestas y proveer una interfaz que le permita al cliente mantener el estado de la aplicación por su cuenta. Por ejemplo, en el mismo ejemplo de una petición de datos en múltiples páginas, el cliente debería incluir el número de página a recuperar en vez de pedir "la siguiente", tal como se muestra en la siguiente figura:



Un servicio web sin estado genera una respuesta que se enlaza a la siguiente página del conjunto y le permite al cliente hacer todo lo que necesita para almacenar la página actual. Este aspecto del diseño de un servicio web REST puede descomponerse en dos conjuntos de responsabilidades, como una separación de alto nivel que clarifica cómo puede mantenerse un servicio sin estado.

7.6. Responsabilidad del servidor

- Genera respuestas que incluyen enlaces a otros recursos para permitirle a la aplicación navegar entre los recursos relacionados. Este tipo de respuestas tiene enlaces embebidos. De la misma manera, si la petición es hacia un padre o un recurso contenedor, entonces una respuesta REST típica debería también incluir

enlaces hacia los hijos del padre o los recursos subordinados, de manera que se mantengan conectados.

- Genera respuestas que indican si son susceptibles de caché o no, para mejorar el rendimiento al reducir la cantidad de peticiones para recursos duplicados, y para lograr eliminar algunas peticiones completamente. El servidor utiliza los atributos Cache-Control y Last-Modified de la cabecera en la respuesta HTTP para indicarlo.

7.7. Responsabilidades del cliente de la aplicación

- Utiliza el atributo Cache-Control del encabezado de la respuesta para determinar si debe cachear el recurso (es decir, hacer una copia local del mismo) o no. El cliente también lee el atributo Last-Modified y envia la fecha en el atributo If-Modified-Since del encabezado para preguntarle al servidor si el recurso cambió desde entonces. Esto se conoce como *GET Condicional*, y ambos encabezados van de la mano con la respuesta del servidor 304 (No Modificado) y se omite al recurso que se había solicitado si no hubo cambios desde esa fecha. Una respuesta HTTP 304 significa que el cliente puede seguir usando la copia local de manera segura, evitando así realizar las peticiones GET hasta tanto el recurso no cambie.
- Envía peticiones completas que pueden ser serviciadas en forma independiente a otras peticiones. Esto implica que el cliente hace uso completo de los encabezados HTTP tal como está especificado por la interfaz del servicio web, y envía las representaciones del recurso en el cuerpo de la petición. El cliente envía peticiones que hacen muy pocas presunciones sobre las peticiones anteriores, la existencia de una sesión en el servidor, la capacidad del servidor para agregarle contexto a una petición, o sobre el estado de la aplicación que se mantiene entre las peticiones.

Esta colaboración entre el cliente y el servicio es esencial para crear un servicio web REST sin estado. Mejora el rendimiento, ya que ahorro ancho de banda y minimiza el estado de la aplicación en el servidor.

7.8. REST expone URIs con forma de directorios

Desde el punto de vista del cliente de la aplicación que accede a un recurso, la URI determina qué tan intuitivo va a ser el web service REST, y si el servicio va a ser utilizado tal como fue pensado al momento de diseñarlo. La tercera característica de los servicios web REST es justamente sobre las URIs.

Las URI de los servicios web REST deben ser intuitivas, hasta el punto de que sea fácil adivinarlas. Pensemos en las URI como una interfaz auto-documentada que necesita de muy poca o ninguna explicación o referencia para que un desarrollador pueda comprender a lo que apunta, y a los recursos derivados relacionados.

Una forma de lograr este nivel de usabilidad es definir URIs con una estructura al estilo de los directorios. Este tipo de URIs es jerárquica, con una única ruta raíz, y va abriendo ramas a través de las subrutas para exponer las áreas principales del servicio. De acuerdo a esta definición, una URI no es solamente una cadena de caracteres delimitada por barras, sino más bien un árbol con subordinados y padres organizados como nodos. Por ejemplo, en un servicio de hilos de discusiones que tiene temas varios, se podría definir una estructura de URIs como esta:

http://www.miservicio.org/discusion/temas/{tema}

La raiz, */discusion*, tiene un nodo */temas* como hijo. Bajo este nodo hay un conjunto de nombres de temas (como ser tecnología, actualidad, y más), cada uno de los cuales apunta a un hilo de discusión. Dentro de esta estructura, resulta fácil recuperar hilos de discusión al tipear algo después de */temas/*.

En algunos casos, la ruta a un recurso encaja muy bien dentro de la idea de "estructura de directorios". Por ejemplo, tomemos algunos recursos organizados por fecha, que son muy prácticos de organizar usando una sintaxis jerárquica.

El siguiente ejemplo es intuitivo porque está basado en reglas:

http://www.miservicio.org/discusion/2008/12/23/{tema}

El primer fragmento de la ruta es un año de cuatro dígitos, el segundo fragmento es el mes de dos dígitos, y el tercer fragmento es el día de dos dígitos. Puede resultar un poco tonto explicarlo de esta manera, pero es justamente el nivel de simpleza que buscamos. Tanto humanos como máquinas pueden generar estas estructuras de URI porque están

basadas en reglas. Como vemos, es fácil llenar las partes de esta URI, ya que existe un patrón para crearlas:

http://www.miservicio.org/discusion/{año}/{mes}/{dia}/{tema}

Podemos también enumerar algunas guías generales más al momento de crear URIs para un servicio web REST:

- ocultar la tecnología usada en el servidor que aparecería como extensión de archivos (.jsp, .php, .asp), de manera de poder portar la solución a otra tecnología sin cambiar las URI.
- mantener todo en minúsculas.
- sustituir los espacios con guiones o guiones bajos (uno u otro).
- evitar el uso de strings de consulta.
- en vez de usar un 404 Not Found si la petición es una URI parcial, devolver una página o un recurso predeterminado como respuesta.

Las URI deberían ser estáticas de manera que cuando cambie el recurso o cambie la implementación del servicio, el enlace se mantenga igual. Esto permite que el cliente pueda generar "favoritos" o bookmarks. También es importante que la relación entre los recursos que está explícita en las URI se mantenga independiente de las relaciones que existen en el medio de almacenamiento del recurso.

7.9. REST transfiere XML, JSON, o ambos

La representación de un recurso en general refleja el estado actual del mismo y sus atributos al momento en que el cliente de la aplicación realiza la petición. La representación del recurso son simples "fotos" en el tiempo. Esto podría ser una representación de un registro de la base de datos que consiste en la asociación entre columnas y tags XML, donde los valores de los elementos en el XML contienen los valores de las filas. O, si el sistema tiene un modelo de datos, la representación de un recurso es una fotografía de los atributos de una de las cosas en el modelo de datos del sistema. Estas son las cosas que serviciamos con servicios web REST.

La última restricción al momento de diseñar un servicio web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las

peticiones/respuestas. Aquí es donde realmente vale la pena mantener las cosas simples, legibles por humanos, y conectadas.

Los objetos del modelo de datos generalmente se relacionan de alguna manera, y las relaciones entre los objetos del modelo de datos (los recursos) deben reflejarse en la forma en la que se representan al momento de transferir los datos al cliente. En el servicio de hilos de discusión anterior, un ejemplo de una representación de un recurso conectado podría ser un tema de discusión raíz con todos sus atributos, y links embebidos a las respuestas al tema.

```
<discusion fecha="{fecha}" tema="{tema}">
   <comentario>{comentario}</comentario>
   <respuestas>
        <respuesta de="
gaz@mail.com " href="/discusion/temas/{tema}/gaz"/>
        <respuesta de="
gir@mail.com " href="/discusion/temas/{tema}/gir"/>
   </respuestas>
</discusion>
```

Por último, es bueno construir los servicios de manera que usen el atributo HTTP Accept del encabezado, en donde el valor de este campo es ul tipo MIME. De esta manera, los clientes pueden pedir por un contenido en particular que mejor pueden analizar. Algunos de los tipos MIME más usados para los servicios web REST son:

MIME-Type	**Content-Type**
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

Esto permite que el servicio sea utilizado por distintos clientes escritos en diferentes lenguajes, corriendo en diversas plataformas y dispositivos. El uso de los tipos MIME y del encabezado HTTP Acepto es un mecanismo conocido como *negociación de*

contenido, el cual le permite a los clientes elegir qué formato de datos puedan leer, y minimiza el acoplamiento de datos entre el servicio y las aplicaciones que lo consumen.

7.10. Conclusión

No siempre REST es la mejor opción. Está surgiendo como una alternativa para diseñar servicios web con menos dependencia en middleware propietario (por ejemplo, un servidor de aplicaciones), que su contraparte SOAP y los servicios basados en WSDL.

De algún modo, REST es la vuelta a la Web antes de la aparición de los grandes servidores de aplicaciones, ya que hace énfasis en los primeros estándares de Internet, URI y HTTP. XML sobre HTTP es una interfaz muy poderosa que permite que aplicaciones internas, como interfaces basadas en JavaScript Asincrónico + XML (AJAX) puedan conectarse, ubicar y consumir recursos. De hecho, es justamente esta gran combinación con AJAX que generó esta gran atención que tiene REST hoy en día.

Resulta muy flexible el poder exponer los recursos del sistema con un API REST, de manera de brindar datos a distintas aplicaciones, formateados en distintas maneras. REST ayuda a cumplir con los requerimientos de integración que son críticos para construir sistemas en donde los datos tienen que poder combinarse fácilmente y extenderse. Desde este punto de vista, los servicios REST se convierten en algo mucho más grande.

Actualmente no se está utilizando el servicio REST en nuestro prototipo, pero a continuación mostramos la utilidad que tendría para futuras funcionalidades:

7.11. Utilidad de REST en futuras funcionalidades.

REST, puede ayudarnos en las siguientes funcionalidades de la siguiente manera:

Por ejemplo, ya que cuando se dispone de distintos tipos de usuarios es necesario informar a la siguiente página que tipo de usuario está intentando acceder. Para ello nos es útil este servicio puesto que usa métodos HTTP de manera explícita, que ventajas nos aporta esto, pues que por ejemplo si tenemos la información o datos de usuario que se ha conectado como su password u otro tipo de datos de carácter personal no se visualizan en la url ya que estará contenidos en el documento u objeto especificado tal y como se ha explicado en el punto 7.3.

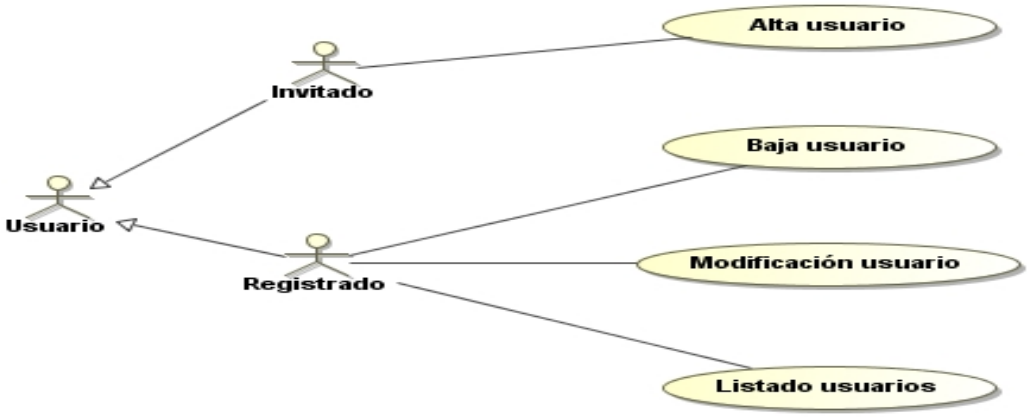
Además como REST no mantiene el estado habría que adaptar la paginación y sistema de votación, para que en lugar de esperar que la siguiente página obtenga el dato, se lo pasemos directamente cuando la solicitamos. Por este motivo nos servirá de gran ayuda también para los accesos a la base de datos para alta, modificación, baja puesto que al no mantener estado no será necesario realizar estas operaciones en la siguiente página sino que se harán a tiempo real en la misma página de manera que si perdemos la sesión la base de datos ya quedará modificada y no esperara respuesta de la siguiente página.

Modelado de usuarios

Los roles Administrador y Usuario son globales. De momento no se han definido roles para un usuario, al ser un prototipo de interfaz, hemos tratado de manera global este concepto, en un futuro debería tenerse en cuenta si es un usuario desconocido o se ha logueado.

Casos de Uso Administración de usuarios

De momento no se ha desarrollado esta funcionalidad, ya que en este proyecto presentamos el prototipo de la interfaz, pero si es necesario tenerlo en cuenta para el futuro de ésta.



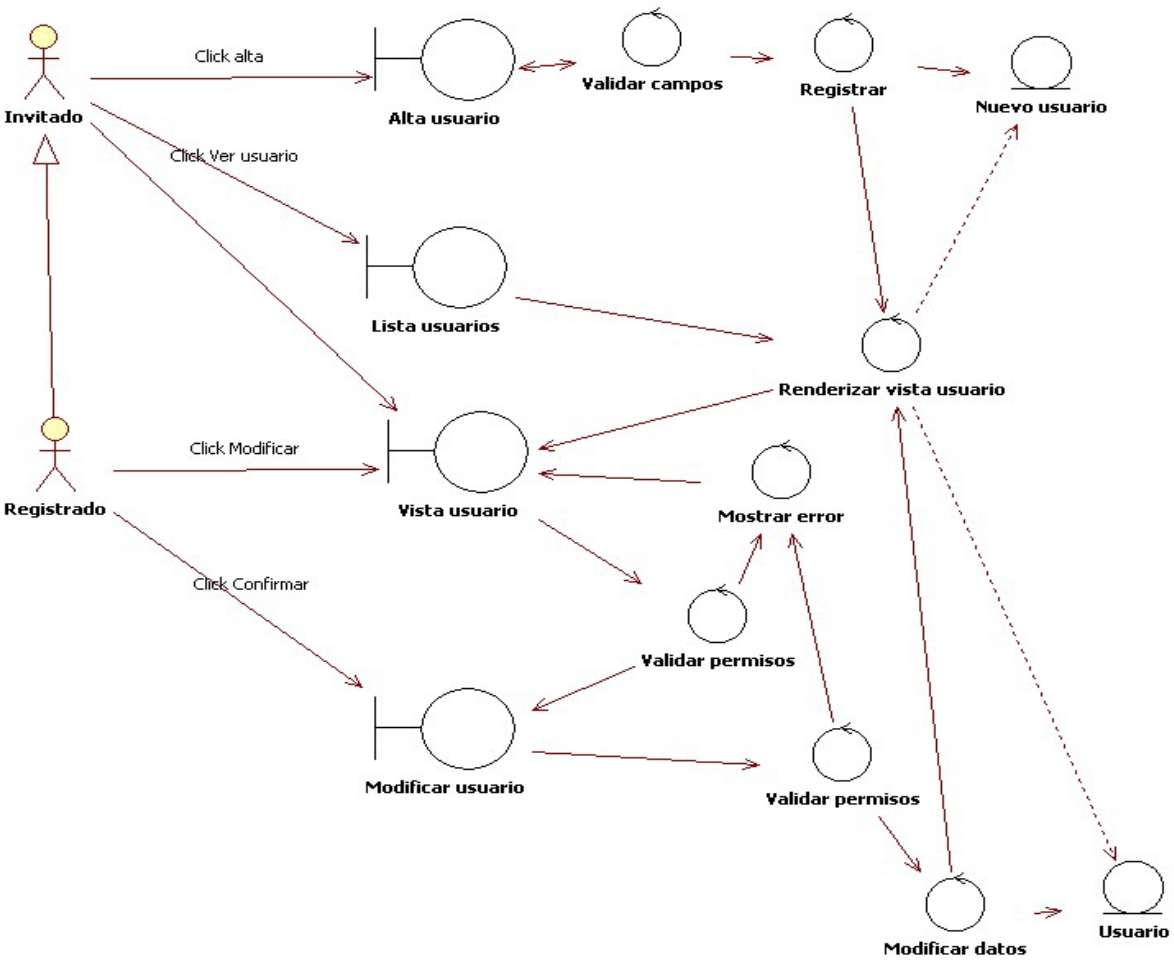
Administrador es una especialización de registrado. Como sus casos de uso son los mismos, se ha obviado.

Tanto modificar usuario, como baja usuario tienen como precondiciones que el usuario sea el mismo o un administrador.

En caso de que no se cumpla alguna de estas dos condiciones, se generará una excepción.

Modelo análisis usuarios

Como se ha comentado en líneas anteriores, esto es un modelo a tener en cuenta para un futuro.



8. PRESUPUESTO

A continuación se siguen las reglas de actuación para medir el tamaño del proyecto según el método de puntos de función.

En el campo de los sistemas de información, como en otros campos, es necesario medir para conseguir un mejor conocimiento, tanto del producto como del proceso que se desarrolla para obtener el producto.

Mediante dicho conocimiento se puede incrementar la calidad del producto software y se pueden introducir mejoras continuas en el proceso de su desarrollo.

La medida más común del producto software es la que hace referencia a su tamaño.

Las medidas más comunes que se aplican al proceso de desarrollo son el esfuerzo realizado y el coste.

8.1. MÉTODO DE LOS PUNTOS DE FUNCIÓN

El Método de los Puntos Función se ha convertido en un estándar aceptado para la medición del tamaño del software de una aplicación. Este método se basa en la teoría de que existe una relación directa entre las funciones del negocio, entendidas como conjunto de datos y procesos descritos por el usuario, y el tamaño de la aplicación o producto software. En otras palabras, a más funciones del negocio, más tamaño del producto. Es, por tanto, una medida del software independiente de la tecnología utilizada para su desarrollo e implementación.

Se debe determinar el alcance de la medición y los límites del proyecto o aplicación a ser medido.

Después, se estudian las especificaciones funcionales del negocio proporcionadas por el usuario, agrupándolas en funciones de datos y funciones transaccionales.

Las funciones de datos se corresponden con agrupaciones lógicas de datos. Se dividen en :

1.- Fichero Interno (Internal Logical File, de aquí en adelante ILF): grupos lógicos de datos mantenidos en el interior de la aplicación.

2.- Fichero Externo (External Interface File, de aquí en adelante EIF): grupos lógicos de datos referenciados por la aplicación, pero mantenidos por otras aplicaciones.

Las funciones transaccionales se refieren a los procesos elementales del negocio que mantienen o consultan las funciones de datos. Se dividen en :

1.- Entrada (External Input, de aquí en adelante EI).

2.- Salida (External Output, de aquí en adelante EO).

3.- Consulta (External Inquiry, de aquí en adelante EQ).

La identificación de las funciones es la parte más delicada del método.

Una vez identificadas las funciones, se procede a asignar a cada una de ellas su tamaño en puntos función. Para ello, es necesario realizar las siguientes actividades:

1.- Determinar la complejidad funcional de cada función, que uede tomar los siguientes valores: Baja, Media o Alta.

2.- Asignar a cada función un valor en puntos función en base a su tipo y complejidad. Dicho valor recibe el nombre de medida no ajustada de la función.

Existen importantes características funcionales y consideraciones de diseño lógico que no están suficientemente representadas por la medida no ajustada. La forma como se han tenido en cuenta todos estos aspectos es mediante un factor de ajuste, cuyo valor, comprendido entre 0,65 y 1,35, se obtiene rellenando un cuestionario formado por 14 preguntas, cada una de ellas relativa a una característica general de la aplicación.

La medida ajustada o simplemente medida de la aplicación se obtiene multiplicando la medida no ajustada por el factor de ajuste. Puesto que el factor de ajuste puede variar entre el margen especificado anteriormente, el valor de la medida ajustada puede ser hasta un 35% menor o mayor que el de la medida no ajustada.

8.1.1. ¿QUÉ SON LOS PUNTOS DE FUNCIÓN?

Es una métrica que permite traducir en un número el tamaño de la funcionalidad que brinda un producto de software desde el punto de vista del usuario, a través de una suma ponderada de las características del producto.

Componentes:

EI : Procesos en los que se introducen datos y que suponen la actualización de cualquier archivo interno.

EO: Procesos en los que se envía datos al exterior de la aplicación.

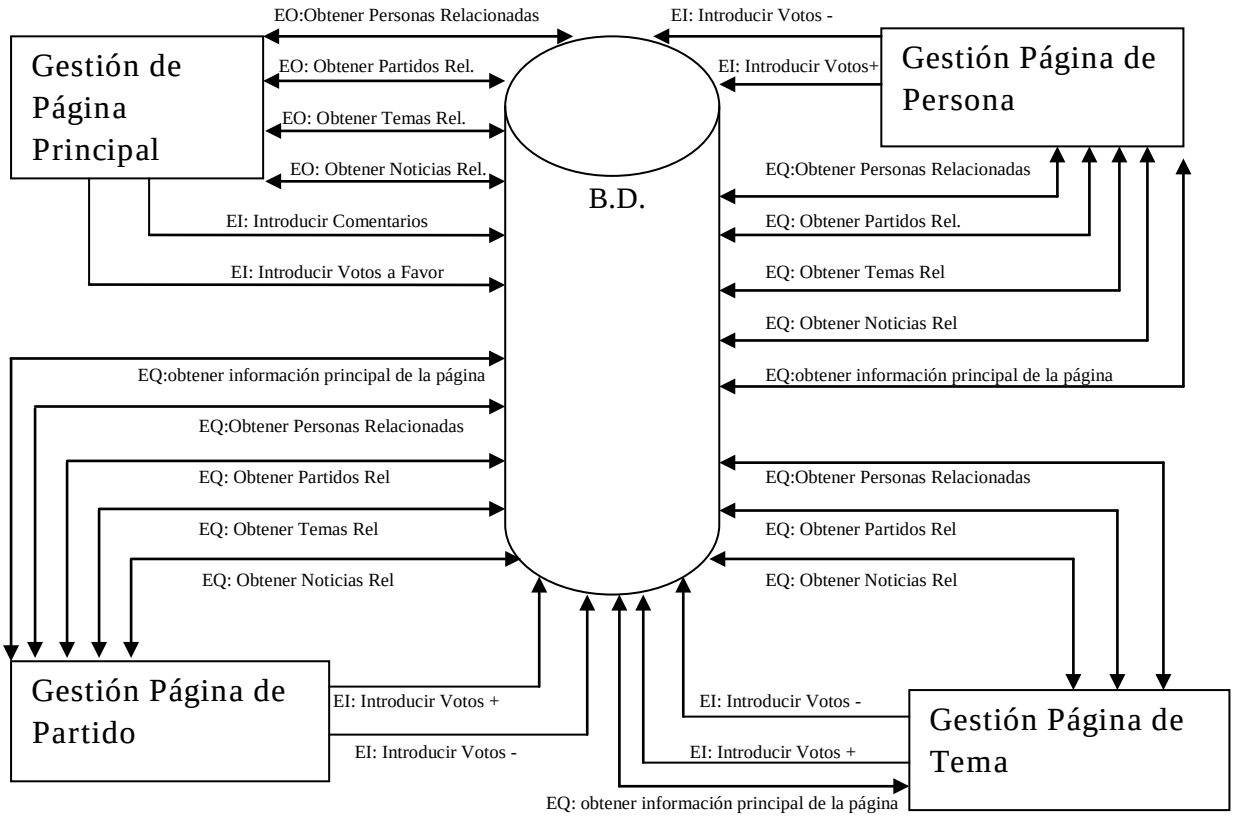
EQ: Procesos consistentes en la combinación de una entrada y una salida, en el que la entrada no produce ningún cambio en ningún archivo y la salida no contiene información derivada.

ILF: Grupos de datos relacionados entre sí internos al sistema.

EIF: Grupos de datos que se mantienen externamente.

8.1.2. PROCEDIMIENTO DE ESTIMACIÓN DE LOS PUNTOS DE FUNCIÓN.

8.1.2.1. OBTENER INFORMACIÓN DEL SISTEMA



Cabe destacar que en la figura aparecen dos entradas por cada página, como se verá más adelante si la funcionalidad de cada una de las entradas es la misma y afecta a los mismos campos se considerará como una única entrada.

8.1.2.2. IDENTIFICAR LOS COMPONENTES DEL SISTEMA

Componentes	Entrada	Salida	Relaciones
EO: Recuperar información sobre personas relacionadas en la página principal		Nombre, apellidos, foto, voto positivo, voto negativo y biografía.	Habla_de_b, persona, referenciado, noticia y post.
EO: Recuperar información sobre partidos relacionados en la página principal		Nombre, logotipo, siglas, voto_positivo y voto_negativo	Noticia, post, habla_de_a y partido.
EO: Recuperar información sobre temas relacionadas en la página principal		Nombre, count y permalink.	Habla_de_c y post
EO: Recuperar información sobre noticias relacionadas en la página principal		Id_post, cod_post, permalink, titulo, texto, autor, fecha (post), fecha (noticia), logo, url, entradas_crawler, num_comentarios, votos_positivos (noticia), foto, descripción, título (canal), tendencia.	Post, noticia, canal y tema.
EQ: Recuperar información principal sobre la página de una Persona.	Nombre y apellidos	Nombre, apellidos, foto, biografía, cargo, voto_positivo, voto_negativo.	Persona, Referenciado y pertenece.
EQ: Recuperar información sobre personas relacionadas en la página sobre una Persona.	Nombre, apellidos y post.	Nombre, apellidos, foto, voto positivo, voto negativo y biografía.	Habla_de_b, persona, referenciado y post.
EQ: Recuperar información sobre partidos relacionados en la página sobre una Persona.	Post	Nombre, logotipo, siglas, voto_positivo y voto_negativo	Habla_de_a y partido.
EQ: Recuperar información sobre temas relacionadas en la página sobre una Persona.	Nombre y apellidos	Nombre, count y permalink.	Habla_de_c, habla_de_b y post
EQ: Recuperar información sobre noticias relacionadas en la página sobre una Persona.	Nombre y apellidos	Id_post, cod_post, permalink, titulo, texto, autor, fecha (post), fecha (noticia), logo, url, entradas_crawler, num_comentarios, votos_positivos (noticia), foto, descripción, título (canal), tendencia.	Post, noticia, habla_de_b, canal y tema.

EQ: Recuperar información principal sobre la página de una Partido.	Nombre	Todos los campos del fichero partido.	Partido
EQ: Recuperar información sobre personas relacionadas en la página sobre una Partido.	Post	Nombre, apellidos, foto, voto positivo, voto negativo y biografía.	Habla_de_b, persona y referenciado.

EQ: Recuperar información sobre partidos relacionados en la página sobre una Partido.	Nombre y post	Nombre, logotipo, siglas, voto_positivo y voto_negativo	Habla_de_a y partido.
EQ: Recuperar información sobre temas relacionadas en la página sobre una Partido.	Nombre	Nombre, count y permalink.	Habla_de_c, habla_de_a y post
EQ: Recuperar información sobre noticias relacionadas en la página sobre una Partido .	Nombre	Id_post, cod_post, permalink, titulo, texto, autor, fecha (post), fecha (noticia), logo, url, entradas_crawler, num_comentarios, votos_positivos (noticia), foto, descripción, título (canal), tendencia.	Post, noticia, habla_de_a, canal y tema.
EQ: Recuperar información principal sobre la página de un Tema.	Nombre	Foto	Tema y habla_de_c
EQ: Recuperar información sobre personas relacionadas en la página sobre un Tema.	Nombre	Nombre, apellidos, foto, voto positivo, voto negativo y biografía.	Habla_de_b, persona, referenciado, habla_de_c y post.
EQ: Recuperar información sobre partidos relacionados en la página sobre un Tema.	Nombre	Nombre, logotipo, siglas, voto_positivo y voto_negativo	Habla_de_c, post, habla_de_a y partido.
EQ: Recuperar información sobre noticias relacionadas en la página sobre un Tema .	Nombre	Id_post, cod_post, permalink, titulo, texto, autor, fecha (post), fecha (noticia), logo, url, entradas_crawler, num_comentarios, votos_positivos (noticia), foto, descripción, título (canal), tendencia.	Post, noticia, habla_de_c, canal y tema.
EI: Introducir Comentario sobre la noticia en Página Principal.	votosActuales, id_Post, comentario, alias y fecha.	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flecha unidireccional.	Post y Comentario.
EI: Introducir votos a favor sobre un partido (En cualquiera de las páginas)	VotosActuales y Nombre del Partido	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flecha unidireccional.	Partido
EI: Introducir votos en contra sobre un partido (En cualquiera de las páginas)	VotosActuales y Nombre del Partido	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flecha unidireccional.	Partido

EI: Introducir votos a favor sobre una Persona(En cualquiera de las páginas)	VotosActuales y Nombre y apellidos.	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flecha unidireccional.	Persona
EI: Introducir votos en contra sobre una Persona(En cualquiera de las páginas)	VotosActuales y Nombre y apellidos.	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flecha unidireccional.	Persona
EI: Introducir votos a favor sobre una Noticia(En cualquiera de las páginas)	votosActuales e idPost.	Una entrada no produce salida, por eso en el esquema se indica este hecho con una flechaunidireccional.	Post

8.1.2.3 CALCULAR NÚMERO DE ELEMENTOS Y SU COMPLEJIDAD

En el anexo se facilitan las tablas de ponderaciones que asignan la complejidad (A=ALTA, M= MEDIA, B=BAJA) según el tipo de elemento (EI, EO, EQ) , DETs (número de atributos) y FTRS (número de ficheros).

Componentes	DETs	FTRs	COMPLEJIDAD
EO: Recuperar información sobre personas relacionadas en la página principal	6	5	A
EO: Recuperar información sobre partidos relacionados en la página principal	5	4	M
EO: Recuperar información sobre temas relacionadas en la página principal	3	2	B
EO: Recuperar información sobre noticias relacionadas en la página principal	17	4	A
EQ: Recuperar información principal sobre la página de una Persona.	7+2=9	3	M
EQ: Recuperar información sobre personas relacionadas en la página sobre una Persona.	6+3=9	4	A
EQ: Recuperar información sobre partidos relacionados en la página sobre una Persona.	5+1=6	2	M
EQ: Recuperar información sobre temas relacionadas en la página sobre una Persona.	3+2=5	3	B
EQ: Recuperar información sobre noticias relacionadas en la página sobre una Persona.	17+2=19	5	A
EQ: Recuperar información principal sobre la página de una Partido.	1+9=10	1	B
EQ: Recuperar información sobre personas relacionadas en la página sobre una Partido.	6+1=7	3	M
EQ: Recuperar información sobre partidos relacionados en la página sobre una Partido.	5+2=7	2	M
EQ: Recuperar información sobre temas relacionadas en la página sobre una Partido.	3+1=4	3	B
EQ: Recuperar información sobre noticias relacionadas en la página sobre una Partido .	17+1=18	5	A
EQ: Recuperar información principal sobre la página de un Tema.	1+1=2	2	B
EQ: Recuperar información sobre personas relacionadas en la página sobre un Tema.	6+1=7	4	A
EQ: Recuperar información sobre partidos relacionados en la página sobre un Tema.	5+1=6	4	A

EQ: Recuperar información sobre noticias relacionadas en la página sobre un Tema .	17+1=18	5	A
EI: Introducir Comentario sobre la noticia en Página Principal.	5	2	M
EI: Introducir votos a favor sobre un partido (En cualquiera de las páginas)	2	1	B
EI: Introducir votos en contra sobre un partido (En cualquiera de las páginas)	2	1	B
EI: Introducir votos a favor sobre una Persona(En cualquiera de las páginas)	3	1	B
EI: Introducir votos en contra sobre una Persona(En cualquiera de las páginas)	3	1	B

EI: Introducir votos a favor sobre una Noticia(En cualquiera de las páginas)	2	1	B

Componentes	**DET**	**RET**	**COMPLEJIDAD**
ILF: Habla_de_b	4	1	B
ILF: Habla_de_a	3	1	B
ILF: Habla_de_c	2	1	B
ILF: Referenciado	4	1	B
ILF: Persona	5	1	B
ILF: Partido	9	1	B
ILF: Canal	16	1	B
ILF: Post	12	1	B
ILF: Noticia	4	1	B
ILF: Tema	2	1	B
ILF: Comentario	4	1	B

8.1.2.4. OBTENER LOS PF SIN AJUSTAR (PFSA)

Por tanto los PFSA (Puntos de Función Sin Ajustar) se calculan como la suma de los productos de cada componente por su peso determinado en la tabla correspondiente.

PFSA = PFTe + PFTo + PFTq + PFTif + PFTef

Componente	Bajo	Medio	Alto	Total
EI	5 * 3 = 15	1 * 4 = 4	0 * 6 = 0	PFTe = 19
EO	1 * 4 = 4	1 * 5 = 5	2 * 7 = 14	PFTo = 23

EQ	4 * 3 = 12	4 * 4 = 16	6 * 6 = 36	PFTq = 64
ILF	11 * 7 = 77	0 * 10 = 0	0 * 15 = 0	PFTif = 77
EIF	0 * 5 = 0	0 * 7 = 0	0 * 10 = 0	PFTef = 0
				PFSA = 183

8.1.2.5. OBTENER LOS PF AJUSTADOS (PFA)

Nº DE FACTOR	FACTOR DE AJUSTE	VALOR 0..5
1	Comunicación de Datos	4
2	Proceso Distribuido	0
3	Rendimiento	3
4	Uso de la Configuración	2
5	Número de Transacciones	4
6	Entrada de Datos Online	5
7	Eficiencia con el Usuario Final	2
8	Actualización Online	3
9	Lógica de Proceso Interno Compleja	1
10	Reutilización	4
11	Facilidad de Instalación	1
12	Facilidad de Operación	5
13	Instalaciones Múltiples	4
14	Facilidad de Cambios	4
	Ajuste Complejidad Técnica (ACT)	**42**

VAF = 0,65 + (0,01*ACT) = 1,07

PFA = VAF * PFSA = 1,07 *183 = **195,81**

8.1.2.6. CÁLCULO DEL ESFUERZO

Esfuerzo horas/persona = PFA / [1/8 persona/hora] = 195,81 / 0,125 = 1566,48

8.1.2.7. CÁLCULO DE LA DURACIÓN DEL PROYECTO

Duración del proyecto en horas = 1566,48 horas/persona / 4 personas = 391,62 horas x miembro

Duración en meses = 391,62 / 100 horas/mes=3,9162 = 3 meses y 19 días

Nota: horas/mes productivas estimadas en el proyecto, calculadas de 20 días laborables y 5 horas productivas estimadas de las 8 de jornada laboral normal diaria

8.1.2.8. CÁLCULO DEL PRESUPUESTO DEL PROYECTO

Costo Total del Proyecto = [(Sueldo del Jefe de Proyecto + Sueldo del Analista+ Sueldo del Programador + Sueldo del Diseñador Gráfico) * Meses de Duración]+ Otros costos necesarios durante la realización del proyecto.

Para el cálculo del presupuesto ya que tenemos los costes por hora de cada miembro, sumaremos estos costes y los multiplicaremos por las horas de duración.

Costo Total del Proyecto = [(44 euros/hora + 36 euros/hora + 28 euros/hora + 28 euros/hora) * 391,62 horas de duración] = 136 euros/hora * 391,62 horas = **53260,32 euros**

9. LÍNEAS FUTURAS

9.1 Arquitectura lógica

La arquitectura lógica de la aplicación sigue el patrón **Modelo Vista Controlador (MVC)**. Éste separa los datos de una aplicación, la interfaz de usuario y la lógica de control o negocio en tres componentes distintos:

- **Modelo**: es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de éstos. El SGBD para gestionar los datos corresponde a este componente.
- **Vista**: presenta el modelo en un formato adecuado para interactuar, normalmente la interfaz de usuario. La vista estará formada por un conjunto de páginas webs que facilitarán al usuario la interacción con el Sistema.
- **Controlador**: responde a eventos, normalmente acciones del usuario que provocan cambios en el modelo y probablemente en la vista.

El patrón MVC se utiliza muy a menudo en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el SGBD y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Según este modelo nuestro prototipo podría mejorar si estructuramos un poco mejor la implementación, es decir, actualmente nuestras JSP están realizando tanto la funcionalidad de vista como la de controlador. Sería conveniente crearnos un controlador, que se encargase de evaluar y decidir qué vista es la que debe mostrarse en cada momento y que este gestione todos los cálculos y acciones del usuario, dejando nuestras JSP mucho más limpias y claras. Consiguiendo así implementar totalmente este modelo.

9.2. Control de versiones:

Sería interesante utilizar un servidor para el control de versiones. Este servidor ofrece la posibilidad de almacenar todo el desarrollo del proyecto. A parte de ser una medida de *backup* ofrece la posibilidad de mantener actualizada toda la información y organizarla en versiones en cualquier máquina en la que se trabaje. Es una herramienta ideal para realizar trabajos en grupo ya que la información estará actualizada en todo momento, de manera que cada vez que alguien necesite trabajar sobre alguna parte del proyecto podrá consultar la última versión y trabajar sobre una copia de ésta, hasta que consolide sus cambios y vuelva a subirla al control de versiones convirtiéndose en una nueva versión. De esta manera pueden trabajar varias personas a la vez sobre partes idénticas del desarrollo sin pisarse los cambios los unos a los otros. Por ejemplo: Dos personas se han

descargado a su equipo la última versión de una JSP, es decir, ambas tienen la misma versión y están trabajando con diferentes objetivos. La primera que finalice subirá sus cambios al control de versiones, sin que se produzca ningún problema. Cuando finalice la segunda, existirá una versión más actualizada en el control de versiones que la que estaba modificando, de esta manera nos daremos cuenta que deberemos bajarnos la

última versión para poder añadir los nuevos cambios, sin pisar el trabajo de nuestro compañero.

9.3. Accesibilidad:

Para cumplir con los estándares de accesibilidad, sería necesario según que estándar deseemos cumplir, revisar la parte de Javascript que hemos utilizado en la aplicación y buscar otra forma alternativa a éste para implementar su funcionalidad. También sería interesante para la funcionalidad de introducir comentarios evitar el uso de ventanas y cargarlo todo sobre la misma página.

También cabe destacar, que si en un futuro se usan servicios REST mejoraría el acceso a la información del sistema pues simplifica y organiza la estructura de las URLs. Esto permite acceder a los recursos del sistema de una manera más cómoda.

10. CONCLUSIONES

Finalmente podemos concluir con que para este prototipo se han logrado todos los objetivos marcados a excepción de que nuestras páginas pasen la validación de los niveles de adecuación. Tal y como se ha comentado en “líneas futuras”, este problema se resolvería eliminando el Javascript que posee la aplicación y buscando una manera alternativa para su adecuación. Además deberían eliminarse la apertura de nuevas ventanas, y como posible alternativa se podría ofrecer esa información sobre la misma página. No se ha llegado a cumplir este objetivo puesto que supondría rediseñar parte del desarrollo, y se ha sido consciente de este problema en la finalización del proyecto.

Este proyecto forma parte de un proyecto mayor en el que han participado más personas, cada una de ellas ha realizado su proyecto de fin de carrera de su parte correspondiente. Normalmente el formar parte de un equipo es una ventaja porque existen especialistas cada uno de su materia, y en conjunto pueden resolver la mayoría de los inconvenientes que se les planteen con cierta facilidad, gracias a que con la comunicación entre ellos y aportando cada uno sus conocimientos se hace frente a cualquier inconveniente. En el caso del desarrollo de este proyecto, este punto ha sido más bien un inconveniente puesto que cada componente del equipo ha formado parte de éste en un determinado periodo de tiempo y no se ha llegado a formar en sí un equipo. Lo que quiero decir con esto, es que no se ha trabajado como ocurriría en un proyecto real, sino que más bien te encontrabas trabajo hecho, el cuál debías estudiar y analizar para poder realizar el tuyo propio. Esto presenta tanto sus ventajas como sus inconvenientes, la ventaja es que el conocimiento no tiene lugar y en lugar de adquirir sólo conocimientos de mi proyecto en cuestión, he adquirido conocimientos del resto de partes dándome una visión global que me ha hecho comprender mucho mejor mi trabajo. Los inconvenientes han sido el tener que dedicar mucho más tiempo del que se emplearía sólo en realizar éste proyecto y la dificultad de entender el trabajo que otra persona ha realizado sin poder comunicarte con él para entender su planteamiento.

11. BIBLIOGRAFÍA

Introducción a HTML, CSS y Java Script:

Existe una página en Internet : http://librosweb.es/ , que ha sido de gran utilidad a la hora de adquirir información a la hora de elaborar este proyecto. A continuación detallo alguno de los libros que he consultado:

1.- Título: Introducción a CSS Autor: Javier Eguíluz Pérez (enlace: http://librosweb.es/css/index.html)

2.- Título: CSS avanzado Autor: Javier Eguíluz Pérez (enlace: http://librosweb.es/css_avanzado/)

3.- Referencia de CSS 2.1. (enlace: http://librosweb.es/referencia/css/index.html)

4.- Título: Introducción a JavaScript Autor: Javier Eguíluz Pérez (enlace: http://librosweb.es/javascript/index.html)

5.- Título: Introducción a XHTML Autor: Javier Eguíluz Pérez (enlace: http://librosweb.es/xhtml/index.html)

Introducción a jsp:

1.- JSP es una especificación más de la tecnología Java. En Javasoft, la página que trata de JSP es:

http://www.javasoft.com/products/jsp/index.html y
http://www.javasoft.com/products/jsp/docs.html

2.- Libros:

Editorial: McGrau-Hill JSP: Manual de Referencia por Philip Hanna
JSP: Ejemplos Prácticos Autor: Andrew Patzer Editorial Anaya Multimedia

Manual donde aprenderemos la tecnología Java para la creación de páginas web con programación en el servidor. Enlace:http://www.desarrolloweb.com/manuales/73/

3.- Web:

A la hora de resolver dudas o recoger información para las diferentes funcionalidades implementadas con esta tecnología se ha accedido al siguiente enlace: http://www.javahispano.org/ , buscando por la tecnología en cuestión.

Accesibilidad Web:

1.- La Wikipedia (enlace: http://es.wikipedia.org/wiki/Accesibilidad_web).

2.- Introducción a la accesibilidad web. Enlace: http://www.w3c.es/Traducciones/es/WAI/intro/accessibility.

3.- Analizador de accesibilidad TAW. Enlace para validaciones: http://www.tawdis.net/.

4.- Pautas de accesibilidad al contenido en la web 1.0. Enlace: http://www.discapnet.es/web_accesible/wcag10/WAI-WEBCONTENT-19990505_es.html.

5.- ¿Qué es la Accesibilidad Web? Enlace: http://www.nosolousabilidad.com/articulos/accesibilidad.htm

6.- Preguntas frecuentes sobre la accesibilidad http://www.discapnet.es/web_accesible/WCAG-REC-fact.html

Resolución de pantalla y Navegadores:

Artículo sobre resolución de pantalla: http://www.desarrolloweb.com/articulos/1286.php

Diferencias y semejanzas entre navegadores:

http://www.descargan.com/firefox_vs_internet_explorer_diferencias_y_similitudes_15.html

¿Como elegir un navegador web?:

http://www.maestrosdelweb.com/editorial/%C2%BFcomo-elegir-un-navegador-web/

Otros Articulos:

Firefox sobre Internet Explorer: http://mononeurona.org/pages/display/681
Razones para elegir uno de estos dos últimos: http://www.genbeta.com/navegadores/razones-para-elegir-firefox-o-internet-explorer

JDBC

Descargar manuales: http://www.todoprogramas.com/manuales/programacion/java/371.asp

http://www.conocimientosweb.net/dcmt/ficha14842.html

http://java.sun.com/javase/technologies/database/

MySQL

Toda la información, manuales y documentación relativa en: http://dev.mysql.com/doc/

REST:

Transparencias: http://www.slideshare.net/brenes/introduccion-servicios-web
Introducción a los servicios REST: http://www.dosideas.com/java/314-introduccion-a-los-servicios-web-restful.html

12. ANEXO

Tabla de ponderaciones para EI, EQ y EO

CLASIFICACION DE ENTRADAS Y CONSULTAS	1-4 Atributos	5-15 Atributos	Más de 15 Atributos
0 o 1 ficheros accedidos	BAJA 3	BAJA 3	MEDIA 4
2 ficheros accedidos	BAJA 3	MEDIA 4	ALTA 6
Más de 2 ficheros accedidos	MEDIA 4	ALTA 6	ALTA 6

Tabla 1

CLASIFICACION DE SALIDAS	1-5 Atributos	6-19 Atributos	Más de 19 Atributos
0 o 1 ficheros accedidos	BAJA 4	BAJA 4	MEDIA 5
2 o 3 ficheros accedidos	BAJA 4	MEDIA 5	ALTA 7
Más de 3 ficheros accedidos	MEDIA 5	ALTA 7	ALTA 7

Tabla 2

Tabla de ponderaciones para ILF y EIF

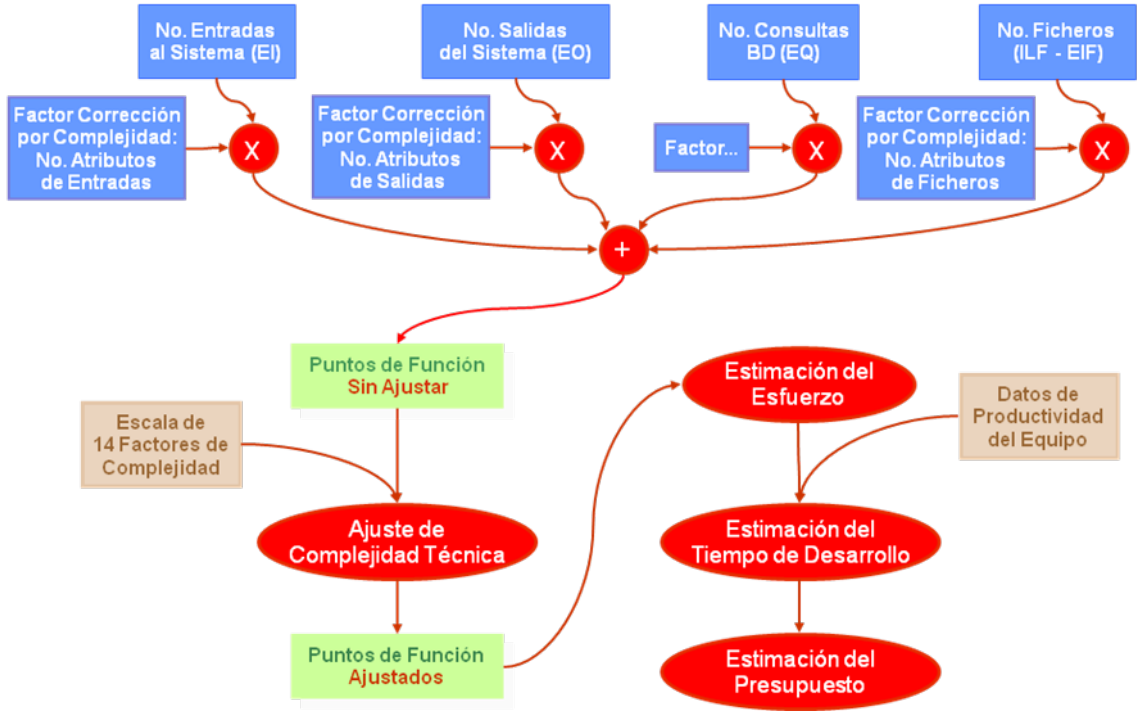
FICHEROS LÓGICOS INTERNOS	1-19 Atributos	20-50 Atributos	Más de 50 Atributos
1 Entidad o registro lógico	BAJA 7	BAJA 7	MEDIA 10
2 - 5 Entidades o registros lógicos	BAJA 7	MEDIA 10	ALTA 15
Más de 5 Entidades o registros lógicos	MEDIA 10	ALTA 15	ALTA 15

Tabla 3

FICHEROS LÓGICOS EXTERNOS	1-19 Atributos	20-50 Atributos	Más de 50 Atributos
1 Entidad o registro lógico	BAJA 5	BAJA 5	MEDIA 7
2 - 5 Entidades o registros lógicos	BAJA 5	MEDIA 7	ALTA 10
Más de 5 Entidades o registros lógicos	MEDIA 7	ALTA 10	ALTA 10

Tabla 4

Proceso de Estimación Mediante PF



PFTe : Total Puntos de Función para las entradas del sistema.
PFTo : Total Puntos de Función para las salidas del sistema.
PFTq: Total Puntos de Función para las consultas del sistema.

PFTif: Total Puntos de Función para los archivos internos del sistema.
PFTef: Total Puntos de Función para los archivos externos del sistema.