REPÚBLICA BOLIVARIANA DE VENEZUELA
UNIVERSIDAD DEL ZULIA
FACULTAD EXPERIMENTAL DE CIENCIAS
DIVISIÓN DE PROGRAMAS ESPECIALES
LICENCIATURA EN COMPUTACIÓN

Herramienta Web para la Clasificación y Recuperación de Información Digital Basada en Etiquetas

Informe de Trabajo Especial de Grado presentado como requisito para optar al título de Licenciado en Computación

Autor: Br. Edinson Padrón Urdaneta

Tutor: MSc. Gerardo Pirela Morillo

Maracaibo, marzo de 2014

Herramienta Web para la Clasificación y Recuperación de Información Digital Basada en Etiquetas

Br. Edinson Padrón Urdaneta
CI. No.: 19.216.488
Teléfono: +58 414 6574149
Venezuela, Zulia, Maracaibo, Urb. La Paz, Calle 96D, Avenida 56
Correo electrónico: edinson.padron.urdaneta@gmail.com

MSc. Gerardo Pirela Morillo
CI.: 12.404.565
Teléfono: +58 412 1734718
Correo electrónico: gepirela@fec.luz.edu.ve

Padrón Urdaneta, Edinson. **"Herramienta Web para la Clasificación y Recuperación de Información Digital Basada en Etiquetas "**. Trabajo Especial de Grado. Universidad del Zulia. Facultad Experimental de Ciencias. División de Programas Especiales. Maracaibo, Venezuela. 2014. 31p.

RESUMEN

Palabras claves: Aplicación web, Etiquetado, Clasificación, Recuperación, Información digital

Correo electrónico: edinson.padron.urdaneta@gmail.com

Padrón Urdaneta, Edinson. **"Herramienta Web para la Clasificación y Recuperación de Información Digital Basada en Etiquetas"**. Trabajo Especial de Grado. Universidad del Zulia. Facultad Experimental de Ciencias. División de Programas Especiales. Maracaibo, Venezuela. 2014. 31p.

ABSTRACT

KeyWords: Web application, Tagging, Classification, Retrieval, Digital information

Email: edinson.padron.urdaneta@gmail.com

ÍNDICE DE CONTENIDO

Pág.

ÍNDICE DE TABLAS

Pág.

ÍNDICE DE FIGURAS

Pág.

INTRODUCCIÓN

CAPÍTULO I

El Problema

1. Planteamiento del Problema y Justificación de la Investigación

La época actual es llamada la “era de la información” (Giuliano, 1983) debido a la relevancia de ésta en la sociedad. Furth (1994) escribió: “La información es actualmente tan vital, y tan intangible, como el aire que respiramos, el cual está lleno de ondas de radio”. Hoy en día es el bien más importante y valioso para toda compañía, país y grupo social, por lo que su refinamiento a partir de la abundante cantidad de datos que son creados cada segundo es un proceso primordial y que ofrece una ventaja significativa.

En el año 2010 se produjo y almacenó un estimado de dos exabytes de información (esto incluye todos los medios: libros, revistas, documentos, Internet, fotografías, televisión, radio, música, entre otros) de los cuales, el 93% era digital (Burgin, 2010). Dos años después, la cantidad de información digital generada, se elevó a 2.5 exabytes al día (IBM, 2012), evidenciando un crecimiento exponencial, con la Internet como principal catalizador de este fenómeno.

A la Internet se le suma el creciente uso de dispositivos móviles (teléfonos inteligentes, tabletas, lectores electrónicos, entre otros) que permiten la accesible y cómoda creación y consumo de información gracias a la web, lo cual provoca una cuantiosa producción de datos, además de contribuir con la heterogeneidad y descentralización de los mismos, ocasionando así que su tratamiento sea una tarea penosamente difícil de realizar.

En la actualidad, los distintos repositorios presentan al usuario la información digital en ellos almacenados mediante una estructura jerárquica de directorios en forma de árbol invertido. Esta representación, nacida en 1969 con el lanzamiento del sistema operativo Multics en respuesta a la necesidad de un sistema de almacenamiento secundario en un ambiente de multiprogramación (Multics, 1968), rígida por causa de su inherente propiedad taxonómica, limita la manera en la que puede ser organizada la información, mediante una relación padre-hijo, y hace de la búsqueda de la misma un proceso arduo debido al uso de engorrosas rutas de directorios. Además, esta aproximación dista en gran medida del método natural usado

por el hombre para almacenar y recuperar información de su memoria a largo plazo, el cual consiste en codificarla semánticamente para su almacenamiento (Baddeley, 1966), asociando los datos que percibe (Atkinson y Shiffrin, 1968). Por lo tanto, el ser humano no es una máquina que trabaje en base a datos aislados, éste busca interrelacionarlos, describiéndolos de manera inherentemente subjetiva, breve y dinámica; haciendo necesario un método semejante para clasificar y recuperar la información digital, siendo éste más flexible, intuitivo y natural.

Afortunadamente, la web ha dado origen a un nuevo sistema de clasificación. Originalmente diseñado e implantado por el servicio del.icio.us (Mathes, 2004), el sistema ofrece la posibilidad de describir recursos mediante un conjunto de palabras clave llamadas etiquetas (*tags,* por su término en inglés), las cuales son consideradas por los usuarios como relevantes para caracterizar dichos recursos de acuerdo a sus necesidades sin depender de un vocabulario controlado o de una estructura previamente definida, estableciendo así una relación entre el recurso y un concepto en su mente, con el objeto de organizar el contenido para uso futuro de una manera fácil y flexible (Specia y Motta, 2007). Este esquema de catalogación se ha extendido en uso por una gran variedad de servicios web debido a su popularidad entre los usuarios, quienes ven en éste una manera más natural, sencilla, rápida y personal de clasificar la información digital que les es de interés, hallándola posteriormente con mínimo esfuerzo, siendo ésta una mejor alternativa al problema de darle tratamiento a una gran cantidad de datos, contrastada con la estructura jerárquica que ofrecen los sistemas de archivos modernos.

Hoy en día, existe una serie de herramientas que extienden las capacidades de los sistemas de archivos al permitir el uso de etiquetas; sin embargo, estas herramientas presentan algunas limitaciones, tales como: el tratamiento exclusivo de una fracción de la información digital existente, su operación bajo un número reducido de plataformas, la prestación de un conjunto restringido de funcionalidades, entre otros.

Con este trabajo de investigación se buscó desarrollar una herramienta web integral que garantizara la interoperabilidad entre los distintos sistemas operativos y dispositivos al permitir un acceso convenientemente centralizado a la información digital sin importar la distribución física de ésta, brindando así transparencia de localización mediante el uso de un servidor central que contenga la herramienta y aplicaciones que enlacen los repositorios de los usuarios con dicho servidor. Además, se implantó una manera sencilla, versátil y personal de clasificar

y recuperar la información digital contenida en los mencionados repositorios al permitir su descripción y recuperación mediante el uso de etiquetas definidas a conveniencia. Adaptando así la tecnología nacida en la web al ámbito de los actuales sistemas de archivo.

2. Alcance del problema

La investigación estuvo delimitada al desarrollo de un sistema cliente-servidor donde el cliente, desarrollado para trabajar solo bajo los entornos Windows®, OS X® y Linux®, se comunica con una aplicación web, manteniendo sincronizadas las etiquetas asociadas a los archivos ubicados en los repositorios del usuario, permitiéndole así clasificarlos y recuperarlos mediante el uso de las mencionadas etiquetas.

3. Objetivos

3.1. Objetivo General

- Desarrollar una herramienta web para la clasificación y recuperación de información digital basada en etiquetas

3.2. Objetivos Específicos

- Realizar una revisión documental sobre herramientas de clasificación y recuperación de información digital
- Implementar la aplicación servidor para el almacenamiento y gestión de etiquetas de recursos digitales
- Desarrollar el módulo de la aplicación cliente encargada del rastreo personalizable de la información a etiquetar
- Desarrollar el módulo de la aplicación cliente correspondiente a la interfaz web de usuario
- Realizar las pruebas de caja gris de la herramienta

CAPÍTULO II

Marco Teórico

1. Antecedentes de la Investigación

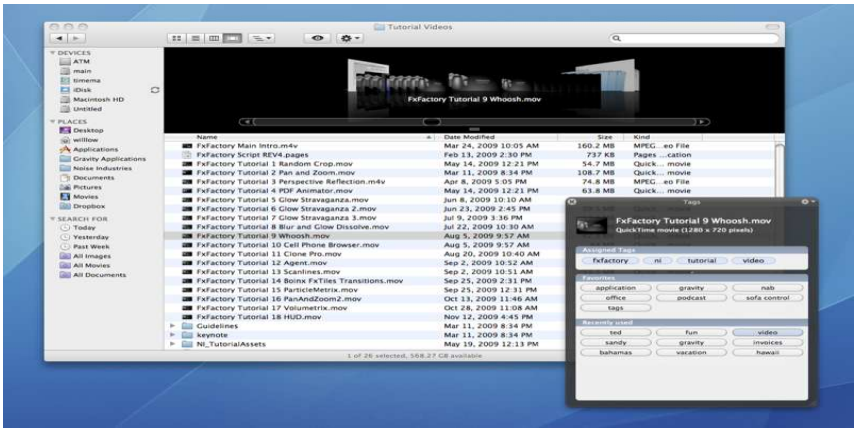
En la actualidad, los sistemas de clasificación basados en etiquetas son ampliamente usados por innumerables aplicaciones web, sin embargo, su uso como complemento a los sistemas de archivo modernos presenta una reducida gama de opciones, dentro de la cual se pueden mencionar:

1.1. Tags

Tags es una aplicación que permite etiquetar archivos y directorios en *finder* (explorador), correos electrónicos en *Mail* (gestor de emails), fotos en *IPhoto* (gestor de imágenes) y enlaces en *Safari* (navegador web); con el objetivo de mantener organizados dichos recursos y poder hallarlos de manera rápida y sencilla.

Tags también ofrece una vista previa de la información, atajos de teclado y una excelente integración con el sistema operativo. Desafortunadamente, esta aplicación solo está disponible para el sistema operativo *OS X*®, está sujeta a una licencia privativa y no es capaz de sincronizar recursos localizados en dispositivos diferentes.

Figura 1. Tags



Fuente: Tags (2014)

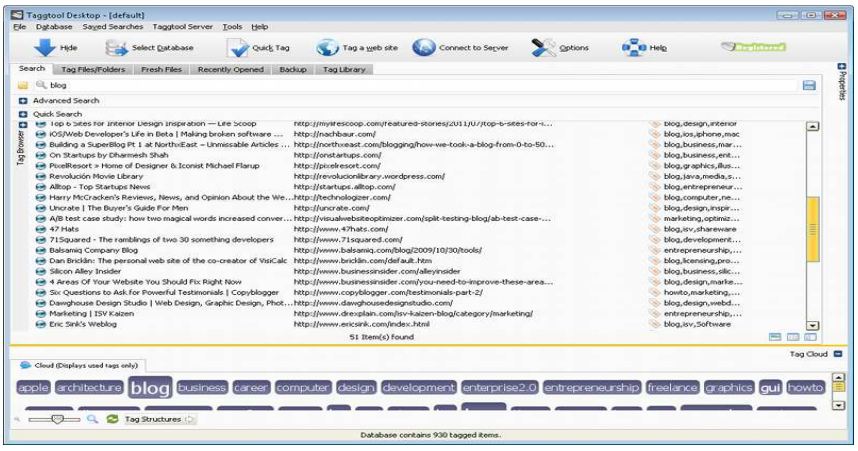
1.2. TagTool

TaggTool permite añadir etiquetas a archivos, directorios y «bookmarks». Adicionalmente, el usuario puede proporcionar una descripción del recurso, además de una calificación cuantitativa (*rating).* Esta herramienta ofrece la posibilidad de importar metadatos pre-existentes en los recursos y realizar búsquedas en base a estos.

TaggTool también posee una alta integración con el sistema operativo y brinda una vista previa de los recursos gestionados. Además, facilita un sistema de etiquetado automático basado en la naturaleza de los recursos, y muestra una nube de etiquetas para la visualización de las mismas.

Al igual que Tags, TaggTool está sujeto a una licencia privativa y no es capaz de sincronizar recursos localizados en dispositivos diferentes. Otra desventaja que presenta este sistema es que está disponible solo para el sistema operativo *Microsoft Windows*®.

Figura 2. TaggTool



Fuente: TaggTool (2014)

1.3. Tables

Tabbles ofrece la posibilidad de clasificar, buscar, organizar y compartir (a través de una LAN o la Internet) archivos, carpetas y «bookmarks», mediante el uso de etiquetas. Adicionalmente, la herramienta ofrece un sistema de auto-etiquetado, además de una excelente integración con el sistema operativo.

A pesar de ser capaz de gestionar, compartir y sincronizar recursos localizados en dispositivos distintos, Tabbles está disponible únicamente para el sistema operativo Microsoft Windows® bajo una licencia privativa.

Figura 3. Tabbles

Fuente: Tabbles (2014)

2. Bases Teóricas

2.1. Sistema de Archivo

Un sistema de archivo es un módulo central de todo sistema operativo moderno que consiste de estructuras lógicas y rutinas que controlan la creación, modificación, eliminación y acceso a los datos que residen en una unidad de disco, partición o volumen lógico. Dicho sistema organiza los datos en una estructura jerárquica mediante el uso de directorios, los cuales fungen como contenedores de punteros a múltiples archivos (EMC, 2012). Algunos de los sistemas de archivo más ampliamente usados en la actualidad son: FAT, FAT32, NTFS, HFS Plus, ext2, ext3 y ext4.

2.2. Transparencia de Localización

Se refiere a una consideración de diseño de los sistemas operativos distribuidos. Se habla de transparencia de localización cuando un usuario accede a un recurso en particular sin que sea necesario conocer la ubicación del mismo dentro de la red de trabajo a fin de acceder a él. El no conocer la localización de un recurso implica que todo acceso a éste es realizado por medio de

un nombre que no depende de la ubicación dónde reside actualmente, ni la ubicación dónde fue creado (Borghoff y Schlichter, 2000).

2.3. Aplicación Web

Es una aplicación desarrollada mediante el uso de tecnologías web (html, css, javascript, entre otros) cuyo entorno de ejecución (*runtime enviroment*) está conformado por un motor web (*web engine*). Como ejemplos de este tipo de aplicaciones se pueden mencionar: sistemas de reservación, sitios de compras en línea, juegos, aplicaciones multimedia, mapas, aplicaciones interactivas de diseños, sistemas de correo, entre otros (W3C, 2012).

Los cuatros componentes típicos de una aplicación web son: un navegador, un servidor web, programas de aplicación y un servidor de base de datos. Las responsabilidades del navegador incluyen: presentar una interfaz de usuario, comunicarse con un servidor web, ejecutar *scripts* embebidos, gestionar la *cache* y las *cookies*. El servidor web se encarga, entre otras cosas, de comunicarse con los clientes, proveer contenido estático, invocar programas de aplicación para la generación de contenido dinámico y gestionar las conexiones (Grove, 2009).

CGI, *servlets*, lenguajes de plantilla (*template languages*) y lenguajes de guión (*script languages*) son algunos de los entornos de desarrollo para la construcción de los programas de aplicación mencionados anteriormente, y están típicamente acompañados por un sistema de gestión de base de datos relacional (*RDBMS*, por sus siglas en inglés) accedido por los programas de aplicación a través de un *ODBC* / *ORM* (Grove, 2009).

2.4. ORM

Es una herramienta que permite conectar un objeto, a veces llamado “modelo de dominio”, con una base de datos relacional de manera automática mediante el uso de metadatos como descriptores del objeto y los datos en sí mismos. Algunas de las ventajas que ofrece el uso de un ORM sobre otras técnicas de acceso de datos se listan a continuación: en primer lugar, se automatiza la conversión bidireccional entre objetos y tablas; además, se provee el almacenamiento en *cache* de los objetos de manera transparente en el lado de la aplicación, mejorando así el rendimiento del sistema; y por último, brinda una API de acceso a la base de datos, abstrayendo por completo el sistema de gestión de bases de datos relacional

(Mehta, 2008). *ODB*, *JDBC*, *Peewee*, *SQLAlchemist* y *Yii* son algunos de los ORM usados en la actualidad.

2.5. AJAX

Es una tecnología empleada para prevenir la carga de una página en su totalidad al permitir la transferencia de datos entre el cliente y el servidor web de manera asíncrona, logrando que las páginas se carguen más rápidamente al retrasar la descarga de grandes recursos (Zakas, 2010).

2.6. JSON

Es un formato ligero y fácil de analizar sintácticamente para representar datos, basado en el uso de la sintaxis dispuesta para los literales de tipo arreglo y objeto en el lenguaje de programación JavaScript, usado generalmente para transmitir información entre distintos sistemas de software (Zakas, 2010). La serialización de información estructurada, tarea que se solía realizar con XML, es actualmente llevada a cabo mediante el uso del formato JSON, tendencia que crece con el pasar del tiempo.

2.7. WSGI

Es una interfaz simple y universal entre servidores y aplicaciones web para el lenguaje de programación Python (Eby, 2010). En términos simples, esta especificación establece la CGI (*Common Gateway Interface*) para el lenguaje de programación Python.

3. Descripción de las Herramientas Usadas

- Python: es un lenguaje de programación de código abierto y de propósito general. Cuenta con una sintaxis simple y legible. Soporta los paradigmas: orientado a objetos, funcional, y procedural. El dominio de aplicación del lenguaje va desde sistemas de administración, desarrollo de paginas web, y educación hasta el análisis computacional de inversión, el desarrollo de video juegos y el control aéreo (Lutz, 2010).

- Flask: es un *microframework* web para el lenguaje de programación Python. Su principal

característica recae en poseer un conjunto de funcionalidades reducido y simple, brindando gran detalle a la extensibilidad de la herramienta, permitiéndole al usuario elegir los componentes a integrar, adaptándose así a las necesidades del proyecto a desarrollar y a las preferencias de los usuarios de manera flexible e indolora (Flask.org, 2014).

- Pewee: es un ORM simple, escrito en python, que provee una interfaz ligera para realizar consultas SQL (Leifer, 2014).

- SQLite: es una librería que implementa un motor de base de datos SQL transaccional y auto contenido. Sobresale entre sus características el poseer una arquitectura que no concibe la existencia de un proceso servidor, almacenando los datos gestionados en un archivo ordinario (SQLite.org, 2014).

- Twitter bootstrap: es un *framework* HTML, CSS y JavaScript para desarrollar proyectos web adaptables (*responsive*) y orientados a dispositivos móviles (*mobile first*) (getbootstrap.com, 2014).

- JQuery: es una librería escrita en JavaScript rápida, simple y con una gran repertorio de utilidades. Hace de la manipulación del DOM, el manejo de eventos, la creación de animaciones y el uso de Ajax, tareas mucho más sencillas de realizar a través de una API multiplataforma (JQuery.com, 2014).

CAPÍTULO III

Marco Metodológico

1. Descripción de la Metodología Empleada

Para el desarrollo de la herramienta, fruto de esta investigación, se empleó la metodología *Scrum*, la cual se define como un "marco de trabajo en el cual las personas abordan problemas adaptativos complejos, entregando productos del más alto valor posible de manera productiva y creativa" (Schwaber y Sutherland, 2011). *Scrum* está fundamentada en el empirismo, el cual afirma que el conocimiento viene de la experiencia y que la toma de decisiones debe basarse en lo que se sabe. También, vale mencionar que esta metodología emplea un modelo iterativa e incremental para optimizar la capacidad de predicción y el control de riesgos.

El corazón del *Scrum* es el *Sprint*, el cual es un lapso de tiempo, cuya duración no supera la de un mes, durante el cual un incremento de un producto terminado, usable y potencialmente entregable es creado. Un *Sprint* abarca lo que se va a construir, la elaboración de un plan flexible para llevar a cabo dicho proceso, la construcción en sí misma y el producto resultante.

En cuanto a las pruebas del sistema, se optó por la aplicación de dos tipos: pruebas funcionales orientadas a tareas (*Task-Oriented Functional Tests*) y pruebas de errores forzados (*Forced-Error Tests*), ambas ejecutadas bajo el modelo de caja gris.

De acuerdo a Hung Nguyen, Bob Johnson y Michael Hackett, las pruebas funcionales orientadas a tareas consisten de casos de prueba positivos que son diseñados para verificar las características del programa al asegurarse de que cada una de ellas se comporte como se espera, tomando en cuenta las especificaciones, guías de usuario, requerimientos y documentos de diseño. Cada característica es probada para:

- Validar que la tarea se ejecute adecuadamente dadas condiciones de dato adecuadas bajo condiciones operativas favorables.
- Verificar la integridad del resultado final de la tarea.
- Asegurar la integridad de la característica cuando se usa junto a otras.

Por su parte, las pruebas de errores forzados, según Nguyen, Johnson y Hackett, son casos de prueba negativos diseñados para forzar a un programa a pasar a condiciones de error. Debe generarse una lista de todos los mensajes de error que el programa emite. Esta lista es usada como base en el desarrollo de casos de prueba. Algunas recomendaciones a la hora de llevar a cabo este tipo de pruebas se describen a continuación:

- Verificar que todas las condiciones de error comunes sean detectadas y manejadas correcta y consistentemente.
- Verificar que el programa se recupere correctamente para toda condición de error.
- Verificar que los estados inestables del programa, causados por un error, también sean corregidos.

Por último, el modelo (o pruebas, como también es conocido) de caja gris incorpora elementos de las pruebas de caja negra y blanca. En las pruebas de caja gris se considera el resultado desde el punto de vista del usuario, el conocimiento técnico específico del sistema y el ambiente operativo; además, estas pruebas evalúan el diseño de la aplicación bajo el contexto de la interoperabilidad entre los componentes del sistema.

El modelo de caja gris es ideal para poner a prueba de manera efectiva aplicaciones web debido a que éstas abarcan numerosos componentes, tanto de software como de hardware. Estos componentes deben ser probados en el contexto del diseño del sistema para evaluar su funcionalidad y compatibilidad.

2. Desarrollo de la Herramienta Balo la Metodología Scrum

El desarrollo de la herramienta web, Omnitag, se llevó a cabo a lo largo de nueve *Sprints*, cada uno de un mes de duración. La distribución de las tareas llevadas a cabo a lo largo del ciclo de desarrollo se ha plasmado en la siguiente tabla, la descripción de las mismas se plantea justo después de la mencionada tabla.

Tabla 1. Distribución de Actividades

Sprint/Tarea	1º	2º	3º	4º	5º	6º	7º	8º	9º
Diseño e Implementación de la Base de Datos									
Diseño e Implementación de la Interfaz Web de Usuario									
Codificación e Implementación de la Aplicación Servidor									
Codificación e Implementación de la Aplicación Cliente									
Pruebas de Caja Gris de la Herramienta									

Fuente: Padrón (2014)

2.1. Primer Sprint

Durante este *sprint* se culminó la revisión de literatura relacionada con los sistemas de clasificación y recuperación de información digital basados en etiquetas, con la recopilación de los antecedentes de esta investigación. Además, se comenzó a diseñar e implementar la interfaz web de usuario a través del diseño de bocetos físicos y digitales; y la base de datos, mediante el esquema fundamental de la misma. Adicionalmente, se dio inicio al proceso de codificación e implementación de la aplicación servidor, diseñando la funcionalidad base para recibir y responder peticiones HTTP. Todo esto junto a las respectivas pruebas de caja gris de los componentes desarrollados, acción que se mantuvo durante todo el ciclo de desarrollo de la herramienta.

2.2. Segundo Sprint

Durante este *sprint* se continuó con las actividades iniciadas en el anterior refinando el diseño de la base de datos, añadiendo nuevas características a la aplicación servidor y mejorando la interfaz web de usuario con el fin de hacerla más intuitiva y funcional.

2.3. Tercer Sprint

Al igual que en el anterior, durante este *sprint* se dio continuidad a las labores anteriormente

descritas, añadiendo un nuevo conjunto de características a la aplicación servidor, reflejando éstas mejoras en la interfaz web del usuario y dándole los toques finales a la base de datos, finalizando el proceso de diseño e implementación de la misma.

2.4. Cuarto Sprint

En este *Sprint* se comenzó el diseño de una API pública, implementada en la aplicación servidor, para permitir una comunicación sencilla con la misma, potenciando así la posibilidad de desarrollar una aplicación cliente que se comunique con la aplicación servidor de manera efectiva. También se hicieron un par de cambios menores a la interfaz web de usuario.

2.5. Quinto Sprint

Es en este *Sprint* que se da inicio a la codificación e implementación de la aplicación cliente, conformada, al final de este ciclo, solo por el mecanismo de rastreo de recursos del sistema. Por otro lado, se sigue con la expansión de la API mencionada anteriormente y el refinamiento de la interfaz web de usuario, acciones que se repiten hasta el final del tercer lanzamiento.

2.6. Sexto Sprint

Se desarrolló el módulo que permite al usuario seleccionar a potestad cuáles directorios serán incluidos en el proceso de rastreo, así como el módulo que se encarga de mantener sincronizada con el servidor la información generada a partir del mencionado rastreo.

2.7. Septimo Sprint

A lo largo de este sprint, se implementó la gestión de usuarios en la aplicación servidor, se adaptó la interfaz web de usuario para reflejar dichos cambios y se implementó en ambas aplicaciones un mecanismo de comunicación sencillo, robusto y seguro, basado en *tokens*.

2.8. Octavo Sprint

Durante este *Sprint* se desarrolló e integró un mecanismo de etiquetado automático de la información rastreada.

2.9. Noveno Sprint

Este *Sprint* cerró el ciclo de desarrollo de esta investigación, durante el cual se llevaron a cabo las pruebas más rigurosas del sistema, con la finalidad de detectar y posteriormente corregir errores funcionales que residían en la aplicación.

CAPÍTULO IV

OmniTag

1. Arquitectura de la Herramienta

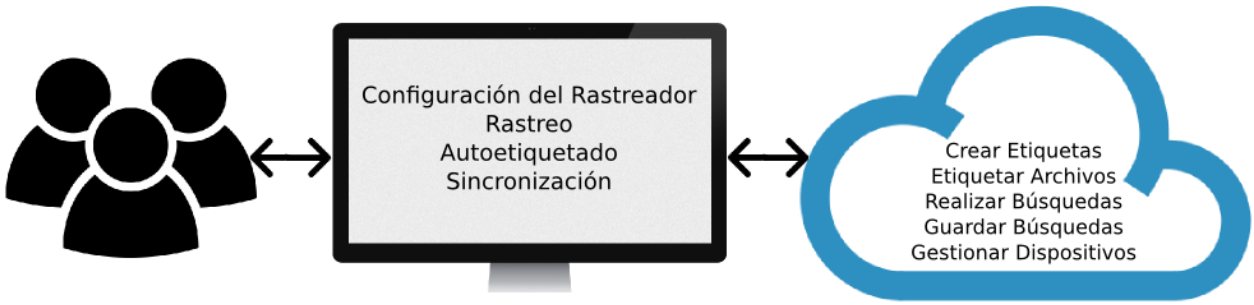
OmniTag es una herramienta que permite a sus usuarios organizar y buscar los archivos, contenidos en sus distintos computadores de escritorio y portátiles, a través del uso de etiquetas bajo una sola plataforma basada en tecnologías web y el lenguaje de programación Python. OmniTag relega a un navegador web el despliegue de su interfaz gráfica de usuario, construida mediante el uso de los lenguajes HTML y CSS junto al *framework* Twitter Bootstrap, para la parte visual; y el lenguaje de programación JavaScript, con la ayuda de la librería JQuery, para la parte interactiva. La lógica de negocios es manejada por el interprete de Python, el cual se vale de una base de datos SQLite para gestionar la información no volátil, base de datos con la que interactúa mediante el ORM Peewee; y el *framework* web Flask, encargado del manejo de las peticiones web, la transmisión de contenido estático, la ejecución de los programas de aplicación para generar contenido dinámico, entre otros.

Funcionalmente, OmniTag está constituida por dos módulos fundamentales: uno cliente y otro servidor. Ambos módulos dependen de un navegador web para interactuar con el usuario, haciendo uso de las tecnologías disponibles en dicho navegador para desplegar una interfaz interoperable con los distintos ambientes operativos, computadores de escritorio y portátiles de amplio uso en la actualidad.

El módulo servidor se ocupa de la gestión de todos los procesos relacionados con la creación, control y mantenimiento de los usuarios y sus correspondientes dispositivos. También se encarga de brindar las funcionalidades de visualización, etiquetado y búsqueda de los recursos almacenados en dichos dispositivos. La tarea de visualización se ofrece al usuario empleando una capa de transparencia de localización para un cómodo tratamiento de los recursos; el etiquetado es flexible, pudiendo aplicar o remover etiquetas a múltiples recursos a la vez, y la búsqueda es un proceso intuitivo permitiendo asociar un número arbitrario de palabras clave bajo un mismo término con el fin de potenciar, facilitar y agilizar la recuperación de los recursos.

Por su parte, el módulo cliente es la pieza destinada a residir en los dispositivos de los usuarios, permitiendo a éstos indicarle cuáles directorios de un repositorio en particular tomará en cuenta a la hora de rastrear los recursos que serán sincronizados con el módulo servidor para su posterior clasificación y recuperación. También es responsable de alojar los parámetros de configuración empleado para la comunicación con el servidor, además de ser el encargado de predefinir etiquetas a un subconjunto de los recursos rastreados, dependiendo su asignación de la naturaleza de los dichos recursos. La **Figura 4** plasma la funcionalidad de estos módulos, así como la interacción que se da entre ellos y aquella que involucra al usuario.

Figura 4. Arquitectura Funcional de la Herramienta



Fuente: Padrón (2014)

2. Implementación de la Herramienta

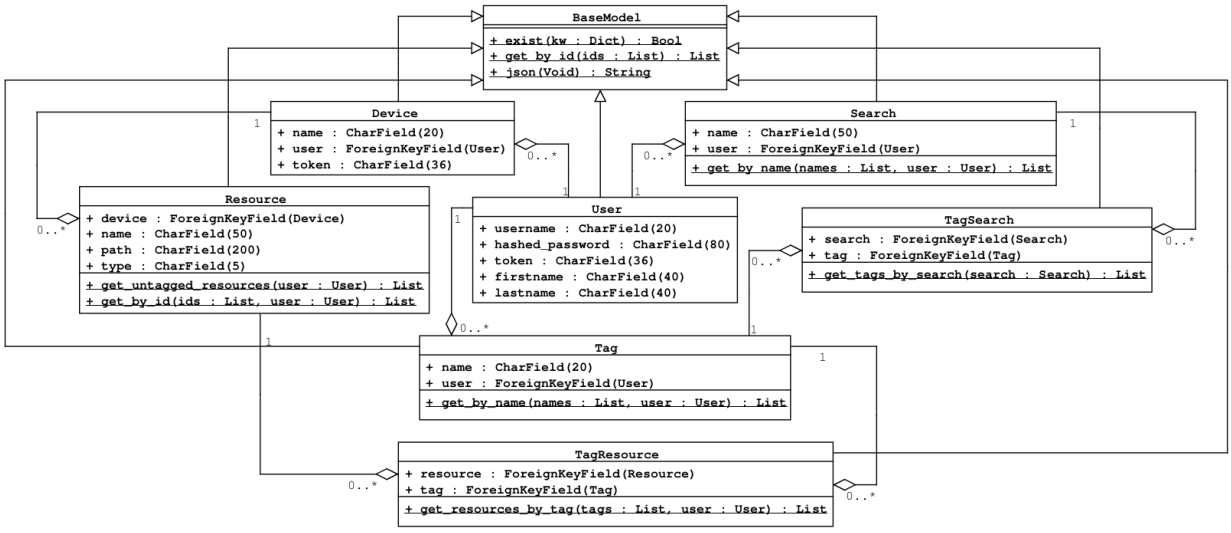
La **Figura 5** ilustra los modelos que forman parte del módulo servidor, cabe mencionar que ésta describe los atributos de los modelos y las relaciones que existen entre ellos tomando en cuenta su representación en forma de tablas bajo un paradigma relacional, estando éstas intrínsecamente vinculadas con sus correspondientes clases (paradigma orientado a objetos) mediante el uso del *Peewee* (ORM). Estos modelos son:

- User: Representa un usuario de la herramienta.
- Device: Representa un dispositivo de un usuario.
- Tag: Representa una etiqueta, la cual pertenece a un usuario único.

- Resource: Representa un recurso alojado en dispositivo en particular.

- Search: Representa asociaciones de etiquetas usadas en búsquedas almacenadas.

Figura 5. Diagrama de Clases de los Modelos



Fuente: Padrón (2014)

Como puede observarse, adicional a los atributos de los modelos, éstos comparten una interfaz común, definida por la clase *BaseModel*, además de tener asociados un conjunto de métodos, a saber:

- *BaseModel*: Pauta la interfaz mínima que debe poseer todo modelo, definiendo tres métodos: *exists*, para verificar la existencia de un registro en particular en la base de datos correspondiente; *get_by_id*, que retorna un conjunto de registros dada una lista de identificadores; y por último, *json*, el cual se encarga de convertir en formato *JSON* un registro en particular.

- *Tag*: Solo define un método: *get_by_name*, que se encarga de devolver todos aquellos registros que correspondan a una lista de nombres en particular.

- *Search*: Similar a *Tag*, esta clase define solo un método *get_by_name*, el cual, a partir de una lista de nombres, retorna todos los recursos cuyo nombre se encuentra en la mencionada lista.

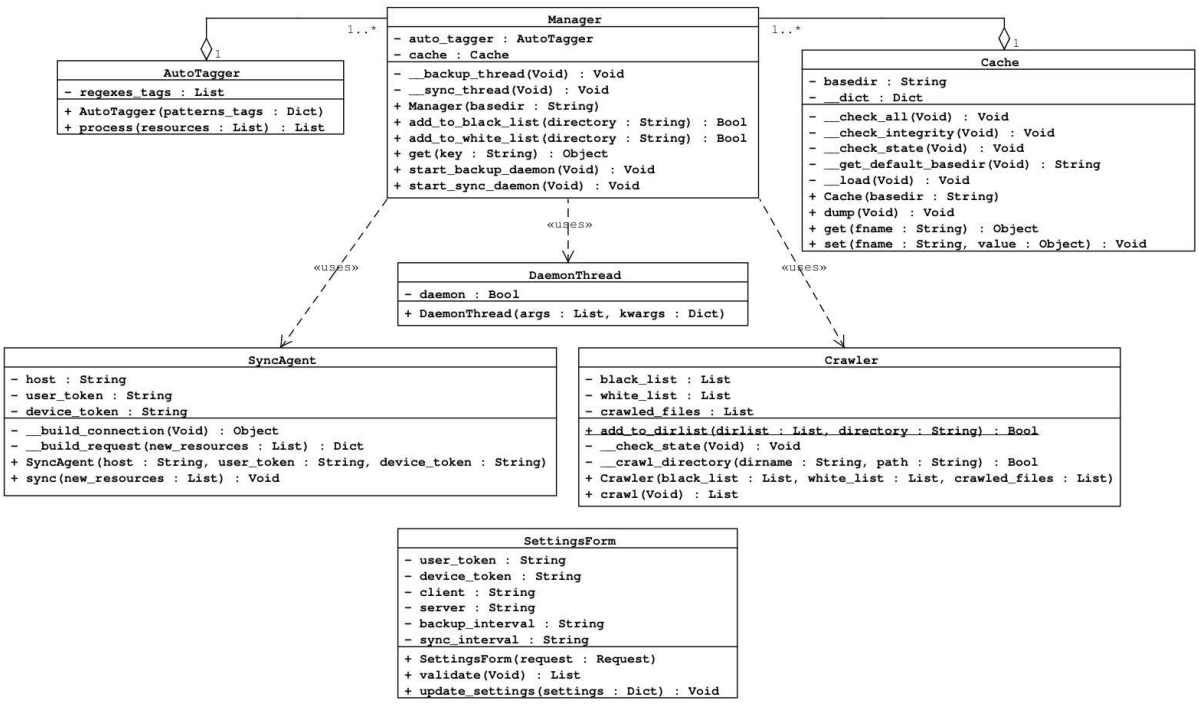
- *Resource*: Esta clase define un método llamado *get_untagged_resources* que se encarga de devolver todos aquellos recursos que no poseen asociación alguna con una etiqueta.

- *TagSearch*: Establece la relación que existe entre las etiquetas y las búsquedas. Esta clase ofrece un método *get_tags_by_search* que retorna todas las etiquetas asociadas a una búsqueda en particular.

- *TagResource*: Funge como vínculo entre las etiquetas y los recursos. El método *get_resources_by_tag* permite obtener todos los recursos que se encuentran asociados a un conjunto arbitrario de etiquetas.

En el caso del módulo cliente, se presenta a continuación un diagrama de clases que muestra la estructura interna del mismo.

Figura 6. Diagrama de Clases del Módulo Cliente



Fuente: Padrón (2014)

- *Manager*: se encargar de la generación, lectura y respaldo de los archivos de configuración y estado del sistema, así como también de la comunicación con el

módulo servidor. Los métodos asociados a esta clase son: *__backup_thread* y *__sync_thread* contienen los hilos que se encargan del respaldo de las variables de estado y la sincronización de los archivos rastreados con el servidor, respectivamente; *start_backup_daemon* y *start_sync_daemon* se encargan de ejecutar los hilos mencionados anteriormente; *__add_to_black_list* y *__add_to_white_list* controlan la inserción de directorios en la lista negra y blanca, de manera correspondiente.

- *DaemonThread*: es la clase usada para generar los hilos de ejecución asíncrona y que extiende a la clase *Thread* del módulo *threading* de la librería estándar del lenguaje de programación Python.

- *Cache*: es la clase sobre la cual *Manager* confía las tareas de respaldo de las variables de estado. Los métodos definidos en esta clase son: *__check_all*, *__check_integrity* y *__check_state*, empleados para verificar la correctitud del estado interno de toda instancia de la clase; *__get_default_base_dir* retorna el directorio base a usar por defecto; *__load* carga en memoria las variables de estado serializadas en el medio de almacenamiento secundario del dispositivo en formato JSON; *dump*, lleva a cabo el proceso inverso que realiza *__load*; y, por último, los métodos *get* y *set* controlan el acceso y redefinición de las variables respaldadas.

- *SyncAgent*: es usada por la clase *Manager* para mantener los archivos rastreados en sincronización con el servidor. Los métodos de esta clase son: *__build_request* construye una petición HTTP a partir de una lista de archivos, petición usada por la conexión generada por *__build_connection*; el método *sync* lleva a cabo la sincronización mediante el uso de la conexión mencionada anteriormente.

- *Crawler*: encapsula las rutinas relacionadas con el rastreo de recursos en el dispositivo del usuario. Los métodos definidos es esta clase son los siguientes: *__check_state* verifica la correctitud del estado interno de toda instancia de la clase; *__crawl_directory* determina, dada la ruta de un directorio, si éste debe o no ser tomado en cuenta en el rastreo de archivos; y, por último, *crawl* ejecuta el rastreo.

- *AutoTagger*: esta clase se encarga de relacionar etiquetas a los archivos rastreados de manera automática en función de la extensión de éstos. El único método de esta clase se

llama *process*, el cuál ejecuta el etiquetado automático de los archivos indicados.

- *SettingsForm*: es empleada para la recepción y validación de los parámetros de configuración proporcionados por los usuarios a través de la interfaz gráfica. Los métodos definidos en esta clase se describen a continuación: *validate* verifica la correctitud de la información enviada por el usuario y *update_setings* se encarga de comunicar al resto de la aplicación los nuevos valores de los parámetros de configuración.

A continuación, se presenta el código fuente de los procesos de rastreo y auto-etiquetado de archivos, componentes más sobresalientes del sistema:

Figura 7. Crawler

```
# ~~ Classes ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class Crawler(object):
    def crawl(self):
        self.__check_state()

        crawled_files = set()
        for directory in self.white_list:
            for path, dirnames, filenames in os.walk(directory):
                crawled_files.update(os.path.join(path, filename) for filename in filenames)
                dirnames[:] = (dirname for dirname in dirnames if self.__crawl_directory(dirname, path))

        new_files = crawled_files.difference(self.crawled_files)

        self.__check_state()

        return new_files
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fuente: Padrón (2014)

Figura 8. AutoTagger

```
# ~~ Classes ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
class AutoTagger(object):
    def process(self, resources):
        assert isinstance(resources, set)

        resources_tags = []

        for resource in resources:
            auto_tags = []

            for regex, tags in self.regexes_tags:
                if(regex.match(resource)):
                    auto_tags.extend(tags)

            resources_tags.append((resource, auto_tags))

        assert isinstance(resources_tags, list)

        return resources_tags
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Fuente: Padrón (2014)

REFERENCIAS BIBLIOGRÁFICAS

- Atkinson, R.C.; Shiffrin, R.M. (1968). Chapter: Human memory: A proposed system and its control processes. The psychology of learning and motivation. pp. 89-195.
- Baddeley, A.D. (1966). The influence of acoustic and semantic similarity on long-term memory for word sequences. The Quarterly Journal of Experimental Psychology. pp. 302-309.
- Borghoff, Uwe; Schlichter, Johann (2000). Computer-Supported Cooperative Work: Introduction to Distributed Applications. Springer Science y Business Media. p. 6.
- Burgin, Mark (2010). Theory of Information: Fundamentality, Diversity and Unification. World Scientific. p. VI.
- Eby, Phillip (2010) (Página consultada el 16 de julio de 2014) [On-line]. Dirección: http://legacy.python.org/dev/peps/pep-3333.
- EMC Education Services (2012). Information Storage and Management: Storing, Managing, and Protecting Digital Information in Classic, Virtualized, and Cloud Environments. John Wiley & sons. p. 6.
- Flask.org (2014)(Página consultada el 19 de julio de 2014) [On-line]. Dirección: http://flask.pocoo.org/.
- Furth, J. (1994). The Information Age in Charts, Fortune International.
- getbootstrap.com (2014) (Página consultada el 19 de julio de 2014) [On-line]. Dirección: http://getbootstrap.com/.
- Giuliano, V.E. (1983). The United States of America in the Information Age, en Information Policy and Scientific Research, Elsevier, Amsterdam, pp. 59-76.
- Grove, Ralph (2009). Web Based Application Development. Jones & Bartlett Publishers. p. 42.

- IBM (2012) (Página consultada el 25 de junio de 2013) [On-line]. Dirección: www-01.ibm.com/software/data/bigdata/
- JQuery.com (2014) (Página consultada el 19 de julio de 2014) [On-line]. Dirección: http://jquery.com/.
- Leifer, Charles (2014) (Página consultada el 19 de julio de 2014) [On-line]. Dirección: http://peewee.readthedocs.org/.
- Mark Lutz (2010). Programming Python. O'Reilly Media, Inc.
- Mathes, Adam (2004) (Página consultada el 25 de junio de 2013) [On-line]. Dirección: www.adammathes.com/academic/computer-mediated-communication/folksonomies.html
- Mehta, Vijay (2008). Pro LINQ Object Relational Mapping in C# 2008. Apress. pp. 4-5.
- Multics (1968) (Página consultada el 25 de junio de 2013) [On-line]. Dirección: www.multicians.org/fjcc4.html
- Nguyen, Hung; Johnson, Bob; Hackett, Michael (2003). Testing Applications on the Web: Test Planning for Mobile and Internet-Based Systems. Second Edition. Wiley Publishing, Inc.
- Schwaber, Ken; Sutherland, Jeff (2011). The Definitive Guide to Scrum: The Rules of the Game.
- Specia, Lucia; Motta, Enrico (2007). Integrating Folksonomies with the Semantic Web. Knowledge Media Institute - The Open University.
- SQLite.org (2014) (Página consultada el 19 de julio de 2014) [On-line]. Dirección: https://sqlite.org/about.html.
- World Wide Web Consortium (2012) (Página consultada el 16 de julio de 2014) [On-line]. Dirección: http://www.w3.org/2012/webapps/charter/
- Zakas, Nicholas (2010). High Performance JavaScript. O'Reilly Media, Inc.