FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Framework para facilitar la construcción de widgets relacionados a Web Augmentation
Autores: Matias Moratti – Fermín Recalt
Director: Gustavo Rossi
Codirector: Sergio Firmenich
Asesor profesional: -
Carrera: Licenciatura en Sistemas

Resumen

En los últimos años ha surgido un interés en aumentar el contenido de un sitio web ya creado, para ampliar la interacción del usuario con el mismo. De esta manera, un desarrollador puede estar visitando una página, y pensar una nueva funcionalidad que podría tener la misma, que agrande el abanico de posibilidades de interacción del usuario que accede a ella. Este concepto tiene el nombre de Web Augmentation, y está relacionado a lo que es la realidad aumentada en el mundo físico, esto es distintas capas de contenido, de diseño y de navegación para personalizar la experiencia del usuario, sin tener que crear una nueva aplicación web desde cero. A partir de ahora, llamaremos Widget a cada script desarrollado que aumenta y/o customiza el contenido de una página web.
Tras el análisis de varios widgets ya existentes en la web, y desarrollando otros de manera de poder analizar el método de creación de cada uno, se llegó a la conclusión de que estos poseen muchas cosas en común a la hora de desarrollarlos, y que, para poder hacerlo, se necesita un nivel intermedio de conocimiento de técnicas de programación. Es por ello que el trabajo consistirá en el desarrollo de un framework que brinde herramientas y soporte para facilitar la creación de widgets basados en Web Augmentation.

Palabras Claves

Framework, Widget, Web Augmentation, Social Web, SocialEye, Navegador, API, JavaScript, HTML, Django, PostgreSQL, JSON, Google Chrome, Extensión, WebRTC.

Conclusiones

Se creó un framework que facilita el diseño y desarrollo de widgets basados en Web Augmentation, enfocado a que usuarios que no tengan un alto nivel de conocimiento informático puedan explotar este nuevo concepto combinándolo con sus propias ideas.
Además, se codificaron widgets de muestra que utilizan el framework, de manera de justificar su desarrollo y el beneficio que brinda a los futuros usuarios.

Trabajos Realizados

Investigación del concepto de Web Augmentation en la Web. Análisis de los widgets existentes. Desarrollo de un framework que facilita la construcción de los mismos. Creación de widgets de muestra diseñados a partir del framework realizado.

Trabajos Futuros

Enriquecer la interfaz de programación con más métodos y atributos que hagan aún más flexible la creación de widgets, aumentando así la productividad de la herramienta. Widget dedicado a la accesibilidad Web. Widget que permita "exportar contenido" entre sitios web. Extensión a "Red social" a widget de Usuarios. Extender herramienta de navegación a otros navegadores

Fecha de la presentación: Julio, 2017

Framework para facilitar la construcción de widgets relacionados a Web Augmentation

Tesina de Licenciatura

Alumnos:
Matias Moratti – Fermín Recalt

Director:
Gustavo Rossi

Codirector:
Sergio Firmenich

2017

Facultad de Informática
Universidad Nacional de La Plata

Dedicado a nuestras familias, por su apoyo
en todas las etapas de nuestra carrera.
A Sergio, por su calidez y profesionalismo.

Índice General

Índice de Figuras

1. *Introducción*

En la última década, el uso de la web ha crecido exponencialmente en popularidad, con la ayuda de tecnologías y dispositivos móviles que permiten que la mayor parte de la población mundial se encuentre las veinticuatro horas conectada, consumiendo información y contenidos. Los sitios web actuales son cada vez más sofisticados, no solo por la cantidad de operaciones o acciones que se pueden realizar en ellos, sino también por el diseño que poseen, incluyendo la posibilidad de ser accedidos por diferentes terminales, de diferentes tamaños y sistemas operativos.
Si bien notamos este constante avance en la web, ubicando sitios cada vez más interactivos y complejos, así también crecen las expectativas y exigencias de los usuarios, las cuales puede que no fueran satisfechas en su totalidad. Este último concepto puede ser muy difícil de dominar, teniendo en cuenta que cada ser humano es diferente, y así son sus preferencias y necesidades.

Además, ha surgido en los últimos años, una necesidad por parte de los usuarios de un software que le permita realizar trabajo en equipo o en forma colaborativa al mismo tiempo, manteniéndose conectados. Esta necesidad no se basa únicamente en cuestiones laborales, sino que engloba a todos los sistemas informáticos que asisten a personas que persiguen un mismo objetivo. Esta combinación de personas agrupadas con un mismo fin, junto con el software adecuado que las asiste al cumplimiento del mismo, se denomina Groupware [1]. La gran mayoría de los sistemas Groupware incluyen aspectos de sistemas distribuidos, y el tipo de estrategia utilizada por el grupo para alcanzar su objetivo determina la característica del software involucrado.

De esta forma, se desprende el interés emergido en los últimos años por aumentar el contenido de un sitio web ya creado, en base a los requerimientos específicos de cada usuario. De esta manera, un usuario con conocimientos técnicos desarrollador puede estar visitando una página, y pensar una nueva funcionalidad que podría tener la misma, que haga más grande al abanico de posibilidades de interacción del usuario que accede a ella. La intención de "aumentar" un sitio web no se reduce sólo al interés de incrementar su contenido, o añadir nueva funcionalidad, sino también a la tendencia particular de personalizar a gusto propio una aplicación que se usa con regularidad. Este fenómeno, conocido como "The Augmented Web" [2], puede ser aplicado con variados objetivos, ya sea por una cuestión relacionada a la estética o gusto personal de cada uno, como también a temas relacionados con una imposibilidad física, como en el caso de usuarios no videntes [3], o incluso tener visión disminuida o problemas con ciertos colores específicos [4].

Este concepto tiene el nombre de Web Augmentation o Augmented Browsing [2][5], y está relacionado a lo que es la realidad aumentada en el mundo físico, esto es distintas capas de contenido, de diseño y de navegación para personalizar la experiencia del usuario, sin tener que crear una nueva aplicación web desde cero [2].
Actualmente, la manera más común de llevar a la práctica este concepto, es utilizando una extensión para el navegador, que se ejecuta en tantas páginas como sea configurada. Navegadores como Mozilla Firefox, Google Chrome, poseen extensiones llamadas GreaseMonkey [6][7] y TamperMonkey [8] respectivamente, que permiten ejecutar, en todos los sitios web, el código escrito en lenguaje JavaScript [9] empleado por un desarrollador que quiera empezar a incursionar en este nuevo concepto.

Esta idea de asociar scripts al navegador, hace que los usuarios puedan alterar los sitios Web una vez estos son cargados en sus navegadores, pero usualmente no tienen la lógica necesaria para tener un soporte back-end que posibilite, por ejemplo, aspectos sociales y de colaboración.
Es así que en esta tesina se presenta el proyecto e implementación de un Framework [10], cuyo objetivo será el de presentar a los usuarios una herramienta con la cual puedan desarrollar Widgets de navegador [11], con la idea de que los mismos se encuentren orientados a la comunicación entre las personas, de modo que los usuarios de internet interactúen y creen contenido en la web de distintas formas.
A partir de ahora, llamaremos Widget a cada script desarrollado que aumenta y/o customiza el contenido de una página web.

La tesina se organiza de la siguiente manera:

- En el capítulo 2, se plantean los objetivos propuestos para esta tesina.
- En el capítulo 3, se realiza una introducción al mundo de Web Augmentation, junto con el fenómeno social en la web, conceptos que motivaron el desarrollo de esta tesina.
- En el capítulo 4, se otorga una breve explicación del funcionamiento del framework, como para dar un primer pantallazo al lector previo a los detalles técnicos de la herramienta.
- En el capítulo 5, se trata del framework SocialEye desde el lado del componente del cliente, lenguaje y herramientas utilizadas, y una descripción de cómo desarrollar un widget extendiendo de las clases que componen el framework.
- El capítulo 6 se enfoca en el framework desde el lado del componente servidor, detallando lenguaje de programación utilizado, librerías externas, y motor de base de datos elegido.
- En el capítulo 7, se introduce la herramienta de navegación del framework, además de su sitio web, el cual entre otras cosas, será el lugar en donde el usuario puede realizar la administración de sus widgets desarrollados.
- En el capítulo 8, se presentan todos los widgets de prueba realizados en esta tesina, describiendo en detalle su utilidad, comportamiento, modelo de datos, y procedimiento de codificarlo basándose en el framework. Además, se plantean futuros nuevos widgets que podrían ser de utilidad para el usuario de internet.
- En el capítulo 9, se expone el estado actual del concepto de Web Augmentation en internet, mencionando aplicaciones desarrolladas relacionadas con este criterio.
- En el capítulo 10, se finaliza con las conclusiones obtenidas del desarrollo de la tesina, además de posibles trabajos futuros que pueden surgir de ella.
- En el capítulo 11, se citan las referencias.

2. *Objetivos propuestos*

Con lo expuesto anteriormente, se propone como objetivos generales:

- Diseñar e implementar una arquitectura de aumentación que permita la creación de scripts sociales de aumentación y una plataforma que permita el uso de los mismos por la comunidad de usuarios.

Para cumplirlo, los objetivos específicos que se definieron son los siguientes:

- Investigar soluciones existentes relacionadas al concepto de Web Augmentation. A partir de la recopilación de estas aplicaciones, se podrá realizar una conclusión acerca de la utilidad de realizar un framework, que permita desarrollar scripts de forma más sencilla.
- Diseñar e implementar un back-end que de soporte genérico al comportamiento social. Esto conlleva el análisis de las herramientas existentes en el mundo del software dedicadas al componente back-end de una solución, y posteriormente el desarrollo del mismo, incluyendo manejo de sesión, almacenamiento y movimiento de datos, ya sea los provenientes de los widgets, como del sitio web del framework, en una base de datos relacional, tomando en cuenta aspectos relacionados a la seguridad de los mismos para su correcta protección.
- Implementar una herramienta de usuario final para la utilización de los widgets. El objetivo es agrupar los widgets en un espacio físico reducido dentro del sitio web, de forma de automatizar y unificar el acceso a los mismos. Además, en esta herramienta se podrá configurar los widgets que cada usuario quiere tener habilitado en cualquier momento.
- Implementar distintos tipos de scripts sociales utilizando ese back-end. Para ello, se analizarán distintos sitios web, buscando y planificando posibles widgets junto con su comportamiento que aumenten el contenido de los sitios web. Posteriormente, se procederá al desarrollo de los mismos con el back-end ya desarrollado como soporte.
- Analizar estos scripts para detectar comportamiento común entre ellos y diseñar un framework que facilite el desarrollo de los mismos. El análisis será enfocado en todos los aspectos que conciernen el desarrollo de un widget, es decir, desde el componente visual y de maquetación, pasando a través de la lógica del widget, hasta la comunicación con el back-end, recepción y envío de datos.
- Reimplementar scripts como widgets del framework definido. Una vez finalizado el desarrollo del framework, serán reimplementados todos los widgets codificados utilizando el mismo, de manera que sirvan de muestra de la solución propuesta.

3. *Background*

En el presente capítulo se procede a introducir al lector en el contexto donde tiene lugar el accionar de la herramienta desarrollada, así como también su producto, representado por una aplicación de usuario tipo de la Web. Esta atmósfera de ubicación para el desarrollo estará compuesta de dos grandes conceptos en auge, los cuales serán combinados a lo largo de todo el trabajo para conseguir el objetivo propuesto:

3.1 Web Augmentation

3.1.1 Introducción

El concepto de Web Augmentation [12][1] es una propuesta para aumentar/customizar un sitio web, no solo sin necesidad de tener que reescribir o reinventar el código propio de la página, sino que sin siquiera tener que leerlo o investigar sobre el mismo.
En este capítulo veremos sus características y su utilidad dentro del mundo actual de la web, así como también sus implementaciones actuales.
El creciente volumen de contenido y acciones en el mundo de internet, combinado con el aumento de usuarios activos en ella, anticipan un incremento en el deseo de mayores formas sofisticadas de controlar la experiencia de la web. Solo basta imaginar la posibilidad de customizar un sitio web de miles de formas según un propio propósito. Hasta el momento, los mashups [13][14][15] son los precursores de esta tendencia, donde los consumidores (compañías y personas) producen nuevas aplicaciones mediante la combinación sinérgica de recursos de terceros. Esta tesina se basa en el enfoque de Web Augmentation (WA), en donde más que crear una nueva aplicación, construye en la parte superior de una ya existente. A diferencia de los mashups, el propósito de WA no es tanto el de crear un nuevo software, sino de enmarcar el reciente desarrollo dentro de la experiencia de usuario de una página web ya existente. Dado que esto se consigue por parte de terceros y por lo tanto no es introducida dentro del sitio original, WA es una tecnología que trabaja desde el lado del cliente: un amplio uso del lenguaje JavaScript usando herramientas de los navegadores (como por ejemplo GreaseMonkey y TamperMonkey en Mozilla Firefox y Google Chrome respectivamente) o mediante plug-ins.

3.1.2 Utilidad

El sólo pensar en el hecho en que un usuario de la Web tenga la posibilidad de tanto extender los contenidos que en ella se encuentran, como de editar los mismos, a fin de mejorar su experiencia de navegación o la de otros usuarios, abre un inmenso abanico de posibilidades a la hora de analizar las distintas acciones que puedan llevarse a cabo con estos objetivos; a punto de pensar que quizás el único techo al momento de ser creativo para modificar a gusto los sitios que navegamos día a día, sea nuestra propia imaginación.
Al reflexionar con el enfoque propuesto anteriormente, sólo tenemos que imaginar qué desearíamos agregar o modificar sobre lo que ya existe en la Web, o mejor aún, cómo podríamos hacer de la misma un lugar de acceso a la información con contenido todavía más y mejor al alcance de los usuarios de lo que hoy es. De esta forma, estaríamos razonando de una forma más amplia, dejando de lado las barreras que puedan representar en algunos casos, determinados aspectos de la navegación Web, y permitiéndonos la posibilidad de aumentar nuestra experiencia en Internet.

Con el objetivo de exponer algunas de las formas de explotar el concepto de Web Augmentation, se mencionan distintas utilidades del término para funciones específicas, volviendo al hecho de que cada usuario, parado en sus propias necesidades, imagina diversas formas de extender la Web:

- Web Augmentation enfocada a usuarios con dificultades visuales: con el objetivo de mejorar la experiencia Web de las personas con inconvenientes para percibir con facilidad tamaños de caracteres, formas, colores, etc; podríamos pensar en disponer de herramientas las cuales le permitan al usuario modificar el contenido de la Web según su necesidad. Esto puede ser: aumentar el tamaño de la letra de los sitios Web, definir las gamas de colores que usen los mismos según una determinada configuración, evitar el exceso de contenido, o el contenido espontáneo en distintos sitios.
- Web Augmentation enfocada a embeber contenido y "combinar" sitios Web: seguro más de un usuario Web alguna vez habrá pensado cuánto más fácil sería si pudiera reunir la información de determinados sitios, a los cuales accede a diario y se relacionan en una temática, en una Web común de acceso. Así, podríamos concentrarnos en aumentar un determinado sitio con información que provenga de otro. A modo de ejemplos imaginarios podríamos mencionar: el sitio de base de datos de películas y series de TV "IMDb", que posea en cada ficha de artículo, el vídeo de un trailer de YouTube que corresponda al contenido; artículos de un periódico Web, donde figuren además contenidos relacionados de otros diarios, de forma que se permita comparar opiniones; para sitios de mercadeo Web (Amazon, eBay, MercadoLibre), se podría sugerir otro destino en Internet para el mismo artículo.
- Web Augmentation enfocada a la traducción de sitios Web: muchas veces las traducciones que se generan desde los navegadores para determinados sitios, no son para nada exactas, dificultando y haciendo tediosa la experiencia en los mismos. De esta forma, podríamos pensar en la posibilidad que, mediante el mismo aporte de los usuarios, se soliciten y aporten traducciones en tiempo real del contenido de los distintos sitios.
- Web Augmentation enfocada a userStyles: con esto nos referimos a la idea de modificar los diseños que presentan los sitios web a los cuales accedemos habitualmente, según nuestra preferencia. Según el alcance que tenga la herramienta que provee el servicio, podemos pensar en editar cualquier aspecto visual de las páginas, como imágenes, colores, tamaños y tipos de fuente, posicionamiento de los elementos, etc.

3.1.3 Metodologías contemporáneas.

Actualmente existen comunidades de scripting y extensiones de navegador donde podemos ver aplicado el concepto de Web Augmentation en scripts ofrecidos en forma de "plugins", ya sea en comunidades libres de usuarios que presentan sus desarrollos, o en los mismos stores de extensiones asociados a cada navegador. Todos estos instrumentos "aumentan" y manipulan nuestro paso por la Web de una u otra forma, pero siempre con el objetivo de que el mismo sea de una forma más personalizada. Como ejemplos de esto podemos encontrar los clásicos bloqueadores de anuncios, también plugins de aplicaciones conocidas, las cuales agregan su "función clave" en los sitios de una forma más simplificada que en sus apps (Skype con sus botones de llamada donde hay teléfonos, o accesos rápidos a servidores de correo electrónico donde se visualizan direcciones de email).

3.2 Social Web

No caben dudas de que el factor Social en la Web ha sido en los últimos años (y lo seguirá siendo), uno de los más influyentes en materia de contenido para la Internet. Es así que los usuarios por sí solos generan información para la red y terminan por definir sitios donde todo el mundo puede interactuar con otras personas, dando origen también al término por excelencia para estos casos: Red Social.
Por lo descrito, podemos notar con facilidad el incremento exponencial del deseo de suplir una necesidad particular por parte de los usuarios, el estar conectados. Pareciera ser que el avance de la Internet y las aplicaciones móviles va tomando un curso el cual involucra cada vez más al usuario común y las conexiones entre los mismos, haciendo de los sitios Web y aplicaciones que invitan al usuario a relacionarse con otros generando contenido el cual pueda ser visto y comentado por toda una comunidad, los más famosos y solicitados en la actualidad.
Vivimos en un mundo donde el ser humano se interesa cada vez más por ser escuchado y formar parte de grupos de personas con intereses comunes, los cuales le permitan al mismo compartir comentarios, vivencias y cualquier otro tipo de experiencias, así como también interesarse por lo que otras personas tienen para aportar. De esta forma, con los pioneros de las mencionadas "Redes Sociales" como lo son Facebook y Twitter, han surgido (y surgen cada día) distintos sitios en la Web donde la principal fuente de contenido es el usuario mismo, el cual desea sentirse conectado con los demás.
Particularmente, se busca con el desarrollo de la herramienta, dejar a libre interpretación del usuario (desarrollador) la invención de nuevas formas de interacción para los clientes de Internet, esto es, desarrollar aplicaciones individuales (las cuales serán expuestas a continuación) que permitan a las personas conectarse por distintos aspectos (gustos, actividades, geografía, sitios web de interés, etc) y por distintas vías (chats privados, comentarios públicos en la web, llamadas y video, etc), aumentando así su experiencia en la Web.
Es por lo anteriormente expresado, que esta Tesina busca explotar el concepto de Web Augmentation junto al fenómeno social de la Web, de forma que los usuarios tengan la posibilidad de relacionarse en la red añadiendo contenido a la misma, el cual pueda ser visto en tiempo real por otros usuarios que naveguen los mismos sitios y utilicen la herramienta en cuestión. Así, las distintas formas de agregar contenido son tan grandes como la aumentación web en sí misma, y en el desarrollo van a estar representadas en forma de Widgets de interacción, creados por un desarrollador que utiliza el Framework provisto. Estos Widgets cumplirán funciones específicas, siempre relacionadas a la interacción entre los usuarios en los distintos sitios Webs, por medio de la generación de contenido de forma colaborativa.
Así, basándonos en el concepto de Groupware, el cual previamente definimos como el accionar de un grupo de personas con un objetivo común, que es total o parcialmente cumplido por el uso de algún software de colaboración, es que encontramos a los Widgets creados como el medio por el cual los usuarios ingresan contenido a la web colaborativamente, con el objetivo particular que haya sido definido al momento de su creación. En este último punto podemos citar distintos tipos de interacciones entre los usuarios que han sido propuestas de forma predefinida: realizar comentarios generales sobre un sitio Web, visualizando las opiniones de otros usuarios; interactuar directamente con otras personas de manera exclusiva, mediante el uso de un chat; crear encuestas Web sobre temas específicos de los sitios.
Como se puede apreciar, relacionar el concepto de Web Augmentation con el fenómeno social que estamos viviendo en la red, crea por sí mismo una visión de invención con un

potencial que sólo puede estar limitado por la imaginación de cada desarrollador, y del usuario mismo.

3.3 Resumen

Actualmente, los usuarios de Internet requieren de paginas más sofitiscadas, en donde puedan acceder al mayor contenido posible, y donde posea la mayor cantidad posible de funcionalidades para interactuar con este contenido. Es por eso que el concepto de WebAugmentation viene avanzando con el correr de los años. Como ya se dijo, la idea es permitir al usuario la mayor interacción posible con el sitio, asi como opciones para añadirle funciones y/o cambiar su interfaz visual de acuerdo a sus necesidades. Además, el incremento exponencial del uso del lenguaje JavaScript, incrementa a su vez esta idea, ya que es el lenguaje utilizado para desarrollar estas herramientas.

La presente tesina combina la idea de red social con Web augmentation, para que cualquier sitio Web existente pueda ser utilizado para iniciar algún tipo de actividad social con el resto de usuarios visitantes del mismo sitio, y que a su vez propone un esquema extensible por la propia comunidad.

4. *Overview de la propuesta*

4.1 Introducción

En este capítulo se realizará una breve explicación del funcionamiento del framework SocialEye, tanto de los componentes presentes en él como también de la comunicación entre todos los actores participantes. El objetivo es brindar un primer vistazo para entender a grandes rasgos la estructura de la aplicación y su interacción con las entidades que forman parte de la navegación web, antes de abordar específicamente aspectos más técnicos relacionados al desarrollo del framework. Se presentará un gráfico que demuestra este comportamiento y que será la base que justifique el razonamiento de dicha solución.

4.2 Comportamiento del framework

Para proceder con la descripción de los roles y componentes que componen el framework y las interacciones que existen entre ellos, se presentará un gráfico realizado con el objetivo de ayudar a clarificar el razonamiento.

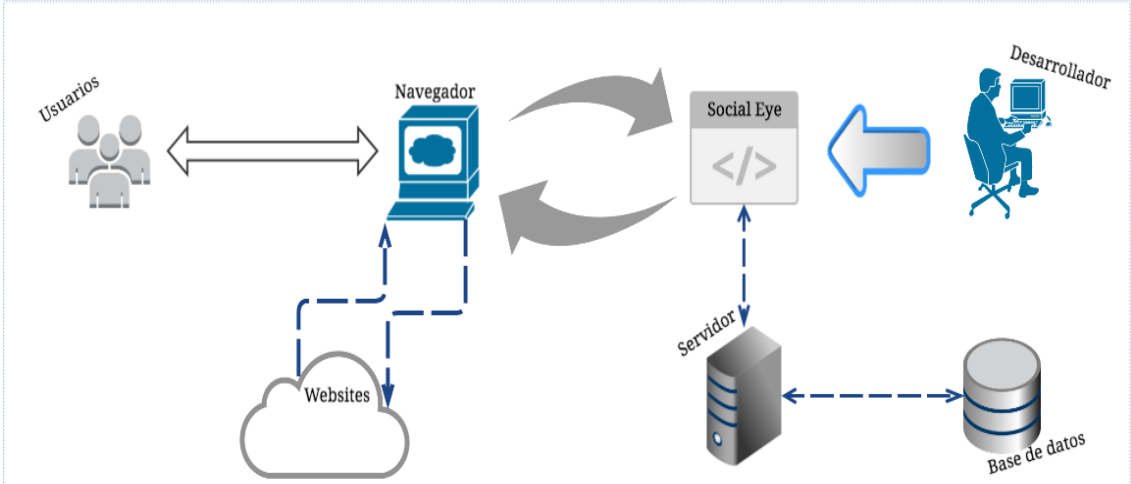

Figura 4.1 - Gráfico de perspectiva del Framework

Los roles que participan en la interacción son:

- Usuarios: aquellos que utilizan la herramienta de navegación y los widgets. Pueden ser también desarrolladores.
- Desarrollador: desarrolla widgets a través del framework SocialEye. Puede ser también un usuario.

Los componentes presentes son los siguientes:

- Navegador: permite el acceso a la web y almacena y administra extensiones.
- Websites: cualquier sitio web con sus respectivos servidores en todo el mundo.
- Servidor: responde a las peticiones de la herramienta de navegación y recupera y persiste los datos propios de cada widget en la base de datos.
- Base de datos: almacena la información del framework y de cada uno de los widgets.
- SocialEye: comprende tanto el framework como la extensión (herramienta de navegación) que se ejecuta en el navegador. Es el componente principal que se

encarga de realizar la interacción entre los usuarios y los widgets, dado que se comunica con el servidor para obtener y persistir la información ya sea del usuario como así también de todos los widgets almacenados, y visualiza su contenido en el navegador.

En primer lugar, centraremos el foco en el desarrollador, que utiliza el framework SocialEye para desarrollar un widget con el objetivo de ampliar el contenido y la comunicación entre los usuarios de la web. Este usuario, después de navegar en la página del framework y de leer su respectiva documentación, es capaz de diseñar e implementar un widget social sabiendo que dispone de una arquitectura que le brinda todo el comportamiento social.
Una vez que el widget está listo para ser consumido por todos los usuarios de SocialEye, el desarrollador es el encargado de subirlo desde el sitio web oficial del framework, y desde ese momento el widget está listo para ser ejecutado.
Este es el ciclo que se produce desde el lado del desarrollador, ahora bien, falta mencionar que es lo que ocurre con la otra sección del gráfico, es decir, de los usuarios que utilizan los widgets. Para ello, el framework cuenta con la herramienta de navegación, la cual será detallada más adelante, que es una extensión del navegador chrome, y que es el nexo entre las dos partes del gráfico. Recordemos que las extensiones [16] (también conocidas como complementos) son pequeños programas que se instalan dentro del navegador (Google Chrome, Firefox, Opera, etc.) y añaden o mejoran funciones del mismo. La herramienta de navegación de SocialEye se sitúa entonces sobre el navegador, insertando contenido JavaScript y HTML [17] en el DOM [18], por lo tanto la página web en la que estamos navegando nunca se entera de la existencia de esta, y esto posibilita la versatilidad de que sea ejecutada en todas ellas.

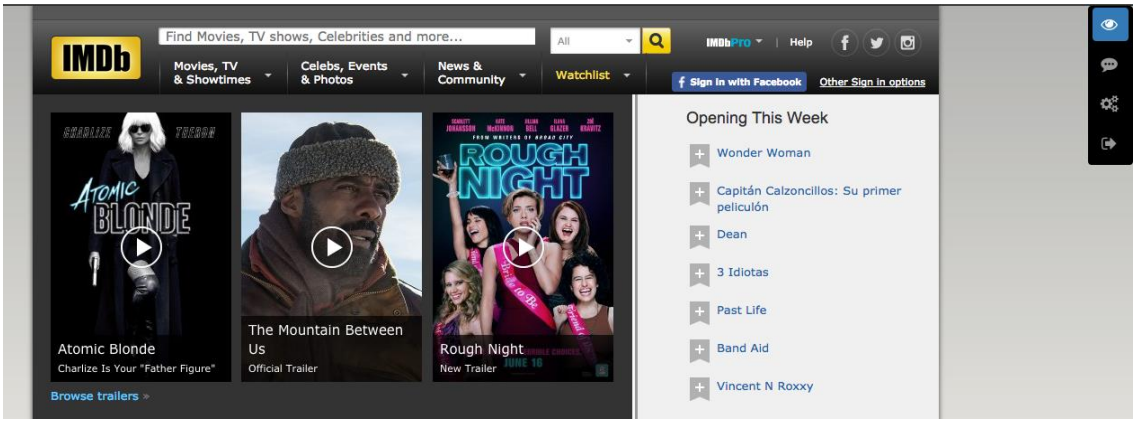

Figura 4.2 - Herramienta de navegación

El usuario de SocialEye de esta manera, al tener instalada la extensión en su navegador, puede utilizar los widgets existentes para augmenter cualquier Web e interactuar con ellos. Pero para que esta interacción sea posible, la herramienta debe comunicarse con el servidor de SocialEye, que a través de su base de datos le provee todos los datos de los widgets presentes y de la información que cada uno tiene persistida. Debemos hacer énfasis en que el sitio web en el que nos encontramos navegando, no es consciente de este circuito, ni de todas las conexiones HTTP que son realizadas entre la herramienta de navegación y el servidor de SocialEye.

4.3 Resumen

En este capítulo hemos brindado una breve introducción del funcionamiento del framework SocialEye, pasando desde el desarrollador que desarrolla un widget, hasta el usuario final que lo utiliza, mencionando la interacción que se presenta entre los componentes para permitir que la ejecución de la herramienta de navegación no afecte el sitio web visitado, que jamás se entera de la presencia de la misma. Además señalamos la importancia que tienen el servidor y la base de datos del framework, que posibilitan que cada widget persista la información para que posteriormente pueda ser consumida, incrementando la robustez de la solución.
Además, en la explicación del enfoque de la tesina, se aprovechó para definir de forma general las distintas entidades que forman parte del mismo. Aquí, se detalló brevemente el papel de cada uno, desde los usuarios finales de la herramienta, los cuales interactúan por medio del uso de los widgets, hasta los desarrolladores, quienes usan el framework como medio para generarlos. De esta forma, y apoyándose en los componentes "tecnológicos" como los sitios webs, el servidor para el manejo de datos y el framework mismo, se concibió el mapa general de la tesina, modelo que se hará referencia en los distintos capítulos de la misma.
El objetivo de este capítulo fue introducir al lector en la arquitectura general del enfoque de la presente tesina, brindándole un vistazo de su comportamiento para que en los capítulos siguientes, en donde se detallan todos los aspectos técnicos, se encuentre más familiarizado con el enfoque de la idea, finalidad de esta tesina.

5. *Framework Client-Side*

Después de haber brindado una generalización del enfoque propio de la tesina, procederemos en el presente capítulo a presentar la perspectiva del framework del lado del cliente (Client-Side), esto es, definir al mismo como una estructura la cual su lógica se ubica por completo en lo que llamamos "lado del cliente" del procesamiento, que no es más que el navegador en cuestión.
Mediante el uso del lenguaje de programación Javascript, sumado al de potentes librerías basadas en el mismo, concebimos una arquitectura de software la cual se caracteriza por una comunicación rápida y eficiente con el servidor de datos, mediante el cual obtiene dinámicamente la información necesaria de cada Widget, para luego ejecutar todo el procesamiento en el navegador del usuario.
Si nos enfocamos en lo que conllevaría el desarrollo de un widget, nos encontramos que todos los usuarios del mismo se enfrentarían a los mismos desafíos:

- Diseñar un modelo que se adecue a los datos que el widget necesita persistir para luego reutilizar.
- Crear los componentes visuales, que posteriormente se expondrán en cualquier sitio web
- Realizar la comunicación entre los componentes visuales y el modelo diseñado.

Fácilmente podemos observar que decantamos en el patrón MVC (Modelo-Vista-Controlador), que puede no ser conocido por cualquier persona que recientemente se haya iniciado en el mundo de la informática. Por lo tanto, este framework puede resultar de mucha utilidad, dado que resuelve en su gran mayoría la implementación de dicho patrón, dejando solamente los detalles particulares de cada widget al desarrollador. Además, es muy probable que esa persona realice sus primeros pasos en tecnologías client-side (HTML,CSS,JS), dejando de lado aquellas relacionadas al back-end, por eso es importante proveerles una API que se encargue de comunicarse con el servidor, genérico, desarrollado para todos los widgets.
Por último, no podemos dejar de lado que cada widget debe funcionar en forma coordinada, se ejecute cuando deba ejecutarse, realice peticiones al servidor cuando lo necesite, cuestiones que el desarrollador en este caso no debe saber, sino que las utiliza de forma innata, las hereda, enfocándose solamente en los aspectos técnicos propios del widget que va a desarrollar, como es su funcionalidad, y no en cuando su widget se va a ejecutar y cuando va a dejar de hacerlo. Este concepto, denominado inversión de control [19], junto con lo mencionado previamente, motivaron a realizar el framework desarrollado en esta tesina.
Con esta introducción, nos embarcamos en la especificación de cómo están logrados los detalles que hacen al Framework del lado del cliente, definiendo APIs [20] y dando detalles de los principales métodos, comunicación con el servidor de datos, librerías utilizadas en la implementación y otros aspectos que serán tratados en el desarrollo del apartado actual.

5.1 Estructura

Al descomponer la arquitectura de la herramienta en el cliente, nos encontramos con que la misma está compuesta por dos entidades definidas:

Manager: objeto que cumple el rol de definir las funcionalidades de gestión de la herramienta instalada en el navegador. Con estas funciones, las cuales varias son widgets generados con el framework, nos referimos a: login/logout, registro, configuración de widgets del usuario, recuperación de widgets, etc.
Framework: el framework propiamente dicho, el cual se compone por:

Clase Widget: la clase principal y razón de ser del Framework. Mediante la instanciación de esta clase comenzará para el desarrollador la invención de un nuevo Widget, representado por un objeto que lleva su mismo nombre y al cual se le definirán las propiedades y comportamiento deseado.

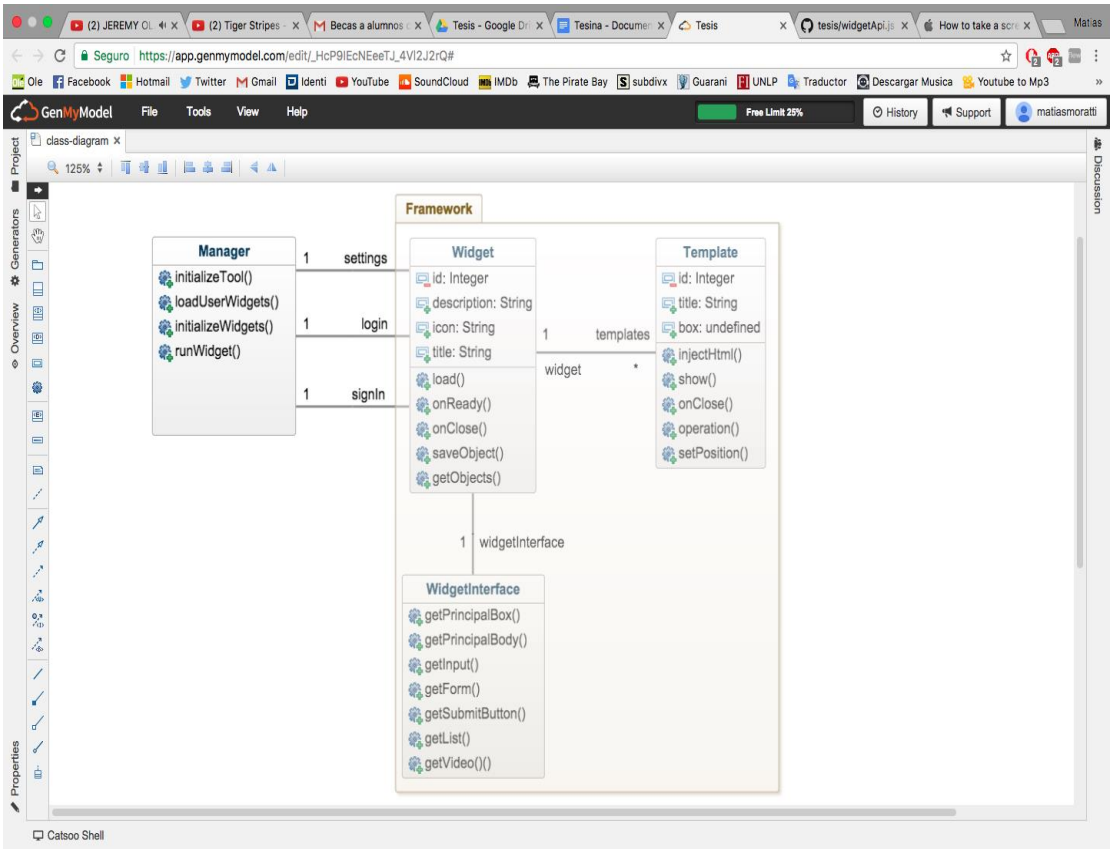

Figura 5.1 - Modelo de clases

La clase Widget simboliza en una gran estructura encapsulada, la mayor parte de la idea del Framework, ya que al definir objetos nuevos estamos creando partes inéditas de la herramienta asociada, las cuales tendrán su grado de originalidad e innovación como principales aportes a la misma.
Al crear un nuevo objeto Widget, el desarrollador tendrá la tarea de plasmar en él la idea que propone, mediante la definición de las distintas propiedades y comportamientos que presentará el complemento. Aquí, el mismo se va encontrar con la labor de definir a su Widget aspectos tales como un nombre, una descripción general donde detalle en pocas palabras su objetivo y utilidad, un icono que lo represente, entre otros.
En cuanto a los eventos que definirán el accionar del widget, el desarrollador tendrá a su disposición distintos métodos que deberá redefinir para lograr que su creación se comporte como tiene previsto. Estos métodos determinarán cómo responde el widget en determinadas situaciones:

- Cuando es abierto por el usuario: qué ventanas, iconos o imágenes debe mostrar, qué funciones debe habilitar o deshabilitar, etc.
- Cuando el widget está listo: ya fue seleccionado por el usuario, se abrió y visualizó el contenido programado.
- Cuando el usuario oculta el widget: si el programador decide que le complemento debería realizar alguna acción cuando el usuario lo cierra, aquí se ubica ese comportamiento.

De esta forma, el programador define lo que vendría a ser el esqueleto del widget, la estructura base por la cual inicia su desarrollo, el cual será complementado luego con el uso de las facilidades que provee la API del Framework a presentar en el transcurso de esta sección.

Diseño del widget: aquí la herramienta hace un apartado para definir la lógica de cómo los widgets presentarán visualmente la información al usuario. Para esto, el Framework ofrece dos posibilidades al programador, totalmente combinables en el desarrollo:

- Objeto Template: representa un archivo con código HTML el cual es asociado a un widget para mostrar la información que corresponda en determinado momento. Cada widget tendrá ligados un conjunto de templates, los cuales serán visualizados cuando el desarrollador lo defina. Para esto, el widget entiende mensajes que harán que determinado template que se le haya asociado, se visualice en el navegador.

  Dejando los detalles del funcionamiento de los templates para más adelante, podemos destacar que un objeto template tendrá en principio una estructura de ventana definida, a fin de mantener un estilo para la herramienta. Luego, estará la posibilidad de "inyectarle" a este template archivos con código HTML, a medida que el widget requiera presentar determinada información. A su vez, cada creación de un template e inyección de código HTML, podrá llevar consigo un conjunto de datos dinámico los cuales el usuario presentará en la sección de código que vea conveniente.

  Un objeto template deberá presentar un título para la ventana a mostrar, y tendrá la posibilidad de editar dinámicamente aspectos como la posición en la pantalla del navegador, el ancho y el alto de la ventana, así como también determinado comportamiento al cerrar la misma.

  La clase template y su funcionamiento tienen como propósito ofrecer libertad al desarrollador a la hora de diseñar su widget. Este método de creación de la parte visual de los widgets está destinado a desarrolladores con conocimientos en el lenguaje HTML, para que tengan la posibilidad de explotar su saber con el fin de diseñar un widget con una interfaz lo más expresiva y amigable posible.
- Objeto WidgetInterface: representa un único elemento asociado a un widget en el momento de su creación, y tendrá como objetivo la creación de la estructura de elementos visuales que correspondan al mismo. Para esto, la clase WidgetInterface ofrece una API cuyos métodos tienen como objetivo simplificar la creación de los elementos HTML que harán a la visual del widget. Estos métodos cumplirán la función de retornar al usuario un objeto que represente un elemento HTML ya creado, o bien una estructura de elementos ya dispuesta que simbolice, por ejemplo, una ventana con una lista asociada, un ícono con un diseño específico, un formulario, etc.

  Como podemos apreciar, la clase WidgetInterface está dedicada a ofrecer una forma de diseño más rápida y sencilla en la creación del widget, otorgando una completa API al desarrollador que puede que no tenga suficientes conocimientos

de HTML o simplemente desee crear la parte visual de su widget mediante una vía más ágil. Cabe destacar la evidencia de que, mientras un objeto WidgetInterface nos ofrece obtener el diseño de nuestro widget sin hacer grandes esfuerzos, puede que a su vez nos limite en los casos de desear un diseño específico.

Sea cual fuere el método de implementación que el desarrollador desee emplear para el diseño de su widget, el Framework ofrece la posibilidad de asociar a cada complemento creado distintas plantillas de código CSS [21], con el fin de perfeccionar la visual de cada widget de la forma que el programador disponga.
Así, dependiendo de la forma de implementación de la interfaz (se destaca nuevamente que es posible combinar ambas), el desarrollador podrá asociar propiedades a los elementos del widget directamente en sus templates HTML (caso de uso de la clase Template) o mediante código Javascript (caso de uso de la clase WidgetInterface).
Como se ha podido apreciar, ambas vías de desarrollo de la interfaz gráfica de un widget sos totalmente combinables. Siendo incluso el empleo de las dos técnicas muchas veces la mejor práctica a la hora de la implementación. Esta apreciación se sustenta en el hecho de que algunas veces podemos desear solo inyectar una porción pequeña de elementos al diseño, por lo que quizás no sería la mejor decisión crear un archivo HTML para este fin. En estos casos, puede que utilicemos el objeto template para representar grandes conjuntos de elementos o diseños complejos, y que inyectemos, mediante el uso de la API que nos ofrece WidgetInterface, las porciones más pequeñas de elementos, por ejemplo un nuevo componente a una lista.

5.2 Lenguajes y herramientas utilizados

En la presente sección se procede a realizar una breve descripción de los lenguajes de programación que han sido utilizados para desarrollar la sección client-side del Framework, así como también las librerías y otros instrumentos que han sido útiles para agilizar y perfeccionar su creación.
Algunos de los lenguajes y herramientas en los cuales se sustenta el Framework han sido mencionados previamente en las distintas secciones del documento. Procedemos aquí a realizar una descripción de cada uno, junto a su aporte en este desarrollo específico:

- Javascript: como no podía ser de otra manera, el desarrollo del Framework que implica su implementación del lado del cliente se encuentra, casi en su totalidad (haciendo un apartado a la parte visual), definido por el lenguaje de programación Javascript.

Ya asentado como el lenguaje por excelencia para ser interpretado en un navegador, Javascript nos provee, junto con sus librerías, de todo lo necesario para la creación de una herramienta la cual se presenta como un complemento del mismo:

- Programación orientada a objetos [22].
- Tipado débil y dinámico para ofrecer libertad al desarrollador a la hora de crear objetos que manejen los datos específicos de su widget.
- Sencilla y eficaz conexión asincrónica con el servidor de datos por medio de librerías.
- Manejo simple de los elementos del DOM.
- Variedad de desarrollos posibles mediante el uso de un gran repertorio de librerías basadas en el lenguaje.

Entre las numerosas cualidad que posee el lenguaje, el hecho de que Javascript posea un tipado dinámico fue de las más provechosas para este desarrollo. Así, como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una cadena.
Esta característica es explotada en gran medida en el desarrollo del Framework y además brinda una gran flexibilidad al desarrollador a la hora de crear los objetos que den forma a la representación de su widget.
A fin de evidenciar de forma más clara este concepto que se remarca, decimos que aprovechamos el tipado dinámico de Javascript principalmente por el hecho de que, en la herramienta desarrollada, el almacenamiento de los datos asociados a los widgets no sigue una estructura determinada. La arquitectura del Framework carece de un modelo relacional fijado para cada uno de los widgets. Por esto mismo es que el usuario tiene la posibilidad de crear sus propios objetos con las características deseadas y sin la necesidad de seguir una estructura o tipado determinado. De esta forma, los elementos se persistirán asociados al widget, para que el mismo complemento pueda consultarlos, editarlos o eliminarlos durante su funcionamiento.
Por otro lado, cabe destacar que, en Javascript, las funciones suelen ser llamadas "ciudadanos de primera clase"; a fin de definir a las mismas como objetos en sí mismos, y como tales, poseen propiedades y métodos. Por esto mismo, utilizamos las mismas para representar todos los objetos del Framework.

- HTML: propuesto por el Framework como una de las dos formas de presentar la interfaz de un widget, el lenguaje HTML nos brinda esta posibilidad de una forma simple y a su vez muy flexible para la representación.

Como lenguaje de marcado, HTML representará la estructura visual de los widgets en forma de elementos HTML, los cuales simbolizarán texto, imágenes, videos, links, etc.
De la forma que el lenguaje lo propone, el usuario podrá definir atributos específicos para los elementos, tanto de estilos, eventos, identificadores, como cualquier otra propiedad alcanzada por el lenguaje.
Una vez definidos los dos principales lenguajes en los cuales se basa el Framework del lado del servidor, procedemos a la importante tarea de explicar cómo se conjuntan y relacionan entre ellos, a fin de definir la lógica del widget para luego visualizar la información y funcionalidad al usuario.
Para cumplir este objetivo, el Framework hace uso de la librería Javascript Underscore.js [23], la cual con unas pocas líneas de código, permite crear un template con código HTML en un objeto y, lo que es mejor, visualizar datos en el código HTML referenciando los objetos Javascript que se manejan en la lógica de la aplicación. Además, la librería termina por brindar la interpretación de código Javascript en cualquier parte del HTML, por lo que tendremos la posibilidad de crear bucles para iterar sobre los datos, sentencias condicionales, etc.
De esta forma, y antes de explicar la implementación en detalle en la sección *Api para el desarrollador*, podemos a esta altura hacernos una idea del funcionamiento del framework, el cual en primer medida recupera objetos almacenados en la base de datos, opera con ellos en base a la implementación del desarrollador, y finalmente la información almacenada en los mismos llega al usuario por la creación de un template en manos de esta muy útil librería.

- CSS: el Framework brinda la posibilidad al usuario de asociar a su desarrollo archivos de hojas de estilo CSS.

Como lenguaje de especificación visual, CSS tiene como objetivo definir y crear los atributos para los elementos de un documento escrito en un lenguaje de marcado, como HTML.
De esta forma, como bien mencionamos en la descripción del lenguaje HTML, el desarrollador podrá definir propiedades en el lenguaje CSS para los elementos de los templates que haya creado.
A esta altura, asociando los lenguajes Javascript, HTML y CSS al Framework, podemos ya dejar en evidencia la correcta separación del código en sus distintas funciones, así como también el potencial de la herramienta, la cual nos permite un completo desarrollo de los complementos, integrando lógica, interfaz y diseño.

- JQuery [24]: conocida como la librería Javascript más utilizada, nos da la posibilidad en primer medida de simplificar la manera de codificar en el lenguaje, ofreciendo funciones rápidas con las cuales la mayoría de las ocasiones hacemos más, escribiendo menos código. Así mismo, nos facilita la interacción con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción utilizando AJAX [25], lo cual se detalla a continuación.

- Ajax: pieza clave de todo el desarrollo del Framework, representa una técnica de desarrollo web para crear aplicaciones las cuales se ejecutan en el cliente, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.
Mediante el uso de Ajax, explotamos su característica asíncrona, para brindar desde la herramienta la posibilidad de solicitar los datos al servidor (widgets a cargar, objetos asociados a los mismos) y cargar los mismos en segundo plano sin interferir con la visualización ni el comportamiento de la página.
Como se podrá imaginar, al desarrollar una herramienta que trabaja "por encima" de cualquier página web, el asincronismo para recuperar los datos resulta fundamental en la constante comunicación con el servidor.

- Bootstrap [26]: se utiliza el famoso Framework para definir la base del diseño de la herramienta. A fin de proyectar una interfaz pareja en toda la estructura de la herramienta de navegador, se utilizan algunas de las clases de Bootstrap, las cuales ofrecen propiedades que brindan un grado de detalle apropiado para la aplicación.
Aprovechamos este apartado para mencionar un concepto que se expondrá en detalle a la brevedad, que consiste en el hecho de que el Framework impondrá ciertas "reglas" para el diseño a los desarrolladores. Esto dado que se busca mantener una visual uniforme en toda la herramienta. Por lo tanto, se ofrecerá la base del diseño de Bootstrap, realizando aclaraciones sobre la libertad que tendrán los usuarios para definir detalles propios.
Como se imaginará el lector, estas "limitaciones" en el diseño estarán relacionadas con propiedades que puedan significar un cambio sobre la visual considerado abrupto o inesperado para el usuario. Ejemplos de estas propiedades podrían ser: color, tipo y tamaño de fuente, rellenos, resaltados, etc.

5.3 Comunicación con el servidor y persistencia de datos

Si bien se hizo hincapié en este capítulo en el hecho de que el Framework, y por lo tanto la herramienta generada a partir de este, realizan la mayor parte del procesamiento del lado del cliente en código interpretado por el navegador, es de suma importancia aclarar el concepto de que SocialEye se ubicará, como se dice, "por encima de cualquier página web", funcionando como un complemento del navegador. Esto lleva a entender que, cualquiera sea la página que estemos navegando, la herramienta estará disponible para su uso particular.
Aclaramos el concepto anterior aquí con el fin de detallar cómo la herramienta, del lado del cliente, lleva a cabo la actividad fundamental de comunicarse con el servidor de datos para tratar con la información. Así, al ubicarse la aplicación delante de la web que estemos navegando, no existirá el evento de "recargar" la página que implica una comunicación sincrónica con el servidor. En cambio, el trato con el servidor de datos se realiza de manera asíncrona, manteniendo la comunicación abierta en segundo plano. Esto, como se mencionó previamente, lo logramos a partir del uso de la técnica AJAX de la librería JQuery, la cual nos permite realizar peticiones al servidor de datos y obtener la información de respuesta en un mismo bloque de código, rápidamente y sin recargar ninguna página, hecho que no se admite en nuestro caso.
A modo de ilustrar el funcionamiento de AJAX en la herramienta, mostramos a continuación un fragmento de código en el cual se define una función interna al Framework, encargada de recuperar todos los widgets creados por medio de una invocación AJAX:

```
function getWidgets(){
    var data;
    $.ajax({
        url: "https://127.0.0.1:8000/widgetRest/widget/",
        type: "GET",
        dataType: 'json',
        async: false,
        data : {
        },

        success: function (response) {
            data = response
        },

        error: function (xhr, errmsg, err) {
            alert("Error al intentar recuperar los widgets");

        }
    });
    return data;
}
```

Figura 5.2 - Recuperación de Widgets

Tan solo con definir la ruta específica al servidor y propiedades del llamado (método de envío de datos, tipo de datos a procesar, método de comunicación), tenemos un requerimiento al proveedor de datos donde evaluaremos también qué hacer ante una comunicación exitosa (success), o fallida (error).
Un detalle importante a remarcar aquí, es el hecho de que toda transferencia de información entre el servidor y la aplicación va estar definida por objetos JSON [27], el formato de texto ligero para el intercambio de datos por excelencia en Javascript. Los objetos JSON son colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y puestas entre llaves. El nombre tiene que ser una cadena y el valor puede ser de cualquier tipo. *Ejemplo:*
{"departamento" : 8,"nombredepto" : "Ventas", "director" : "juan rodriguez" , "empleados" : [{"nombre" : "Pedro", "apellido" : "Fernandez"} , {"nombre" : "Jacinto", "apellido" : "Benavente"}]}

La programación en Javascript asociada al uso de JSON para la transferencia de datos termina siendo una excelente combinación para nuestro propósito, ya que como trataremos más adelante, mediante esta filosofía es que permitimos a los usuarios definir sus propios objetos y almacenarlos a su gusto para luego poder consultarlos cuando su aplicación lo requiera, todo por medio de la API del Framework.
Así, como se supo explicar en la sección *Servidor de datos*, los objetos creados por el usuario son persistidos en la base del servidor con el mismo formato JSON, siempre asociados al widget en cuestión.
Habiendo expuesto brevemente la definición de comunicación asincrónica con el servidor y el tratamiento de los datos que involucran a la herramienta, podemos anticipar algo de lo que explicaremos con más detalle en la sección donde se exhibirá la API para el desarrollador, y es que la misma nos proveerá de una extensa variedad de métodos con los cuales podremos recuperar objetos definidos bajo cualquier tipo de dato, actualizar los mismos a gusto (incluso modificando su estructura y atributos), crear nuevos asociados al widget y eliminar existentes.

5.4 Api para el desarrollador

5.4.1 Extendiendo de la clase Widget

Es momento de comenzar a desarrollar nuestro widget, y para ello, como primer paso, debemos extender de la clase Widget, la de mayor influencia del framework, y la que nos va a permitir heredar las funciones principales y tambíen modificar otras según lo que el usuario desee. El constructor de la clase Widget no recibe parámetros, por lo tanto, para instanciar un objeto Widget simplemente hay que crearlo y asignarlo a una variable para poder comenzar a implantar el comportamiento que deseamos para nuestro widget.
La clase Widget está compuesta por 8 atributos, de los cuales dos no son vistos ni utilizados por el usuario, sino que refieren a una función propia, privada, de cada widget. A continuación mencionaremos la utilidad de cada una de las propiedades:

- idWidget: es el identificador de cada widget, y es asignado automáticamente por el framework, por lo tanto el usuario no debe preocuparse por este campo.
- descripcion: breve reseña de la utilidad y función que brinda el widget.
- icono: este campo está compuesto por la identificación que posee cada uno de los iconos pertenicientes al sitio http://fontawesome.io/. Dado que el objetivo es mantener una armonía entre las vistas de cada uno de los widgets, decidimos

que los mismos utilicen íconos de este sitio para mantener un cierto estilo. Desde ya, este campo debe ser inicializado por el usuario.

- intervalPing: este atributo no es visible para el desarrollador, sino que su función es ser un cronómetro en donde, cada un tiempo definido, se realiza una llamada al componente servidor para indicar que el usuario sigue utilizando, o no, el widget correspondiente.
- title: es el nombre del widget, por lo tanto debe ser conciso y bien descriptivo para que la leerlo ya se pueda deducir de qué trata su utilidad.
- Interface: objeto correspondiente a la clase WidgetInterface que, como se explicó anteriormente, puede ser utilizada como no, dependiendo del grado de conocimiento del desarrollador en cuanto al aspecto de diseño. Más adelante, en este mismo capítulo, se detallará las funciones que brinda.
- filesHTML: este campo se identifica con la segunda posibilidad que el framework otorga a la hora del diseño. Tal como se mencionó previamente, en este atributo se almacenan los archivos de tipo HTML que el desarrollador brinda juntó al desarrollo de su widget, para pulir con más detalle el componente visual del mismo.
- templates: atributo relacionado con el campo anterior, dado que el mismo contiene el diccionario de datos que va a utilizar, si es necesario, cada archivo HTML que posee el widget. Cada template está compuesto por el id, el título del widget, el widget actual, y una variable de tipo boolean, que indica si el template debe ser mostrado en una nueva caja, o simplemente sin una caja que lo contenga.

5.4.2 Redefiniendo el comportamiento del widget

Los 8 atributos mencionados en la sección previa son fundamentales para comenzar a desarrollar nuestro widget, pero nos está faltando un detalle muy importante también, que es el de otorgarle al widget el comportamiento que deseamos para cada una de las etapas del ciclo de vida del mismo. Estas son:

- Inicio: es decir, cuando el widget es cargado en la herramienta del framework
- Apertura: el momento en que el widget es abierto en la página web.
- Cierre: cuando se cierra el widget.

A continuación detallaremos los métodos que heredamos de la clase Widget, y que debemos implementar para definir las acciones que tomaremos en cada uno de los ciclos de vida del mismo:

`loadWidget = function ()`: es la primera función llamada cuando un widget es puesto en ejecución. En ella, el usuario debe escribir el código necesario para que su aplicación inicie y esté lista para ser consumida. Entre otras acciones, en esta función se podrían obtener los datos de la aplicación almacenados en la base de datos del framework, y crear la primera vista de la misma con estos datos obtenidos, si es que es necesario. Es requisito obligatorio que esta función sea implementada por el usuario.

`getObjectsInUrl = function()`: función que nos provee los objetos almacenados de nuestra aplicación en la base de datos del framework. Puede ser utilizada en el método anterior, cuando mencionamos como posible acción para iniciar nuestro widget a obtener la información ya persistida del mismo. Esta función recibe un parámetro como mínimo, aunque si el usuario lo necesita, puede sumarse uno más. El primero, obligatorio, es la url de la cual se está pidiendo los datos, dado que los mismos, cuando son almacenados, se identifican primero con el widget en cuestión, desde ya, pero también con la url de donde fueron enviados. El segundo parámetro,

opcional, es un mapa de tipo clave valor, utilizado para realizar la consulta para obtener los registros. Esto significa, que se puede filtrar la consulta simulando una de tipo SQL, en donde cada clave de este mapa vendría a ser cada atributo en la cláusula "where", mientras que el valor, el tipo buscado para ese campo. Si bien se mencionó que esta función se puede utilizar, entre otros casos, cuando se inicializa el widget, está claro que también puede ser llamada desde cualquier otro método siempre que se necesite obtener datos del servidor. Además, esta función no es implementada por el usuario, sino que es brindada por el framework y solo debemos utilizarla cuando lo deseemos.

`onReady = function ()`: hablamos de dónde escribir las primeras acciones a realizar cuando nuestro widget es ejecutado, ahora bien, existe el caso en que un usuario que está consumiendo nuestra aplicación, cierra la ventana principal de la misma, por el motivo que fuere, y que luego de un momento, desea volver a abrirla. Para estos casos, esta función es la encargada de ejecutarse, y en ella debemos asegurarnos de hacer lo necesario, si hace falta, de dejar en estado consistente nuestro widget para que sea nuevamente utilizado. Esto puede ser, por ejemplo, volver a obtener datos de la base de datos, para actualizarlos y que el usuario visualice constantemente la información lo más actualizada posible. También podemos mostrar un mensaje, o cambiar lo que sea necesario en el diseño. Por último, mencionar que esta función no recibe parámetros.

`onCloseWidget = function ()`: posteriormente a haber presentado los principales métodos a implementar para cumplir con el ciclo de vida de un widget, nos resta saber qué ocurre cuando el usuario cierra la ventana del mismo. Previamente mencionamos la función que se debe implementar cuando se restaura la aplicación, ahora es el turno de la acción contraria. Este procedimiento no es obligatorio de implementar, a menos que se requiera hacer alguna acción con el estado actual del widget, o con el diseño. De otro modo, el framework se encarga de ocultar la ventana principal, y de restaurarla cuando sea necesario, para invocar a la función onReady() descrita anteriormente.

`getUsersConnected = function ()`: esta función fue creada como ayuda para aquellos widgets que necesiten obtener la lista de usuarios conectados en tiempo real en la página actual. En casos donde se requiera estos datos, como puede ser en un chat (se observará posteriormente en el widget de chat), puede ser muy útil. Por si es necesario, este método posee una variante (`getUsersConnectedInWidget`), en donde se retornan solo los usuarios que estén utilizando el widget específico en cuestión.

`isUserConnected = function ()`: se desprende de la función anterior, y su objetivo es retornar true, si el usuario que recibe como parámetro, se encuentra actualmente conectado en la página actual. La función retornará false en caso contrario. Similar al procedimiento explicado anteriormente, también existe una variante (`isUserConnectedInWidget`), que retorna true solo si el usuario se encuentra en linea en el sitio y en el widget actual en cuestión.

Hasta aquí hemos visto las funciones que nos brinda SocialEye para unir nuestro widget con el framework, y que debemos implementar, a excepción de `getObjectsInUrl()`, que ya está dada y solo debemos invocarla cuando la necesitemos. Es momento de centrarnos en otras de estas funciones, relacionadas con el manejo de datos, y persistencia de los mismos, que también heredamos de la clase Widget, y que nos ayudarán en gran medida a desarrollar nuestra aplicación.

`getWidgetElement = function ()`: función que como su nombre lo indica, retorna uno o varios elementos de nuestro widget, dado el parámetro enviado. Se debe aclarar, que en este caso lo que se trata de hacer, es obtener un elemento del

componente visual de nuestro desarrollo, y no de la base de datos, para ello, ya explicamos la función `getObjectsInUrl()`. Es una de las más utilizadas del framework, dado que muy probablemente en muchos lugares de nuestro desarrollo vamos a necesitar pedir objetos de nuestra vista. El parámetro que debe recibir es un selector css, otorgando flexibilidad al usuario de cómo identificar a cada elemento HTML que se inserta en un template, y permitiendo obtener varios elementos que posean un mismo selector.
`saveObject = function ()`: cuando necesitemos persistir un objeto de nuestro widget a la base de datos, esta es la función a llamar. Recibe como parámetro un objeto javascript, que posteriormente es trasladado a un objeto JSON y almacenado. La función retorna 0 en caso de error, y cualquier otro número cuando el resultado haya sido exitoso.
`updateObject = function ()`: si necesitamos actualizar un objeto ya persistido, sin crear uno nuevo, esta es la función a invocar. Los parámetros que debe recibir son en primer lugar el objeto a actualizar, similar a como si se quisiera persistir uno nuevo. En segundo lugar, se debe enviar los parámetros por los cuales se identificara desde el lado del servidor, el objeto que se desea actualizar, en forma de clave valor como se explicó anteriormente.

5.4.3 Creando una interfaz propia

Hasta el momento se ha hecho hincapié mayormente en la parte lógica que nos brinda el framework para construir nuestro propio widget. Es el momento de centrarnos en el aspecto visual, un componente no menor, muy importante para el usuario final de un widget, que desea tener un alto nivel usabilidad a la hora de utilizarlo. El framework SocialEye fue desarrollado pensando en dos tipos de usuarios con respecto al diseño web: aquellos que no poseen mucho conocimiento en cuanto a la confección de un diseño HTML, y aquellos que saben un poco más, a tal punto de poder codificar un archivo HTML y crear un sitio web. Por lo tanto, el componente visual puede ser desarrollado utilizando ambas, o alguna de las dos soluciones, que serán detalladas a continuación. Cabe destacar, que si nos inclinamos por la primera opción, el resultado será un diseño más simple, intuitivo y útil al fin, pero más simple que la segunda opción, en donde el usuario puede diseñar el widget a su gusto.

5.4.3.1 Inyectando templates (enfocado a diseñadores)

Comenzaremos entonces con el enfoque dedicado a usuarios con más experiencia en desarrollo, en donde él mismo puede diseñar el/los archivos HTML, e inyectarlos de manera muy sencilla.
Objeto Template: la clase principal para inyectar nuestros archivos HTML, es la clase Template. Representa una caja si nos referimos al diseño, el contenedor del widget, en donde dentro de él, estarán los elementos que forman la aplicación en sí. Cada widget puede disponer de tantos templates como necesite para poder lograr el apartado visual que se busca. Cada uno de ellos se compone de tres campos, todos ellos definidos por el usuario cuando se dispone a crear un template. A continuación, se describe la función de cada uno:

- idTemplate: id del template, identificación única que escogerá el usuario para cada template que cree.
- title: título que se observará en la esquina del template, es decir, es el nombre que tendrá, y posteriormente será visualizado por toda aquella persona que utilice el widget ya desarrollado.

- htmlFile: nombre exacto del archivo HTML que está guardado en el el archivo .zip que se adjunta cuando se sube al servidor el widget ya terminado. Es de carácter obligatorio que el valor a introducir en este campo incluya también la extensión HTML.

El uso y significado de cada uno de estos campos, se verá mejor reflejado en el siguiente capítulo, donde se mostrarán los widgets construidos utilizando el framework ya desarrollado, incluyendo imágenes tanto del código como del widget ya finalizado.
El proceso para crear y agregar un template al widget es muy sencillo, simplemente hay que invocar a la función `createTemplate()`, heredada de la clase Widget, de la siguiente manera:

```
widget.createTemplate(idTemplate, title, htmlFile, data, boxInvisible)
```

Los primeros cuatro parámetros fueron descritos previamente, mientras que el último forma parte de la manera visual que se desea mostrar el archivo HTML envíado. Un template, como se especificó, fue desarrollado con el concepto de ser una "caja", que contenga los objetos HTML que el usuario codificó. Esta "caja", permite agrupar lo agregado por el usuario de forma de mantener una cohesión en el diseño del framework y de cada widget construido. Sin embargo, puede darse el caso que ciertos archivos HTML no tengan la necesidad de estar dentro de la caja de un template, dado su diseño o finalidad, y simplemente se muestren flotando en el sitio web en cuestión. Para elegir una u otra opción, se encuentra el quinto parámetro (`boxInvisible`), un booleano que inserta los elementos del archivo HTML envíado por el usuario dentro de una nueva caja cuando su valor es verdadero (true), y lo hace de manera suelta en el página web cuando su valor es falso (false). Una vez invocada esta función, el componente vista de nuestro widget estará visible en el navegador para poder ser utilizado, y a partir de allí se abre un abanico de posibilidades que el programador puede aprovechar para manipular cada uno de los templates que conforman su widget. Estas funciones son las siguientes:

- `closeTemplate = function()`: cuando se requiera cerrar uno de los templates del widget, esta es la función a llamar. Recibe como parámetro el id del template en cuestión.
- `onCloseTemplate = function()`: en ciertos casos puede ser necesario realizar alguna acción a la hora cerrar un template. Para ello, debemos invocar esta función, enviando como parámetros en primer lugar el id del template, y en segundo lugar, una función, que será llamada cuando el método anterior sea llamado.
- `showTemplate = function()`: carga en pantalla el template con el id recibido como parámetro.
- `setTemplatePosition = function()`: función dedicada exclusivamente al aspecto visual. El objetivo es otorgarle al template la posición que se desea dentro de la página web. Para eso, este método recibe tres parámetros, el id del template, un top, y un left, en donde los últimos dos refieren a un número entero que representa la respectiva posición.
- `setTemplateWidth = function()`: especifica el ancho de la caja que contiene al template. Recibe como parámetro el id del template y el número con el ancho correspondiente.

- `setTemplateHeight = function()`: similar a la función anterior, pero dedicada a darle un alto correspondiente al widget. Recibe los mismos parámetros que la anterior.
- `injectInTemplate = function()`: puede darse el caso, en que necesitemos añadir un archivo HTML a un template ya creado, dependiendo del contexto de ejecución del widget desarrollado. Para ello, existe esta función, que recibe el id del template (al que se le desea añadir más contenido), el nombre exacto del archivo HTML (como se detalló anteriormente), el diccionario con los datos dinámicos para completar el HTML, y un valor booleano, para determinar si el contenido se insertará dentro de la caja, o sobre el sitio web (como se mencionó previamente).

El sistema de templates fue desarrollado para permitir mayor flexibilidad a la hora de encarar el diseño del widget, principalmente para usuarios que poseen un nivel básico/medio de programación, pero así para aquellos novatos, que tengan la iniciativa de capacitarse y aprender más y se animen al desafío de utilizarlo. Hay que destacar, que eligiendo esta alternativa, no quita que podamos a su vez utilizar la otra que se explicará a continuación.

5.4.3.2 Utilizando WidgetInterface (para un diseño simple)

Es momento de centrarnos en la alternativa a la creación de Templates, haciendo foco en aquellos usuarios novatos que recién están comenzando con su camino en la programación, pero que eso no quita que puedan desarrollar su propio widget, utilizando la estructura WidgetInterface que provee el framework SocialEye. Su objetivo es brindar al usuario un conjunto de funciones que le permitan crear y agregar elementos HTML al componente visual de su widget. De esta manera, se obtiene un código mucho más limpio y legible, porque lineas escritas en lenguaje JavaScript correspondientes a la creación de un elemento HTML serán reemplazadas por llamadas a funciones del objeto Widget que harán el trabajo por nosotros. En el momento en que es instanciada la clase Widget, se inicializa el atributo interface con la instancia de un objeto WidgetInterface, que utilizará el widget para obtener los elementos HTML que son requeridos. A continuación, detallaremos las funciones disponibles para comenzar a crear elementos y armar el diseño e interfaz del widget. Cabe destacar, que todas reciben como mínimo un parámetro de tipo opcional, que asigna, o no, un id al elemento creado.

- `addPrincipalBox = function()`: podría considerarse la primer función a invocar, dado que su resultado es la creación de la caja principal contenedora del widget, que incluye el título del widget brindado por el usuario cuando subió al servidor su desarrollo como encabezado de la caja.
- `getPrincipalBox = function()`: al igual que la función anterior, crea una caja, con la diferencia de que no la añade automáticamente al componente vista, sino que lo retorna al usuario como elemento `div`.
- `getForm = function()`: retorna un elemento de tipo `form`.
- getInput = function(): retorna un elemento HTML `input`, del tipo recibido como parámetro obligatorio.
- `getTextArea = function()`: retorna un elemento `textarea`.
- `getPrincipalList = function()`: esta función fue desarrollada con el objetivo de obtener un elemento HTML de tipo `li`, que posteriormente será utilizado como lista principal del contenedor. Es decir, si el comportamiento del widget se basa en una lista, como puede ser una lista de comentarios, este

método es ideal dado que ya provee una lista con un estilo simple acorde al framework.

- `getLi = function()`: similar a la función anterior, con la diferencia que el objetivo es retornar una lista sin contemplar previamente un posible uso. Es decir, no provee un estilo predefinido.
- `getPrincipalBody = function()`: retorna un elemento `div`, creado con el objetivo de ser el contenedor principal de los demás componentes que conforman el widget. Para ello, viene con un estilo predefinido.
- `getSubmitButton = function()`: retorna un elemento `button` que posee un estilo pensado para un botón de tipo `submit`.
- `getListButton = function()`: similar a la función anterior, pero con la finalidad de proveer un botón para una lista.
- `getDiv = function()`: retorna un elemento `div`.
- `getSpan = function()`: retorna un elemento `span`.
- `getP = function()`: retorna un elemento `p` (párrafo).
- `getA = function()`: retorna un elemento `anchor`, usado para diseñar un link.
- `getI = function()`: retorna un elemento de etiqueta `i`, creado para HTML5 [28], cuya función es mostrar texto en modo diferente. También es utilizado para visualizar íconos.
- `getLabel = function()`: retorna un elemento de etiqueta `label`, cuyo contenido es el string recibido como parámetro obligatorio.
- `getVideo = function()`: retorna un elemento de etiqueta `video`, una de las funciones de HTML5 que suscitan más interés. Esta etiqueta se suele presentar como alternativa a Flash para el contenido multimedia, pero tiene más aplicaciones.

Como vimos, a excepción de la primera de todas las funciones, que ya crea e inserta el elemento en el sitio web, las demás fueron diseñadas con el objetivo de que el usuario les asigne el lugar adecuado según su criterio, utilizando una función nativa de Javascript como append para añadirlo a un elemento ya creado, que puede obtener mediante una función que ya mencionamos que provee el framework, que es `getWidgetElement`.
Como conclusión, esta alternativa del objeto Interface brinda un gran abanico de oportunidades a un usuario reciente para desarrollar su propio widget con un estilo acorde al framework SocialEye.

5.5 Resumen

A lo largo del capítulo hemos expuesto la esencia del Framework creado y de la herramienta basada en su utilización, definiendo al navegador en el cliente como su ambiente natural donde lleva a cabo todo su procesamiento.
Haciendo una presentación inicial del lenguaje Javascript, pilar de toda implementación a interpretarse en un navegador, definimos la estructura de clases del Framework, haciendo una división aquí entre lo que sería la base lógica de la herramienta, y la representación del diseño de la interfaz:
Para la primera parte especificamos la clase Widget, caracterizándola como la pieza clave de la creación y punto de partida en el desarrollo del usuario, quien deberá plasmar en una instancia de ella su idea innovadora, mediante la definición de sus atributos y comportamientos característicos.

Acto seguido hablamos de los objetos Template y WidgetInterface, para explicar la posibilidades de crear una interfaz de usuario mediante código HTML y también a través de la creación de objetos Javascript asociados al DOM. Al detallar los usos de cada método, aclaramos también el porqué fueron creados, al determinar los tipos de desarrolladores apuntados a utilizarlos. También hablamos de la posibilidad de asociar plantillas CSS a los widgets, a fin de mejorar el diseño mediante la definición de atributos que modifiquen los componentes visuales.
Nos tomamos un momento para hablar de los lenguajes de programación y librerías utilizados a lo largo del desarrollo. Aquí explicamos brevemente la razón de ser de cada uno, expusimos también sus características y funcionamiento, terminando por asociar todo al desarrollo de la herramienta, explicando el porque los elegimos, donde los utilizamos y qué aportan a la creación del Framework.
Hicimos un apartado para explicar la metodología utilizada por la herramienta para la comunicación desde el lado del cliente con el servidor de datos. Aquí hablamos de la importancia del asincronismo para el trato entre ambas partes, debido a la característica de "extensión de navegador" de la herramienta generada. Explicamos el papel fundamental que juega AJAX para esto último, exponiendo resumidamente su funcionamiento.
Definimos a JSON como el formato de texto en común entre aplicación Client-Side y servidor de datos para la transferencia de la información entre ambos y la representación de todo objeto asociado a la herramienta.
Finalmente, nos adentramos de forma precisa en los detalles del uso del Framework, en la sección *Api para el desarrollador*. Empezando por la instanciación de la clase Widget, pasando por los detalles de diseño y exhibiendo cada uno de los atributos y comportamientos definidos para la creación de los mismos, terminamos por determinar el alcance de la herramienta para el desarrollador que aquí exponemos.

6. *Servidor de datos*

6.1 Introducción

Como toda aplicación que debe persistir datos para su funcionamiento, estos deben almacenarse en una base de datos para que puedan ser reutilizados y/o modificados por parte del usuario. Para ello, se ha creado un modelo con las clases necesarias del framework, y dentro de lo que se llama controlador, las funciones que acceden a la base de datos mediante estas clases del modelo e interactúan con ellos.
En este capítulo entonces se procederá a explicar el componente de la arquitectura desde el lado del servidor, que implica qué clases del modelo se crearon y por qué, y las funciones desarrolladas que procesan los datos del lado del servidor y se comunican con la base de datos del framework.

6.2 Motor de base de datos

Después de realizar un análisis intenso de los motores de bases de datos [29] más conocidos y más utilizados en el mercado, nos inclinamos por PostgresSQL. Las razones son varias, entre las que se encuentran:

- Open Source: uno de los motivos más importantes, dado que apoyamos todo lo que sea código abierto. El motor posee una licencia bsd [30], y puede ser descargado de su página principal bajo ningun costo. Además, dado a todos los desarrollador que están interesados en él, posee un gran soporte y está en constante mantenimiento.
- Multiplataforma: se encuentra disponible en prácticamente todas las versiones de Unix, y también en Windows, y posee un cliente muy intuitivo y fácil de utilizar llamado PgAdmin.
- Alto Volumen: otra de las razones más importantes para elegirlo. Dado que el volumen de datos del framework será alto a medida de que su uso se incremente, Postgres posee un alto rendimiento y eficiencia ante muchas transferencias de datos simultáneas.
- Tipo de datos JSON: PostgresSQL posee un tipo de datos exclusivo para guardar un JSON, una característica fundamental para nosotros y que nos ayudó a tomar la decisión final. Como se verá posteriormente, el framework persistirá datos de este tipo, y este motor de base de datos, con esta propiedad, nos facilitó el trabajo notablemente.

Clases del modelo
El framework está construido basándose en sólo dos clases, por un lado la clase Widget, y por otro lado la clase Element. Estas clases, como se verá posteriormente, se corresponderán con una tabla en la base de datos.
Con respecto a la clase Widget, describe a un Widget desarrollado por el usuario, y está formada por los siguientes atributos:

- Widget_name: String que se refiere al nombre del Widget. Debe ser único, es decir que no pueden existir dos widgets que posean el mismo nombre.
- Widget_tittle: Título del widget que se mostrará cuando se necesite describir e informar acerca del mismo.
- Users: propiedad relacionada a los usuarios que tienen activado el widget en cuestión para poder utilizarlo.

- Widget_icon: icono gráfico perteneciente al widget y que será utilizado para acceder al mismo mediante la barra de herramientas del framework.
- Description: descripción acerca del Widget y de su propósito y/o utilidad.
- File: corresponde al path de ubicación del archivo en lenguaje javascript, y que posee la implementación del widget desarrollado por el usuario del framework.
- Owner: refiere al usuario que desarrolló el widget

Se puede observar, como ya se dijo anteriormente, que la clase Widget justamente describe las propiedades básicas y necesarias que debe poseer un widget para su funcionamiento.
En cuanto a la clase Element, fue desarrollada para representar a todo objeto que pertenece a un widget y será persistido en la base de datos. Es una clase muy flexible, debido a que refiere a cualquier tipo de objeto que se desee guardar, por lo tanto, según el propósito de cada Widget, este objeto podrá variar. A continuación describiremos cada uno de sus atributos:

- Widget: Como dijimos que esta clase representa a cualquier elemento de cualquier Widget que se va a persistir, este atributo sirve para identificar a qué widget pertenece el elemento.
- Url: refiere a la url completa en donde fue guardado el elemento. Es importante que se sepa en qué página web fue creado para luego mostrarlo en la misma.
- Username: nombre de usuario de la persona que generó el elemento utilizando el Widget en cuestión.
- Date: fecha de creación del elemento.
- Element: este atributo es fundamental, y es el responsable de generar la abstracción en cuanto a que todo elemento dentro de un Widget se guarde en esta clase. Es un tipo de dato JSONField, que solo lo posee el motor de base de datos Postgres como se comentó al principio de este capítulo. Al ser un JSON, nos permite flexibilidad, debido a que en su interior se puede definir cualquier cantidad de objetos con distintas longitudes y tipos.

Se puede observar que no se ha mencionado cómo se realiza el manejo de los usuarios, es decir, donde y como se almacena la información que identifica a cada uno. Esto se debe a que el framework Django (utilizado para desarrollar el componente Servidor de la aplicación) ya nos provee una API que realiza este trabajo por nosotros, es decir, mediante la cual se puede crear, modificar, eliminar y autentificar usuarios. Así también nos provee de una clase que, como se puede deducir, se llama User, que se traslada a una tabla en la base de datos, y que, entre otros, posee los siguientes campos:

- username (obligatorio): treinta caracteres como máximo. Sólo acepta caracteres alfanuméricos (letras, dígitos y el carácter subrayado).
- email (opcional). Dirección de correo electrónico.
- password (obligatorio): contraseña de ingreso..
- last_login: Fecha y hora de la última vez que el usuario se identificó. Se asigna automáticamente a la fecha actual por defecto.
- date_joined: Fecha y hora en que fue creada esta cuenta de usuario. Se asigna automáticamente a la fecha actual en su momento.

6.3 Tecnología utilizada

Para desarrollar el componente Servidor de la aplicación, se utilizó Django [31], un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como Modelo–vista–controlador [32]. La elección del mismo se basó en su practicidad y rapidez a la hora de crear un software web y, entre otras, por el mapeador objeto-relacional [33] que nos brinda, que nos facilita la interacción con la base de datos utilizando el concepto de objeto y no de tabla. Además, proporciona una api de bases de datos robusta, que incluye compatibilidad con la mayoría de los motores de bases de datos existentes.
Luego de esta breve introducción de Django, podemos ahondar un poco más en la implementación del lado servidor de la plataforma, es decir en el componente controlador, si lo vemos basándonos en el modelo MVC (Modelo-Vista-Controlador). Está compuesto de varias funciones, en donde cada una es llamada según la url del requerimiento entrante. Esta es una de las características que nos provee Django, un despachador de URLs basado en expresiones regulares, que facilita notoriamente esta tarea. Ahora bien, una vez que se despacha el requerimiento a la función que le corresponde, la misma ejecutará su tarea asignada.
En cuanto a las funciones desarrolladas en el servidor, por el hecho de que la lógica principal del framework se encuentra en el cliente, la mayoría se limita principalmente a la recuperación y almacenamiento de datos, sumadas a las operaciones de usuario, como lo es el acceso a la aplicación.
Bajo la clasificación mencionada, podemos mencionar algunas de las funciones de recuperación y almacenamiento de datos en el servidor las cuales se corresponden directamente con algunos de los métodos de la API Restful definida:

- `Objects`: Si el requerimiento es de tipo POST, crea y guarda en la base de datos un objeto de tipo Element. En cambio, si el requerimiento es de tipo GET, busca en la base de datos un objeto de tipo `Element`, basándose en los parámetros recibidos, y retorna un elemento de tipo JSON.
- `updateObject`: Actualiza un objeto de tipo `Element` en la base de datos.

Con el mismo objetivo del manejo de datos, nos encontramos con métodos que hacen a la gestión del framework y la herramienta

- `Scripts`: Retorna un JSON con los titulos de todos los widgets creados.
- `getWidget`: busca en la base de datos y retorna un JSON con el widget que posee el id recibido por parámetro.
- `User_ping`: función utilizada para controlar cuando un usuario está activo en la aplicación. Si el requerimiento es de tipo POST, actualiza el contador del objeto `UserActiveUrl` en la base de datos. Por otro lado, si el requerimiento es de tipo GET, busca en la base de datos todos los usuarios activos y los retorna en un formato JSON.
- `addUserWidget`: agrega un widget en la colección de widgets activados para el usuario con id recibido por parámetro. El id del widget a almacenar es obtenido igualmente por parámetro.
- `Framework`: retorna el script principal del widget para que sea utilizado en el lado del cliente.
- `Interface`: retorna el script que implementa los componentes visuales para crear un framework desde el lado del cliente.

- `removeUserWidget`: elimina de la base de datos el widget elegido por el usuario. Este widget debe pertenecer al usuario, de otro modo esta operación no es posible.
- `widgetsByUser`: Busca en la base de datos los widgets que posee activado el usuario con id recibido por parámetro, y los devuelve en formato JSON.

Por último, tenemos los métodos mencionados que requieren algún tipo de lógica en el servidor:

- `Logout`: cierra la sesión del usuario logueado en la aplicación.
- `Registration`: Crea un nuevo usuario en la aplicación y lo guarda en la base de datos.

6.4 Librerías externas

En esta sección nos centraremos en los componentes de software externos que se debieron utilizar desde el lado del servidor para desarrollar el framework. Recordemos que una librería es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. A diferencia de un programa ejecutable, el comportamiento que implementa una librería no espera ser utilizada de forma autónoma, sino que su fin es ser utilizada por otros programas, independientes y de forma simultánea. Por otra parte, el comportamiento de una biblioteca no tiene por qué diferenciarse en demasía del que pudiera especificarse en un programa. Es más, unas librerías pueden requerir de otras para funcionar, dado que el comportamiento que definen refina, o altera, el comportamiento de la biblioteca original, o bien la hace disponible para otra tecnología o lenguaje de programación.
En este caso haremos referencia a dos librerías que han sido fundamentales para el desarrollo de esta tesina: django-tokenapi [34] y django SSL Server [35].

6.4.1 Django-tokenapi

Como todos sabemos, hoy en día, los usuarios se han acostumbrado a iniciar sesión en una aplicación, y a partir de allí, realizar todas las acciones y operaciones que desee sin tener que volver a autenticarse por cada una de ellas. Una sesión es típicamente, pero no siempre, con estado, significando que al menos una de las partes comunicantes necesita salvar información sobre el historial de sesión para ser capaz de comunicarse, o sin estado, donde la comunicación consta de peticiones independientes con respuestas.
Para poder llevar a cabo el protocolo de sesión, se tuvo que buscar una solución alternativa a las sesiones web del lado del servidor, dado que cada petición HTTP que se ejecuta desde el lado del cliente en el framework SocialEye, es una nueva para el servidor, por lo tanto delegar todo el manejo de sesión a la API que provee el framework Django no nos resolvía de todo el problema.
La solución se encontró con la librería Django-tokenapi, basada en el mecanismo de autenticación con token de sesión, un identificador único que está generado y enviado desde un servidor a un cliente para identificar la sesión de interacción actual. El cliente envía el token como una cookie HTTP [36] y/o lo envía como parámetro en GET o POST. De esta manera, una vez que el usuario se autentifica, el servidor retorna el token al componente cliente del framework, el cual lo almacena en el objeto localStorage del navegador, una propiedad de HTML5, que permite almacenar datos en nuestro navegador web, de manera muy similar a como lo hacen las cookies, por un tiempo indefinido, sin importar que el navegador se cierre.

Un ejemplo de respuesta de la librería ante la primera autenticación de un usuario, es la siguiente:

```
{"success": true, "token": "2uy-420a8efff7f882afc20d",
"user": 1}
```

Posteriormente, al efectuar cada petición HTTP al servidor, se debe enviar el nombre de usuario autenticado, sumado al identificador del token, en el header del requerimiento. Ambos se encuentran almacenados como ya se dijo en el localStorage:

```
$.ajaxSetup({

          beforeSend: function (xhr) {

              xhr.setRequestHeader('Authorization',
"Basic " + btoa(localStorage['user'] + ":" +
localStorage['token']));

          }

      });
```

Esta función se ejecuta en el momento previo de ejecutar una petición HTTP, añadiendo la información del token en forma de header.

6.4.2 Django SSL Server

Actualmente todas las aplicaciones se encuentran en constante alerta ante la amenaza de los hackers de vulnerar su seguridad y obtener datos e información de los usuarios que las utilizan. Una herramienta para ayudar a prevenir estos ataques es el protocolo SSL (Secure Sockets Layer) [37], cuyo objetivo es proporcionar conexiones seguras en internet. Se utiliza criptografía asimétrica para autentificar a la contraparte con quien se están comunicando, y para intercambiar una llave simétrica. Esta sesión es luego usada para cifrar el flujo de datos entre las partes. Esto permite la confidencialidad del dato/mensaje, y códigos de autenticación de mensajes para integridad y como un producto lateral. Varias versiones del protocolo están en aplicaciones ampliamente utilizadas como navegación web, correo electrónico, fax por Internet, mensajería instantánea, y voz-sobre-IP (VoIP).

A la hora de implementar los widgets predeterminados, nos encontramos con la necesidad de utilizar este protocolo para que funcionen distintas características fundamentales de distintos widgets, como por ejemplo el chat y la videollamada. Por lo tanto, hacemos uso de esta librería que nos permite implementar este protocolo de forma gratuita. Django-sslserver es un servidor utilizado para la etapa de desarrollo de aplicaciones en Django, que posee habilitado el protocolo SSL. Esto permite sortear los obstáculos a la hora de desarrollar software que necesite de este protocolo, en la etapa de desarrollo. Al momento de que el mismo sea productivo, es recomendable pagar para obtener un certificado original.

6.5 Resumen

En este capítulo se especificó el componente Servidor de la aplicación, el cual fue desarrollado utilizando Python y Django, un framework creado en el mismo lenguaje de programación y con PostgreSQL como motor de base de datos, teniendo en cuenta potencial, y el hecho de que es open source, multiplataforma, puede manejar un alto volumen de información, y nos brinda de un tipo de dato fundamental para nuestro framework, que es JSONField. Con respecto a las clases creadas en el modelo, las que se destacan son Widget y Element, dado que son el corazón de la aplicación, y el centro de la mayoría de las operaciones que se ejecutan. En cuanto al controlador y la lógica en el servidor, además de mencionar la herramienta elegida para programarlo, se detallaron las funciones utilizadas, junto a su misión, de manera de poder entender el circuito de la aplicación. El objetivo de las mismas se relaciona con las operaciones básicas que se ejecutan sobre una base de datos, es decir, alta, modificación y acceso a los registros. Por último, se detallaron las dos librerías externas que nos sirvieron de mucha utilidad para el desarrollo del componente servidor: django-tokenapi, para el manejo de sesión, y django-sslserver, para la seguridad del framework.

7. Herramienta de usuario final

Posteriormente al análisis técnico del desarrollo del framework y de la aplicación web, comenzaremos a adentrarnos en la interacción que tendrá el usuario con los distintos puntos de entrada de la aplicación, empezando por su utilización dentro de una página web, siguiendo por el apartado de configuración de los widgets que cada uno escogerá, para terminar con el envío, por parte del usuario, de su propio widget desarrollado.

7.1 Estructura

La visualización de la aplicación consiste en un ícono con la figura de un ojo ubicado en la esquina superior derecha de la página web cargada. Al momento de presionarlo, y luego de iniciar sesión, se desplegará hacia abajo una barra de navegación, que está compuesta por todos los íconos de los widgets que el usuario configuró como activos en la opción de configuración. Para acceder a la misma, uno de los íconos que ya nombramos, tendrá el icono típico de una opción de ajuste. Por otro lado, cada widget presente se ejecutará haciendo click en su figura correspondiente, permitiendo la posibilidad de tener abierto al mismo tiempo tantos como se desee, dibujando una línea vertical, recta, de color rojo en la parte izquierda de la figura del icono de cada widget que esté siendo utilizado. En la figura 7.1 observamos la barra de navegación, constituída por los widgets, el apartado de configuración, y por último con el botón de cerrar sesión.

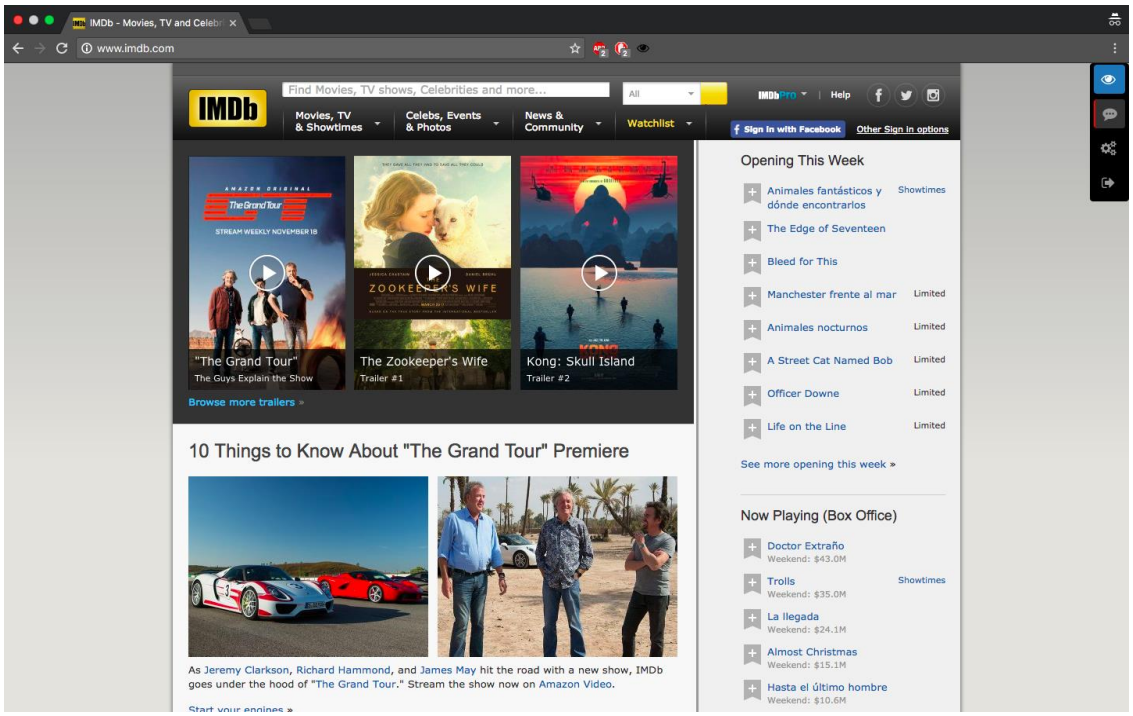

Figura 7.1 - Presentación de la Herramienta

Es importante destacar, que la barra de la aplicación se encontrará visible en todo momento, por más que el usuario se deslice por la página web en cualquier sentido.

7.2 Perfil del usuario

En esta sección nos centraremos en el perfil de un usuario, desde la creación de su cuenta, hasta la configuración de los widgets que desea mantener activos.

Como se explicó anteriormente, cuando un usuario accede a una página web, encontrará el icono con la figura de ojo que identifica a la aplicación. Al presionarlo, se observará un cuadro que pertenece al inicio de sesión, en donde el usuario ingresará su usuario y contraseña para poder acceder a la herramienta, tal como se muestra en la figura 7.2

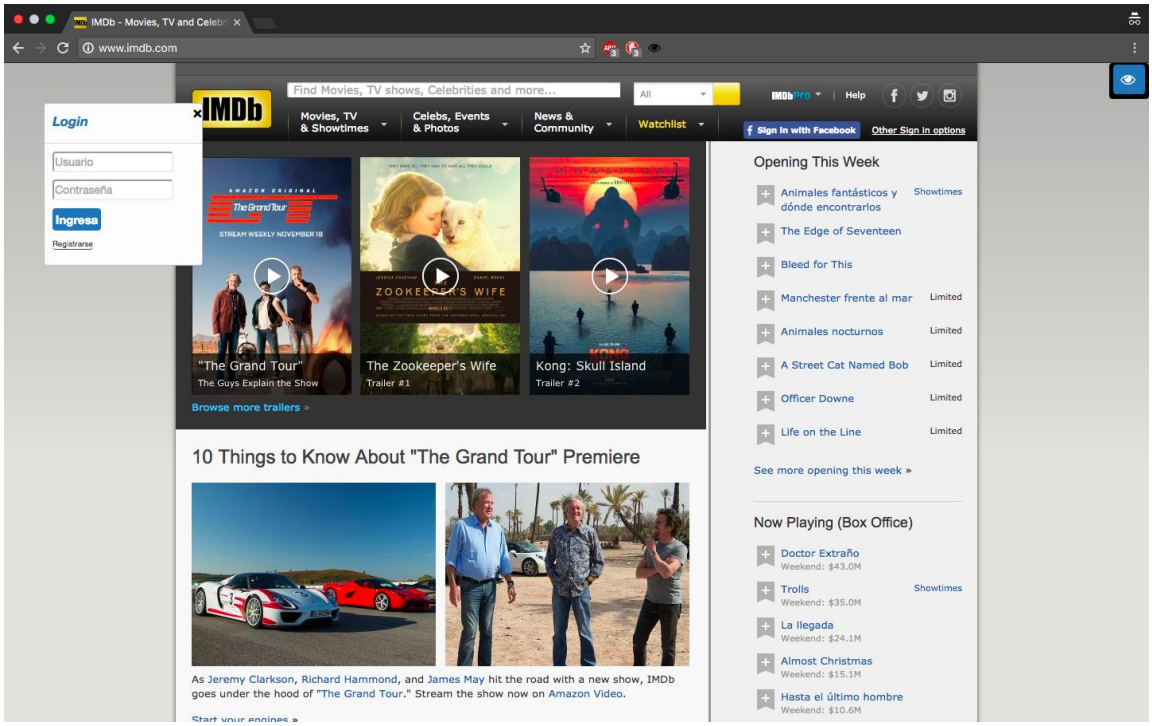

Figura 7.2 - Login

Si la persona no posee una cuenta creada, podrá crear de dos formas distintas, ya sea haciendo click en el link con el texto "Registrarse" ubicado debajo del botón de ingresar, como se observa en la imagen previa, o desde la página web del framework. A continuación se detallará la primera opción, dejando la segunda para la siguiente sección de este capítulo.
Cuando el usuario se registra desde la aplicación, se visualizará un formulario (figura 7.3) en donde deberá ingresar un nombre de usuario y una contraseña. Si la validación es exitosa, el perfil estará creado y el usuario estará habilitado para iniciar sesión como se mencionó anteriormente.

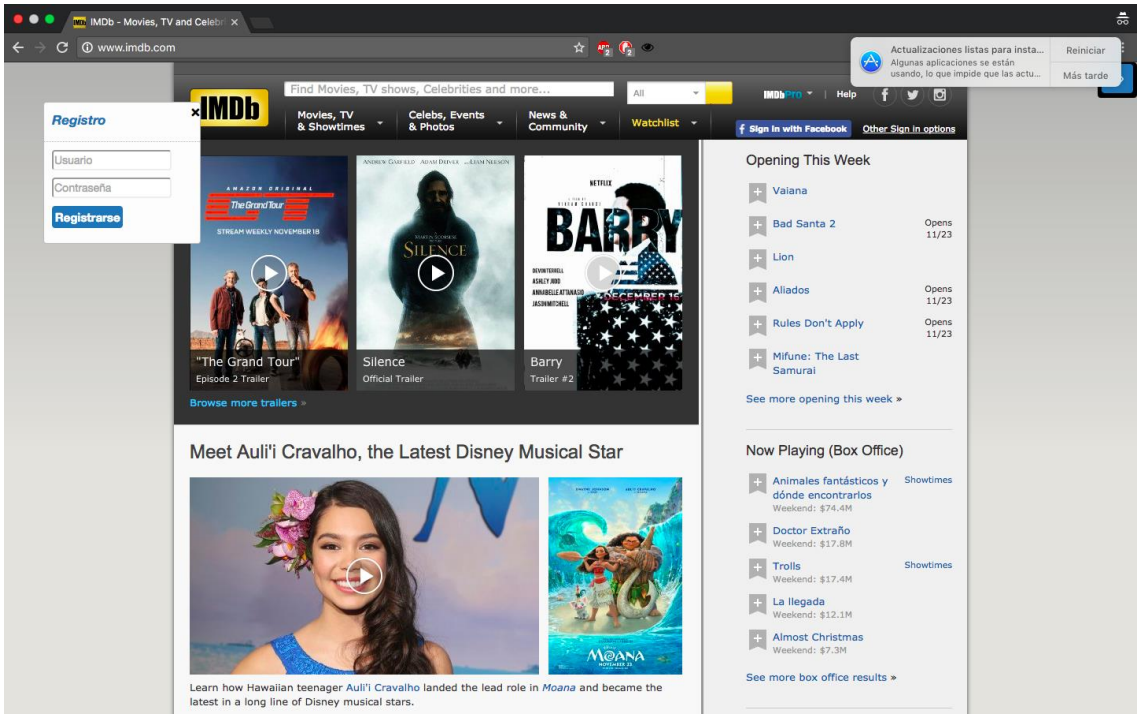

Figura 7.3 - Registro

Una vez ingresado, la persona seguramente deseará configurar los widgets a mantener activos, basándose en sus gustos e intereses personales. Para ello, está presente la opción de configuración, a la cual se accede, como se mencionó previamente, presionando en el icono de ajuste, el cual mostrará en pantalla los widgets presentes como se observa en la figura 7.4, para que el usuario escoja según su preferencia.

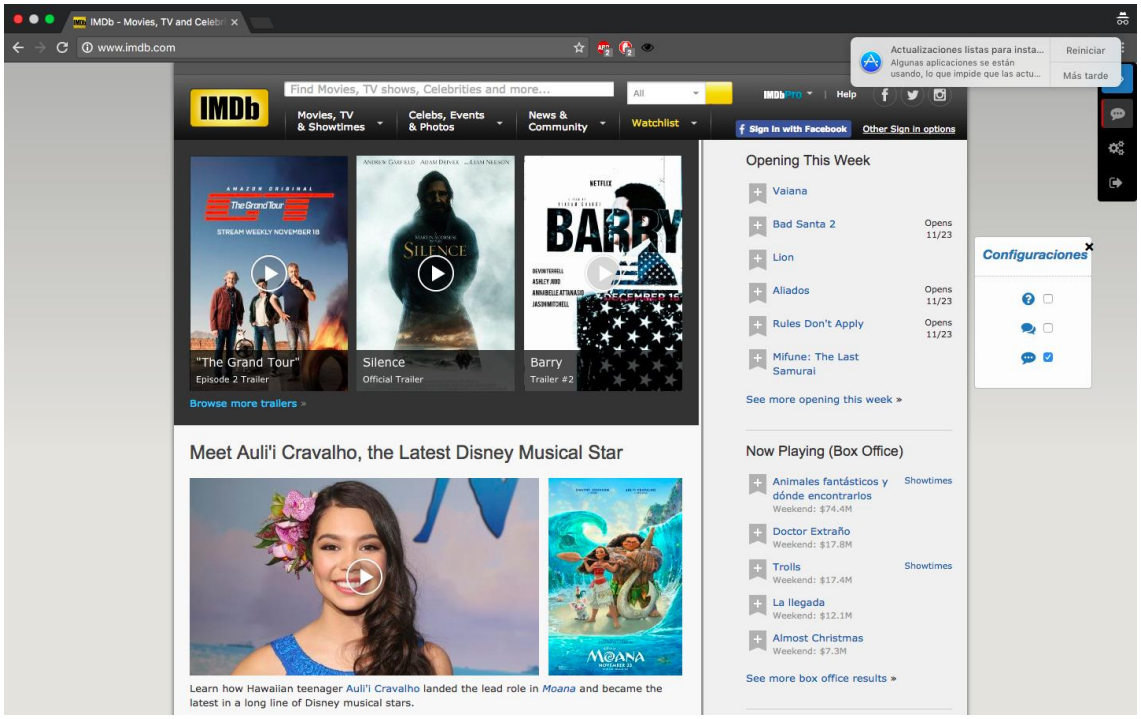

Figura 7.4 - Configuración de la Herramienta

7.3 Sitio web

Como todo software desarrollado que posee su propio sitio web de presentación en donde, además, se presenta información acerca de las funciones que brinda, SocialEye no podía ser la excepción. Es por ello que se diseñó un sitio web para exponer el framework desarrollado, sus características y, por otro lado, para que el usuario pueda realizar lo esencial, es decir, cargar su propio widget creado con el framework en cuestión.

7.3.1 Home

En la página de inicio del sitio, nos encontramos con toda la información acerca del framework desarrollado, incluyendo objetivos, motivación, para que el usuario que ingrese pueda comprender a simple vista la facilidad que otorga el widget para encarar el diseño y codificación de un widget basado en WebAugmentation.
En la parte superior de la página se encuentra una barra de navegación con pestañas, que incluyen la posibilidad de registrarse en el sitio de la misma forma que puede realizarse desde la herramienta de navegación tal como se mencionó en la sección anterior.

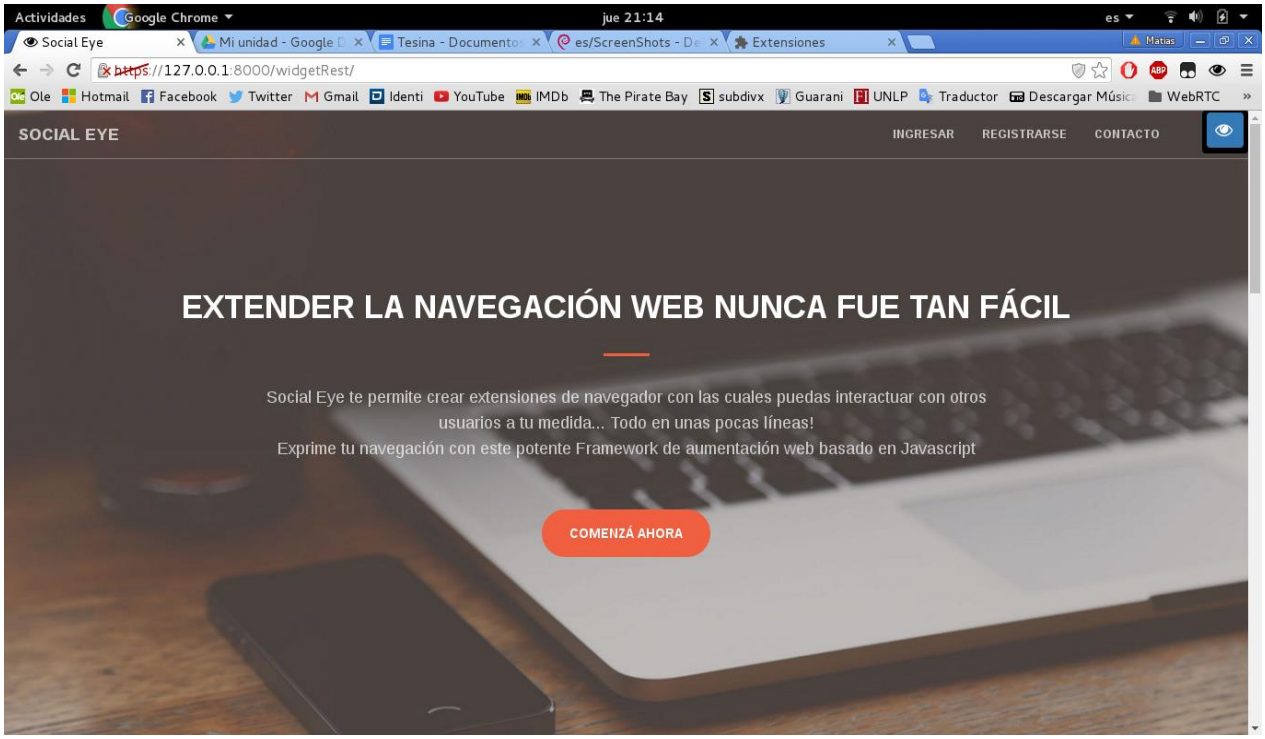

Figura 7.5 - Home de la Web

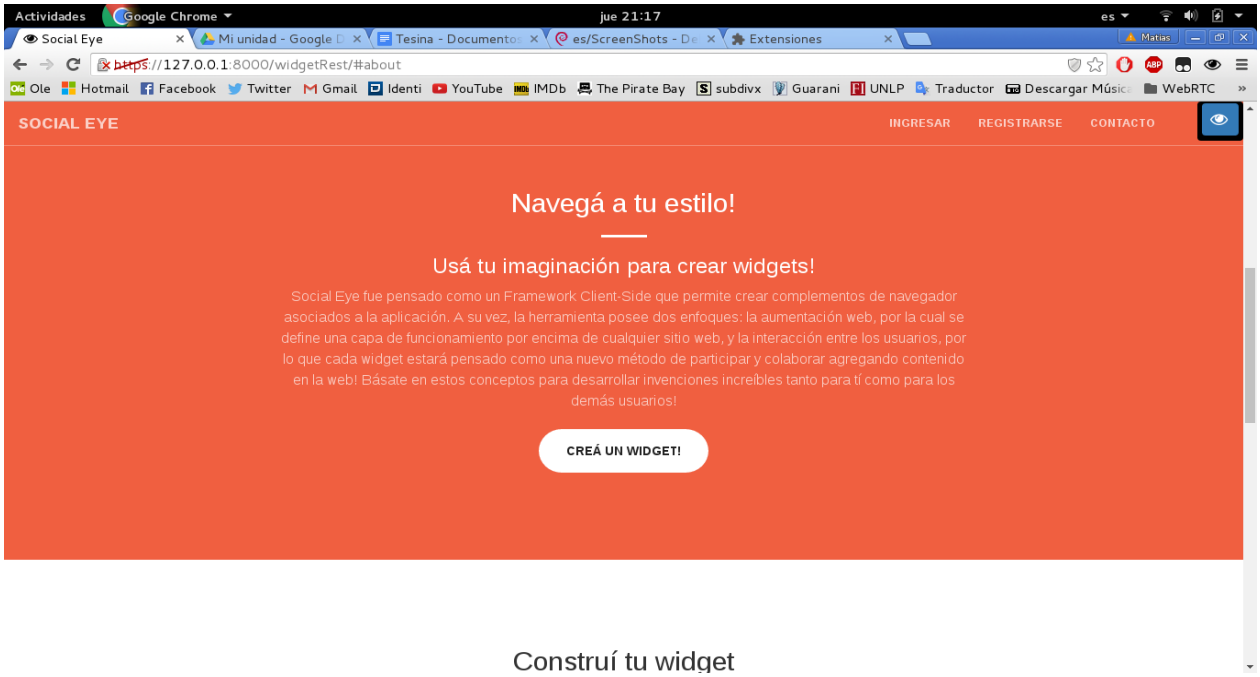

Figura 7.6 - Presentación y motivación del Framework

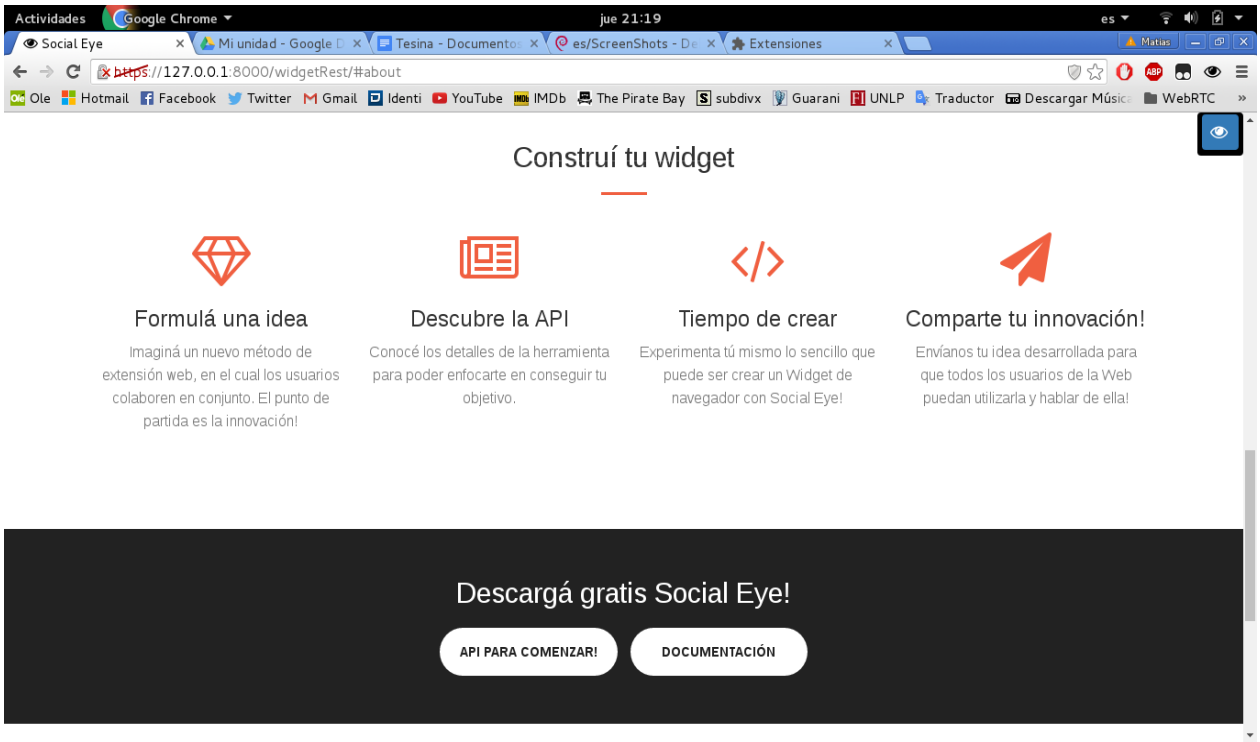

Figura 7.7 - Pasos para la creación y links de descargas

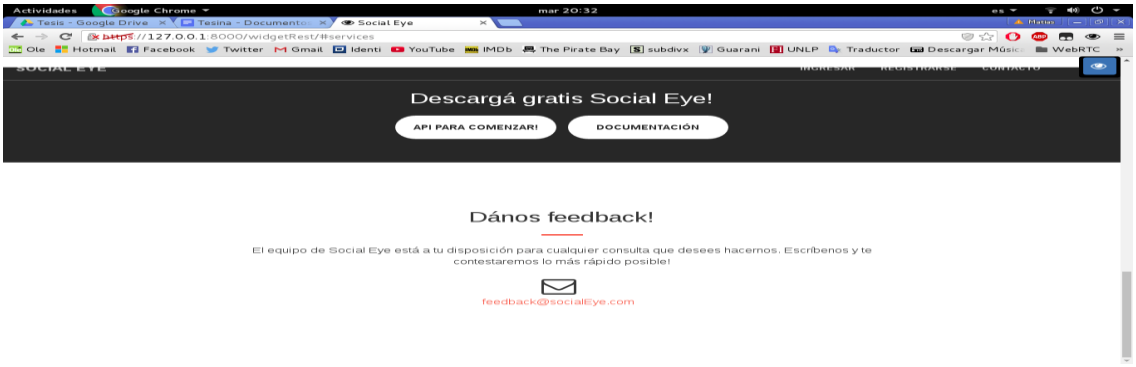

Figura 7.8 - Footer de la Web

7.3.2 Registro y login

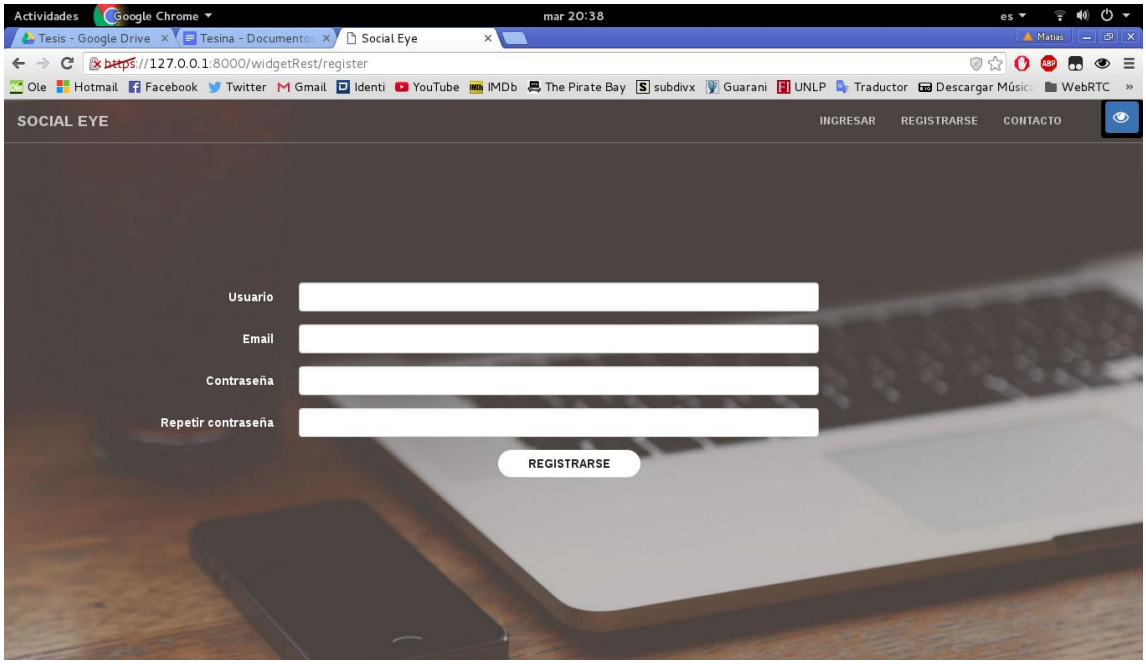

Figura 7.9 - Registro en la Web

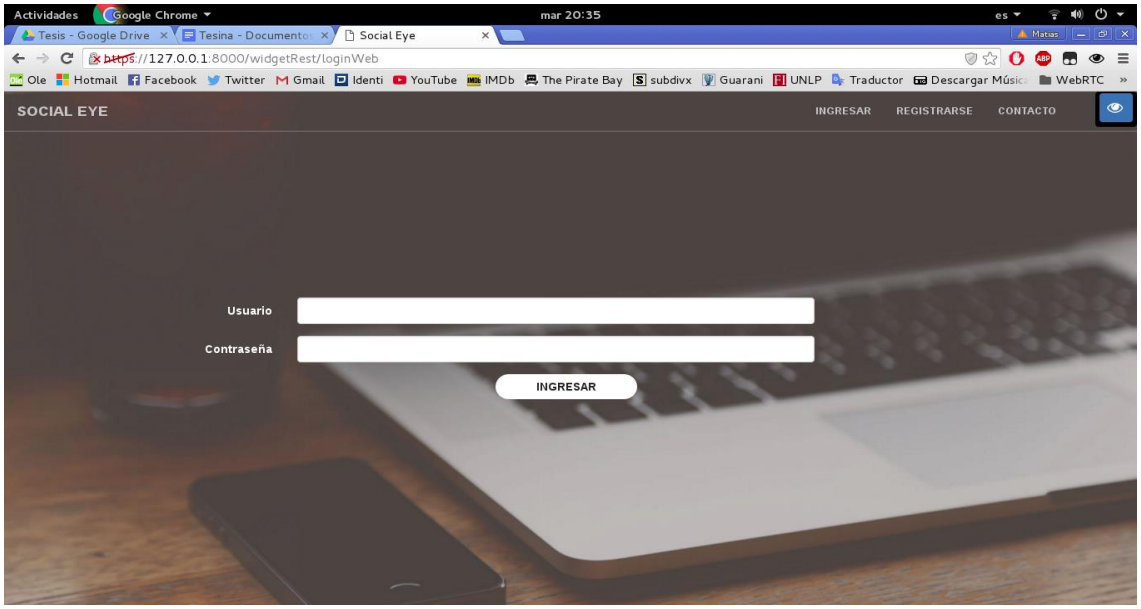

Figura 7.10 - Login en la Web

7.3.3 Administración de widgets del usuario

Luego de detallar aspectos relacionados al sitio web del framework, y como un usuario puede crear su cuenta y posteriormente iniciar sesión, en esta sección se describirá el conjunto de acciones para realizar lo que creemos, es la principal función del framework desarrollado y es, la de cargar el widget codificado por el usuario para que pueda comenzar a ser utilizado por todos los demás usuarios de SocialEye.

7.3.3.1 Listado de widgets asociados

En la barra de navegación se encuentra una pestaña llamada “Mis Widgets”, la cual agrupa las funciones básicas relacionadas al manejo de widgets: listado, creación, edición y eliminación.

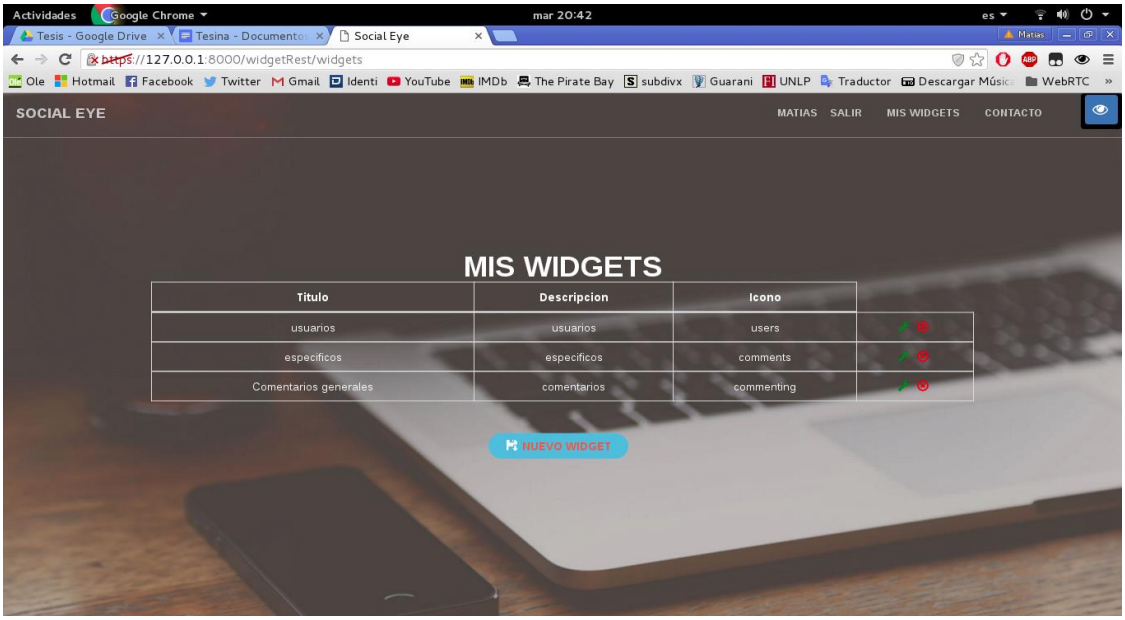

Figura 7.11 - Listado de Widgets

Como se observa en la imagen, el usuario actual tiene tres widgets asociados, sobre los cuales puede tomar la decisión de editar sus características (botón verde de herramienta) o eliminarlos (botón de cruz roja). También, con un click sobre el botón "Nuevo Widget" se dará inicio a la subida de un widget recién creado.

7.3.3.2 Carga de un widget

Una vez que el usuario haya codificado su propio widget utilizando el framework SocialEye, deberá cargarlo en el sitio para que el mismo se encuentre a disposición de todos aquellos que hayan creado su cuenta en SocialEye y quieran comenzar a utilizarlo. Accediendo a la ya mencionada pestaña "Mis Widgets", se procederá pulsar en el botón "Nuevo widget".

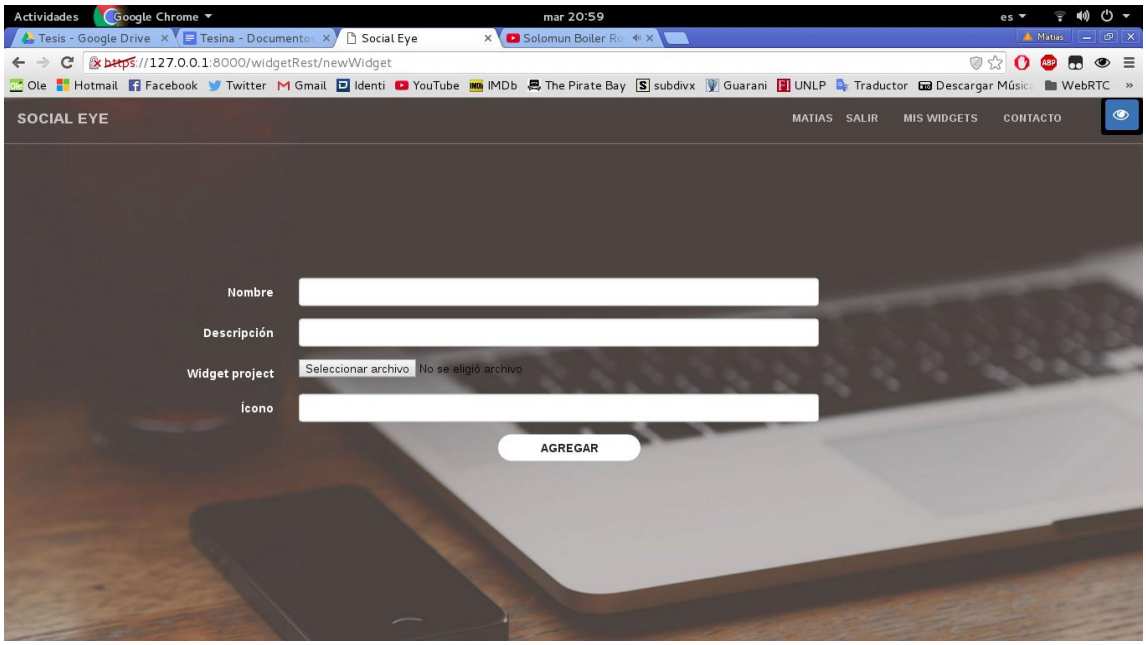

Figura 7.12 - Carga de un Widget

El formulario para poder cargar un nuevo widget consta de cuatro campos, en donde todos son obligatorios para poder cumplir con el objetivo:

- Título: es el título del widget, y es único, es decir que no puede existir más de un widget que repita el título. Al ser un elemento fundamental dado que representa al widget, sobretodo indicando básicamente su funcionalidad, es aconsejable que sea lo más representativo posible.
- Descripción: en este campo se debe especificar con mayor grado detalle la funcionalidad que va a cumplir el widget y su principal objetivo.
- Widget project: aquí debemos seleccionar desde nuestra computadora, el proyecto del widget que será subido al servidor. Recordemos que debe estar comprimido en un formato .zip.
- Icono: Todo widget debe tener un icono que lo represente, de manera de que sea más amigable para los demás usuarios que lo van a utilizar. Para ello, se puede buscar y elegir un icono de la página font-awesome, y colocar el nombre del icono en este campo. Por ejemplo, en un widget de comentarios, se podría escoger la imagen del ícono "commenting", y ese nombre escribirlo en este campo, para que luego automáticamente aparezca en la barra de navegación representando al widget en cuestión.

Una vez finalizada la carga de estos cuatro campos, se procederá a pulsar el botón "Agregar", manera de subir el widget al servidor. A continuación, se agregará el nuevo widget a la lista correspondiente.

7.3.3.3 Eliminación de un widget

Si el usuario previamente envió un widget propio al servidor, el mismo aparecerá en la tabla que se observa al presionar sobre la pestaña "Mis Widgets", permitiendo la posibilidad de eliminarlo, ya sea por diferentes motivos que el usuario puede tener en ese momento. Para ello, como se muestra en la figura 7.11, en la última columna de la tabla, se presenta un ícono con la forma de una cruz, de color rojo, representando la función de eliminar. Al presionarlo, aparecerá un diálogo indicando si estamos seguros de la acción a realizar, y luego de confirmar, el widget habrá sido eliminado.

7.4 Resumen

En este capítulo hemos presentado la herramienta de navegación del framework, su diseño, ubicación, y estructura, seguido de la interacción del usuario para ingresar a cada uno de los widgets que posee activados. Relacionado con la activación, detallamos cómo cada usuario puede pulir su perfil, de manera de seleccionar aquellos widgets que desea tener activados para utilizar y cuáles no. A continuación, nos centramos en la página web del framework, utilizado para presentarlo junto con su misión. Describimos la creación de una cuenta ya sea desde la herramienta de navegación como desde el sitio, y para finalizar la inserción y eliminación de un widget creado por el usuario, funciones imprescindibles para utilizar la aplicación.

8. *Widgets predeterminados*

El presente capítulo tendrá como finalidad presentar los distintos widgets que fueron desarrollados puramente con la utilización del Framework exhibido.
La principal idea de esta sección es la de mostrar y explicar distintos temas en los cuales las aplicaciones implementadas se fundamentan, con el objetivo de impulsar al usuario a generar sus propias invenciones. Además, se verán en detalle algunas implementaciones que luego podrán servir de guía para los desarrolladores a la hora de estructurar su widget, basándose en su imaginación y en los conceptos básicos del uso del Framework. El orden de presentación de cada widget se basa en la dificultad de desarrollo que representa cada uno, comenzando desde lo más simple para finalizar con aquel que posee mayor complejidad.

8.1 Comentarios Generales

8.1.1 Utilidad

Este widget surgió de la idea de poder armar una especie de foro en aquellas páginas web que se prestan para tener uno, pero que obviamente no lo poseen. Podemos pensar en sitios de diarios, noticias, blogs de distintos temas en donde se presentan novedades acerca del ámbito en que se destacan, que son visitados por muchas personas o seguidores, y desean poder debatir acerca de lo que leen, o simplemente dejar un comentario acerca de sus sentimientos, deducciones, o nuevas ideas para un futuro. Hoy en día existen millones de estas páginas web, que no brindan esta posibilidad a sus usuarios, y como este framework fue pensado con el objetivo de ampliar el contenido web y la interacción entre los usuarios, creemos que este widget es muy útil para estas circunstancias. Dado que la finalidad es para poder comentar acerca de lo que se está observando, este widget administra los comentarios por página web, más específicamente por url. Por lo tanto, de esta manera los comentarios van a quedar divididos por sección, y no se van a mezclar con los comentarios de otro tópico.

8.1.2 Comportamiento

Es momento de detallar el funcionamiento del widget, y para ello se presentarán capturas del mismo para poder describir lo más fiel posible su comportamiento. Más adelante, nos concentraremos en el aspecto del desarrollo en sí mismo, es decir en la codificación, añadiendo capturas del mismo y señalando cómo se utilizó el framework para construirlo.
Para empezar, tomaremos como ejemplo la página web del diario Hoy de la ciudad de La Plata, la cual no cuenta por defecto con la opción de dejar comentarios acerca de una nota, por lo tanto es una elección excelente para poner a prueba el widget. En la figura 8.1 se puede observar parte de la noticia, y al costado derecho de la imagen, el widget de comentarios generales ejecutándose.

Figura 8.1 - Widget de Comentarios Generales - Presentación

Una de las ventajas del widget es que es muy simple e intuitivo, solo hace lo que tiene que hacer para cumplir con su objetivo, que es el de agregar y leer comentarios. Además, al igual que todos los widgets (es una funcionalidad provista por el framework), puede moverse por toda la pantalla, arrastrándolo con el mouse, para que el usuario elija donde le sienta más cómodo según la disposición del sitio donde se encuentra.

Como se puede observar, en la página actual no se han registrado comentarios todavía, por lo tanto, procederemos a ejemplificar este suceso. Para ello, el usuario escribe en el rectángulo correspondiente a un comentario, para dejar su opinión.

Figura 8.2 - Widget de Comentarios Generales - Agregando un comentario

En la figura superior vemos el texto ya escrito por el usuario, luego, el mismo procede pulsando el botón “Agregar”, de modo de enviar el comentario.

Figura 8.3 - Widget de Comentarios Generales - Visualización del comentario

El comentario ha sido guardado, junto con el nombre de usuario de quien lo escribió, el día y el horario, para que otra persona que lo lea pueda imaginar con qué contexto fue pensado.
Para finalizar, simularemos que otro usuario ingresa y desea responder al comentario realizado, compartiendo su pensamiento.

Figura 8.4 - Widget de Comentarios Generales - Lista de comentarios

El comentario de *Marcelo* es colocado debajo del correspondiente a Fermín, y así sucederá con el resto de los comentarios que sean añadidos para esta página. Cabe recordar que los mismos serán vistos solo en este sitio,para ser más especifico solo en esta url, y para cada una de ellas, la lista de comentarios será única.
Podemos concluir que el widget es muy útil para la finalidad para la que fue desarrollado, sobre todo para aquellas páginas web que no poseen esta característica. A continuación nos embarcaremos a la codificación, destacando las ventajas que nos brinda el framework y lo simple que resultó desarrollar el widget utilizándolo.

8.1.3 Modelo del widget

Ya hemos descrito la forma en que los objetos del framework son almacenados en la base de datos, precisamente en la tabla element, conformada por los atributos url, username, date,widget, y element. Mencionamos también que el campo element está destinado a ser el campo que diferencie a cada uno de los widget, dado que al ser un JSON, nos permite flexibilidad, debido a que en su interior se puede definir cualquier cantidad de objetos con distintas longitudes y tipos. En este apartado, como así también en el correspondiente a cada widget desarrollado, se expondrá los campos particulares pertenecientes al widget de comentarios generales.
Cuando dijimos que el orden de presentación de cada widget estaría basado en el grado de dificultad de cada uno, esa evaluación fue hecha teniendo en cuenta también este aspecto. Es por eso que este widget cuenta solo con el campo `texto`, destinado a almacenar la cadena de comentarios pertenecientes a un comentario. La justificación

está basada en que es el único dato faltante para identificar un comentario, dado que los restantes campos de la tabla `element`, nos proveen tanto la fecha, widget, y usuario que realizó el comentario.

8.1.4 Desarrollo

Llegó el momento de centrarnos en cómo ha sido el desarrollo de este widget utilizando el framework SocialEye. Como se mencionó en el comienzo de esta sección, el orden de descripción de cada widget está basado en el grado de dificultad que representa cada uno según nuestro punto de vista.
En primer lugar, debemos instanciar nuestro widget para poder heredar todas las funciones que nos provee el framework para codificarlo y, a partir de ello, otorgarle el comportamiento que pretendemos.

```
1  var comentarios = new Widget();
2  comentarios.tittle = "Comentarios sobre la Web";
3
4  comentarios.loadWidget = function () {
5    items = comentarios.getObjectsInUrl(window.location.href);
6
7    var data = {items : items};
8
9    comentarios.createTemplate('principal', comentarios.tittle, 'comentarios_L76B80n.html', data);
10   comentarios.onCloseTemplate('principal', function(){
11         comentarios.close();
12   });
13   comentarios.setTemplatePosition('principal', '10%', '10%');
14 }
```

Figura 8.5 - Widget de Comentarios Generales - Instanciación

En la figura 8.5 presentamos la correspondiente instanciación del widget en la variable llamada `comentarios`, junto con la implementación del método `loadWidget` de la clase `Widget`. A partir de este momento cada vez que necesitemos de una funcionalidad provista por el framework, debemos llamar a la función correspondiente desde la variable en donde instanciamos nuestro Widget, en este caso `comentarios`.
Ahora bien prestemos atención a partir de la línea número nueve, dejando a un lado las dos anteriores por un momento. Como se puede observar, la interfaz del widget se diseñó utilizando la estrategia de los templates, dado que se invoca a la función que nos crea y agrega un template, con “principal” como id, debido a que será el más importante del widget, y con el título que escogimos en la segunda línea que se corresponde al del template. Pero nos está faltando el cuerpo principal del template, que le brinde la impronta al widget que estamos desarrollando, es por ello que los restantes argumentos están dedicados a esta cuestión. El primero es el nombre específico del archivo HTML con el que queremos rellenar este template, y el segundo, como se observará a continuación, necesario en este caso, el diccionario con los datos a utilizar en el HTML.

```
1  <div id='principalGenerales' class='actionBox'>
2    <ul id='listaComentariosGenerales' class='commentList'>
3      <% _.each(items, function(item) { %>
4        <li>
5          <div class='commentText'>
6            <span class='sub-text'>
7              <%= item.username %> dijo el <%= item.date %>
8            </span>
9            <p>
10             <%= item.element.texto %>
11           </p>
12         </div>
13       </li>
14       <% }); %>
15   </ul>
16   <form>
17     <textarea class='form-control' type='text' id='textoComentarioGeneral'
18      placeholder='Escriba su comentario...'/>
19     <button class='submitButton' id='agregarComentarioGeneral'>
20       Agregar
21     </button>
22   </form>
23 </div>
```

Figura 8.6 - Widget de Comentarios Generales - Template de Comentarios

La figura 8.6 se corresponde al archivo "comentarios.html" e implementa el diseño principal del widget de comentarios generales. Como podíamos imaginar, se encuentra basado en un objeto HTML de tipo lista, dado que es un conjunto de comentarios que se deben mostrar uno debajo del otro. Debemos prestar mayor atención al código encerrado entre <% y %>, porque es la función más interesante a la hora de crear un HTML para un template. Es mediante estos tags la manera en que vamos a insertar en el HTML datos obtenidos en forma dinámica del widget. Se sabe de antemano que nosotros a esta altura tendremos definidos los nombres de las variables que guardaremos en el diccionario de datos que será transferido al template, por lo tanto, en la línea número tres, utilizamos la estructura de control `each`, para recorrer la lista de comentarios a la que le escogimos el nombre `items`. Posteriormente, por cada comentario (`item`), insertamos los campos que necesitamos para mostrarlos, estos son, el nombre de usuario (`username`), fecha de creación (`date`), y por último, texto del comentario, propio del modelo de este widget (`texto`). Para finalizar el HTML, se incluye un botón que se corresponde al agregado de nuevos comentarios, utilizado cuando un usuario desee hacerlo.

Podemos observar como de una forma tan simple, entre el sistema de templates, y un pequeño archivo HTML ya tenemos la base de la interfaz de nuestro widget, desarrollada para obtener datos dinámicamente.

Retornando al comportamiento del widget, solo nos resta referirnos al último argumento del método `createTemplate()` para poder invocarlo. Ya mostramos el archivo HTML base del diseño del Widget y cómo obtiene los datos del diccionario recibido. Ahora es momento de armar dicha estructura, y para ello tenemos que centrarnos en las líneas 5 y 7 de la figura 8.5. En la primera de ellas, utilizamos el método `getObjectsInUrl()` que heredamos de la clase Widget, para obtener los comentarios del sitio donde nos encontramos actualmente, es por eso que le enviamos como parámetro dicha url. Posteriormente, en la línea 7, construimos el diccionario, que

en este caso solo cuenta con un solo objeto, que es la lista de los comentarios que nos retorna la función anteriormente invocada.
¡Eso fue todo! Con muy pocas líneas de código se logró poner en funcionamiento el widget de comentarios generales. Con respecto a las dos últimas de ellas, refieren, en primer lugar, a la forma de actuar del widget cuando el template principal es cerrado. Dado que no se debe realizar ninguna acción en particular simplemente se invoca a la función que heredamos para cerrar el widget. Por último, ubicamos el template recientemente generado a la posición deseada.
Hasta este momento, el widget se observaría como lo muestra la figura 8.3. Lo que sigue a continuación se encuentra relacionado a la funcionalidad propia del widget, al cumplimiento de su objetivo.

```
1  comentarios.onReady = function () {
2      $(comentarios.getWidgetElement("#listaComentariosGenerales")).scrollTop(
3        $(comentarios.getWidgetElement("#listaComentariosGenerales"))[0]
4        .scrollHeight);
5
6      $(comentarios.getWidgetElement("#agregarComentarioGeneral")).on
7      ('click', function (event) {
8          if ($(comentarios.getWidgetElement("#textoComentarioGeneral")).val()
9           != "") {
10             var c = {};
11             c['texto'] = $(comentarios.getWidgetElement("#textoComentarioGeneral")).val();
12             result = comentarios.saveObject(c);
13             if (result != 0) {
14                 var textoComentario =
                   $(comentarios.getWidgetElement("#textoComentarioGeneral")).val();
15                 var data = {spanText: spanText,
16                             textoComentario : textoComentario};
17
18                  comentarios.injectInTemplate('principal','lineaComentario.html',
                    data,"#listaComentariosGenerales");
19             }
20         }
21         return false;
22     });
```

Figura 8.7 - Widget de Comentarios Generales - Definiendo Comportamiento

En la figura 8.7 podemos observar la función `onReady()`, donde definimos el comportamiento de nuestro widget de comentarios generales. A simple vista, no parece una cantidad muy numerosa de líneas de código, y podemos asegurar que solo con ellas se logró desarrollar la lógica este widget.
Echemos un vistazo a cada una de ellas, comenzando en la línea número 2, en donde utilizamos el método `getWidgetElement()` para obtener la lista de comentarios generales y posteriormente llevar la barra de scroll hacia la cima, de manera de que siempre que restaure el widget, sea el primer comentario de la lista el que se encuentre visible. Teniendo definida la parte de mostrar los comentarios generales que se encuentran actualmente el sitio situado, en este momento lo que nos resta para tener el widget finalizado es tener la posibilidad de insertar un comentario. Para ello, utilizamos el evento `onClick` que nos provee el lenguaje JavaScript, de manera de capturar el momento en que se presiona sobre el botón de "Agregar comentario". Posteriormente, nos resta validar que haya texto en el `textarea` donde se escribe un comentario, para evitar agregar uno vacío (línea de código número 5).
Lo único que queda es guardar el comentario en el servidor, para ello, debemos crear el objeto que vamos a guardar. Como mencionamos previamente cuando hablamos del

modelo de datos de este widget, solo nos interesa que el objeto a guardar posea el atributo `texto`, correspondiente al comentario realizado. Desde la línea 6 a la 9, creamos dicho objeto, y utilizamos la función `saveObject()` para almacenarlo. Luego, nos aseguramos que se haya guardado correctamente, preguntando por el código retornado en la variable `result`. Recordemos que cualquier número a excepción del 0, indica que la operación ha resultado exitosa.
Una vez guardado el comentario, debemos mostrarlo en la parte gráfica, es decir, agregarlo a la lista con los comentarios ya existentes en la página.

```
1  <li>
2   <div class="commentText">
3     <span class="sub-text">
4       <%= spanText %>
5     </span>
6     <p>
7       <%= textoComentario %>
8     </p>
9   </div>
10 </li>
```

Figura 8.8 - Widget de Comentarios Generales - Template Comentario Nuevo

En la figura 8.8 tenemos otro archivo HTML creado para este widget, relacionado con la inserción de un nuevo comentario. El mismo lo utilizaremos para añadir el comentario recientemente agregado a la lista de comentarios ya existentes. Podemos observar que utiliza dos campos, el primero será donde expondremos el usuario y la fecha de creación del comentario, mientras que el segundo, el texto propio del mismo.
Ahora si, retornando a la figura 8.7, debemos centrarnos en la línea de código número 16, dado que las anteriores están relacionadas en el armado de la fecha a mostrar. Podemos observar que se crea la variable `data`, que contiene los atributos correspondientes a los dos campos que acabamos de mencionar, que debe recibir el template para poder insertarlos en el HTML. Para finalizar, inyectamos en el template "principal", en el objeto identificado con selector css `#listaComentariosGenerales`, el archivo "lineaComentario.html" junto con el diccionario de datos recién creado, y por último, dejamos en blanco el `textarea` donde se escribe el comentario, de manera de dejarlo listo para recibir otra opinión de otro usuario.
De esta manera se desarrolló el widget de comentarios generales utilizando el framework SocialEye, el cual permitió codificarlo en muy pocas líneas, y sin ninguna complejidad. A medida que se avance con la exposición de cada uno de los widgets de prueba, la dificultad irá en ascenso, dada la lógica que poseen cada uno de ellos.

8.2 Comentarios específicos

8.2.1 Utilidad

Este widget se encuentra muy emparentado con el correspondiente a comentarios generales, dado que la función principal es similar, es decir, realizar comentarios y compartir opiniones con los demás usuarios acerca de lo que se está observando en el sitio web. La diferencia subyace en que en este caso lo que se desea es centralizar, o focalizar el asunto a debatir, de manera de que no sea tan general. Recordemos que en el widget de comentarios generales, existe un único debate por página web, de forma de que los comentarios son acerca de todo lo que la misma contiene. En este caso, en cambio, la ideal es tener varias "cajas" en un mismo sitio, especificando el debate por cada contenido que pueda tener. En este momento usted puede estar preguntándose cuál es el máximo de distintos debates que puede contener una sola página. La respuesta a esa pregunta es una decisión que se analizó con atención, y que se basa en la posibilidad de crear un nuevo debate por cada link que posee el sitio. La justificación se centra en que generalmente en la mayoría de los casos estos elementos HTML son utilizados para resaltar texto, y de esta manera acceder a su contenido específico cuando se hace click sobre ellos, por lo tanto creemos que cada link representa un tema o tópico diferente, que puede ser debatido de forma independiente.

8.2.2 Comportamiento

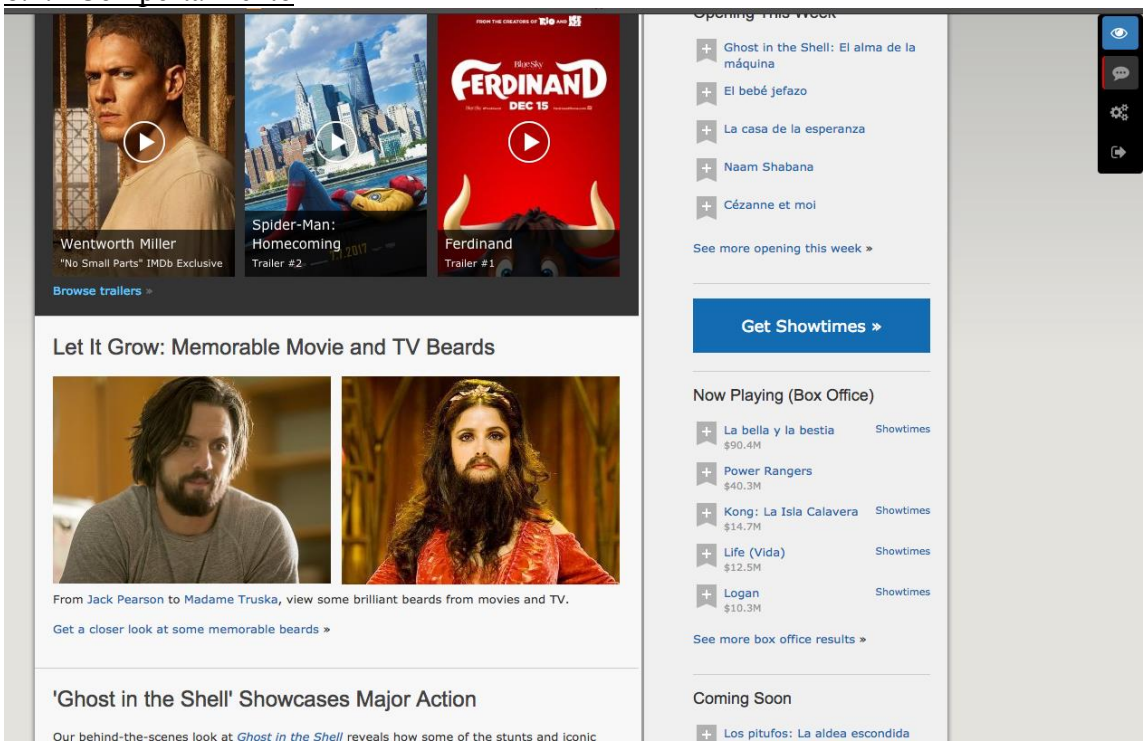

Figura 8.9 - Widget de Comentarios Específicos - Presentación

Empecemos a analizar el funcionamiento del widget de comentarios específicos desde el principio, momento en que la herramienta de navegación es abierta. Se puede observar que el widget se encuentra activo, esto quiere decir que a partir de este momento, cada vez que se presione sobre algún link del sitio en el que nos encontramos (imdb.com), en lugar de redirigirnos a la dirección web que contiene el link, se visualizará el box con los comentarios relacionados al tópico que resalta el mismo.

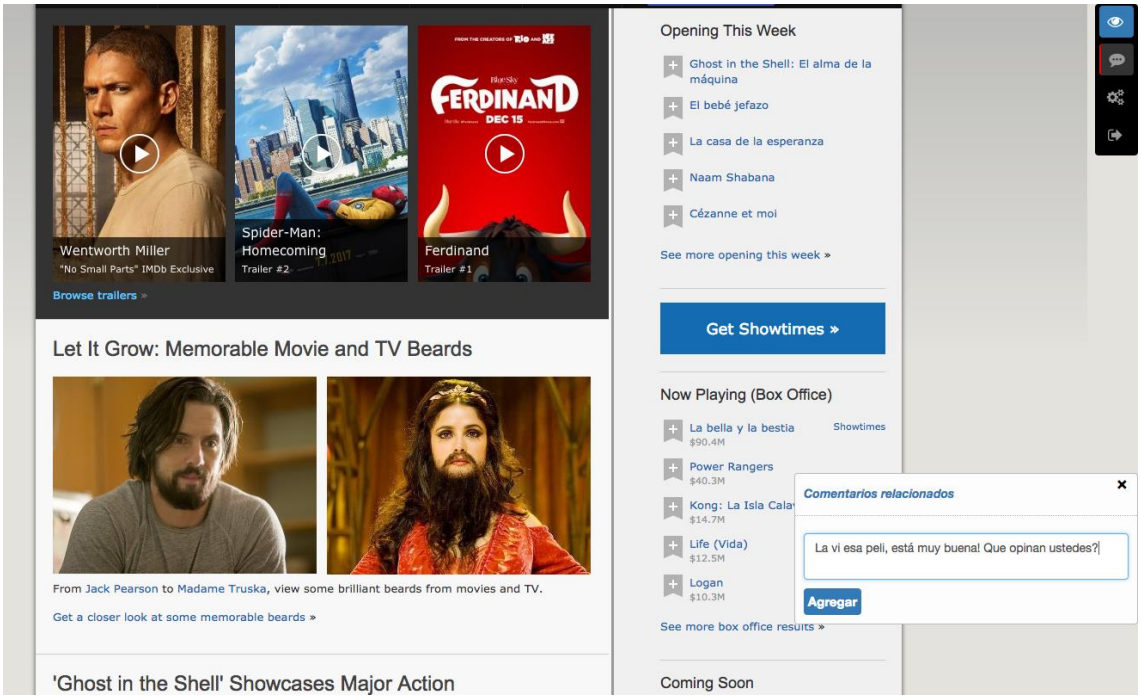

Figura 8.10 - Widget de Comentarios Específicos - Asociado de un comentario

Si el usuario presiona sobre el link con el texto “La bella y la bestia”, se abre una caja de comentarios cercano al link, con todos los comentarios generados a partir de ese link. Como todavía no existe ninguno, se procederá a generar el primero, pulsando en el botón “Agregar”, de la misma forma que en el widget anterior.

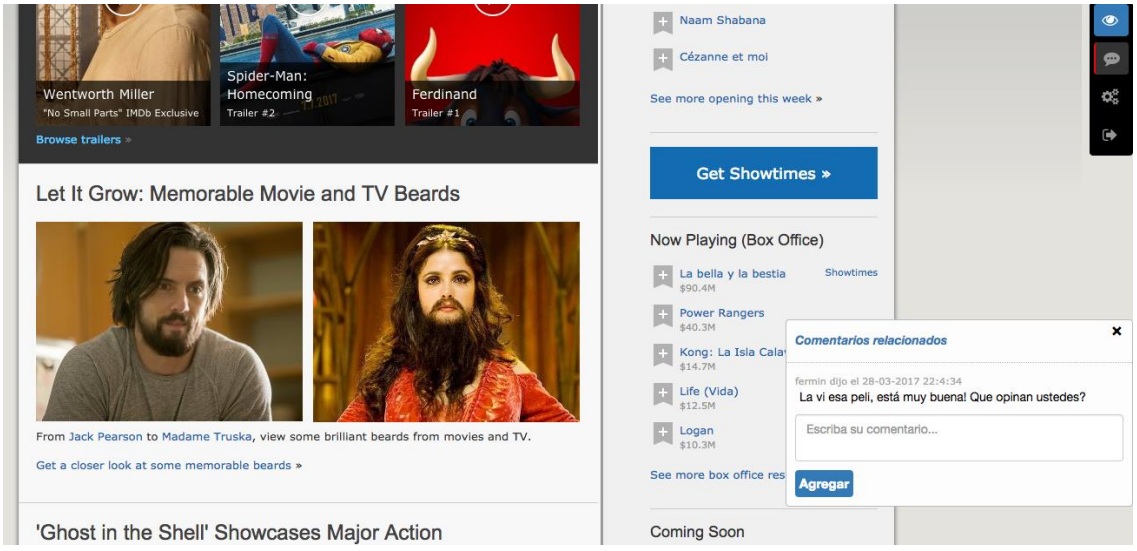

Figura 8.11 - Widget de Comentarios Específicos - Visualización del Comentario

El comentario ha sido agregado y se encuentra disponible para que otros usuarios puedan observarlo y comentar acerca de esa película. Usted ahora puede preguntarse cómo un usuario se entera de que existe un comentario en alguno de los links cuando recién ingresa al sitio. La respuesta es simple, y es que cada debate iniciado, llamando iniciado a aquel que posee al menos un comentario, estará representado como un ícono (con un símbolo de comentario desde luego), sobre el link correspondiente.

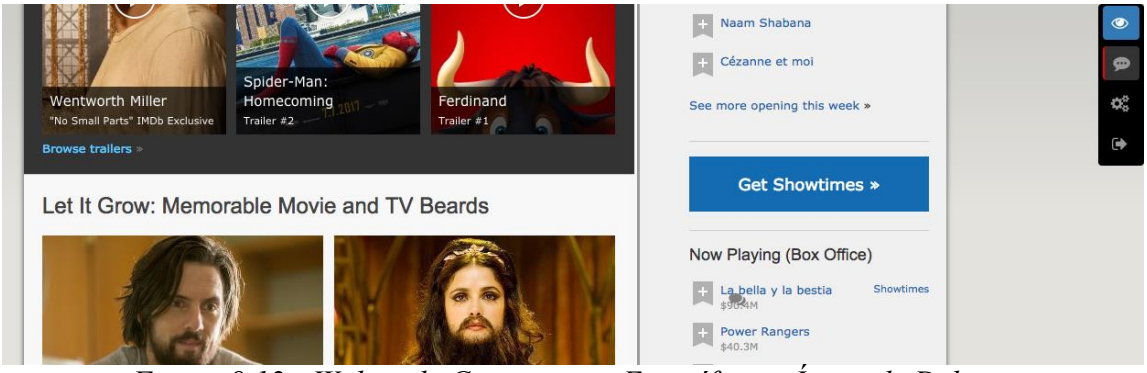

Figura 8.12 - Widget de Comentarios Específicos - Ícono de Debate

Podemos observar el ícono mencionado sobre el link, indicando la presencia de un debate existente. Con tan solo presionar sobre él, o sobre el mismo link, se visualizará en pantalla el debate con los comentarios correspondientes.

8.2.3 Modelo del Widget

El modelo del widget de comentarios específicos es bastante similar a su predecesor, con el agregado de un campo adicional. Esto quiere decir que en el atributo `element`, además del campo `texto`, se almacena el `tag`, para identificar la ubicación del link en donde fue generado el comentario. Para poder señalar cuando existe uno o más comentarios en una determinada url cuando se accede a una página web, se utiliza este campo, que representa la ruta en formato de XPath (XML Path Language), y en esa posición se presenta el icono con símbolo de comentario representando la existencia de los mismos. XPath es un lenguaje que permite construir expresiones que recorren y procesan un documento XML. La idea es parecida a las expresiones regulares para seleccionar partes de un texto sin atributos (texto plano). Permite buscar y seleccionar teniendo en cuenta la estructura jerárquica del XML. Actualmente los sitios web son muy dinámicos, por lo tanto puede ocurrir con frecuencia que la posición de los objetos cambien, o que un objeto no se encuentre más presente, por lo tanto aparecerían comentarios acerca de un tópico que ya no es el mismo que se encuentra en esa posición. El framework SocialEye contempla esta situación, teniendo en cuenta otros parámetros de referencia aparte del XPath, como el contenido del objeto, para decidir si incluir o no ciertos comentarios.

8.2.4 Desarrollo

Es momento de proseguir con el desarrollo del widget, centrándonos en la implementación utilizando el framework SocialEye. Si bien se puede suponer que la estructura del código es muy parecida a la de su predecesor, como por ejemplo el mismo archivo HTML para representar la "caja" en donde se muestran los comentarios, tiene sus diferencias por el hecho de manejar distintos debates dentro de una misma página.

```
1  var especificos = new Widget();
2
3  especificos.loadWidget = function() {
4      params = {};
5      params.tipo = 'comment';
6      items = especificos.getObjectsInUrl(window.location.href, params);
7      setPositionItems(items);
8      var data = {
9          items: items,
10         file: 'comentariosEspecificos.html'
11     };
12     especificos.createTemplate('principal', 'Comentarios Especificos',
13     'comentariosEspecificos.html', data,true);
14 }
15
16 function setPositionItems(items) {
17   $.each(items, function(index,value) {
18     var obj = getElementByXpath(value.element.tag);
19     value.element.data = value.element.tag;
20     value.element.tag = value.element.tag.replace(/[/]/g,'_');
21     value.element.tag = value.element.tag.replace(/\[/g,'-');
22     value.element.tag = value.element.tag.replace(/\]/g,'close');
23     value.element.positionTop = getOffset(obj).top;
24     value.element.positionLeft = getOffset(obj).left;
25   })
```

Figura 8.13 - Widget de Comentarios Específicos - Inicialización

Empezaremos con la función `loadWidget()`, destinada como ya dijimos a inicializar el widget y obtener del servidor los datos primordiales que necesitamos para poner en funcionamiento del mismo, acción que se lleva a cabo en la línea número 6 con la función `getObjectsInURL()`. Posteriormente, debemos indicar en qué posición se encuentra el comentario, para ello se creó la función `setPositionItems()`, la cual a partir del tag (XPath) del objeto, calcula las coordenadas exactas del mismo en el HTML. Estas coordenadas son utilizados luego para la creación del template principal en la línea 12, para indicar la posición donde debe mostrarse el objeto.

```
1  <% _.each(items, function(item) { %>
2    <a class='socialEye iconoComentarioEspecifico' title='Mostrar comentarios'
3    style='position:absolute; top:<%= item.element.positionTop %>px;
4     left:<%= item.element.positionLeft %>px'>
5    <span class='fa-stack fa-lg socialEye'>
6    <i id='<%= item.element.tag %>' data-social='<%= item.element.data %>'
7      class='fa fa-comments fa-stack-1x socialEye iconoClick'>
8  </i></span></a>
9    <% }); %>
10
```

Figura 8.14 - Widget de Comentarios Específicos - Template ícono comentario

La imagen corresponde al archivo comentariosEspecificos.html, utilizado como template principal, para mostrar con un símbolo de comentario en aquellos links donde exista al menos uno. Podemos observar que se hace de uso de la posición que

calculamos en la función `setPositionItems()` mencionada previamente para otorgarle al objeto la ubicación correcta.

```
especificos.onReady = function() {
    $('body').on("click", '.iconoComentarioEspecifico', showSpecificBox);
    changeClickListeners();
}

especificos.onCloseWidget = function() {
    revertClickListeners();
}

function changeClickListeners() {
    var all =
    toArray(document.getElementsByTagName('a')).concat(toArray(document.getElementsByTagName('div')
    ));
    for (var i = 0, max = all.length; i < max; i++) {
        all[i]._onclick = all[i].onclick;
        all[i].onclick = function(e) {
            if (!($(e.target).parents().hasClass('socialEye'))) {
                e.stopPropagation();
                e.preventDefault();
                showSpecificBox(e);
            }
        }
    }
}
```

Figura 8.15 - Widget de Comentarios Específicos - Definición de comportamiento

```
function revertClickListeners() {
    var all =
    toArray(document.getElementsByTagName('a')).concat(toArray(document.getElementsByTagName('div'))
    );
    for (var i = 0, max = all.length; i < max; i++) {
        if (!($(all[i]).parents().hasClass('socialEye')))
            all[i].onclick = all[i]._onclick;
    }
}
```

Figura 8.16 - Widget de Comentarios Específicos - Definición de comportamiento

Es momento de continuar con la función `onReady()`, ejecutada cuando se abre/restaura el widget. Lo principal que debemos hacer aquí, es lograr que cada vez que se haga click sobre cualquier link que la página posea, se abra la "caja" con el debate, en lugar de rederigirnos al link en cuestión. Para ello, se encuentran las funciones `changeClickListeners()` y `revertClickListeners()`, en donde cada una realizan lo opuesto, es decir, la primera se encarga de desactivar la redirección en los links, mientras que la segunda de otorgarle a los mismos nuevamente su condición de link. Esto último debe hacerse cuando el widget se cierra, es por ello que la función se invoca en `onCloseWidget()`.
Hasta aquí, nos enfocamos en la puesta en marcha del widget, ya sea desde traer los comentarios de la página web actual, hasta señalizar en donde se encuentran estos comentarios. Lo que sigue a continuación, es obtener los comentarios existentes cuando se hace click ya sea en el icono con símbolo de comentario, como en el link correspondiente, y agregar uno nuevo, muy similar al widget anterior.

```
1  function showSpecificBox(e) {
2      var tagAux = getXPath(e.target);
3      params = {};
4      params.tag = tagAux;
5      params.tipo = 'comment';
6      var boxObject = especificos.getObjectsInUrl(window.location.href, params);
7      positionLeft = e.pageX;
8      positionTop = e.pageY;
9      tagForId = tagAux.replace(/[/]/g,'_');
10     tagForId = tagForId.replace(/\[/g,'-');
11     tagForId = tagForId.replace(/\]/g,'close');
12     data = {
13       idWidget: this.idWidget,
14       positionLeft: positionLeft,
15       positionTop: positionTop,
16       tag: tagForId,
17       items:boxObject
18     }
19     especificos.injectInTemplate('principal', 'boxEspecifico.html',data,'',true);
```

Figura 8.17 - Widget de Comentarios Específicos - Carga de comentarios

Nos encontramos con la función `showSpecificBox()`, la cual es invocada desde `onReady()`, y es ejecutada cada vez que se presiona sobre un link. Se divide en dos partes, la primera se encarga de obtener los comentarios presentes en ese link, si es que existen, y mostrar la caja con los mismos, y la segunda parte que se visualizará en unos momentos se encarga de agregar un nuevo comentario.
En esta primera parte, como podemos observar, se obtiene el XPath del objeto presionado, para posteriormente obtener del servidor los comentarios que se encuentren en ese lugar (línea 6). Lo último que resta, es mostrar la "caja" con los comentarios, si es que existen, de ese link. De otra manera, se muestra vacía lista para recibir comentarios nuevos. Para ello, insertamos en el template principal, el archivo "boxEspecifico.html", que procesará y visualizará los respectivos comentarios. Este último archivo no se expondrá dado que es similar al widget predecesor, con la diferencia de que se ubica en el lugar del link presionado.

```
$(".agregarComentarioEspecifico").on('click', function(e) {
        var tag = e.target.id.substr(17);
        var usuarioComentario = especificos.getUser();
        var textoComentarioTag = $(e.target).parent().find("textarea");
        if (textoComentarioTag.val() != "") {
            obj = {}
            obj['texto'] = textoComentarioTag.val();
            obj['tipo'] = 'comment';
            obj['tag'] = tag;
            result = especificos.saveObject(obj);
            if (result != 0) {
                var parrafo = especificos.getP();
                parrafo.innerHTML = textoComentarioTag.val();
                var divComentario = especificos.getDiv();
                var lineaComentario = especificos.getLi();
                divComentario.appendChild(parrafo);
                lineaComentario.appendChild(divComentario);
                getListWithId("listaComentario"+tag).appendChild(lineaComentario);
                getListWithId("listaComentario"+tag).animate({
                    scrollTop: lista.children[0].scrollHeight
                });
            }
        }
        return false;
    });
```

Figura 8.18 - Widget de Comentarios Específicos - Guardado de un comentario

Para finalizar, tenemos la segunda parte de la función `ShowSpecificBox()`, encargada de guardar un nuevo comentario. Como podemos observar, es similar al widget de comentarios generales, utilizando la función `saveObject()` para guardar el objeto en el servidor, y posteriormente agregándolo a la "caja" correspondiente.

8.3 Encuestas

8.3.1 Utilidad

En los últimos años, con el incremento del uso de las redes sociales, se ha observado un crecimiento también en la utilización de encuestas, para medir la opinión de la gente en distintos temas de la actualidad, ya sea política, deportes, educación, turismo, etc. Generalmente, estas encuestas circulan en redes sociales como twitter, o sitios dedicados y exclusivos a realizar este tipo de procedimientos, por lo que debemos adecuarnos a las reglamentaciones, tiempos que estos nos imponen. Imaginemos la situación de estar navegando cualquier página web, ya sea de noticias, de compra y venta de artículos, de libros, películas, etc, y nos surja la necesidad de crear una encuesta para obtener opiniones acerca de lo que estamos observando, de manera de poder decidir acerca de una compra, de un destino turístico, de lo que se nos ocurra. De esta idea surgió el widget de encuestas, desarrollado para crear y votar encuestas relacionadas al contenido del sitio web que se está navegando, y sin ningún tipo de restricción acerca de cantidad de preguntas ni del tiempo de vigencia de la encuesta. Como todos los demás widgets, el objetivo principal de fondo es ampliar la interacción entre los usuarios de internet, de distintas maneras posibles.

8.3.2 Comportamiento

Es momento de presentar el funcionamiento del widget de encuestas. Para ello, al igual que todos los demás, expondremos capturas de pantalla para situarnos lo más cerca de la realidad, dejando así bien clara su usabilidad.

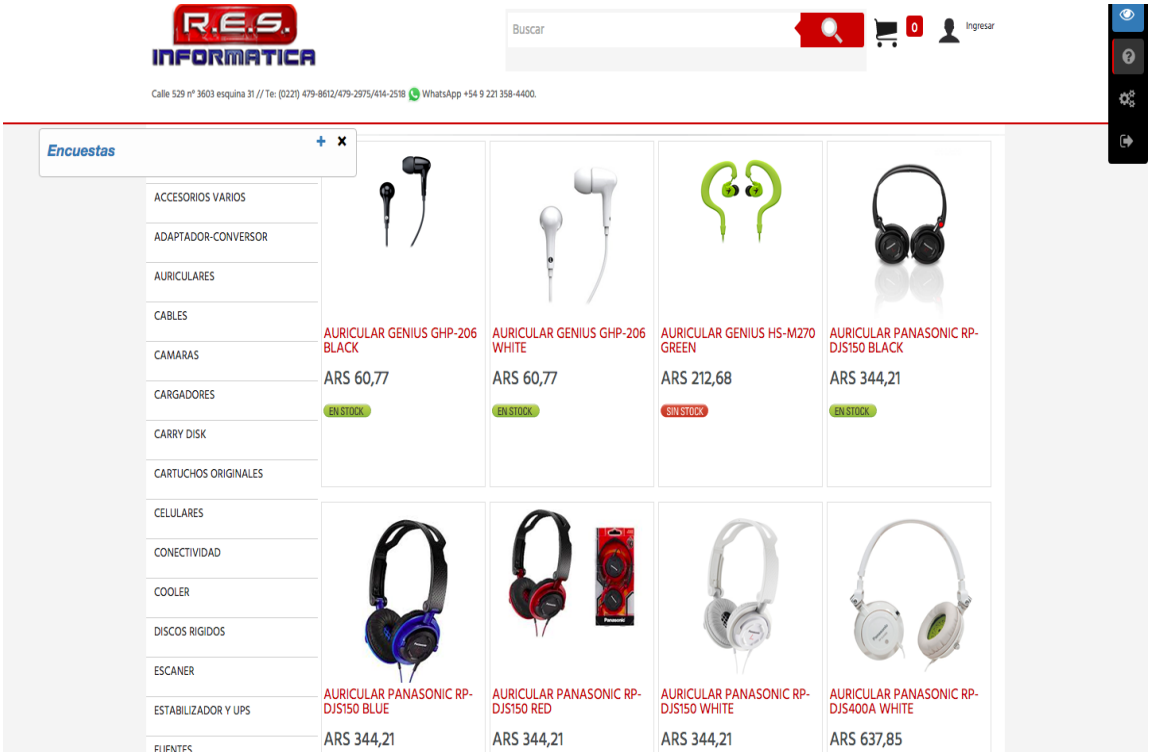

Figura 8.19 - Widget de Encuestas - Presentación

En la figura 8.19, se encuentra la interfaz principal perteneciente al widget de encuestas. Podemos detectar que es similar al de comentarios generales, manteniendo el estilo propio que llevarán todos los widgets. En este momento la lista se encuentra vacía, dado que no hay ninguna encuesta generada en este sitio. En primer lugar, prestemos atención al contenido del mismo, donde el propósito es vender productos informáticos y electrónicos. Luego, supondremos la actividad de un usuario tipo de internet, el cual suele realizar muchas de sus compras en internet, y en este momento está en búsqueda de un producto en particular (auriculares), pero con el inconveniente usual de no tener conocimiento del mismo, por lo cual la elección se torna complicada. Dado esto, estamos ante un buen caso para generar una nueva encuesta, y para ello el usuario en cuestión hace click en el icono con el símbolo de suma, para comenzar el proceso de creación.

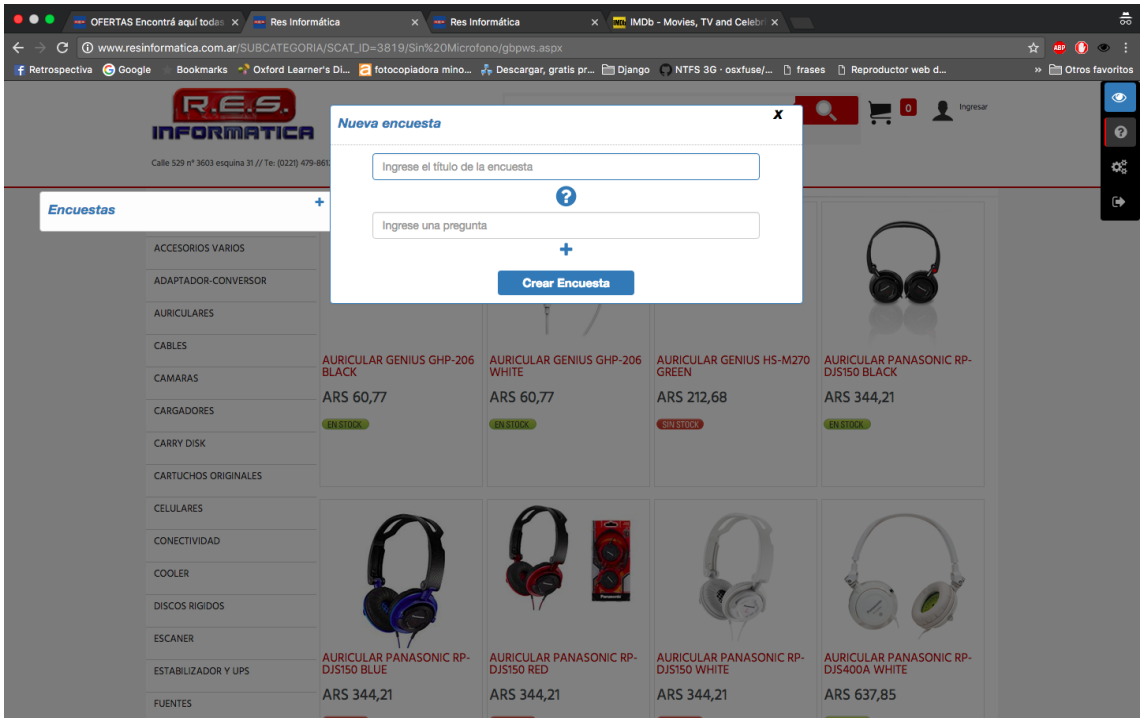

Figura 8.20 - Widget de Encuestas - Nueva Encuesta

Luego de esto, se presenta la imagen 8.20, que contiene el modal diseñado para agregar una nueva encuesta. En primer lugar, el usuario ingresa un título para la misma, destinado a describir con qué tópico va a estar relacionada. Posteriormente, se deben armar las preguntas que formarán parte de la encuesta, junto con sus respectivas opciones de respuesta. La condición mínima para poder crear una encuesta, es que la misma posea al menos una pregunta, que a su vez contenga al menos dos opciones de respuesta. En la imagen podemos observar dos íconos, el primero (símbolo de interrogación) debajo del título que llevará la encuesta, correspondiente a añadir una nueva pregunta, y el segundo (símbolo de suma), destinado a agregar las opciones de respuesta de cada consulta.

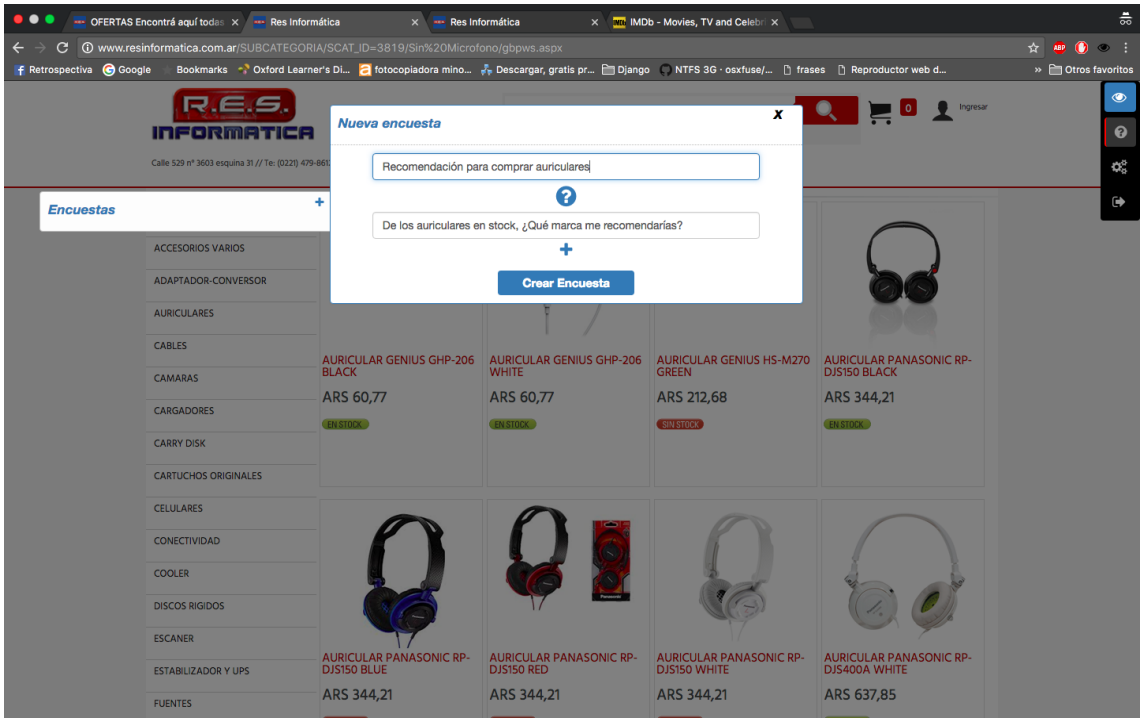

Figura 8.21 - Widget de Encuestas - Datos de Presentación

Podemos observar en la figura 8.21 que el usuario ya ha escogido tanto el título como la primera pregunta de su encuesta, por lo tanto es momento de elegir sus opciones, para lo cual el usuario hace click en el ícono con el símbolo de suma.

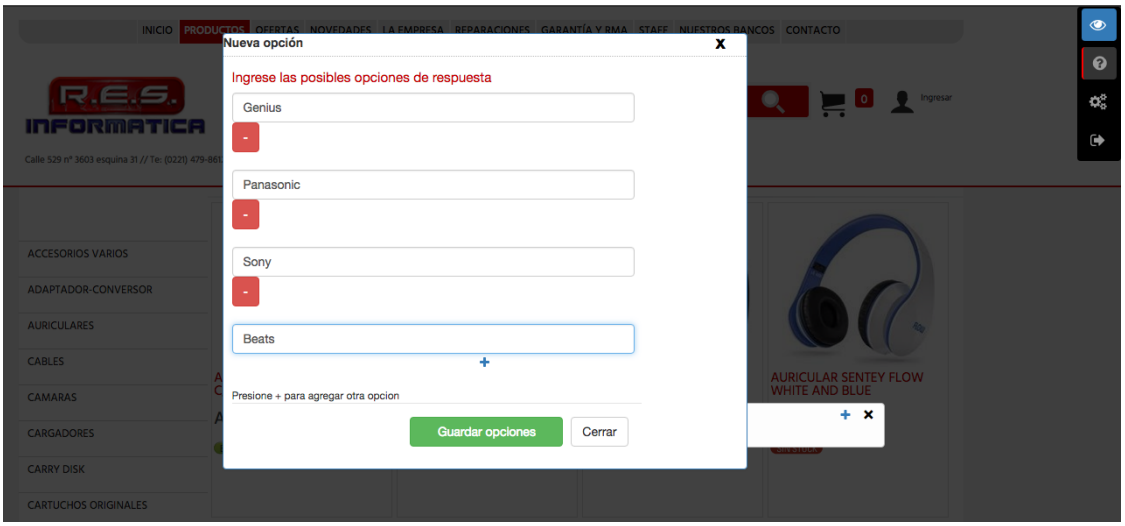

Figura 8.22 - Widget de Encuestas - Definiendo Opciones de Respuesta

En la figura 8.22 se visualiza la pantalla para elegir las opciones de respuesta a la pregunta. Con el símbolo de suma y de resta se añaden o quitan según la preferencia del usuario. En este caso ya se han agregado las distintas alternativas, por lo tanto el usuario procede a guardar las opciones. A partir de ese momento, las respuestas ya están almacenadas dentro de la pregunta, en caso de desear editarlas, simplemente basta con repetir el proceso.

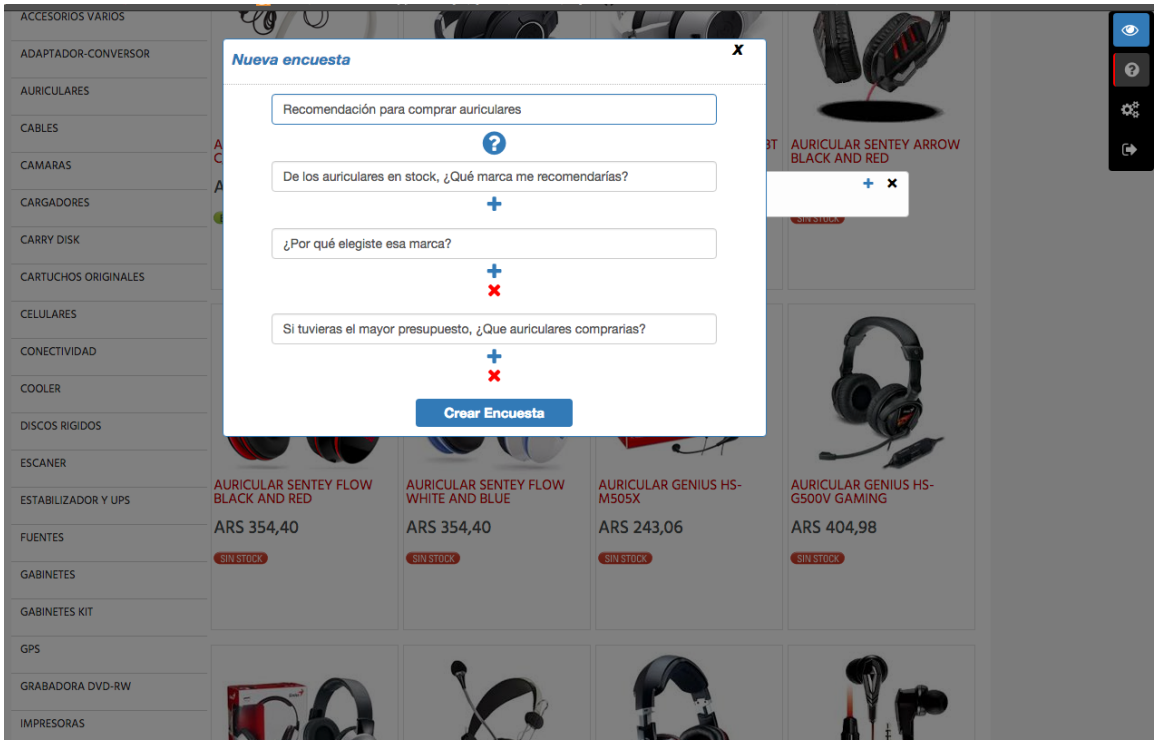

Figura 8.23 - Widget de Encuestas - Lista de Preguntas

En la figura 8.23 se observan las tres preguntas que formarán parte de esta encuesta, solo resta presionar en el botón “Crear Encuesta”, para finalizar el proceso de inserción.

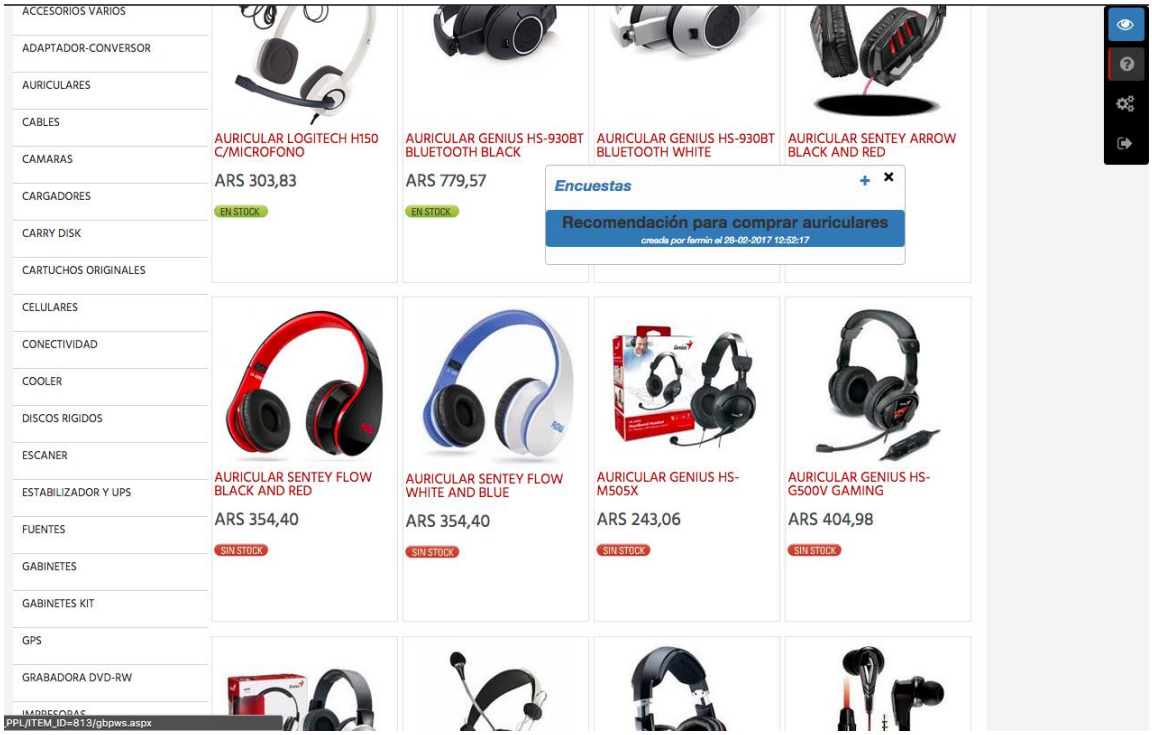

Figura 8.24 - Widget de Encuestas - Lista de Encuestas

La encuesta ha sido agregada, y ya forma parte del listado que contiene todas aquellas que fueron creadas en la página web actual, por lo tanto es momento a proceder a la otra parte de este widget, que es nada más ni nada menos de votar en una encuesta. Para

ello, tomemos como ejemplo el de un usuario de un widget que ingresa al sitio web, y observa la encuesta generada, y dado que conoce del tema porque ha comprado auriculares y le gusta la música, desea ayudarlo contestando las preguntas, entonces presiona sobre la encuesta que se encuentra en el listado.

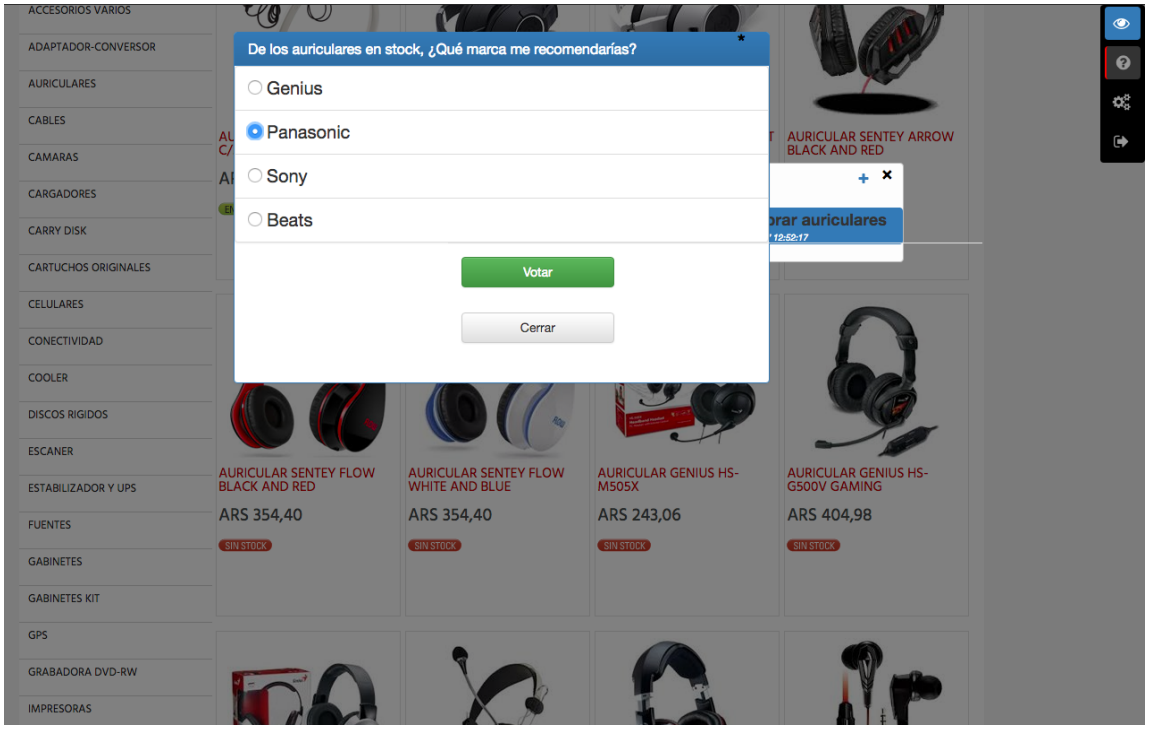

Figura 8.25 - Widget de Encuestas - Votación

En la figura 8.25 se muestra la interfaz para votar en una encuesta. En la esquina superior, aparece la pregunta actual, mientras que en el centro del modal, están las respectivas opciones de respuesta, y el botón para realizar la votación. El proceso de elección se dirige de una pregunta a la otra, es decir, cuando se presiona en “Votar”, se procede a la siguiente, repitiendo luego el primer paso, hasta culminar con todas las que conforman la encuesta. Posteriormente, automáticamente se presentan los resultados parciales para que el votante tenga una referencia hacia donde se inclina la tendencia. En cualquier momento, es posible presionar en el botón “Cerrar” de manera que la votación se cancela.

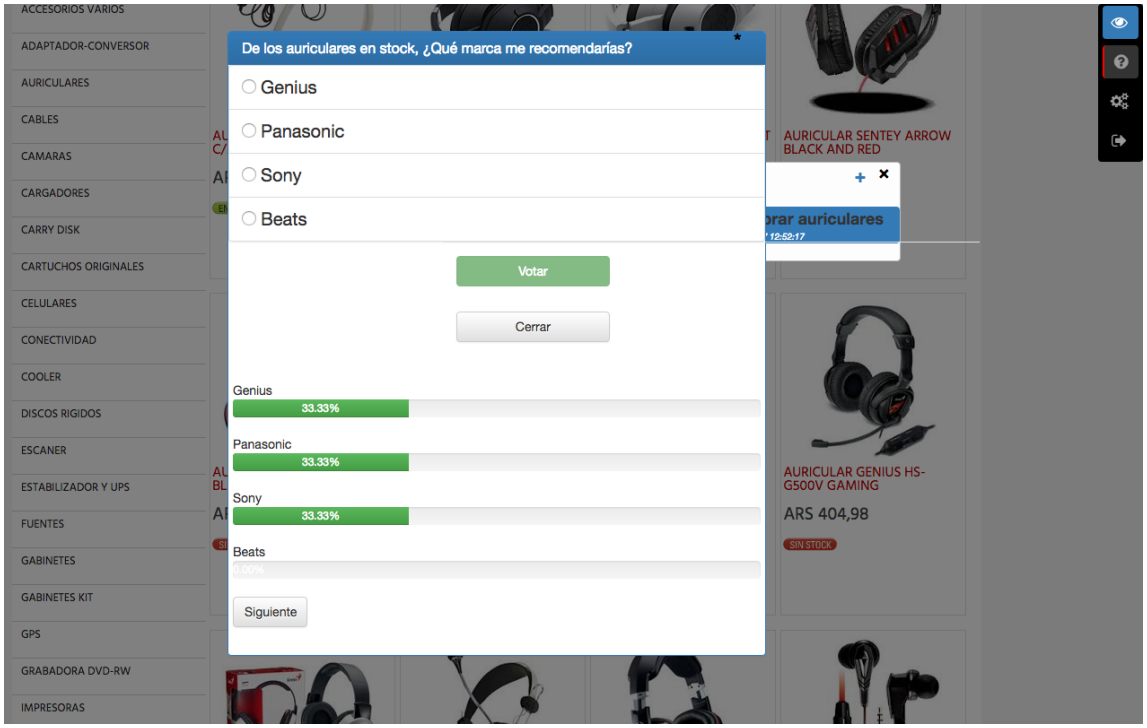

Figura 8.26 - Widget de Encuestas - Visualización de Resultados (Pregunta 1)

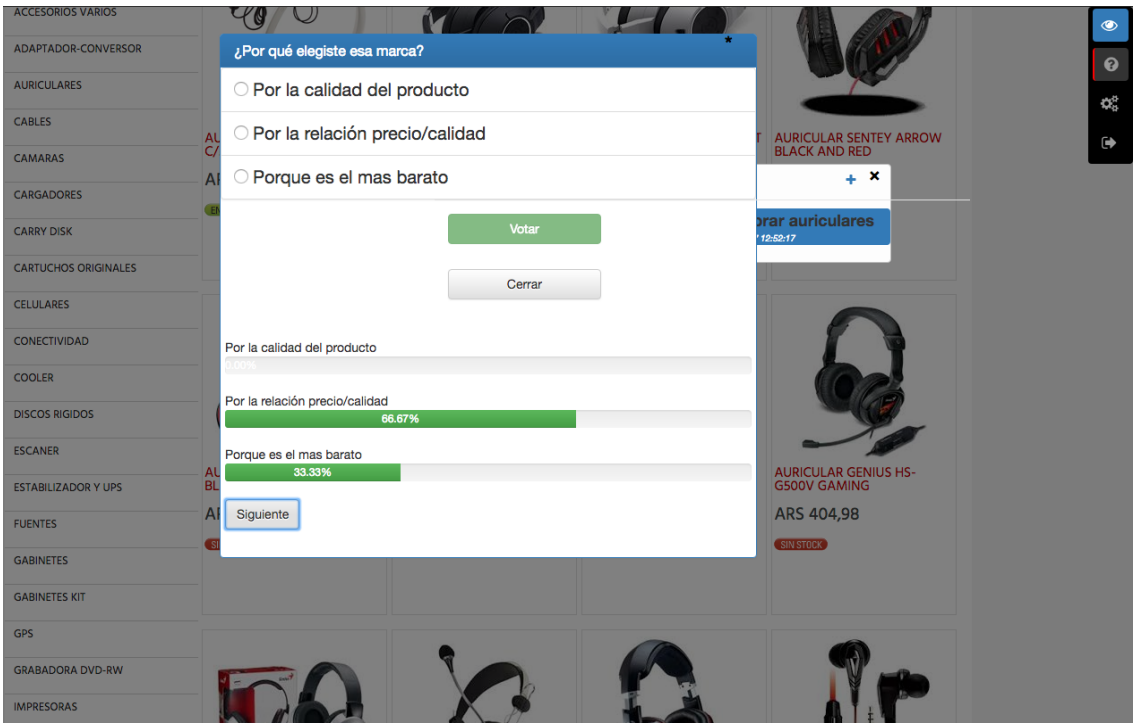

Figura 8.27 - Widget de Encuestas - Visualización de Resultados (Pregunta 2)

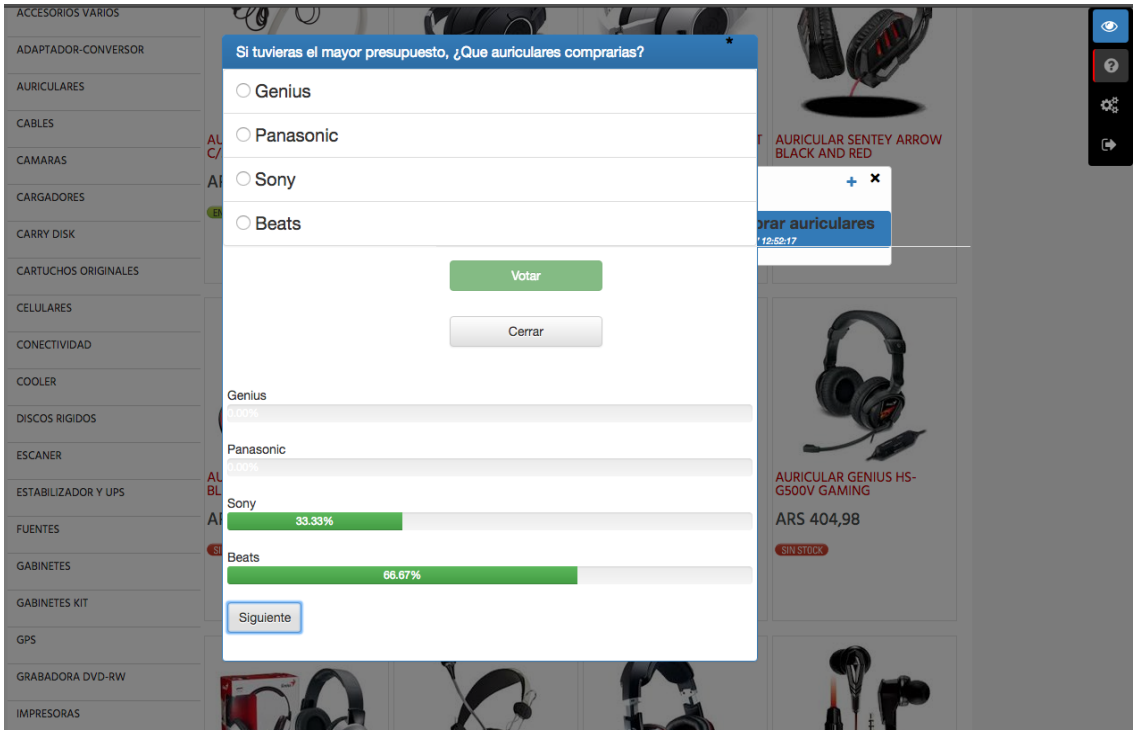

Figura 8.28 - Widget de Encuestas - Visualización de Resultados (Pregunta 3)

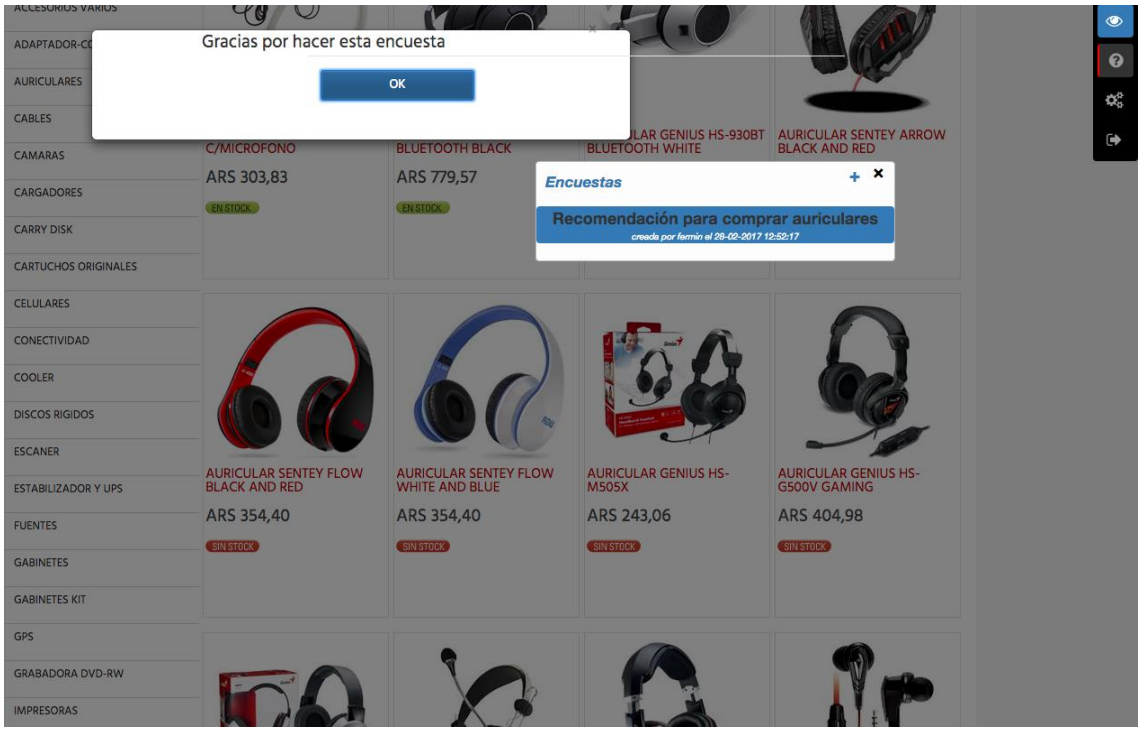

Figura 8.29 - Widget de Encuestas - Fin de Encuesta

En las cuatro imágenes anteriores se puede observar el proceso de presentación de los resultados, comenzando desde la primer pregunta, y pasando por cada una de ellas a través del botón “Siguiente”. El método se basa en mostrar una barra con la proporción de votos que posee cada posible opción de respuesta, junto con el porcentaje específico, para tener un número exacto si así se desea.

Aquí finaliza la primera parte del widget de encuestas, dedicada a introducir el objetivo de su desarrollo junto con la demostración de su funcionamiento. A partir de este momento continuaremos con la parte técnica, comenzando con el modelo para terminar con la codificación del widget.

8.3.3 Modelo del Widget

Previamente cuando hablamos del modelo en el widget de comentarios generales, se hizo la aclaración de que el foco lo íbamos a situar en el campo element de la tabla `Element`, dado que los demás campos son similares para todos los widgets. Este caso no es tan simple como el anterior, sino que posee un poco más de complejidad.

En el momento en que una encuesta es guardada, lo que ocurre de fondo es que en la tabla `Element`, se almacena un objeto que contiene en el campo `element` la información esencial a la hora de crear una encuesta, es decir, todas las preguntas y las opciones de respuesta disponibles por cada una. Tomando como ejemplo la encuesta agregada para mostrar el comportamiento del widget, el campo `element` contiene los siguientes datos:

```
{
 "tipo": "encuesta",
 "preguntas": {
   "1": {
     "options": [
       {
         "id": 0,
         "opcion": "Genius"
       },
       {
         "id": 1,
         "opcion": "Panasonic"
       },
       {
         "id": 2,
         "opcion": "Sony"
       },
       {
         "id": 3,
         "opcion": "Beats"
       }
     ],
     "pregunta": "De los auriculares en stock, ¿Qué marca
me recomendarías?"
   },
   "2": {
     "options": [
       {
         "id": 4,
```

```
          "opcion": "Por la calidad del producto"
        },
        {
          "id": 5,
          "opcion": "Por la relación precio/calidad"
        },
        {
          "id": 6,
          "opcion": "Porque es el mas barato"
        }
      ],
      "pregunta": "¿Por qué elegiste esa marca?"
    },
    "3": {
      "options": [
        {
          "id": 7,
          "opcion": "Genius"
        },
        {
          "id": 8,
          "opcion": "Panasonic"
        },
        {
          "id": 9,
          "opcion": "Sony"
        },
        {
          "id": 10,
          "opcion": "Beats"
        }
      ],
      "pregunta": "Si tuvieras el mayor presupuesto, ¿Que
auriculares comprarias?"
    }
  },
  "description": "Recomendación para comprar auriculares"
}
```

El objeto está compuesto por todas las preguntas que conforman la encuesta, en donde cada pregunta es un objeto que contiene el texto de la pregunta en sí, junto con todas las opciones de respuesta posibles. Además, obviamente, posee el campo `description` dedicado a la descripción de la encuesta, es decir, su título general para identificar a que hace referencia. Dejamos pendiente el campo `tipo` para dentro de unos momentos.

Usted se puede estar preguntando qué ocurre cuando realizamos el proceso de votación, y la respuesta a esa pregunta, se basa en que cada voto de cada opción es almacenado como un nuevo objeto en la tabla `Element`. El contenido del campo `element` es el siguiente:

```
{
 "tipo": "voto",
 "idOpcion": "1",
 "idEncuesta": "60"
}
```

Como podemos observar, se guarda el id que la opción tiene en esa pregunta, así como también el id de la encuesta que contiene a dicha opción. Además, nos encontramos nuevamente con el campo `tipo`, cuyo objetivo es identificar qué tipo de objeto es el almacenado en `element`. La justificativo de esta elección, está centrada en la necesidad de distinguir un objeto de otro a la hora almacenar los datos, para posteriormente recuperarlos en base a su tipo. Para dejar un poco más en claro el concepto, pensemos en el funcionamiento del widget de encuestas. En primer lugar, debemos obtener las encuestas agregadas en la página web en la que nos encontramos, pero luego de hacer una votación, para mostrar los resultados actuales de esa encuesta, debemos recuperar todos sus votos. Por lo tanto, pensamos en el campo `tipo` para distinguir estos dos tipos de objetos, y traerlos del servidor cuando cada uno sea requerido, ya que contamos con la opción de armar una consulta parametrizada y filtrar por atributos, como ya se ha explicado. Recordemos que el campo `element` es de tipo JSON, eso quiere decir que cada widget es libre de tener su modelo interno de datos en ese atributo como le sea más conveniente, en este caso se planificó de esta manera.

8.3.4 Desarrollo

Es momento de comenzar a analizar el desarrollo del widget de encuestas, para observar las facilidades que nos otorgó el framework SocialEye para codificarlo, tal como se hizo con los widgets previos.

```
1  <div>
2    <ul id='listaEncuestas'>
3      <% _.each(items, function(item) { %>
4        <li class='socialEye'>
5          <button class="listButton socialEye filaEncuesta" id="encuesta<%= item.id %>">
6            <div class="tituloEncuesta">
7              <%= item.element.description %>
8            </div>
9            <div class="subtituloEncuesta">
10             creada por <%= item.username %> el <%= item.date %>
11           </div>
12         </button>
13       </li>
14       <% }); %>
15   </ul>
16 </div>
```

Figura 8.30 - Widget de Encuestas - Template de Encuestas

En primer lugar, en la figura 8.30 observamos el archivo “encuestas.html”, el cual es insertado dentro del template principal del widget. Dado que esa primera pantalla es una

lista con las encuestas almacenadas en el sitio, esto explica que el contenido del archivo se base en un elemento HTML de tipo lista, conformado por cada una de las encuestas, situadas en la variable `items`. Además, se encuentra el botón correspondiente a abrir cada una de ellas, y para ello posee el id de la encuesta como id propio del elemento.

```
encuestas.loadWidget = function () {
    params = {};
    params['tipo'] = "encuesta";
    items = encuestas.getObjectsInUrl(window.location.href, params);
    $.each(items, function (i, item) {
        elementosEncuestas[item.id]=item.element;
    });


    var data = {idWidget: encuestas.idWidget,
              title : encuestas.tittle,
              items : items,
              file: 'encuestas.html'};
    encuestas.createTemplate('principal', 'Encuestas', 'encuestas.html', data);
    encuestas.injectInTemplate('principal', 'encuestas_header.html','.titleBox');
    encuestas.onCloseTemplate('principal', function(){
          encuestas.close();
     });
};
```

Figura 8.31 - Widget de Encuestas - Carga del Widget

A continuación, se presenta la función `loadWidget()`, fundamental para inicializar el widget cuando es necesario. En primer lugar, se procede a obtener las encuestas almacenadas en el sitio web actual, utilizando `getObjectsInUrl()`, las cuales conforman un diccionario en la variable `elementosEncuestas` que será utilizado durante el funcionamiento del widget. Posteriormente, se inicializa el objeto necesario para crear el template principal y que abastecerá de los datos que requiere el archivo encuestas.html que se describió recientemente, y se invoca a la función `createTemplate()` para construirlo. En este momento se tiene instanciado el template principal del widget de encuestas, de las últimas dos líneas de código solo hace falta aclarar que la primera de ellas agrega el archivo HTML encargado de agregar el símbolo de suma que nos trasladará a la pantalla para añadir una nueva encuesta, mientras que la segunda, redefine la función `onCloseTemplate()`, para agregar comportamiento cuando el widget se cierre.

En este punto se debe aclarar que dado que este widget posee un número mayor de líneas de código, expondremos aquellas que creemos fundamentales para el comportamiento del mismo, y sobre todo aquellas que hacen mayor uso del framework, objetivo de esta tesina.

```
encuestas.onReady = function () {
    $(encuestas.getWidgetElement(".filaEncuesta")).on('click',function(){
        idEncuestaActual=getOriginalId(encuestas.idWidget,this.id);
        encuestas.injectInTemplate('principal', 'encuestas_votacion.html','#listaEncuestas');
        votosResultados = {};
        $(encuestas.getWidgetElement("#votar")).prop( "disabled", false );
        votosAGuardar = [];
        votos={};
        $(encuestas.getWidgetElement("#resultados")).hide();
        numeroActualResultado=1;
        numPreguntaEncuesta=0;
        questions = elementosEncuestas[idEncuestaActual].preguntas;
        $(encuestas.getWidgetElement("#vote")).modal('show');
        siguientePregunta();
    });
```

Figura 8.32 - Widget de Encuestas - Comportamiento Inicial

En la imagen 8.32 se empieza a abordar la función `onReady()`, contenedora de toda la lógica del widget. Se puede observar que el primer paso que se da, es el de mostrar el modal para realizar el proceso de votación de una encuesta. Para ello, se inyecta el modal que es el encargado de mostrar esa funcionalidad (encuesta_votacion.html).

```
$(encuestas.getWidgetElement("#votar")).on('click',function(){
      if ($(encuestas.getWidgetElement("#listaOpciones")).find('input:radio').is(":checked")) {
          votosAGuardar.push($('#listaOpciones input:checked').attr('id'));
      }
      else {
          bootbox.alert("Debe seleccionar una opcion");
          return false;
      }
      siguientePregunta();
  });
```

Figura 8.33 - Widget de Encuestas - Acción de Votar

Continuando con lo referido a votar en una encuesta, cuando se presiona sobre el botón "votar", el accionar se basa en guardar en un arreglo el id del voto seleccionado, y pasar a la siguiente pregunta, en donde lo que se realiza es inyectar un nuevo HTML con la lista de opciones, para que el usuario continúe con la siguiente elección. En el momento en que se llega a la última pregunta, se crea un objeto por cada opción votada, y se almacena en el servidor, para proseguir con la vista de los resultados parciales, como se observa en la figura que se presenta a continuación, correspondiente a la función `siguientePregunta()`.

```
function siguientePregunta(){
      numPreguntaEncuesta++;
      if (numPreguntaEncuesta<=Object.keys(questions).length){
          $(encuestas.getWidgetElement("#tituloPregunta")).html(questions[numPreguntaEncuesta].pr
          egunta);
          $(encuestas.getWidgetElement("#listaOpciones")).empty();
          var data = {questions : questions,numPreguntaEncuesta:numPreguntaEncuesta};
          encuestas.injectInTemplate('principal',
          'encuestas_lista_opciones.html',data,'#listaOpciones');
      }
      else {
          $.each(votosAGuardar,function (index,option){
              voto.idEncuesta = idEncuestaActual;
              voto.idOpcion = option;
              voto.tipo = 'voto';
              encuestas.saveObject(voto);
          });
          params['tipo'] = 'voto';
          params['idEncuesta'] = idEncuestaActual;
          $(encuestas.getWidgetElement("#votar")).prop( "disabled", true );
          votosResultados=encuestas.getObjects(params);
          $(encuestas.getWidgetElement("#resultados")).show();
          siguienteResultado();
      }
  }
```

Figura 8.34 - Widget de Encuestas - Función SiguientePregunta()

```
<% _.each(questions[numPreguntaEncuesta].options, function (option) { %>
  <li class='list-group-item opcion'>
    <div class='radio'>
      <label>
        <input id='<%= option['id'] %>' class='<%= option['id'] %>' type='radio'
        name='optionsRadios'/>
        <%= option['opcion'] %>
      </label>
    </div>
  </li>
<% }); %>
```

Figura 8.35 - Widget de Encuestas - Template con Listado de Opciones

El Archivo encuestas_lista_opciones.html, presentado en la figura 8.35, se encarga de mostrar las opciones asociadas a cada pregunta.

En cuanto a los resultados parciales, el concepto es similar al de votación, es decir, se recorren las preguntas, y se realiza un promedio con los votos para poder exponer el porcentaje. A continuación, se observa el archivo “encuestas_lista_resultados.html” inyectado en el template para presentar los resultados.

```
<% _.each(questions[numeroActualResultado].options, function (option) { %>
  <li class='list-group-item opcion'>
    <div class='radio'>
      <label>
        <input id='<%= option['id'] %>' type='radio' name='optionsRadios'/>
        <%= option['opcion'] %>
      </label>
    </div>
  </li>
<% }); %>
```

Figura 8.36 - Widget de Encuestas - Template de Resultados en Preguntas

Posteriormente al flujo de votación y muestra de resultados de una encuesta, es momento de proseguir con la creación de una encuesta. Para ello, como se detalló en la sección anterior, se creó un modal dedicado a esta funcionalidad, que se visualiza una vez apretado el botón con símbolo de uma ubicado en la esquina superior derecha del widget.

```
<div class='modal fade' id='encueste' tabindex='-1' role='dialog' aria-labelledby='voteLabel' aria-hi
  <div class='titleBox' id='divNuevaEncuesta'>
    <button class='botonCerrar cerrarCrear'>x</button>
        <label>Nueva encuesta</label>
  </div>
  <div class='actionBox'>
    <div class='modal-body'>
      <form id='formEncuesta'>
        <input type='text' id='titEncuesta' class='form-control inputEncuesta' placeholder='Ingrese e
          <a id='agregarPregunta' title='Nueva pregunta'><i class='fa fa-question-circle fa-stack-1x'
        </div>
        <div id='divPreguntas'>
        <input type='text' id='pregunta<%= numPregunta %>' class='form-control inputPregunta' placeho
          <a id=<%= numPregunta %> title='Nueva opción' class='agregarOpcion'><i class='fa fa-plus fa
        </div>
        <div id='divBotonNuevaEncuesta'><button id='agregarEncuesta' class='submitButton'>Crear Encue
        <div>
      </form>
      <div class='modal-body'>
        <ul id='listaOpciones' class='list-group'/>
    </div>
    </div>
</div>
```

Figura 8.37 - Widget de Encuestas - Template Nueva Encuesta

La imagen pertenece al archivo HTML del modal de creación de encuesta, encargado de mostrar las preguntas agregadas junto con sus correspondientes opciones de respuesta. El usuario tiene la posibilidad de ir cargando las preguntas, y en el momento en que lo desee, cargar las posibles respuestas, sin necesidad de que sea inmediatamente que se añade una pregunta. Para ello tiene disponible otro modal en el que se insertan las opciones para la pregunta seleccionada, pero no se detallará el código del mismo dado que es trivial, pasando al momento de guardar la encuesta ya creada.

```
function enviarEncuesta(preguntas) {
    var inputs = $('#divPreguntas').find('input');
    $.each(inputs, function (index, value) {
        var cortado = $.trim($(value).val());
        if (cortado ==""){
            if (!(typeof preguntas[index+1]==="undefined") && !(typeof
            preguntas[index+1].options==="undefined")) {
                if (preguntas[index+1].options.length > 1) {
                    bootbox.alert("El titulo de la pregunta esta vacio pero contiene al menos dos
                    opciones como respuesta");
                }
            }
        }
        else {
            if ((typeof preguntas[index+1]==="undefined") || (typeof
            preguntas[index+1].options==="undefined")) {
                bootbox.alert("La pregunta no posee opciones de respuesta");
            }
            else{
                bootbox.alert("La pregunta debe poseer al menos opciones de respuesta");
            }
        }
    });
```

Figura 8.38 - Widget de Encuestas - Almacenamiento de una Encuesta

Una vez presionado el botón para añadir una nueva encuesta, lo que debe realizarse es guardar la misma en la base de datos. La función `enviarEncuesta()` fue creada con tal propósito, y en la imagen previa observamos la primer parte del proceso, que consiste en verificar que la encuesta a agregar cumpla ciertas validaciones obligatorias, estas son, que haya al menos una pregunta, y que en cada una de ellas existan al menos dos opciones posibles de respuestas.

```
    if (hayAlMenosUna==false) {
        if (pasoValidacion==true)
            bootbox.alert("La encuesta debe tener al menos una pregunta");
        return false;
    }
    var encues = {};
    encues.description = $("#titEncuesta").val();
    encues.preguntas = preguntas;
    encues.tipo = "encuesta";
    idAgregado=encuestas.saveObject(encues);
    var item = {};
    item.username = localStorage['username'];
    var d = new Date();
    var hs = d.getHours();
    var mins = d.getMinutes();
    var secs = d.getSeconds();
    item.date = $.datepicker.formatDate('dd-mm-yy', d) + " " + hs + ":" + mins + ":" + secs;
    item.id = idAgregado;
    item.element = {};
    item.element.description = $("#titEncuesta").val();
    item.element.preguntas = preguntas;
    elementosEncuestas[item.id]=item.element;
    lista[0].appendChild(agregarFilaAEncuesta(item));
    return true;
}
```

Figura 8.39 - Widget de Encuestas - Definición de Objeto Encuesta

Una vez que se cumplen todas las condiciones para poder añadir la encuesta, lo que sigue a continuación es crear el objeto que se almacenará en el campo `element`. Esto se puede observar entre las líneas número seis y la número diez, donde se crea el objeto

con los atributos `descripción`, `preguntas` y `tipo`. El primero hace referencia a la descripción general que tiene la encuesta, en qué tópico se enfoca, mientras que preguntas es un objeto conformado por todas las preguntas, valga la redundancia, y las opciones de respuesta para cada una de ellas. Por último, como mencionamos en la sección anterior, el campo tipo diferencia entre los dos objetos distintos que pueden persistirse en el widget de encuestas, estos son, encuestas y votos. Luego de crear dicho objeto, utilizamos la función `saveObject()` para almacenarlo, restando solamente mostrar en el listado de encuestas la recientemente creada. Por lo tanto, se toman los datos del cuestionario añadido, agregando la fecha actual, y se envía como parámetro de la función `agregarFilaEncuesta()`, que no hace más que insertar al listado actual de encuestas, una nueva fila.

8.4 Interacción de usuarios

Es hora de exponer un widget predeterminado algo más complejo, ya que el mismo integrará un conjunto de funcionalidades que tendrán a los usuarios interactuando "peer to peer". A su vez, el mismo contará con el uso de librerías y herramientas externas, a fin de mostrar la total compatibilidad del Framework con cualquier otro instrumento de desarrollo.

8.4.1 Utilidad

La función del widget que formularemos a continuación propone la interacción directa entre dos usuarios que utilizan Social Eye. Así, un usuario que abre el widget en cuestión, podrá contactar a cualquier otra persona que se encuentre disponible (está utilizando el mismo complemento y navegando el mismo dominio web). Los usuarios que se encuentren presentes bajo las condiciones detalladas, serán mostrados para el cliente que abrió el widget en forma de lista.
Los distintos métodos de interacción que propone el plugin son:
*Chat: intercambio de mensajes escritos entre los usuarios en forma de conversación.
*Videollamada: posibilidad de realizar una "call" en tiempo real entre ambos usuarios, utilizando un micrófono, parlantes y una cámara web.
Quizás la sola idea de proponer estas funciones puede pensarse como un desarrollo complejo; sin embargo, a lo largo de esta sección detallaremos cómo el uso del Framework simplifica en muchos aspectos su implementación.

8.4.2 Comportamiento

Comenzaremos a detallar cada una de las funcionalidades del widget, presentando qué puede hacer y cómo lo hace.
El escenario que propondremos será el de un usuario de Social Eye navegando una página web cualquiera, y utilizando a su vez el widget presentado. De esta forma, por la siguiente imágen podemos darnos cuenta cómo se vería tal circunstancia.

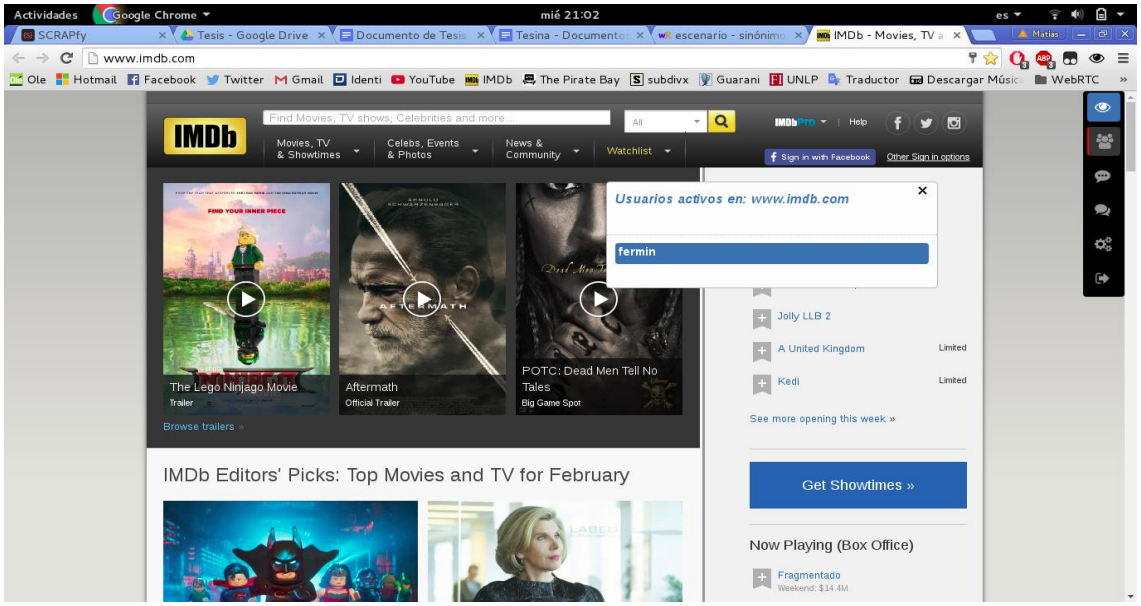

Figura 8.40 - Widget de Interacción de Usuarios - Lista de Usuarios Conectados

Por la figura 8.40 presentaremos la primera función del widget: la lista de usuarios conectados. A modo de mostrar un ejemplo simple, tenemos el suceso en el cual el usuario "Matias" abre el widget en cuestión y se encuentra con la lista de usuarios activos en la web actual, compuesta en este caso por un único usuario "Fermin". Acto seguido, Matias hace click sobre el recuadro que identifica al usuario Fermin, con el fin de entablar una conversación con el mismo.

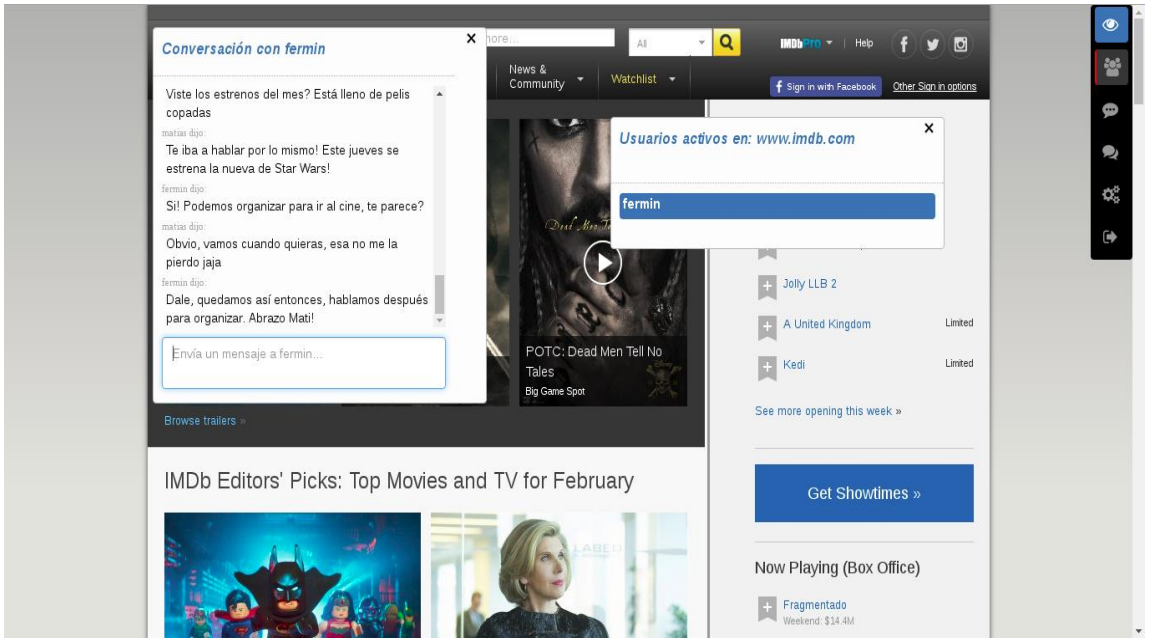

Figura 8.41 - Widget de Interacción de Usuarios - Chat

Como podemos apreciar en la figura 8.41, ya existía una conversación previa entre los usuarios, por lo cual el historial de la misma se visualiza en el recuadro de los mensajes. Con solo escribir un texto en el recuadro de "Envía un mensaje a fermín…" y pulsar la tecla "Enter", Matías podrá conversar en tiempo real con el usuario en cuestión, y verá su nuevo mensaje como parte de la conversación.

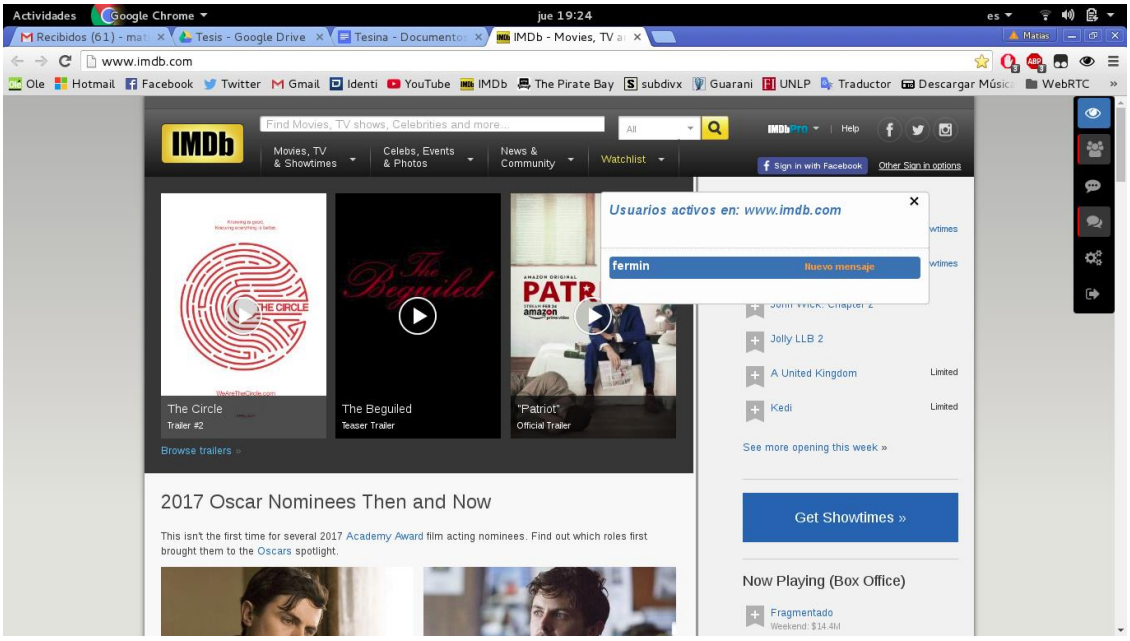

Figura 8.42 - Widget de Interacción de Usuarios - Notificación de Mensaje

Como con cualquier aplicación o servicio de mensajería, puede darse el caso en el cual el usuario no tiene abierta la conversación con otro al momento en el que este último le envía un mensaje, puede que esté hablando con otra persona o que simplemente no tenga ninguna conversación abierta pero sí "prendido el widget". En la figura 8.42 se muestra la función del widget que, con motivo de informar al usuario cuando se sucede una situación de este estilo, hará aparecer en el recuadro del usuario que lo está contactando, un pequeño aviso con el detalle "Nuevo mensaje". Una vez que el usuario se percate que tiene una conversación sin atender y haga click en el recuadro correspondiente para abrirla, el aviso desaparecerá.

Una vez presentada la vía por la cual el widget de usuarios nos permite contactar mediante mensajes directos a otras personas que estén usando el complemento, pasaremos a mostrar la segunda función principal del plugin: las videollamadas.

Figura 8.43 - Widget de Interacción de Usuarios - Ícono de Videollamada

Como la videollamada necesita de una conexión segura para establecerse, el ícono que nos permitirá iniciar esta funcionalidad sólo se verá presente al navegar en páginas webs que cuenten con HTTPS. En este caso, en la figura 8.43 podemos ver el ícono de una cámara a la derecha del box de mensaje, mientras navegamos en la web de la W3.

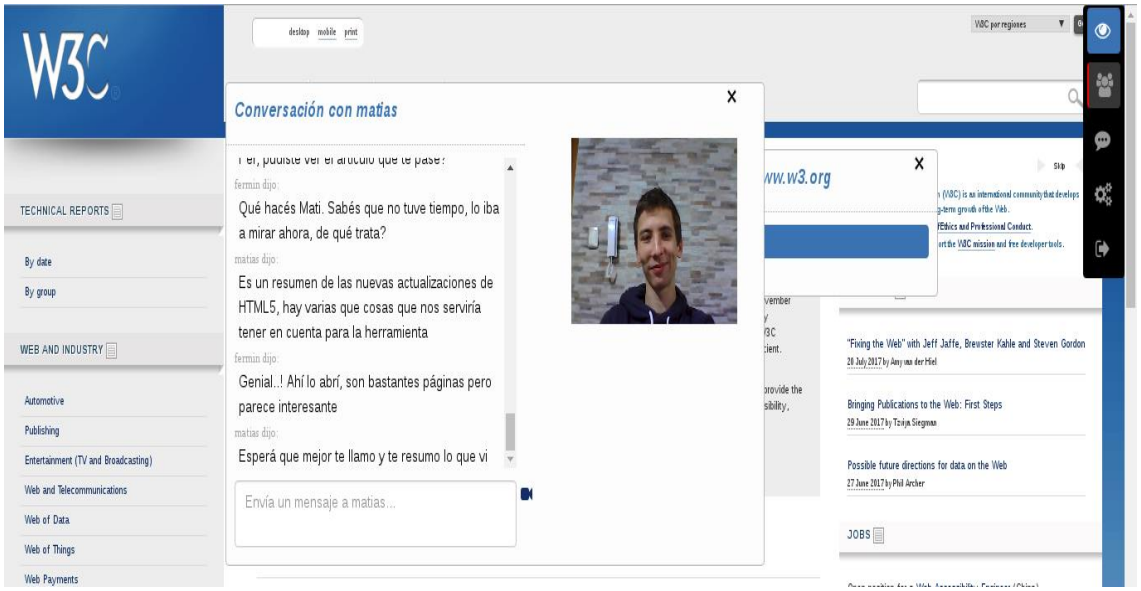

Figura 8.44 - Widget de Interacción de Usuarios - Solicitud de Videollamada

Ni bien el usuario hace click sobre el ícono en cuestión, la videollamada inicia. La figura 8.44 nos muestra lo sucedido: la ventana de conversación se amplía a la derecha descubriendo el recuadro donde se ubica lo reproducido por la cámara web de quien inicia la videollamada, a la espera de la respuesta del otro usuario.

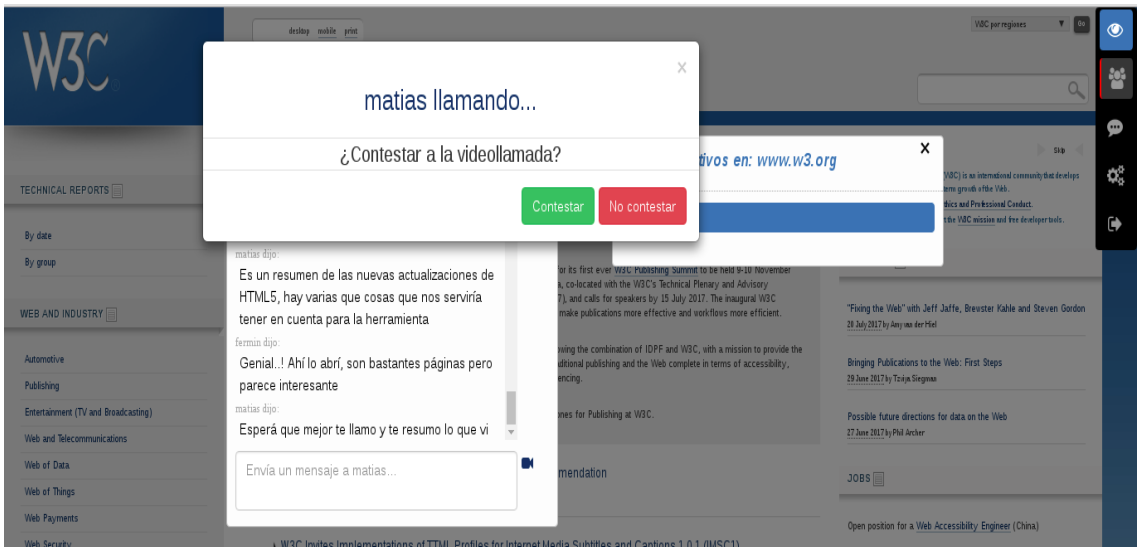

Figura 8.45 - Widget de Interacción de Usuarios - Respuesta de Videollamada

Del lado del usuario que está siendo llamado, tendremos un escenario como el que muestra la figura 8.45, donde al momento de recibir la solicitud, se visualiza un box de aviso indicando qué usuario está intentando realizar la videollamada, y solicitando una respuesta.

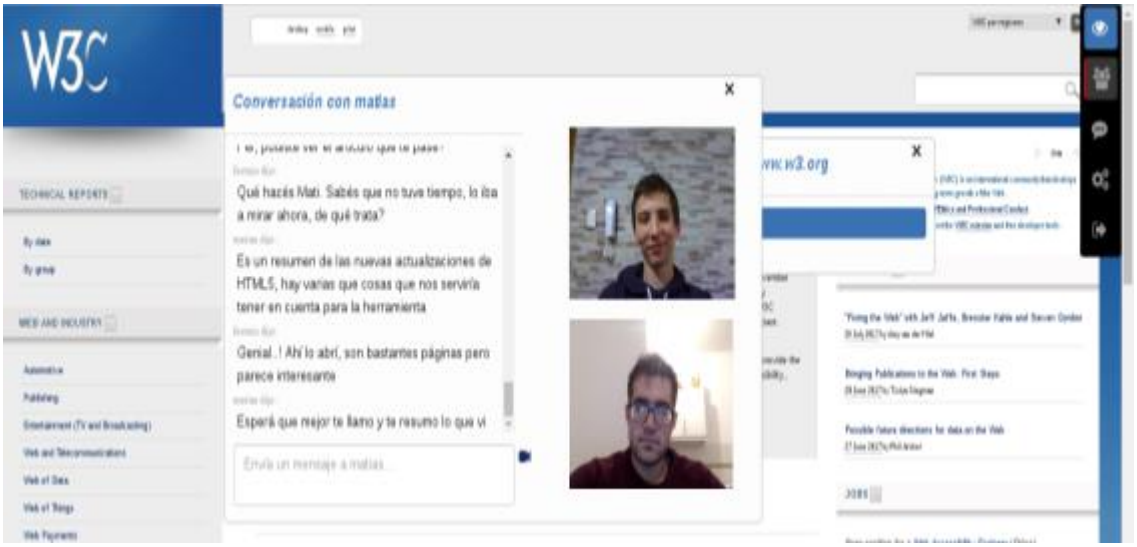

Figura 8.46 -Widget de Interacción de Usuarios - Videollamada establecida

En el caso en que el usuario decida contestar, se establecerá la conexión entre ambos, compartiendo cada uno lo reproducido por su cámara web. Podemos ver en la figura 8.46 que el lugar que estaba vacío en el recuadro de conversación ahora se ve ocupado por el video en tiempo real del usuario contactado. Bastará con que cualquiera de los usuarios vuelva a hacer click sobre el icono de videollamada, para darle fin a la misma y que el recuadro de conversación vuelva a su tamaño original.

Todo lo que respecta a la actividad asociada a una videollamada, se visualizará como mensaje en la misma conversación entre los usuarios. De esta forma el widget nos avisa lo sucedido en los casos en que se haya rechazado, establecido, finalizado, perdido o no contestado una videollamada.
Cabe destacar que, así como sucede con los avisos de "Nuevo mensaje", las solicitudes de llamadas entrantes se mostrarán al usuario contactado de forma inmediata, tenga abierta o no la conversación con el otro usuario.

Figura 8.47 - Widget de Interacción de Usuarios - Estado de Desconexión

Finalmente, pasamos a presentar el último estado posible del widget. Se trata del evento en el cual uno de los usuarios que se encontraba protagonizando una conversación, se desconecta del complemento, ya sea porque cerró el mismo o por cualquier otro suceso (cierre de navegador, desconexión de internet, etc). En este caso, la imagen con la que el otro usuario se encuentra se ilustra en la figura 8.47:
*El título de la conversación ahora agregó el detalle "(desconectado)" asociado al otro usuario.
*Se imposibilita el enviar mensajes al usuario desconectado.
*El usuario desconectado desaparece de la lista del widget.
En caso que el cliente vuelva a conectarse al widget, automáticamente la conversación y herramienta volverán a su estado previo.

8.4.3 Modelo del Widget

La estructura del modelo de datos del widget presentado en esta sección, tendrá al campo `element` representando un solo objeto, el chat.
Como pieza base fundamental del widget, el chat representará cada una de las interacciones entre los usuarios. A continuación ejemplificamos la estructura del mismo:

```
{
 "user1": "pablo.perez",
 "user2": "lucas_93",
 "comentarios": [
      {
        "userName": "pablo.perez",
        "texto": "Lucas, mirá el post que comenté en stack
overflow"
      },
      {
        "userName": "lucas_93",
        "texto":   "lo   vi,   esa   herramienta   podría
servirnos..."
      },
      {
        "userName": "pablo.perez",
        "texto":  "claro,  incluso  varias  personas  me
contestaron con detalles!"
      },
      {
        "userName": "lucas_93",
        "texto":  "si,  estoy  siguiendo  la  discusión  de
SocialEye"
      },
      {
        "userName": "pablo.perez",
        "texto": "está muy bueno!"
      },
   ]
```

}

Como podemos identificar, el modelo de un objeto chat no presenta demasiada complejidad para interpretarlo. Entre los atributos componentes podemos encontrar los userNames de los usuarios involucrados en el chat, seguidos de los distintos comentarios que se hayan enviado entre los mismos. Los comentarios se encuentran representados por una estructura de arreglo, donde cada uno posee dos atributos: el `userName` del usuario que envió el mensaje, y el `texto` correspondiente.

8.4.4 Desarrollo

Como podemos apreciar en base a la sección anterior, el widget presentado cuenta con varias funciones y estados, los cuales terminan por definir una creación compleja por medio de la utilización del framework.
Por lo expresado, en este apartado intentaremos ser los más concisos y específicos posibles en el detalle de la implementación del widget de usuarios, limitandonos a lo que las funciones principales del framework respecta.

Instanciación del widget

La figura 8.48 nos muestra en primer medida la instanciación del nuevo widget de usuarios, junto con la definición de sus propiedades. Luego, el ya mencionado método `loadWidget()` se encarga de preparar la herramienta para su funcionamiento, aquí destacamos un método de la API del framework que se utiliza por primera vez: `getUsersConnectedInWidget()`. Como el propio nombre lo dice, este método se encarga de proveer al desarrollador el conjunto de usuarios que se encuentran utilizando el widget en el dominio web en cuestión. Explorando la API podremos ver otros métodos con funciones similares a éste, en lo que respecta a la recuperación de usuarios de la herramienta.

```
var usuarios = new Widget();
usuarios.tittle = "Usuarios activos en: ";

usuarios.loadWidget = function () {
    var dominio = window.location.hostname;
    enChat = usuarios.getUsersConnectedInWidget();

    var data = {user: usuarios.getUser(),
                enChat : enChat};

    usuarios.createTemplate('usuarios', usuarios.tittle + dominio, 'usuarios.html', data);
    usuarios.onCloseTemplate('usuarios', function(){
          usuarios.close();
    });
    usuarios.setTemplatePosition('usuarios', '10%', '50%');
}
```

Figura 8.48 - Widget de Interacción de Usuarios - Instanciación

Es hora de mostrar lo presentado en la figura 8.40, la lista de usuarios conectados. Para este fin, el desarrollador utiliza el template de la figura 8.49, donde podemos reconocer el uso que se da al conjunto de usuarios recuperado por `getUsersConnectedInWidget()`. De esta forma, se visualizan los botones que representan a los usuarios.
El label apartado con la cadena “Nuevo mensaje”, es parte del módulo de avisos del widget, presentado en la figura 8.41

```
<div id='listaUsuarios' class='actionBox list-group'>
    <% _.each(enChat, function(item) { %>
      <% if (item.userName !== user) { %>
          <button class='listButton usuarioChat' id='<%= item.userName %>'>
            <span class='fa-stack, fa-lg'>
                <i class='fa, fa-user, fa-stack-1x'> </i>
            </span>
            <label id='userLabel' class='userLabel'> <%= item.userName %></label>
            <label id='<%= item.userName %>labelMensaje' class='nuevoMensaje'>Nuevo mensaje</label>
          </button>
      <% } %>
    <% }); %>
</div>
```

Figura 8.49 - Widget de Interacción de Usuarios - Template de listado

Funcionamiento de chats

Mediante el uso del método `onReady()`, el desarrollador define gran parte del comportamiento usual del widget. En este caso, podemos ver por la figura 8.50 las primeras definiciones de eventos para el manejo de los chats, junto a un intervalo para definir la agregación y eliminación dinámica de los usuarios conectados a la lista

```
usuarios.onReady = function(){
    $(".usuarioChat").each(function () {
        this.onclick = chatClick;
    });
    interval = setInterval(function () {
        idsEnChat = [];
        mostrados = $('.usuarioChat');
        enChat = usuarios.getUsersConnectedInWidget();
        $.each(enChat, function (i, item) { //Agrego nuevos usuarios conectados
            idsEnChat.push(item.userName);
            if( (item.userName != usuarios.getUser()) &&
            (usuarios.getWidgetElement("#"+item.userName) == null) ){
                data = {userName : item.userName};
                usuarios.injectInTemplate('usuarios','nuevoUsuario.html', data, "#listaUsuarios");
                $(usuarios.getWidgetElement("#"+item.userName)).on('click', chatClick);
            }
        });
        $.each(mostrados, function (i, item) { //Elimino de la lista los usuarios que ya no están
        conectados
            if($.inArray($(item).attr('id'), idsEnChat ) == -1){
                $(item).remove();
            }
        });
    }, 15000);
```

Figura 8.50 - Widget de Interacción de Usuarios - Comportamiento de Inicio

Un fragmento del método `mostrarChat()`, graficado en la figura 8.51, nos muestra cómo se sería la creación de una conversación con un usuario determinado, o en su defecto, la recuperación de la misma junto a los mensajes asociados: el método `getObject()` recibe en este caso los usuarios protagonistas de una conversación, a fin de recuperar la misma si es que existe. Si se da el último caso, cada uno de los mensajes de la conversación son "inyectados" en el chat por medio del método `InjectInTemplate()`, quien se asociará con el template determinado para insertar cada mensaje en el chat.

```
function mostrarChat(otroUsuario){
    usuarioChatActual = otroUsuario;
    var data = {usuarioChatActual : usuarioChatActual};
    usuarios.createTemplate('chat', "Conversación con " + usuarioChatActual, 'chat.html', data);
    usuarios.setTemplateHeight("chat", "390px");
    if(usuarioChatActual > usuarios.getUser()){
        params['user1'] = usuarioChatActual;
        params['user2'] = usuarios.getUser();
    }else{
        params['user1'] = usuarios.getUser();
        params['user2'] = usuarioChatActual;
    }
    chat = usuarios.getObject(params);
    if(chat == null){
        chat.user1 = params['user1'];
        chat.user2 = params['user2'];
        usuarios.saveObject(chat);
    }else{
        $.each(chat.element.comentarios, function (i, item) {
            data = {lastWriter: lastUserWriting,item: item}
            usuarios.injectInTemplate('chat','mensajeChat.html', data, '#listaComentarios');
            lastUserWriting = item.userName;
        });
    }
    $(usuarios.getWidgetElement("#call"+ usuarioChatActual)).on('click', callClick);
```

Figura 8.51 - Widget de Interacción de Usuarios - Método mostrarChat()

Manejo de conexiones

Si bien no es la finalidad de este trabajo, vamos a ir explicando de forma simplificada el funcionamiento de la conexión entre los usuarios y el intercambio de mensajes con el servidor dedicado para este widget, el cual fue desarrollado utilizando NodeJS [38].

La figura 8.52 presenta la instanciación de un objeto `SocketConnection`, el cual representará la conexión con el servidor y definirá una API con el comportamiento correspondiente para el envío de mensajes y solicitudes de videollamada.

```
conexion = new SocketConnection();
conexion.joinRoom(usuarios.getUser());

if (window.location.protocol == "https:"){
    conexion.startWebRTC();
}
conexion.on('message', function(messageData){
    if(messageData.messenger != usuarios.getUser())
      $(usuarios.getWidgetElement("#"+messageData.messenger + 'labelMensaje')).show();
});
```

Figura 8.52 - Widget de Interacción de Usuarios - Utilizando SocketConnection

El servidor creado maneja el concepto de “rooms”: un room (salón), es donde se sucede el contacto entre los usuarios. En primer medida, al iniciar el widget el usuario se une a un “room general”, donde se encuentran todos los usuarios conectados al widget, y se crea un evento para tratar los avisos de “Nuevo mensaje” (evento “on” con la etiqueta “message”).

Como detalle, podemos ver que para iniciar el módulo basado en WebRTC [39] y utilizado para la comunicación por videollamada, la web actual debe cumplir con el protocolo https.

Al momento de abrir un chat con un usuario en particular, se produce el evento de añadirnos a un "room dedicado", donde los únicos usuarios presentes serán los que participen de esa conversación.
En la figura 8.53, se observa la acción de añadirnos a un room determinado y también la redefinición del evento "on" con la etiqueta message, de modo de insertar el mensaje que se recibe en el chat de conversación.

```
conexion.joinRoom(usuarioChatActual);
conexion.on('message', function(message){
     //Recibo mensaje del usuario con el cual estoy chateando
    if(message.messenger != usuarios.getUser()){
         if(message.messenger == usuarioChatActual){
            item = {};
            item.userName = usuarioChatActual;
            item.texto = message.messageText;
            data = {lastWrite: lastWrite,
                item: item}
            usuarios.injectInTemplate('chat','mensajeChat2.html', data, '#listaComentarios');
        }
        //Recibo mensaje de otro usuario
        else{
            $(usuarios.getWidgetElement("#"+message.messenger + 'labelMensaje')).show();//Muestro
            aviso en la lista de usuarios
        }
    }
});
```

Figura 8.53 - Widget de Interacción de Usuarios - Ingresando en un Room

También debemos definir el comportamiento de enviar un mensaje al usuario en cuestión, con este fin se llama al método "Send" de la conexión, enviando el mensaje escrito en el recuadro.
En la figura 8.54 también veremos la presencia del objeto `Comentario`, el cual se usa para persistir los mensajes de una conversación. Podemos notar que, luego de insertar un comentario al chat, el objeto que lo representa es actualizado en el widget mediante el método `updateObject()`.

```
if(textoComentarioChat.value != ""){
    item = {};
    item.userName = usuarios.getUser();
    item.texto = textoComentarioChat.value;
    data = {lastWrite: (!lastWrite),item: item}
    usuarios.injectInTemplate('chat','mensajeChat2.html', data, '#listaComentarios');
    params = {};
    if(usuarioChatActual > usuarios.getUser()){
        params['user1'] = usuarioChatActual;
        params['user2'] = usuarios.getUser();
    }
    else{
        params['user1'] = usuarios.getUser();
        params['user2'] = usuarioChatActual;
    }
    chat = usuarios.getObject(params);
    comentario = new Comentario(textoComentarioChat.value, usuarios.getUser());
    chat.element.comentarios.push(comentario);
    usuarios.updateObject(chat.element, params);
    conexion.send(usuarioChatActual, 'message', {messenger: usuarios.getUser(), messageText:
}
```

Figura 8.54 - Widget de Interacción de Usuarios - Persistiendo Mensajes

Una vez especificados los detalles de la creación de la conexión entre los usuarios y el envío de mensajes, llegamos a exponer al momento de exponer la sección más "divertida" de interacción del widget: la videollamada.
Como vimos en la sección de prueba del widget, todo comienza con un click sobre el ícono que representa el "videollamar" al usuario con el cual estamos teniendo contacto. Recordemos que este último ícono sólo aparecerá al navegar sobre páginas webs que utilicen HTTPS como protocolo, de modo de establecer una conexión segura. Así, sólo podremos hacer videollamadas bajo ese contexto.
El método `callClick()`, graficado en la figura 8.55, es el que se dispara al momento del click sobre el icono de videollamada. Así tan simple como lo vemos, engloba cada uno de los sucesos que dan inicio al contacto entre los usuarios. En principio, setea la configuración de WebRTC, preparando al objeto conexión para la acción de videollamada: se definen los eventos del socket de conexión, se setea configuración de streaming para la inicialización de la webCam y otros detalles técnicos que explicaremos brevemente junto al objeto SocketConnection.

```
1  function callClick(e){
2      conexion.setWebRTC();
3      conexion.send(usuarioChatActual, 'call', {} );
4      extenderChatBox();
5      appendAndScrollList(getChatMessage("Llamando a " + usuarioChatActual + "..."));
6  }
```

Figura 8.55 - Widget de Interacción de Usuarios - Método callClick()

Acto seguido, de la misma forma que vimos previamente al enviar un mensaje entre usuarios, se llama al método send del objeto conexión, ahora con el tag "call", de modo que el usuario que recibe el mensaje en el room específico, pueda saber que se trata de una solicitud de videollamada.
Mediante los métodos `extenderChatBox()` y `appendAndScrollList()`, se modifica el box del chat de modo que se visualicen ahora los dos videos y que también el widget de una respuesta textual al click sobre el ícono correspondiente.

Procederemos ahora a mostrar breve y simplificadamente la clase `SocketConnection` y su estructura básica, a modo de cierre de la exposición del widget de usuarios, ya que como antes mencionamos no es parte de la finalidad de esta tesina el análisis riguroso del uso de herramientas externas al framework.

```
function SocketConnection(){
  var localStream;
  var localPeerConnection;
  var remotePeerConnection;
  var socket = io.connect('https://127.0.0.1:2013', {secure: true});

  this.send = function(msgRoom, msgType, msgContent){
      var msg = {};
      msg.type = msgType;
      msg.room = msgRoom;
      msg.content = msgContent;
      msg.content.sender = usuarios.getUser();
      socket.emit('message', msg);
  }
}
```

Figura 8.56 - Widget de Interacción de Usuarios - Clase SocketConnection

De forma acotada graficamos por medio de la figura 8.56 los atributos fundamentales de la clase `SocketConnection`, junto al método `send`, pieza clave para el contacto entre los usuarios.
A simple vista tenemos variables que representan el stream local (video compartido al otro usuario), los objetos de conexión entre los pares y el objeto socket, el cual vemos que es el producto de ejecutar el método connect, definido en la librería socket.io, con la configuración de conexión correspondiente.
El método `send`, como pudimos ver previamente en su implementación, recibe como parámetro el "room" al cual va dirigido el mensaje (room general de usuarios o un room específico en el caso de un chat), el tipo de mensaje que se envia (de mensaje, de llamada, de interacción) y el contenido del mismo. Con estos datos, sumados a la identificación del propio usuario que envía el mensaje, el método crea un objeto el cual se emite a través del socket. El mensaje llevará el label "message", el cual es comprendido por el servidor dedicado para este widget, y encargado de administrar los mensajes entre los usuarios.

```
this.startWebRTC  = function(){
    socket.on('call', function (evt){
        if(evt.sender != usuarios.getUser()){
            crearBoxLlamada(evt);
            setTimeout(function(){
                if(callAnswer == 1){ //Aceptó
                    conexion.setWebRTC();
                    conexion.send(evt.sender, 'callAnswer' ,{answer: 'aceptada'});
                    conexion.startPeerConnection(false);
                }
                else{
                    if(callAnswer == 2) //Rechazó
                        conexion.send(evt.sender, 'callAnswer' ,{answer: 'rechazada'});
                    else{ //No contestó
                        conexion.send(evt.sender, 'callAnswer' ,{answer: 'noRta'});
                    }
                }
            }, 10000);
        }
    });
}
```

Figura 8.57 - Widget de Interacción de Usuarios - Iniciando WebRTC

El método `startWebRTC()` (figura 8.57) es el encargado de determinar el comportamiento a la hora de recibir una llamada. En la figura correspondiente podemos ver como se espera la respuesta del usuario y se realiza luego una acción determinada en base a la misma.

```
this.setWebRTC = function(){
    var constraints = {video: true, audio:true};
    navigator.getUserMedia(constraints, successCallback, errorCallback);
    if(!webRTCEnabled){
        socket.on('callAnswer', function(callData){
            var listaComentariosElement = usuarios.getWidgetElement('#listaComentarios');
            if((callData.sender != usuarios.getUser()) && (!llamadaEstablecida)){
                if(callData.answer == 'aceptada'){
                    conexion.startPeerConnection(true);
                    appendAndScrollList(listaComentariosElement, getChatMessage("Conexión de
                    videollamada establecida"));
                }else{
                    if(callData.answer == 'rechazada')
                        appendAndScrollList(listaComentariosElement,
                        getChatMessage(usuarioChatActual + " ha rechazado la llamada"));
                    else
                        appendAndScrollList(listaComentariosElement,
                        getChatMessage(usuarioChatActual + " no ha contestado la llamada"));
                    contraerChatBox();
                }}});
        webRTCEnabled = true;}}
```

Figura 8.58 - Widget de Interacción de Usuarios - Configurando WebRTC

Por último, la figura 8.58 visualiza de forma resumida el accionar del método `setWebRTC()`. Aquí podemos identificar las configuraciones sobre el stream del usuario, así como también el comportamiento referido al momento en el cual se recibe una respuesta a una videollamada que el usuario actual realizó.

8.5 Propuestas de Widgets

La propia utilización de la API para programar nuevos widgets, nos presenta un gran desafío en el cual quizás lo más importante sea utilizar nuestra creatividad, con el fin de generar ideas innovadoras que potencien la navegación web del usuario común. A continuación presentamos brevemente la base de lo que podrían ser futuros desarrollos con SocialEye.

Widget dedicado a la accesibilidad Web: como bien la conocemos, la accesibilidad web tiene como objetivo lograr que las páginas web sean utilizables por el máximo número de personas, independientemente de sus conocimientos o capacidades personales. De esta forma, nos imaginamos que, mediante el uso de SocialEye, podemos lograr un widget que posibilite o facilite el uso de Internet a personas con capacidades limitadas.
Para hacernos una idea, podríamos modificar la estructura de las páginas webs de modo que sean visiblemente más amigables para los usuarios (tamaños de fuente más grandes, colores más nítidos), y que a su vez estos tengan la posibilidad de compartir sus configuraciones y experiencias con otros usuarios con similares limitaciones.
También podríamos permitir a los usuarios añadir contenido semántico a un sitio determinado, donde quizás existan términos técnicos de algún tipo que no fueran explicados. Así, los usuarios con más conocimientos sobre un tema podrían dejar su explicación o comentario de modo de ayudar a los principiantes.

Widget que permita "exportar contenido" entre sitios web: quizás hemos pensado muchas veces (o no por considerarlo imposible) en cuantas operaciones en Internet serían más sencillas y rápidas de realizar si uno o varios sitios web "combinen" su contenido. Expresando mejor la idea, podríamos pensar en casos concretos donde un sitio web presenta de forma embebida alguna funcionalidad o información de otro. Así, podemos recordar los ejemplos que relacionaban a sitios web con temática similar o relacionada:

- IMDb y Youtube: en cada ficha de película o serie de TV del primer sitio, podríamos encontrar el vídeo de un trailer publicado en el segundo, el cual se corresponda al contenido.
- Distintos periódicos online: donde figuren además contenidos relacionados de otros diarios, de forma que se pueda comparar opiniones y así estar mejor informado.
- Sitios de mercadeo Web (Amazon, eBay, MercadoLibre): los cuales se podrían sugerir mutuamente otro destino en Internet para el mismo artículo buscado, a fin de comparar precios, calidad y marcas.

Extensión a "Red social" a widget de Usuarios: entre los trabajos futuros a desarrollar, proponemos una mejora al widget que comprende el contacto entre los usuarios conectados a un sitio en común, de la forma que los mismos puedan intercambiar mensajes en privados y hasta realizar videollamadas.
El perfeccionamiento del cual hablamos trata de proponer una asociación más estrecha entre usuarios que así lo requieran, algo similar a lo que conocemos como "relación de amistad" en la popular red social Facebook. De esta forma, podríamos pensar también en que cada usuario tenga su propio perfil, el cual sería accesible sólo a aquellos a los cuales considera "amigos", donde podría estar volcada información sobre sus gustos, páginas webs que frecuenta más seguido, sus widgets preferidos, etc.

Widget para compartir el foco de navegación en tiempo real: utilizando SocialEye podríamos crear un widget que muestre a uno o más usuarios nuestro sentido de navegación en una web en particular, por ejemplo, donde apuntamos el cursor, en qué botones hacemos click. Si lo pensamos, esto podría ser de gran utilidad en casos en los cuales se quiera brindar una exposición del uso de un sitio web a un grupo de usuarios específicos.

Widget para añadir definiciones a palabras científicas o técnicas que aparezcan en la web: cuando navegamos a través de internet, suele ocurrir que nos encontramos con respuestas a nuestras preguntas que no logramos entender dado el nivel de dificultad del vocabulario utilizado. Podríamos crear un widget que permita a aquel usuario con conocimiento del tema, añadir una definición o explicación de palabras de índole científico, técnico, o que no sea de común ocurrencia.

Widget para agregar contenido y referencias en forma de imágenes y videos: muchas veces sucede que, al momento de leer algún artículo, descripción o cualquier tipo de exposición que nos es desconocida en Internet, puede surgir la necesidad de visualizar determinado objeto, con el fin de clarificar una idea, terminar de entender algún concepto o metodología. Es por esto que se propone crear un framework donde los usuarios puedan cargar imágenes o videos a modo de referencia.

Widget de consultas: así como el sitio web Yahoo! posee un muy popular foro de preguntas y respuestas, podríamos pensar en utilizar SocialEye para crear un widget con este enfoque. Mediante el mismo, los usuarios podrían destacar por algún método, el highlight de un texto por ejemplo, algo que se ha expuesto en el sitio web que no les haya quedado claro, o para lo cual necesiten más información. Así, en forma de colaboración mutua, otros usuarios con los conocimientos necesarios podrían atender a las consultas generadas.

Widget para cambiar el destino de archivos a descargar: actualmente cuando se navega a través de la web, los archivos descargados se almacenan en la ubicación por defecto dentro de la computadora utilizada, configurada en el navegador. Un posible widget podría encargarse de cambiar esta ubicación, por un correo electrónico, u otra ubicación remota, en el caso de que no querramos descargarla localmente y necesitemos enviar los archivos descargados a otra ubicación.

9. *Trabajos relacionados*

En este capítulo se realizará un breve análisis del estado actual en la web del concepto de Web Augmentation, incluyendo algunas herramientas desarrolladas hasta la fecha.

Hasta el momento, se ha avanzado con esta técnica de forma independiente en varios dominios de adaptación, con el fin de mejorar el desarrollo de los artefactos propios de cada uno. Por ejemplo, aumentación web para accesibilidad [3][4], orientada al soporte de tareas de usuario [40] [41], para realizar aumentaciones basadas en contexto desde el navegador Web de un dispositivo móvil [42]. Todos estos desarrollos paralelos, no involucran la capa social en su total conjunto.

Sin embargo, como se mencionó el hecho de que Web Augmentation es una técnica en auge, la cual se encuentra en constante avance año tras año en las comunidades de usuarios, ya existen algunas herramientas orientadas a lo social y colaborativo. Como se podrá apreciar a continuación, algunos de los ejemplos incluso se animan, con distintos enfoques y objetivos, a combinar Web Augmentation con la interacción entre los distintos usuarios, tal y como se propone en esta Tesina. Entre las herramientas involucradas más conocidas podemos encontrar:

- TogetherJS [43]: se trata de una librería Javascript que permite agregar herramientas y capacidades de colaboración a un sitio web. La tecnología se promociona en el hecho de que, referenciando la misma en un sitio web, los usuarios pueden ayudarse mutuamente en su experiencia en el mismo.
  La librería añade una serie de funciones que se ubican por encima del sitio web en cuestión, creando una nueva capa de interacción entre los usuarios.
  TogetherJS ofrece distintas formas de contacto entre los usuarios (chats, audios, visualización de la navegación en tiempo real), las cuales se encuentran limitadas por la capacidad de la herramienta.

- ApacheWave [44]: fue un framework en línea que permitía a sus usuarios comunicarse y colaborar en tiempo real. El proyecto fue anunciado por Google como una aplicación web y una plataforma informática diseñada para unir los servicios de e-mail, mensajería instantánea, wiki, y redes sociales. Se centra en el aspecto colaborativo, apoyado por un analizador ortográfico/gramático, traducción automática entre 40 lenguas, y muchas otras extensiones.
  Finalmente, Google abandonó el desarrollo de Wave, debido a su falta de impacto.

- Diigo: Digest of Internet Information, groups and other stuff es una aplicación basada en la nube, que permite la interacción del usuario con el contenido de la página web visitada. La herramienta incluye marcadores web, bloc de notas, archivo de imágenes y documentos, así como selección de textos destacados. Además, se pueden crear grupos (públicos o privados) para compartir enlaces favoritos. El enfoque principal de la aplicación es que los usuarios puedan expresar su opinión ya sea con notas, y puedan debatir en los grupos en los que pertenece, y a su vez resaltar contenido que le pueda parecer importante o destacado.

- Delicious: es una aplicación de gestión de marcadores sociales en la web. Permite guardar los marcadores que se guardaban en los navegadores, y categorizarlos con un sistema de etiquetado denominado folcsonomías (tags). No sólo puede almacenar enlaces a sitios webs, sino que también permite compartirlos con otros usuarios de delicious y determinar cuántos tienen un determinado enlace guardado en sus marcadores. Tiene una interfaz sencilla que utiliza HTML muy simple y un sistema de URLs legible. Además posee un flexible servicio de sindicación web mediante RSS y una API que permite hacer rápidamente aplicaciones que trabajen con delicious.

- Pocket: es una aplicación y un servicio web que permite al usuario administrar listas de lectura obtenidas desde Internet. El usuario guarda el contenido de una página web a la nube para leerla posteriormente. El artículo guardado es enviado luego a una lista del usuario (la cual se sincroniza en todos los dispositivos) para leerla sin necesidad de internet (offline). Pocket remueve los caracteres innecesarios de los artículos a guardar para permitir al usuario leer de manera más sencilla y que así se ajuste al tipo de pantalla que se esté usando.

- Evernote: es una herramienta cuyo objetivo es la organización de información personal mediante el archivo de notas. Todas las notas, fotos, documentos, archivos de audio y páginas web guardadas en una de las versiones de Evernote se sincronizan automáticamente en las otras plataformas que utilice el usuario. La aplicación tiene también una versión para empresas. En ella se potencia el uso compartido tanto de libros de notas como de documentación corporativa. La hace accesible a todos los empleados según su perfil. En el formato empresarial, pueden tenerse también carpetas privadas para la información o notas que así quieran mantenerse.

- Pearltrees: se refiere a sí mismo como “un lugar para tus intereses”, y permite a los usuarios organizar, explorar y compartir todo aquello que desee, ya sea desde fotos personales, a archivos y notas. Cuenta con una única interfaz gráfica en donde el usuario arrastrar y organizar urls y otros objetos digitales, que a su vez pueden organizarse en colecciones y subcolecciones. La misión de la herramienta es ayudar a los usuarios a “Democratizar la organización del conocimiento”. En el aspecto social, Pearltrees permite a sus usuarios sincronizar la aplicación con sus cuentas de Twitter y Facebook. Esta funcionalidad ayuda a compartir la información más rápidamente, dado que los contactos del usuario en estas redes sociales, pueden volver a compartir a su perfil este contenido, si es que poseen esta opción activada.

El desarrollo del concepto de WebAugmentation asociado a la Red Social, junto al análisis de las herramientas antes mencionadas, terminaron por darle forma a la presente tesina. Así, durante el desarrollo de la misma se intentó hacer hincapié en la motivación de la herramienta implementada y las diferencias que presenta sobre las creaciones asociadas a la aumentación web. SocialEye propone ser más que una herramienta estática de navegador con la cual podemos modificar nuestra experiencia en un sitio con algún enfoque preciso, o contactar otros usuarios por algún medio particular. SocialEye surge como el primer Framework cuyo objetivo es el de brindar un soporte a los usuarios para desarrollar widgets de navegador orientados a los

conceptos de WebAugmentation y SocialWeb. De esta forma, se exhibe una utilidad para desarrollar fácilmente ideas innovadoras con el fin de mejorar la experiencia de los usuarios en los sitios web, presentando un enfoque colaborativo que coloca a los mismos como los generadores de contenido de esta poderosa herramienta.

10. *Conclusiones y trabajos futuros*

Para el momento de finalizar la exposición del desarrollo que esta tesina comprende, nos gustaría realizar una breve conclusión a modo de resumen de cada uno de los aspectos tecnológicos, los entes involucrados y las distintas relaciones entre los mismos, en las cuales este trabajo se fundamenta y enfoca. Para culminar, se presentarán distintas propuestas de extensión de la obra finalizada, ya que consideramos que la misma abre un amplio camino a una rama del desarrollo web muy poco explorada.

En base a lo intensamente investigado, desarrollado y concluido en torno al término Web Augmentation, podemos hoy afirmar que se trata de un fenómeno tan abarcativo y libre a la imaginación como la misma "Realidad aumentada", concepto tan nuevo como renombrado en la actualidad y en el cual nos basamos en varias ocasiones a modo de comparativa con lo que Web Augmentation propone instalar sobre la Web.
Así como la tecnología avanza cada vez más rápido, viendo resultados maravillosos a diario en cortos períodos de tiempo, lo mismo sucede con las expectativas y exigencias de las personas. Aquí nos enfocamos en el específico deseo de los usuarios de tener cada vez más flexibilidad al navegar en Internet, de manipular la información de una forma más completa y agregar contenido pensando específicamente en sus requerimientos. Deteniéndonos en esto último, pudimos concluir que la idea de Web Augmentation nos traslada muchas veces de la difícil tarea de pensar y desarrollar una web en base al usuario, a la de delegar una gran parte de esto a él mismo. Así, "aumentando" su experiencia web, el usuario podrá establecer determinadas preferencias de navegación a su gusto, utilizar determinadas herramientas, modificar distintos aspectos de las páginas web, etc.
Por lo expresado, terminamos viendo a la aumentación web como un nuevo mundo de ideas basadas en los requerimientos de los propios usuarios.

A la par del análisis de Web Augmentation, nos enfocamos en otra rama en auge de la Internet, la Web Social.
En la actualidad podemos ver que le debemos a las famosas "Redes sociales" el simple hecho del poder expresarnos libremente, y con solo un click llegar a cientos, miles y porqué no, millones de personas. Debido a esto, crece en los usuarios la necesidad de estar conectados de forma constante, intercambiando opiniones, gustos e ideales mediante la utilización de alguna red en particular.
Es así que durante el análisis y desarrollo de la presente tesina, concluimos que sería muy útil si pudieramos ofrecer una herramienta y plataforma a los usuarios que usualmente buscan innovar con scripts propios en la web. Brindaríamos a los desarrolladores una forma sencilla y accesible de presentar sus proyectos con un enfoque en particular, definido por la aumentación y colaboración entre los usuarios de Internet.
Así es como decidimos crear un framework que, con un enfoque propio definido por la "Social Web", de soporte al gran número de desarrolladores enfocados a algún tipo de aumentación web (Userscripts, Userstyles), los cuales carecen hoy de una herramienta que proponga estos objetivos. Un enfoque similar es presentado en *"CrowdMock: An Approach for Defining and Evolving Web Augmentation Requirements"* [42]
En tanto, el framework creado es acompañado por la herramienta de navegador que albergará los distintos scripts creados con el mismo. Aquí, se ofrecerá al usuario tipo de

Internet una herramienta destinada a funcionar “por encima” de cualquier sitio web, de la forma de permitir a los usuarios acciones como dar su opinión sobre un tema en particular, o compartir información a otras personas en relación al mismo tópico discutido en ese sitio. Como podemos apreciar, todas estas acciones generalmente se encuentran limitadas por una “red social” en particular; dando un ejemplo, tenemos el caso en que un usuario entra a la web de un diario, en la cual lee una noticia reciente. Aquí, el usuario podría querer expresar ante otras personas lo que la nota le provocó, por lo cual debería utilizar alguna red social de su preferencia (Facebook, Twitter). Nuestro enfoque es: ¿Qué tal si la persona que navega el sitio web podría tener contacto con otros usuarios en su misma situación, sin tener que dejar la página web? Esto implicaría entre otras cosas acotar el conjunto de usuarios que reciben el mensaje a los posiblemente interesados en el tema en concreto (aquellos que se encuentren navegando el sitio). En nuestro caso (los desarrolladores) estaríamos en el gran desafío de ofrecer a los usuarios estos nuevos métodos de interacción y relación sobre cualquier sitio web.

De las reflexiones anteriores nos fundamentamos a la hora de buscar un fenómeno que resulte como “socio” ideal al concepto de Web Augmentation, a fin de darle un enfoque concreto y original a la razón de ser del Framework creado y la herramienta generada mediante su uso.
Así es como llegamos a considerar al resultado obtenido de la relación entre ambos fenómenos, lo que identificamos como “Aumentar la experiencia web de los usuarios mediante el uso de herramientas que supongan la interacción entre los mismos”, como un inexplorado mundo predispuesto a la generación de ideas maravillosas.

A la hora de extender la obra realizada, podemos pensar desde el enfoque de ampliar el funcionamiento del Framework, como el soporte y las prestaciones del sitio web creado. Para el primer caso mencionado, podemos mencionar que la API generada con el Framework presenta solo la base en la cual la herramienta se fundamenta para lograr sus principales objetivos, tales como crear un widget, almacenar objetos asociados a los mismos, darle comportamiento, aspectos visuales, etc. A partir de esto, podemos pensar en enriquecer la interfaz de programación con funcionalidades que hagan aún más flexible la creación de widgets y así aumente la productividad de la herramienta.
En el caso de expandir las funciones del sitio web de SocialEye, podemos pensar en varios aspectos:

- Proveer algún mecanismo de visualización del total de contenidos aumentados por los widgets implementados con el framework.
- Usar métodos de colaboración como like/dislike o puntuaciones, de modo que los usuarios puedan votar los widgets publicados.
- Crear un foro general donde los usuarios de la herramienta puedan solicitar o proponer nuevos widgets que los desarrolladores podrían implementar.

11. *Referencias*

1. Groupware - Patricia Bazán, Alejandro Fernández, Nicolás Del Rio, Lia Molinari, Juan Pablo Perez, Matías Banchoff - Aplicaciones, Servicios y Procesos Distribuidos - Capítulo 7. (2014)
2. Oscar Diaz and Cristobal Arellano. 2015. The augmented web: Rationales, opportunities, and challenges on browser-side transcoding. ACM Trans. Web 9, 2, Article 8 (May 2015).
3. Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina-Medina, N., & Harari, I. (2013). Personalized web accessibility using client-side refactoring. IEEE Internet Computing, 17(4), 58-66.
4. Mangiatordi, A., Sareen, H. Crowdsourcing the inclusive web: providing alternative contents in the cloud.
5. Augmented Browsing (https://en.wikipedia.org/wiki/Augmented_browsing).
6. Bolin, M., Webber, M., Rha, P., Wilson, T., & Miller, R. C. (2005, October). Automation and customization of rendered web pages. In Proceedings of the 18th annual ACM symposium on User interface software and technology (pp. 163-172). ACM.
7. Greasemonkey (http://www.greasespot.net/).
8. Tampermonkey (https://tampermonkey.net/).
9. Javascript (https://www.javascript.com/)
10. Framework (https://www.codeproject.com/Articles/5381/What-Is-A-Framework).
11. Widget (http://whatis.techtarget.com/definition/widget).
12. Bouvin, N. O. (1999, February). Unifying strategies for Web augmentation. In Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots: returning to our diverse roots (pp. 91-100). ACM. ISO 690
13. Mashups (https://es.wikipedia.org/wiki/Mashup_(aplicación_web_híbrida)).
14. Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding mashup development. IEEE Internet computing, 12(5).
15. Wong, J., & Hong, J. I. (2007, April). Making mashups with marmite: towards end-user programming for the web. In Proceedings of the SIGCHI conference on Human factors in computing systems (pp. 1435-1444). ACM.
16. Extensión de navegador (https://www.howtogeek.com/169080/beginner-geek-everything-you-need-to-know-about-browser-extensions/).
17. HTML (https://es.wikipedia.org/wiki/HTML)
18. DOM (https://www.w3schools.com/js/js_htmldom.asp)
19. Fowler, M. (2005, June). Inversion of control
20. API (https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones)
21. CSS (https://www.w3.org/Style/CSS/Overview.en.html)
22. Programación orientada a objetos (https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)
23. UnderscoreJS (http://underscorejs.org/).
24. Jquery(https://jquery.com/)
25. AJAX(https://es.wikipedia.org/wiki/AJAX)
26. Bootstrap(http://getbootstrap.com/)
27. JSON(https://www.w3schools.com/js/js_json_intro.asp)
28. HTML5(https://www.w3schools.com/html/html5_intro.asp)

29. Motor de base de datos https://prezi.com/ry9ckaivktcx/motores-de-base-de-datos/
30. BSD http://www.linfo.org/bsdlicense.html
31. Django (https://www.djangoproject.com/).
32.MVC(http://www.bogotobogo.com/DesignPatterns/mvc_model_view_controller_pattern.php)
33. Mapeo objeto relacional(https://programarfacil.com/blog/que-es-un-orm/)
34. Django token api(https://github.com/jpulgarin/django-tokenapi)
35. Django SSL Server(https://github.com/teddziuba/django-sslserver)
36. Cookie http(https://www.nczonline.net/blog/2009/05/05/http-cookies-explained/)
37. Secure Sockets Layer(https://www.certsuperior.com/QueesunCertificadoSSL.aspx)
38. NodeJS (https://nodejs.org/ - https://blog.xervo.io/build-your-first-http-server-in-nodejs).
39. Proyecto WebRTC (https://webrtc.org/).
40. Bevan, N. (2001). International standards for HCI and usability. International journal of human-computer studies, 55(4), 533-552.
41. Diaz O., De Sosa J., Trujillo S. Activity Fragmentation in the Web: Empowering Users to Support Their Own Webflows
42. Bosetti G., Firmenich S., Gordillo S., Rossi G. An approach for building mobile web applications through web augmentation.
43. TogetherJS (https://togetherjs.com)
44. ApacheWave (https://p2pvalue.eu/awakening-decentralised-real-time-collaboration/)
45.Firmenich, D., Firmenich, S., Rivero, J. M., Antonelli, L., & Rossi, G. (2016). CrowdMock: an approach for defining and evolving web augmentation requirements. Requirements Engineering, 1-29.