ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería Técnica Informática de Sistemas

Red social gastronómica

Realizado por
Manuel Albarrán Guerrero

Dirigido por
José Ramón Portillo Fernández

Departamento
Dpto. de Matemática Aplicada I

Sevilla, (01/09/2012)

Convenios utilizados en esta documentación

Cursiva
Indica neologismos, argot informático y nombres de archivos o extensiones.

`Mono-espaciada`
Se usa para marcar dentro de los párrafos a los elementos de un programa, tales como nombres de variables o funciones, bases de datos, tipos de datos, variables de entorno, declaraciones y palabras clave.

`<Mono-espaciada cursiva>`
Muestra el texto que debe ser remplazado con los valores suministrados por el usuario o por los valores determinado por el contexto.

Helvética
Se usa para destacar nombres de productos, empresas o marcas comerciales.

Resumen

Esta es la documentación de un proyecto final de carrera basado en la creación de una red social para aficionados a la cocina respetando los nuevos estándares semánticos. Todo ello sustentando en una tecnología moderna como MongoDB que permita la escalabilidad de dicho proyecto.

Para el desarrollo de este proyecto además de apoyarnos en librerías existentes, desarrollaremos un *microframework* para PHP que llamaremos Nuts.

A lo largo de esta documentación describiremos los componentes con los que forman el proyecto. Así, por tanto, esta documentación nos instruye sobre diferentes puntos:

- ¿Qué son las redes sociales?
- ¿Qué es la web semántica?
- ¿Qué es MongoDB?
- ¿Qué es Nuts y cómo funciona?
- ¿Cómo funciona nuestra aplicación?
- ¿Cómo instalar nuestra aplicación?

Índice general

Índice de cuadros

Índice de figuras

Índice de código

1 Presentación.

Antes de presentar el proyecto se antoja casi obligatorio presentar los dos vídeos que inspiraron todo el proyecto, pues en ellos se encuentra la esencia de nuestro proyecto.

- Software Libre (spanish audio)[1]
- Google with Recipe View[2]

Ahora sí, ya podemos proceder a la introducción formal de nuestro proyecto.

Nuestro proyecto tiene como objetivo llevar el mundo de la cocina al modelo semántico y social de la web de hoy día. Donde las recetas pueden ser filtradas, clasificadas y procesadas automáticamente por *máquina*[3] como las de Google.

Pero, nuestro proyecto no sólo trata de recetas de cocinas, queremos darle una dimensión más y es un proyecto social, donde creamos una comunidad de cocineros, dónde damos la posibilidad de expresarse y comunicarse entre ellos.

Y cuando hablamos del modelo social de la web, no solo nos referimos a que nuestro proyecto sea en sí una red social, sino que es capaz de interactuar con otras redes sociales, pudiendo compartir nuestras recetas en otras redes sociales o mediante códigos QR nos las podremos llevar a nuestro teléfono o tableta.

[1] **Software Libre (spanish audio)**: http://www.youtube.com/watch?v=FvLJ2JotttM
[2] **Google with Recipe View**: http://www.youtube.com/watch?v=IsUN1dUbbM8
[3] **máquina**: en la terminología de la Web semántica se refiere a cualquier aplicación procesadora de datos.

Por ultimo, nuestro proyecto tratará a las recetas como un ente vivo, que evoluciona y varía a lo largo del tiempo. Basándonos en esta idea Permitimos al usuario crear variantes de una recetas pudiendo aportar su experiencia y nuevos sabores al plato de otro usuario.

2 Definición de objetivos.

2.1– Objetivo Principal.

El objetivo principal del proyecto es el desarrollo de una plataforma social totalmente operativa con un diseño atractivo y usable por un usuario no tecnológico, en la cual se puedan compartir recetas. A su vez, estas recetas tienen que quedar completamente catalogadas y deben presentarse de una forma semántica para poder ser procesadas automáticamente por cualquier máquina.

Otro objetivo de esta plataforma es que debe estar sustentada en tecnologías que permitan escalar en un futuro si fuese necesario. Pues en un mundo cada vez mas conectado y un número de usuarios de servicios sociales creciente este requisito podría no ser opcional en un futuro.

2.2– Objetivos Secundarios.

Para la consecución del objetivo principal vamos a dividirlo en diferentes objetivos secundarios, que nos ayuden a determinar las diferentes fases del proyecto.

- Creación de una plataforma social, donde un usuario pueda registrarse, autenticarse, conectar con otros usuario, comunicarse con ellos a través de la plataforma y publicar contenido.
- Elección e implementación de uno o varios estándares semánticos para la representación de las recetas de cocina.
- Crear las estructuras de datos internas necesarios para hacer viable esta representación semántica.

- Estudiar el modo en el cual estos datos serán introducidos por el usuario en el sistema de información del proyecto, de una manera lo más intuitiva posible dentro de lo que permiten los estándares.
- Crear un buscador de recetas que permita utilizar todos estos datos de manera que el usuario pueda encontrar la receta que busca por múltiples criterios más humanos que el nombre de un plato.

3 Redes sociales.

3.1– ¿Qué son las redes sociales, y qué la diferencian de otro tipo de webs?.

Una red social es una web en la cual los usuarios establecen una relación entre ellos, y generan los contenidos de la propia web; a diferencia de una web clásica donde es el administrador quien publica el contenido y el usuario es completamente pasivo.

En una red social el usuario no solo forma parte en la generación del contenido sino que selecciona que parte de información es la que va a recibir de la web, teniendo así una experiencia única y personalizada.

3.2– Tipos de redes sociales según su estructura.

3.2.1. Simétricas.

Las relaciones entre los usuarios son recíprocas. Un usuario envía la petición de suscripción a su contenido y este debe aceptarla. Al aceptarlo ambos usuarios quedan conectados y ambos quedarán suscrito a las actualizaciones del otro.

3.2.2. Asimétricas.

Las relaciones entre los usuarios no son necesariamente recíprocas. Un usuario A puede estar suscrito a las actualizaciones o noticias de otros usuario B, sin que el usuario B este suscrito al usuario A. Así esta ac-

ción de suscripción no necesita necesariamente de la acción de suscripción contraria, aunque en algunos casos sí puede necesitar de su aprobación.

3.3– Vocabulario básico de las redes sociales.

3.3.1. Perfil/Timeline/Muro.

El perfil, timeline o muro es el sitio que cada usuario personaliza con el contenido que genera. Esta página representa al usuario y nos informa de como se relaciona el usuario con la red social.

3.3.2. Actualización/Noticia.

La actualización o noticia se refiere a cada uno de esos contenidos que cada usuario genera. Este contenido no es solo generado explícitamente por el usuario, sino que también incluye las interacciones con otros usuarios.

3.3.3. Siguiendo/Seguidores/Amigos.

- Siguiendo/Seguidores: Son conceptos relacionado con las redes sociales asimétricas, en las cuales al establecerse una relación entre dos usuarios, el usuario suscrito recibe el nombre seguidor y el usuario al que se suscribe el nombre de seguido.

- Amigos: Es un concepto relacionado con las llamadas redes sociales simétricas. Este es el nombre que se le da a la relación en este caso.

3.3.4. Dashboard/Pagina de inicio/Ultimas noticias/Portada.

Es la página que ve el usuario cuando se autentica en la red. En el se encuentran todas las actualizaciones o noticias de los usuarios a los que se encuentran suscrito.

3.3.5. Mensajes privados.

Es una posibilidad que ofrecen las redes sociales de comunicarse con otros usuarios de la red sin que sea público para el resto de los usuarios.

3.3.6. Notificaciones.

Es un método que usan este tipo de webs para informar al usuario de que existen un tipo de actualización que puede resultar relevante para el usuario. Normalmente se deben a que otro usuario ha interactuado con un contenido generado previamente por el usuario que está siendo notificado.

4 Web semántica.

4.1– ¿Qué es?

Es una aplicación de la web tradicional en la que se realizan un proceso adicional para incorporar un marcado extra. El objetivo de este marcado no es dar estilo, sino significado al contenido.

Este marcado de la información permite la filtración de los datos a las máquinas de una forma sencilla. Estos datos se pueden así luego compartir, procesar, transferir y almacenar de una forma más sencilla.

4.2– ¿Para qué sirve?

Hay millones de recursos disponibles en Internet, esto hace que sea imposible de manejarlos manualmente dada la heterogeneidad de formatos. La Web Semántica ayuda a resolver este problema, permitiendo a los usuarios delegar tareas en software. Gracias a la semántica de la Web, el software es capaz de procesar su contenido, razonar con este, combinarlo y realizar deducciones lógicas para resolver problemas cotidianos automáticamente. [W3C12].

4.3– ¿Cómo funciona?

Gracias a este filtrado de los datos, se puede clasificar la información en una base de conocimiento y ofrecerle al usuario de la aplicación solo lo que necesita. Así en un buscador semántico a diferencia de un buscador tradicional el usuario puede encontrar realmente lo que busca, como podemos ver en el siguiente gráfico.

Figura 4.1: Diferencia entre buscador tradicional y semántico

4.4– Vocabulario.

4.4.1. Metadato.

Metadatos (del griego meta, “después de, más allá de” y latín datum, “lo que se da”, «dato»), literalmente «sobre datos», son datos que describen otros datos. En general, un grupo de metadatos se refiere a un grupo de datos, llamado recurso. El concepto de metadatos es análogo al uso de índices para localizar objetos en vez de datos. Por ejemplo, en una biblioteca se usan fichas que especifican autores, títulos, casas editoriales y lugares para buscar libros. Así, los metadatos ayudan a ubicar datos.3

Metadatos (Meta+datos) es un término que se refiere a datos sobre

los propios datos. Un ejemplo es un folleto que nos informa sobre el lugar y el tipo de un libro. Nos está dando datos sobre otros datos: el libro al que se refiere el folleto. El contenido combinado de los datos y metadatos se conoce generalmente como paquete contenedor.

Para varios campos de la informática, como la recuperación de información o la web semántica, los metadatos en etiquetas son un enfoque importante para construir un puente sobre el intervalo semántico.

4.4.2. W3C.

El World Wide Web Consortium, abreviado W3C, es un consorcio internacional que produce recomendaciones para la World Wide Web.

Está dirigida por Tim Berners-Lee, el creador original de URL (Uniform Resource Locator, Localizador Uniforme de Recursos), HTTP (HyperText Transfer Protocol, Protocolo de Transferencia de Hipertexto) y HTML (Lenguaje de Marcado de Hipertexto) que son las principales tecnologías sobre las que se basa la Web.

4.5– Formatos semánticos.

4.5.1. RDFa.

RDFa son las siglas de *Resource Description Framework in attributes*, que traducido al español es *Marco de Descripción de Recursos en atributos*. RDF es un *framework* para metadatos en la Web desarrollado por W3C.

Los atributos que se utilizan son:

typeof Indica de que tipo es la instancia descrita.

about Una URI que indica el recurso que describen los metadatos y que remite al documento actual por defecto.

rel, rev, href y resource Atributos que establecen una relación o relación inversa con otro recurso.

property Aporta una propiedad para el contenido de un elemento.

content Atributo opcional que se sobrepone al contenido del elemento cuando se usa el atributo property.

datatype Atributo opcional que indica el tipo de datos del contenido.

4.5.2. Microdata.

Es un formato impulsado por Apple, Fundacion Mozilla y Opera Software.

Los atributos que utiliza son:

itemscope Crea el *Item* y indica que los elementos hijos de este contienen información sobre él.

itemtype Una URL válida del vocabulario que describe el *item* y el contexto de propiedades.

itemid Indica el identificador unico de este *item*.

itemprop Indica que el contenido de la etiqueta tiene el valor que especifica `itemprop`. El nombre de las propiedades y el contexto del valor estan descritos por el vocabulario. Las propiedades normalmente consisten en una cadena, pero puede tratarse tambien de una URL usando el href de la propia etiqueta y el elemento `img` puede usar el atributo `src`.

itemref Se usa para indicar que la propiedad no se encuentra en sus elementos hijos.

4.5.3. Microformato (**Microformats**).

Los microformatos son un tipo de marcado semántico definido por sus creadores como: Diseñado para humanos primero y para las maquinas despues. Son un conjunto de formatos de datos simple y abierto construdios tomando en cuenta los standars existentes.

W3C define a los microformatos como:

> *Los microformatos son conjuntos de formatos de datos abiertos y simples, desarrollados sobre estándares ya existentes, ampliamente adoptados, incluyendo HTML, CSS y XML.*

[W3C07]

A diferencia del resto de marcados los microformatos usan propiedades ya existentes en HTML, principalmente `class` y eventualmente `id`, `title`, `rel` o `rev`.

5 Análisis de antecedentes y aportación realizada.

5.1– Antecedentes a nuestro proyecto.

Cuando se inició este proyecto no existía ningún proyecto en español que realizara una presentación semántica de los datos y que además se tratara de una red social. Así que la situación era la siguiente:

Existía una extensión para Wordpress[1] para crear un tipo de post específico de recetas, pero tremendamente complicado de utilizar, con unos formularios excesivamente extensos y detallados. Además esta alternativa era muy limitada, pues estaba integrada dentro del sistema de blog de Wordpress, con lo cual para utilizarla el usuario debía comprar un dominio y contratar un servidor; con el consecuente gasto económico.

Por otro lado existían redes sociales de cocina, pero donde las recetas eran tomadas como simple texto que se publicaba, sin tener en cuenta la componente semántica. Esto hace inviable una verdadera búsqueda por ingredientes o que Google indexe la receta de cocina como tal, ya que la identifica como HTML normal.

5.2– Novedades introducidas.

En nuestro proyecto se encuentra en un punto medio entre ambos polos, no siendo tan complicado como la extensión para Wordpress ni tan simplista como las redes sociales antiguas. Se utiliza un formulario simplificado y con un procesado posterior por parte del servidor. Además

[1] **Wordpress**: http://wordpress.org/extend/plugins/hrecipe/

se les ofrece sugerencias para el etiquetado de la receta, que normaliza sin limitar la clasificación de estas.

6 Requerimientos.

6.1– Conceptos.

6.1.1. HTTP.

Hypertext Transfer Protocol o HTTP (en español protocolo de transferencia de hipertexto) es el protocolo usado en cada transacción de la World Wide Web. HTTP fue desarrollado por el World Wide Web Consortium y la Internet Engineering Task Force, colaboración que culminó en 1999 con la publicación de una serie de RFC, el más importante de ellos es el RFC 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxies) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web o un spider) se lo conoce como "user agent" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

6.1.2. NoSQL.

En informática, NoSQL (a veces llamado "no sólo SQL") es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes, el más destacado que no usan SQL como el principal lenguaje de consultas. Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones JOIN, ni garantizan completamente ACID (atomicidad, coherencia, aislamiento y

durabilidad), y habitualmente escalan bien horizontalmente.

6.1.3. HTML.

HTML, siglas de HyperText Markup Language («lenguaje de marcado de hipertexto»), hace referencia al lenguaje de marcado predominante para la elaboración de páginas web que se utiliza para describir y traducir la estructura y la información en forma de texto, así como para complementar el texto con objetos tales como imágenes. El HTML se escribe en forma de «etiquetas», rodeadas por corchetes angulares (<,>). HTML también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un script (por ejemplo JavaScript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML.

6.1.4. JavaScript.

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, en bases de datos locales al navegador... aunque existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS). Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

JavaScript se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. JavaScript se interpreta en el agente de

usuario, al mismo tiempo que las sentencias van descargándose junto con el código HTML.

6.1.5. Nginx.

Nginx (pronunciado en inglés "engine X") es un servidor web ligero de alto rendimiento.

Es software libre y de código abierto, licenciado bajo la Licencia BSD simplificada. Es multiplataforma, por lo que corre en sistemas tipo Unix (GNU/Linux, BSD, Solaris, Mac OS X, etc.) y Windows.

El sistema es usado por una larga lista de sitios web conocidos, como: WordPress, Hulu, GitHub, SourceForge, TorrentReactor y partes de Facebook.

6.1.6. Apache.

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo. Su nombre se debe a que Behelendorf quería que tuviese la connotación de algo que es firme y enérgico pero no agresivo, y la tribu Apache fue la última en rendirse al que pronto se convertiría en gobierno de EEUU, y en esos momentos la preocupación de su grupo era que llegasen las empresas y "civilizasen" el paisaje que habían creado los primeros ingenieros de internet. Además Apache consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. En inglés, a patchy server (un servidor "parcheado") suena igual que Apache Server.

El servidor Apache se desarrolla dentro del proyecto HTTP Server (httpd) de la Apache Software Foundation.

6.1.7. XAMPP.

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos MySQL, el servidor web

Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, MySQL, PHP, Perl.

El programa está liberado bajo la licencia GNU y actúa como un servidor web libre, fácil de usar y capaz de interpretar páginas dinámicas. Actualmente XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris y MacOS X.

6.1.8. MongoDB.

MongoDB (de la palabra en ingles "humongous" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos tipo JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en ciertas aplicaciones sea mas fácil y rápida.

El desarrollo de MongoDB empezó en octubre de 2007 por la compañía de software 10gen. Ahora MongoDB es una base de datos lista para la producción de uso y con muchas características (feature). Esta base de datos es altamente utilizada en las industrias y MTV Network , Craiglist y Foursquare son algunas de las empresas que utilizan esta base de datos.

El código binario está disponible para los sistemas operativos Windows, Linux, OS X y Solaris.

6.1.9. PHP.

PHP es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor (server-side scripting) pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica usando las bibliotecas Qt o GTK+.

El gran parecido que posee PHP con los lenguajes más comunes de

programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones.

6.1.10. Framework.

La palabra inglesa "framework" define, en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el desarrollo de software, un framework o infraestructura digital, es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, con base a la cual otro proyecto de software puede ser más fácilmente organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y una especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

6.1.11. Plantilla.

Una plantilla es una forma de dispositivo que suele proporcionar una separación entre la forma o estructura y el contenido. Es un medio o un aparato que permite guiar, portar o construir un diseño o esquema predefinido.

Una plantilla agiliza el trabajo de reproducción de muchas copias idénticas o casi idénticas (que no tiene que ser tan elaborado, sofisticado o personal). Si se quiere un trabajo más refinado, más creativo, la plantilla no es sino un punto de partida, un ejemplo, una idea aproximada de lo que se quiere hacer.

Las plantillas, como norma general, pueden ser utilizadas por personas o por sistemas automatizados. Se utilizan plantillas en todos los terrenos de la industria y la tecnología. Una plantilla puede servir como mues-

tra base de una diversidad sobre la que comparten elementos comunes (patrón) y que en sí es lo que constituye la plantilla.

En relación con la edición o composición de textos o imágenes, se compone de cajas y líneas, con unos tamaños y márgenes, para facilitar la escritura de artículos o cartas, con títulos, fotos y diagramas.

Con relación con los sistema computacionales, por ejemplo paquetes de programas basados en la web, utilizan en la actualidad un sistema de plantillas para separar la lógica del programa del formato visualizado. Típicamente, estas plantillas incluirán variable (frecuentemente denotadas como VARIABLE), y posiblemente unos pocos operadores lógicos para permitir una mejor adaptabilidad de la plantilla.

6.1.12. Twig.

Twig es un moderno motor de plantillas para PHP desarrollado por *SesioLabs*[1].

Rápido: Twig compila las plantillas a código PHP sencillo optimizado. La sobrecarga en comparación con el ordinario de código PHP se ha reducido a la mínima expresión.

Seguro: Twig tiene un modo *sandbox*[2] para evaluar el de código de la plantilla no es de confianza. Esto permite a Twig ser usado como lengauje de plantilla para las aplicaciones donde los usuarios pueden modificar el diseño de la plantilla.

Flexible: Twig trabaja sobre un *lexer*[3] y *parser*[4]. Esto permite al desarrollador definir sus propias etiquetas y filtros personalizado.[**?**]

6.1.13. JQuery.

jQuery es una biblioteca de JavaScript, creada inicialmente por John Resig, que permite simplificar la manera de interactuar con los documen-

[1] **SesioLabs**: Grupo de desarrollado famoso por ser los creadores de Symfony.
[2] **sandbox**: Palabra que del inglés significa caja de arena y se refiere a un sistema informático de aislamiento mediante el cual se pueden ejecutar código con seguridad y de manera separada.
[3] **lexer**: Analizador léxico.
[4] **parser**: Analizador sintáctico.

tos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web. Fue presentada el 14 de enero de 2006 en el BarCamp NYC.

jQuery es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres y privativos. jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código, es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

6.1.14. Bootstrap.

Bootstrap es un framework que simplifica el proceso de creación de diseños web combinando CSS y JavaScript. Ha sido desarrollado por Twitter que recientemente liberó su versión 2.0. La mayor ventaja es que podemos crear interfaces apoyándonos en un framework potente con numerosos componentes webs que nos ahorrarán mucho esfuerzo y tiempo.

Bootstrap ofrece una serie de plantillas CSS y ficheros Javascript que nos permiten integrar el framework de forma sencilla y potente en nuestros proyectos webs.

Funciona con todos los navegadores, incluido Internet Explorer usando HTML Shim para que reconozca los tags HTML5. [Rod12]

6.1.15. WYSIWYG.

WYSIWYG es el acrónimo de What You See Is What You Get (en inglés, "lo que ves es lo que obtienes"). Se aplica a los procesadores de texto y otros editores de texto con formato (como los editores de HTML) que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso.

Se utiliza en contraposición a otros procesadores de texto, hoy en día poco frecuentes, en los que se escribía sobre una vista que no mostraba el formato del texto, hasta la impresión del documento.

En el caso de editores de HTML este concepto se aplica a los que permiten escribir la página sobre una vista preliminar similar a la de un

procesador de textos, ocupándose en este caso el programa de generar el código fuente en HTML.

6.1.16. nicEdit.

NicEdit es un editor de texto WYSIWYG para HTML de código abierto que funciona completamente en JavaScript y se distribuye gratuitamente bajo la Licencia MIT.

Al ser basado en JavaScript nicEdit es independiente de la plataforma y se ejecuta en el navegador de internet. Tiene la habilidad de convertir un campo del tipo textarea u otros elementos de html en instancias del editor.

6.2– Requisistos.

El requisito principal para la ejecucíon de nuestro proyecto será contar con un *hosting*[5], que albergue y le preste el servicio necesario. Éste deberá contar con Apache o Nginx instalado para correr nuestro código php. Además deberá contar también con MongoDB.

Como veremos más adelantes en el capítulo 10.1 podemos usar AppFog para suplir este requisito. O también podemos instalarnos nosotros mismos un servidor para pruebas, si nos decidimos por esta opción, entonces podrémos usar XAMPP[6] y *MongoDB 1.8*[7] o superior.

El resto de librerías no necesitaremos instalarlas puesto que hemos decidido incluirlas dentro de nuestro proyecto, para evitar que futuras actualizaciones provoquen incompatibilidades con nuestro proyecto.

Igualmente, haremos un breve repaso de las bibiliotecas externas, dado que pudieran surgir actualizaciones de seguridad que serían necesarias llevar a cabo si se desea uilizar el proyecto en un medio real y no solo en un medio demostrativo.

Twig http://twig.sensiolabs.org/

[5]**hosting**: Servidor dedicado para aplicaciones web, usualmente contratado a otra empresa.

[6]**XAMPP**: http://www.apachefriends.org/es/xampp.html

[7]**MongoDB 1.8**: http://www.mongodb.org/downloads

PHP QR Code http://phpqrcode.sourceforge.net/

Jquery http://jquery.com/

jQuery Tags Input Plugin http://xoxco.com/projects/code/tagsinput/

NicEdit http://nicedit.com/

Infinite scroll http://www.infinite-scroll.com/

Bootstrap http://twitter.github.com/bootstrap/

7 Comparativa con otras alternativas.

En este capítulo nos centraremos en los componentes de nuestro proyectos y en las alternativas posibles a estos.

7.1– Base de datos. MongoDB vs. RDBMS.

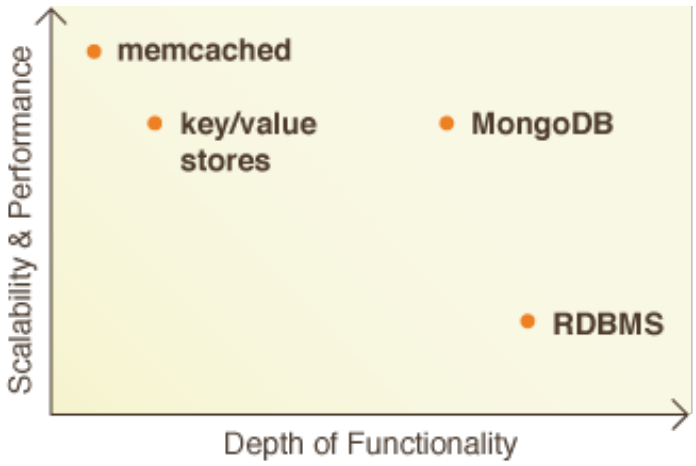

Figura 7.1: MongoDB Escalabilidad y Rendimiento / Amplitud de las funcionalidades.

7.1.1. Flexibilidad.

MongoDB almacena los datos en los documentos JSON (que son *serializados* a BSON). JSON nos proporciona un modelo de datos rico que se asemeja a los tipos nativos de los lenguaje de programación habituales, y su modelo sin esquemas hace que sea mucho más fácil de evolucionar nuestro modelo de datos que con un sistema con esquemas forzadas como un RDBMS.[Mon11]

7.1.2. Potencia.

MongoDB ofrece una gran cantidad de las características de un RDBMS tradicional, tales como los índices secundarios, consultas dinámicas, ordenación, actualizaciones, *upserts* (actualizar si el documento existe, inserte si no existe), y una agregación fácil. Esto le da la amplitud de la funcionalidad que usted está acostumbrado a usar en un RDBMS, con la flexibilidad y capacidad de ampliación que el modelo no-relacional permite.[Mon11]

7.1.3. Velocidad / Escalabilidad.

Al mantener juntos los datos relacionados en los documentos, las consultas pueden ser mucho más rápidas que en una base de datos relacional en donde los datos relacionados se divide en varias tablas y luego tiene que ser ensamblados más adelante. MongoDB también hace que sea fácil de escalar su base de datos. *Autosharding* permite ampliación del *clúster*[1] de forma lineal mediante la adición de más máquinas. Es posible aumentar la capacidad sin ningún tiempo de inactividad, lo cual es muy importante en la web cuando la carga puede aumentar de repente y derribar el sitio web y el mantenimiento prolongado puede costar nuestro negocio y perder de grandes cantidades de ingresos.[Mon11]

7.1.4. La facilidad de uso.

MongoDB trabaja duro para ser muy fácil de instalar, configurar, mantener y usar. Con este fin, MongoDB ofrece pocas opciones de confi-

[1] **clúster**: Termino que se aplica a los conjuntos de computadoras construidos mediante la utilización de hardwares comunes y que se comportan como si fuesen una única computadora.

guración, y en su lugar trata de hacer de forma automática "lo correcto" siempre que sea posible. Esto significa que MongoDB funciona nada más "sacarlo de la caja", y se puede ir directamente al desarrollo de nuestra aplicación, en lugar de gastar un montón de tiempo en afinar las configuraciones de base de datos. [Mon11]

7.1.5. De SQL a Mongo

Ejecutable en MySql	**Ejecutable en Oracle**	**Ejecutable en Mongo**
mysqld	oracle	mongod
mysql	sqlplus	mongo

Cuadro 7.1: Ejecutables

Termino en MySql	**Termino en Mongo**
base de datos	base de datos
tabla	colección
indice	indice
fila	documento BSON
columna	campo BSON
join	embebido y enlazado
clave primaria	campo _id
group by	agregación

Cuadro 7.2: Terminos

Sentencias en SQL	Sentencias en Mongo
CREATE TABLE USERS (a Number, b Number)	db.createCollection("mycoll") (No es necesario)
ALTER TABLE users ADD ...	(No es necesario)
INSERT INTO USERS VALUES(3, 5)	db.users.insert({a:3, b:5})
SELECT a,b FROM users	db.users.find({}, {a:1, b:1})
SELECT * FROM users	db.users.find()
SELECT * FROM users WHERE age=33	db.users.find({age:33})
SELECT a, b FROM users WHERE age=33	db.users.find({age:33}, {a:1, b:1})
SELECT * FROM users WHERE age=33 ORDER BY name	db.users.find({age:33}).sort({name:1})
SELECT * FROM users WHERE age extgreater33	db.users.find({age:{$gt:33}})
SELECT * FROM users WHERE age!=33	db.users.find({age:{$ne:33}})
SELECT * FROM users WHERE name LIKE "%Joe%"	db.users.find({name:/Joe/})
SELECT * FROM users WHERE name LIKE "Joe%"	db.users.find({name:/^Joe/})
SELECT * FROM users WHERE age extgreater33 AND age extless=40	db.users.find({"age":{$gt:33, $lte:40}})

Cuadro 7.3: Sentencias. Parte 1

`SELECT * FROM users ORDER BY name DESC`	`db.users.find().sort({name:-1})`
`SELECT * FROM users WHERE a=1 and b="q"`	`db.users.find({a:1, b:"q"})`
`SELECT * FROM users LIMIT 10 SKIP 20`	`db.users.find().limit(10).skip(20)`
`SELECT * FROM users WHERE a=1 or b=2`	`db.users.find({$or:[{a:1}, {b:2}]})`
`SELECT * FROM users LIMIT 1`	`db.users.findOne()`
`SELECT order_id FROM orders o, order_line_items li WHERE li.order_id=o.order_id AND li.sku=12345`	`db.orders.find( {"items.sku":12345}, {_id:1})`
`SELECT customer.name FROM customers, orders WHERE orders.id="q179" AND orders.custid=customer.id`	`var o = db.orders.findOne( {_id:"q179"}); var name = db.customers.findOne( {_id:o.custid})`
`CREATE TABLE USERS (a Number, b Number)`	`db.createCollection( "mycoll")` (No es necesario)
`SELECT DISTINCT last_name FROM users`	`db.users.distinct( "last_name")`
`SELECT COUNT(*y) FROM users`	`db.users.count()`
`SELECT COUNT(*y) FROM users where AGE extgreater 30`	`db.users.find({age: {"$gt": 30}}).count()`
`SELECT COUNT(AGE) from users`	`db.users.find({age: {"$exists": true}}).count()`
`CREATE INDEX myindexname ON users(name)`	`db.users.ensureIndex( {name:1})`
`CREATE INDEX myindexname ON users(name, ts DESC)`	`db.users.ensureIndex( {name:1, ts:-1})`
`EXPLAIN SELECT * FROM users WHERE z=3`	`db.users.find( {z:3}).explain()`
`UPDATE users SET a=1 WHERE b="q"`	`db.users.update({b:"q"}, {$set:{a:1}}, false, true)`
`UPDATE users SET a=a+2 WHERE b="q"`	`db.users.update({b:"q"}, {$inc:{a:2}}, false, true)`
`DELETE FROM users WHERE z="abc"`	`db.users.remove({z:"abc"});`

Cuadro 7.4: Sentencias. Parte 2

[Mon12]

7.2– Lenguaje de programación del servidor. PHP vs Ruby vs Python

Para la comparación entre estos lenguajes de programación nos vamos a apoyar en unos gráficos desarrollados por udemy[2].

[2] **udemy**: http://www.udemy.com/

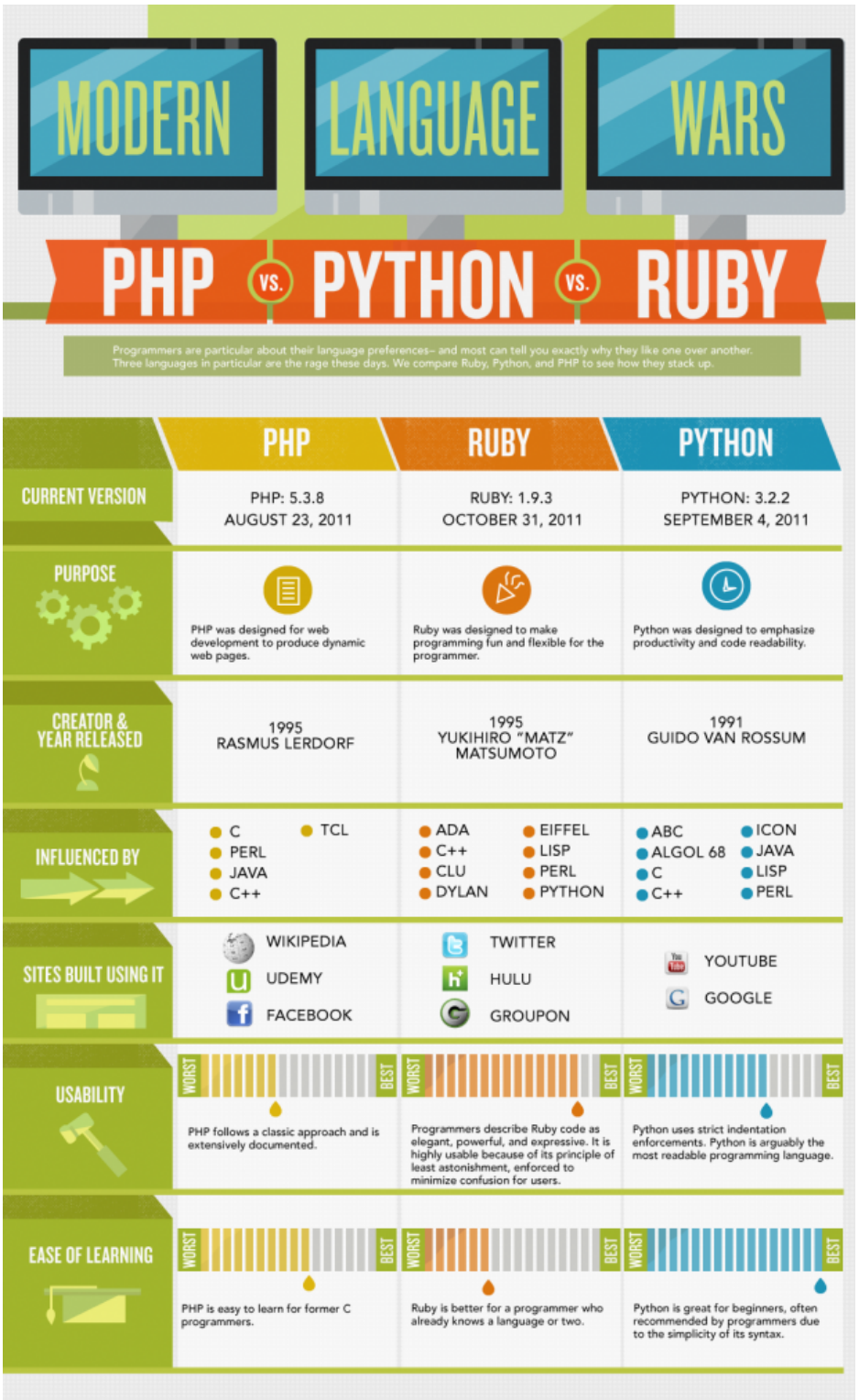

Figura 7.2: Comparación general entre Php Ruby y Python

A tenor del primer gráfico podemos ver que PHP es el único que ha sido diseñado especificamente como lenguaje web.

Ruby se presenta como el lenguaje más usable, pero sin embargo su curva de aprendizaje es bastante lenta y podría causar bastantes quebraderos de cabeza.

Python por su parte tiene la curva de aprendizaje más rápida, y una usabilidad media. Así python se presenta como un buen candidato.

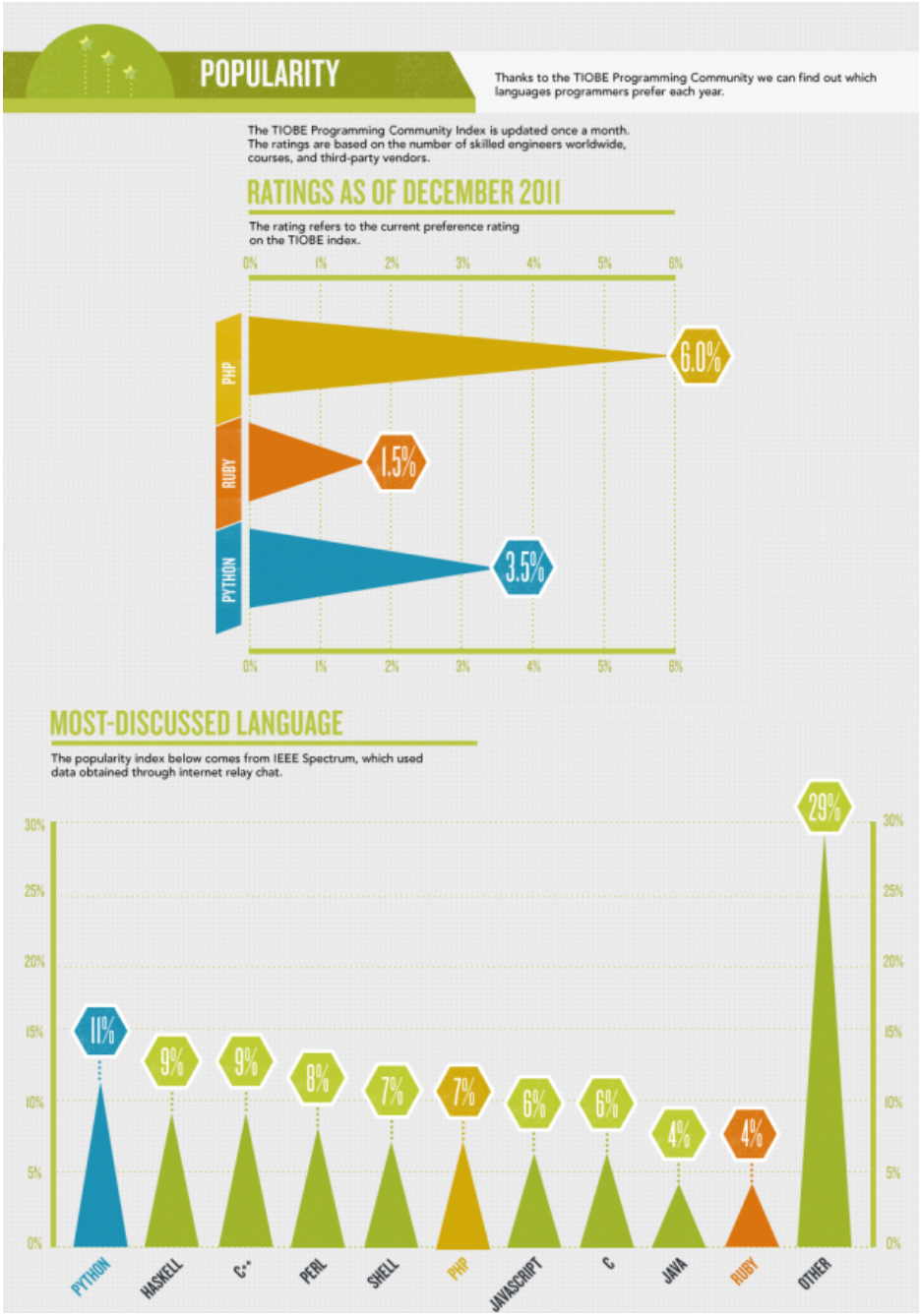

Figura 7.3: Comparación de popularidad entre Php Ruby y Python

En esta segunda imagen en el primer gráfico vemos como los usua-

rios prefiere Php. Mientras en el segundo gráfico vemos como Php a pesar de solo tener un ámbito de desarrollo tiene un buen número de discusiones activas y esta muy cerca de Python que es un lenguaje de propósito general.

Figura 7.4: Comparación de mercado entre Php Ruby y Python

En esta imagen podemos ver como Php es uno de los lenguajes más demandados y más usados en el mundo empresarial. Esto juega no solo a su favor en cuanto a poder aprovechar los conocmientos durante el desarrollo para el futuro, sino que nos asegura una amplia fuente de recursos en internet tanto de documentación, como fragmentos de códigos y bibiliotecas.

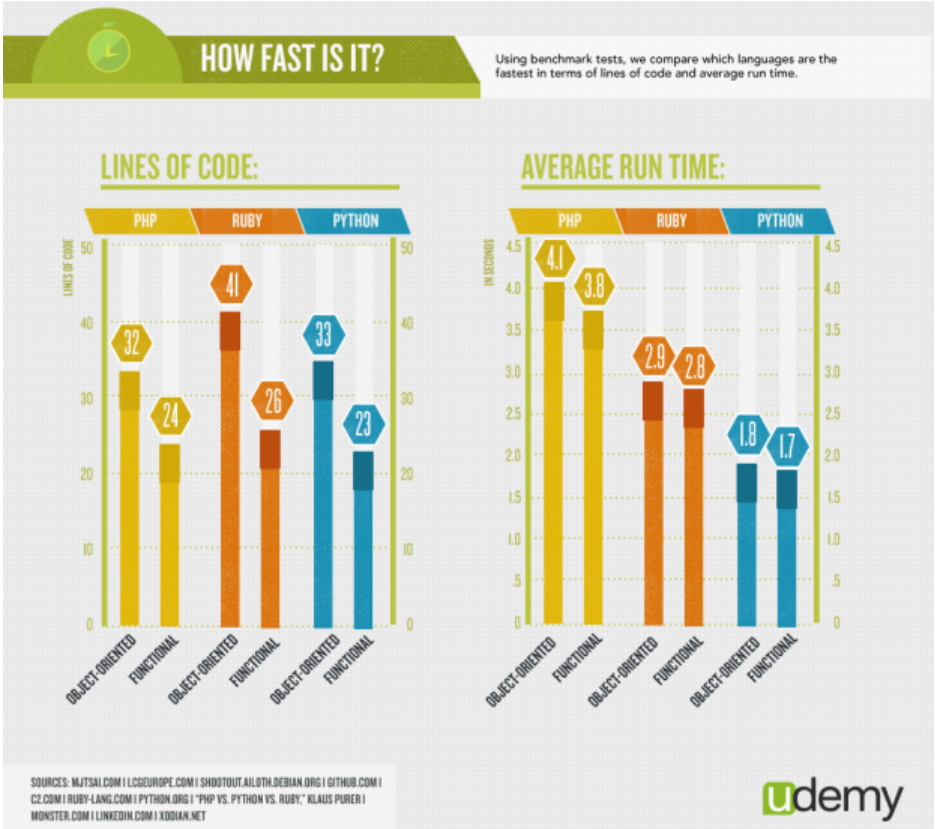

Figura 7.5: Comparación de rapidez entre Php Ruby y Python

En esta imagen vemos como php a pesar de no ser el más rapido en ejecución necesita menos líneas de código en el proyecto, lo que reduce el tiempo de desarrollo y la propensión a errores.

Después de estos datos hemos decidido usar Php, a pesar de que Python parece ser una buena alternativa el hecho de que no exista una documentación tan abundante, y la necesidad de un framework con su peculiar forma de trabajo hacer que lo descartemos. Y en cuanto a Ruby su curva

de aprendizaje tan lenta nos desalienta por completo.

7.3– HTML: XHTML1.0 vs HTML5

Actualmente lo que podemos llamar dos tendencias. Por un lado tenemos XHTML, que es una especificación de HTML llevada al campo de las especificaciones XML, y por otro lado tenemos HTML5, que es la ultima version del lenguaje HTML que lleva asociado no solo el lenguaje de marcado sino una filosofia detrás de modernizar la web y lleva asociado un cambio de todo el modelo incluyendo a javascript y css.

Las ventajas de XHTML es estricta en su especificación que permite una validación y un análisis rápido del código. Pero esto también lo hace que sea más fácil que un código escrito en XHTML resulte erroneo.

Por otro lado HTML5 tiene una especificación más flexible y está pensado específicamente para la web. Gracias a que HTML5 incorpora parte de la lógica de la web en su especificiación permite una reducción en el código y una velocidad de transmisión mayor. Por ejemplo, como una de sus ventajas, hay etiquetas que no necesitan ser cerradas.

Además HTML5 permite al navegador aplicar optimizaciones que no puede aplicar con XHML.

> *It's recommended to use HTML, as text/html. Do not use XHTML. XHTML, as application/xhtml+xml, lacks both browser and infrastructure support and offers less room for optimization than HTML.*
>
> [Goo12]

7.4– Marcado Semántico: Microformatos vs. Microdata

7.4.1. RDFa.

No tiene ningún estandar para marcado de recetas destacable. Sólo existe uno basado en hrecipe, que fue desarrollado por Microformats como veremos más abajo en el punto 7.4.3.

7.4.2. Microdata.

Las especificaciones para una receta en microdata se definen con `itemtype=http://schema.org/Recipe`.

Que a su vez extiende de `Thing`, `CreativeWork`.

Las propiedades que se utilizarán más frecuentemente en una receta seran:

name Nombre de la receta.

author Autor de la receta.

datePublished Fecha de publicación.

image Imagen de la receta.

description Descripcion de la receta.

prepTime Tiempo de preparación.

cookTime Tiempo de cocinado.

totalTime Tiempo total.

cookingMethod Metodo de cocinado.

recipeYield Comensales.

nutrition Información nutricional.

calories Calorias.

fatContent Grasas.

ingredients Ingredientes.

recipeInstructions Instrucciones de la receta.

recipeCategory Categoria de la receta.

recipecuisine Lugar de origen de la receta.

Veamos un ejemplo

Código 7.1: Receta formateada con Microdata

```
<div itemscope itemtype="http://schema.org/Recipe">
  <span itemprop="name">Mom's World Famous Banana Bread</span>
  By <span itemprop="author">John Smith</span>,
  <meta itemprop="datePublished" content="2009-05-08">May 8, 2009
  <img itemprop="image" src="bananabread.jpg" />

  <span itemprop="description">This classic banana bread recipe comes
  from my mom -- the walnuts add a nice texture and flavor to the banana
  bread.</span>

  Prep Time: <meta itemprop="prepTime" content="PT15M">15 minutes
  Cook time: <meta itemprop="cookTime" content="PT1H">1 hour
  Yield: <span itemprop="recipeYield">1 loaf</span>

  <div itemprop="nutrition"
    itemscope itemtype="http://schema.org/NutritionInformation">
    Nutrition facts:
    <span itemprop="calories">240 calories</span>,
    <span itemprop="fatContent">9 grams fat</span>
  </div>

  Ingredients:
  - <span itemprop="ingredients">3 or 4 ripe bananas, smashed</span>
  - <span itemprop="ingredients">1 egg</span>
  - <span itemprop="ingredients">3/4 cup of sugar</span>
  ...

  Instructions:
  <span itemprop="recipeInstructions">
  Preheat the oven to 350 degrees. Mix in the ingredients in a bowl. Add
  the flour last. Pour the mixture into a loaf pan and bake for one hour.
  </span>

  140 comments:
  <meta itemprop="interactionCount" content="UserComments:140" />
  From Janel, May 5 -- thank you, great recipe!
```

```
37    ...
38  </div>
```

7.4.3. Microformat.

Tiene un estandar llamado hrecipe y es el que utilizaremos en nuestro proyecto. Tiene el siguiente esquema:

hrecipe :

fn required. text. the name of the recipe.

ingredient required. 1 or more. text with optional valid (x)HTML markup.

value and type optional. [*experimental*]

yield optional. text.

instructions optional. text with optional valid (x)HTML markup.

duration optional. 1 or more. text.

photo optional. 1 or more. using any element containing a URL, such as IMG. [*experimental*]

summary optional. text. [*experimental*]

author optional. 1 or more. [*experimental*]

published optional. [*experimental*]

nutrition optional. 1 or more. [*experimental*]

value and type . optional. [*experimental*]

tag optional. 1 or more. [*experimental*]

Veamos un ejemplo:

Código 7.2: Receta formateada con Microformat

```
1  <div class="hrecipe">
2      <h1 class="fn">Pommes Frites</h1>
3      <p class="summary">
4          Pommes frites originate in outer space. They
               are served hot.<br />
5          This recipe is only an example. Don't try this
               at home!
6      </p>
7      <p>
```

```
8           Contributed by <span class="author">CJ Tom</
               span> and the
9           <span class="author vcard"><a class="url fn"
               href="http://example.com">Cooky Gang</a></
               span>.
10      </p>
11       <p>Published <span class="published"><span class="
            value-title" title="2008-10-14T10:05:37-01:00">
             </span>14. Oct 2008</span></p>
12      <img src="/img/pommes.png" class="photo" width
           ="100" height="100" alt="Pommes Frites"/>
13      <h2>Ingredients</h2>
14      <ul>
15          <li class="ingredient">
16              <span class="value">500</span>
17              <span class="type">gramme</span> potatoes,
                   hard cooking.
18          </li>
19          <li class="ingredient">
20              <span class="value">1</span> <span class="
                   type">spoonful</span> of salt
21          </li>
22          <li>
23              You may want to provide some
24              <span class="ingredient">Ketchup and
                   Mayonnaise</span>
25              as well.
26          </li>
27      </ul>
28      <h2>Instructions</h2>
29      <ul class="instructions">
30          <li>First wash the potatoes.</li>
31          <li>Then slice and dice them and put them in
               boiling fat.</li>
32          <li>After a few minutes take them out again.</
               li>
33      </ul>
34      <h2>Further details</h2>
35      <p>Enough for <span class="yield">12 children</span
           >.</p>
36      <p>Preparation time is approximately
37          <span class="duration"><span class="value-title
               " title="PT1H30M"> </span>90 min</span>
38      </p>
39      <p>Add <span  class="duration"><span class="value-
           title" title="PT30M"></span>half an hour</span>
           to prepare your homemade Ketchup.</p>
40      <p>This recipe is <a href="http://www.example.com/
           tags/difficulty/easy" rel="tag">easy</a> and <a
           href="http://www.example.com/tags/tastyness/
```

```
        delicious" rel="tag">delicious</a>.</p>
    <p>
        <span class="nutrition">
        Pommes Frites have more than
        <span class="value">1000</span>
        <span class="type">Joule</span>
        Energy</span>,
        while Ketchup and Mayonnaise have
        <span class="nutrition">0 vitamins</span>.
    </p>
</div>
```

7.4.4. Alternativa elegida

Vamos a usar HTML5, con lo cual nos decantaremos por Microformats ya que es un estandar que se integra perfectamente con esta tecnología. Además, microformats como explican sus creadores esta hecho también para humanos, eso nos ofrece un punto más a su favor ya que escribiremos nuestras plantillas y tendremos que escribir la estructura semántica a mano.

8 Análisis de requisitos, diseño e implementación.

8.1– Análisis de requisitos.

Caso de uso	Registro.	**Identificador**	C_1
Precondición	El usuario accede a la aplicación e introduce sus credenciales.		
Postcondición	En la base de datos se crea un usuario nuevo.		
Propósito	El usuario pueda identificarse en un futuro.		
Resumen	Para poder autenticarse en nuestra aplicación el usuario debe contar con un usuario registrado en nuestro sistema.		

Curso normal	**Curso alternativo**
1. Actor accede a la aplicación.	
2. Actor accede al formulario de registro.	
3. El sistema presenta formulario para registrarse.	
4. Actor introduce sus datos.	
5. El sistema verifica datos y permite o no el registro.	
6. El sistema verifica que no existe otro similar y lo registra.	6.1. Falla la verificación y vuelve a mostrar el formulario para la introducción de datos.
7. Fin de caso de uso.	

Cuadro 8.1: Caso de uso: Registro.

Caso de uso	Autenticación.	**Identificador**	C_2
Precondición	El usuario accede a la aplicación e introduce sus credenciales.		
Postcondición	El usuario se conecta la aplicación.		
Propósito	El usuario cuente con un identificador de sesión vinculado a las sesiones anteriores.		
Resumen	Para poder acceder a parte de los servicios que ofrece nuestra aplicación, el usuario debe previamente conectarse y autenticarse como un usuario de la aplicación.		

Curso normal	**Curso alternativo**
1. Actor quiere acceder a la aplicación.	
2. El sistema presenta formulario para autenticarse.	
3. Actor introduce sus datos.	
4. El sistema verifica datos y permite o no la autenticación.	
5. El sistema verifica datos lo autentica.	5.1. Falla la verificación y vuelve a mostrar el formulario para la introducción de datos.
6. Fin de caso de uso.	

Cuadro 8.2: Caso de uso: Autenticación.

Caso de uso	Cambiar imagen de usuario.	**Identificador**	C_3
Precondición	El usuario se encuentra autenticado en el sistema.		
Postcondición	El usuario cuenta con una nueva imagen identificativa.		
Propósito	Cambiar la imagen que identifica al usuario.		
Resumen	El usuario puede cambiar la imagen de su avatar/-perfil que le identifica ante los demas usuarios.		

Curso normal	**Curso alternativo**
1. Actor accede a su perfil.	
2. Actor selecciona la opcion `Cambiar foto de perfil`.	
3. Actor selecciona la imagen que desea en su ordenador.	
4. Automáticamente la imagen del usuario se cambia a la nueva imagen.	
5. Fin de caso de uso.	

Cuadro 8.3: Caso de uso: Cambiar imagen de usuario.

Caso de uso	Eliminar Usuario.	**Identificador**	C_4
Precondición	El usuario se encuentra autenticado en el sistema con una cuenta de usuario válida.		
Postcondición	El usuario con el que estaba autenticado ya no existe, y el usuario ya no se encuentra autenticado en el sistema.		
Propósito	El usuario pueda abandonar el servicio definitivamente.		
Resumen	Los datos de un usuario son eliminados del sistema. El usuario no se va a poder autenticar mas con el mismo nombre a no ser que se registre de nuevo.		

Curso normal	**Curso alternativo**
1. Actor accede al formulario de ajustes	
2. Actor selecciona `Eliminar Cuenta`	
3. Fin de caso de uso.	

Cuadro 8.4: Caso de uso: Eliminar Usuario.

Caso de uso	Seguir.	**Identificador**	C_5
Precondición	El usuario autenticado no sigue al usuario objetivo.		
Postcondición	El usuario autenticado sigue al usuario objetivo.		
Propósito	Añadir un nueva relación entre el usuario autenticado y el objetivo.		
Resumen	Un usuario puede seguir a otros usuarios para informarse de sus actualizaciones.		

Curso normal	**Curso alternativo**
1. Actor accede al perfil del usuario al que desea seguir	1.1. Actor accede a una receta del usuario al que desea seguir
2. Actor selecciona la opción `Seguir`	
3. Fin de caso de uso.	

Cuadro 8.5: Caso de uso: Seguir.

Caso de uso	Dejar de seguir.	**Identificador**	$C6$
Precondición	El usuario autenticado sigue al usuario objetivo.		
Postcondición	El usuario autenticado no sigue al usuario objetivo.		
Propósito	Eliminar la relación entre el usuario autenticado y el objetivo.		
Resumen	El usuario puede dejar de seguir a otros usuarios cuando ya no le interese.		

Curso normal	**Curso alternativo**
1. Actor accede al perfil del usuario al que desea seguir	
2. Actor selecciona la opción `Dejar de Seguir`	
3. Fin de caso de uso.	

Cuadro 8.6: Caso de uso: Dejar de seguir.

Caso de uso	Enviar mensaje.	**Identificador**	c_7
Precondición	El usuario se encuentra autenticado.		
Postcondición	Se crea un nuevo mensaje entre el usuario autenticado y el destinatario.		
Postcondición	El mensaje queda marcado como no leído para el usuario receptor.		
Postcondición	El mensaje queda marcado como leído para el usuario remitente.		
Propósito	Crear nuevos mensajes entre dos usuarios.		
Resumen	El usuario puede enviar un mensaje privado a otro usuario del sistema sin que este pueda ser visible por los demás.		

Curso normal	**Curso alternativo**
1. Actor accede al perfil de usuario destinatario	1.1. El actor accede a su panel de mensajes
2. Actor selecciona `Enviale un mensaje`	2.2. El actor selecciona un hilo de mensajes
3. Actor introduce el texto del mensaje	
4. Actor envía el formulario	
5. Fin de caso de uso.	

Cuadro 8.7: Caso de uso: Enviar mensaje.

Caso de uso	Añadir actualización de estado.	**Identificador**	$C8$
Precondición	El usuario se encuentra autenticado.		
Postcondición	Se crea una nueva actualizcion con como propietario el usuario autenticado.		
Propósito	Crear actualizaciones de estados.		
Resumen	El usuario puede agregar nuevas actualizaciones de estado a su perfil.		

Curso normal	**Curso alternativo**
1. Actor accede a su perfil.	
2. Actor pone el foco sobre el campo de texto de las actualizaciones.	
3. El sistema muestra entonces el botón de envio del formulario oculto.	
4. Actor rellena el formulario.	
5. Actor envía el formulario.	
6. Fin de caso de uso.	

Cuadro 8.8: Caso de uso: Añadir actualización de estado.

Caso de uso	Añadir comentario a un estado.	**Identificador**	$C9$
Precondición	El usuario esta autenticado y existe un estado que comentar.		
Postcondición	Un nuevo comentario queda almacenado en la base de datos.		
Postcondición	Una nueva notificación se genera para el dueño del estado.		
Propósito	Crear un comentario asociado al estado de un usuario.		
Resumen	El usuario puede escribir comentarios sobre los estados de otros usuarios o sobre el suyos propios.		

Curso normal	**Curso alternativo**
1. Actor accede a un perfil.	
2. Actor rellena el campo de texto bajo el comentario que quiere comentar.	
3. Actor envía el formulario.	
4. Fin de caso de uso.	

Cuadro 8.9: Caso de uso: Añadir comentario a un estado.

Caso de uso	Actualizar los datos del perfil.	**Identificador**	C_{10}
Precondición	El usuario esta autenticado en el sistema.		
Postcondición	Los datos del perfil del usuario se han actualizado.		
Propósito	Cambiar los datos del usuario del sistema.		
Resumen	El usuario puede cambiar los datos datos que el usuario conoce sobre el y que se muestran a los demas usuarios.		

Curso normal	**Curso alternativo**
1. Actor accede a su perfil.	
2. Actor selecciona `Editar perfil`	
3. El sistema sustituye los datos del perfil por un formulario	
4. Actor cambia los datos necesarios de los campos	
5. El sistema automáticamente actualiza la base de datos sin que Actor envíe el formulario explicitamente	
6. Fin de caso de uso.	

Cuadro 8.10: Caso de uso: Actualizar los datos del perfil.

Caso de uso	Salir.	**Identificador**	C_{11}
Precondición	El usuario se encuentra autenticado en el sistema.		
Postcondición	El usuario ya no se encuentra autenticado en el sistema.		
Propósito	Terminar la sesión del usuario.		
Resumen	El usuario puede abandonar el sistema cuando lo desee y pasar a estar identificado como un usuario anónimo.		

Curso normal	**Curso alternativo**
1. El usuario selecciona la opción `salir`.	
2. Fin de caso de uso.	

Cuadro 8.11: Caso de uso: Salir.

Caso de uso	Eliminar actualización.	**Identificador**	C_{12}
Precondición	El usuario se encuentra autenticado y tiene al menos una actualización.		
Postcondición	La actualización queda eliminada.		
Propósito	Borrar una actualización.		
Resumen	El usuario autenticado puede eliminar una actualización de la que es propietario.		

Curso normal	**Curso alternativo**
1. Actor sitúa el cursor sobre la actualización.	
2. El sistema muestra el icono para la eliminación de la actualización.	
3. Actor selecciona esta acción	
4. Fin de caso de uso.	

Cuadro 8.12: Caso de uso: Eliminar actualización.

Caso de uso	Nueva receta.	**Identificador**	C_{13}
Precondición	El usuario se encuentra autenticado.		
Postcondición	Se crea una nueva receta con el usuario autenticado como propietario.		
Propósito	Crear una nueva receta.		
Resumen	El usuario puede agregar nuevas recetas al sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en su perfil.	1.1. Actor se encuentra en cualquier página
2. Actor selecciona en el menu izquierdo la opción `Nueva Receta`	2.2. Actor selecciona en la barra superior la opción `Nueva receta`
3. Actor rellena los campos del formulario y lo envía	
4. El sistema valida los datos	
5. El sistema da por válidos los datos	5.3. El sistema da por inválidos los datos
6. El sistema presenta la receta	6.4. El sistema muestra de nuevo el formulario
7. Fin de caso de uso.	

Cuadro 8.13: Caso de uso: Nueva receta.

Caso de uso	Editar receta.	**Identificador**	C_{14}
Precondición	El usuario se encuentra autenticado y es dueño de la receta que va a ser modificada.		
Postcondición	La receta queda modificada como el usuario ha indicado.		
Propósito	Modificar recetas ya creadas.		
Resumen	El usuario puede modificar una receta de la que es propietario.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en la receta a modificar	
2. Actor selecciona `Editar`	
3. Actor modifica los campos del formulario y lo envía	
4. El sistema valida los datos	
5. El sistema da por válidos los datos	5.1. El sistema da por inválidos los datos
6. El sistema presenta la receta	6.2. El sistema muestra de nuevo el formulario
7. Fin de caso de uso.	

Cuadro 8.14: Caso de uso: Editar receta.

Caso de uso	Crear alternativa.	**Identificador**	C_{15}
Precondición	El usuario esta autenticado y existe una receta para ser derivada.		
Postcondición	Se crea una nueva receta con una referencia especial hacia la receta original.		
Propósito	Crear recetas derivadas de otras.		
Resumen	El usuario puede crear una receta derivada de la que esta viendo, con una referencia visible hacia esta.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en la receta a derivar	
2. Actor selecciona `Crear alternativa`	
3. Actor rellena los campos del formulario y lo envía	
4. El sistema valida los datos	
5. El sistema da por válidos los datos	5.1. El sistema da por inválidos los datos
6. El sistema presenta la receta	6.2. El sistema muestra de nuevo el formulario
7. Fin de caso de uso.	

Cuadro 8.15: Caso de uso: Crear alternativa.

Caso de uso	Comentar receta.	**Identificador**	C_{16}
Precondición	El usuario esta autenticado.		
Precondición	Existe una receta.		
Postcondición	Se crea un nuevo comentario sobre la receta con el usuario autenticado como propietario.		
Propósito	Crear comentarios sobre recetas.		
Resumen	El usuario tiene la posibilidad de comentar una receta que este viendo.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta a comentar	
2. Actor rellena el formulario de comentarios	
3. Actor envía el formulario	
4. Fin de caso de uso.	

Cuadro 8.16: Caso de uso: Comentar receta.

Caso de uso	Añadir receta a favoritos.	**Identificador**	C_{17}
Precondición	El usuario esta autenticado.		
Precondición	Existe una receta.		
Precondición	El usuario no tiene añadida esta receta como favorita.		
Postcondición	El usuario tiene añadida esta receta como favorita.		
Propósito	Crear un enlace entre un usuario y una receta.		
Resumen	El usuario puede agregar una receta a la lista de sus recetas favoritas para un fácil acceso.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `Guardar como favorito`	
3. Fin de caso de uso.	

Cuadro 8.17: Caso de uso: Añadir receta a favoritos.

Caso de uso	Eliminar receta de favoritos.	**Identificador**	C_{18}
Precondición	El usuario esta autenticado.		
Precondición	Existe una receta.		
Precondición	El usuario tiene añadida esta receta como favorita.		
Postcondición	El usuario ya no tiene añadida esta receta como favorita.		
Propósito	Eliminar el enlace entre un usuario y una receta.		
Resumen	El usuario puede eliminar una receta de sus favoritos si asi lo desea.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `Eliminar de favoritos`	
3. Fin de caso de uso.	

Cuadro 8.18: Caso de uso: Eliminar receta de favoritos.

Caso de uso	Cambio de imagen de la receta.	**Identificador**	C_{19}
Precondición	El usuario esta autenticado en el sistema.		
Precondición	El usuario es propietario de la receta.		
Postcondición	La receta tiene una nueva imagen de miniatura.		
Propósito	Cambiar la imagen en miniatura de la imagen.		
Resumen	El usuario puede cambiar la imagen que representa a la imagen y que ven el resto de usuarios en las búsquedas y perfiles.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `Seleccionar foto como principal`	
3. El sistema muestra que la opcion la foto se ha seleccionado satisfactoriamente	
4. Fin de caso de uso.	

Cuadro 8.19: Caso de uso: Cambio de imagen de la receta.

Caso de uso	Compartir en Twitter.	**Identificador**	C_{20}
Precondición	Existe una receta.		
Postcondición			
Propósito	Mostrar pantalla de comparticion de Twitter al usuario.		
Resumen	El susuario puede compartir una receta en Twitter.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `Twittear`	
3. Actor sigue los pasos que le indique Twitter en su plataforma	
4. Fin de caso de uso.	

Cuadro 8.20: Caso de uso: Compartir en Twitter.

Caso de uso	Compartir en Google+.	**Identificador**	C_{21}
Precondición	Existe una receta.		
Postcondición			
Propósito	Mostrar pantalla de comparticion de Google+ al usuario.		
Resumen	Mostrar pantalla de comparticion de Google+ al usuario.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `g+1`	
3. Actor sigue los pasos que le indique Google en su plataforma	
4. Fin de caso de uso.	

Cuadro 8.21: Caso de uso: Compartir en Google+.

Caso de uso	Compartir en Facebook.	**Identificador**	C_{22}
Precondición	Existe una receta.		
Postcondición			
Propósito	Mostrar pantalla de comparticion de Facebook al usuario.		
Resumen	Mostrar pantalla de comparticion de Facebook al usuario.		

Curso normal	**Curso alternativo**
1. Actor accede a la receta	
2. Actor selecciona `Recomendar`	
3. Actor sigue los pasos que le indique Facebook en su plataforma	
4. Fin de caso de uso.	

Cuadro 8.22: Caso de uso: Compartir en Facebook.

Caso de uso	Buscar receta (texto).	**Identificador**	C_{23}
Precondición			
Postcondición			
Propósito	Mostar al usuario los resultados de la búsqueda de texto.		
Resumen	Mostrar al usuario un primer resultado general sobre lo que busca.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en la página principal, sin iniciar sesión	1.1. Actor se encuentra encualquier pagina
2. Actor utiliza el campo de búsqueda central	2.2. Actor utiliza el campo de búsqueda de la barra superior
3. Fin de caso de uso.	

Cuadro 8.23: Caso de uso: Buscar receta (texto).

Caso de uso	Buscar receta (ingrediente).	**Identificador**	C_{24}
Precondición			
Postcondición			
Propósito	Mostar al usuario los resultados de la búsqueda de ingredientes.		
Resumen	Mostrar al usuario todas las recetas con el ingrediente que el usuario desea.		

Curso normal	**Curso alternativo**
1. Actor utiliza un campo de búsqueda normal	1.1. Actor utiliza el campo `Ingredientes` dentro del formulario avanzado de búsqueda
2. Actor introduce la palabra clave `ingrediente:` y acontinuación el ingrediente deseado, entre parentesis si tuviese espacios.	2.2. el usuario introduce el ingrediente dentro de dicho campo
3. El usuario envia el formulario	
4. Fin de caso de uso.	

Cuadro 8.24: Caso de uso: Buscar receta (ingrediente).

Caso de uso	Buscar receta (ingrediente negado).	**Identificador**	C_{25}
Precondición			
Postcondición			
Propósito	Mostar al usuario los resultados de la búsqueda de ingredientes negado.		
Resumen	Mostrar al usuario todas las recetas que no tiene el ingrediente que el usuario no desea.		

Curso normal	**Curso alternativo**
1. Actor utiliza un campo de búsqueda normal	1.1. Actor utiliza el campo `Sin Ingredientes` dentro del formulario avanzado de búsqueda
2. Actor introduce la palabra clave `-ingrediente:` y acontinuación el ingrediente no deseado, entre parentesis si tuviese espacios.	2.2. el usuario introduce el ingrediente dentro de dicho campo
3. El usuario envia el formulario	
4. Fin de caso de uso.	

Cuadro 8.25: Caso de uso: Buscar receta (ingrediente negado).

Caso de uso	Buscar receta (etiqueta).	**Identificador**	C_{26}
Precondición			
Postcondición			
Propósito	Mostar al usuario los resultados de la búsqueda por etiqueta.		
Resumen	Mostrar al usuario todas las recetas que contengan una determinada etiqueta introducida por el creador de la receta.		

Curso normal	**Curso alternativo**
1. Actor utiliza un campo de búsqueda normal	1.1. Actor utiliza el campo `Etiqueta` dentro del formulario avanzado de búsqueda
2. Actor introduce la palabra clave `etiqueta:` y acontinuación la etiqueta deseada, entre parentesis si tuviese espacios.	2.2. el usuario introduce la etiqueta dentro de dicho campo
3. El usuario envia el formulario	
4. Fin de caso de uso.	

Cuadro 8.26: Caso de uso: Buscar receta (etiqueta).

Caso de uso	Buscar receta (mixto).	**Identificador**	C_{27}
Precondición			
Postcondición			
Propósito	Mostar al usuario los resultados de la búsqueda mixta.		
Resumen	Mostrar al usuario avanzado el resultado de la combinatoria de varios criterios de filtro combinados de cualquier modo.		

Curso normal	**Curso alternativo**
1. Actor utiliza un campo de búsqueda normal	1.1. Actor utiliza los campos extras dentro del formulario avanzado de búsqueda
2. Actor introduce una combinacion de los anteriores vocabularios	2.2. Actor introduce en los diferentes campos los parametros deseados
3. Actor envia el formulario	
4. Fin de caso de uso.	

Cuadro 8.27: Caso de uso: Buscar receta (mixto).

Caso de uso	Buscar receta (vista detalle).	**Identificador**	C_{28}
Precondición			
Postcondición			
Propósito	Mostrar al usuario detalles de las recetas.		
Resumen	El usuario puede ver detalles sobre la receta sin necesidad de entrar en ella.		

Curso normal	**Curso alternativo**
1. Actor ha realizado cualquier búsqueda	1.1. Actor se encuentra en el modo `miniatura` y selecciona la opcion `detalle`
2. Fin de caso de uso.	

Cuadro 8.28: Caso de uso: Buscar receta (vista detalle).

Caso de uso	Buscar receta (vista miniaturas).	**Identificador**	C_{29}
Precondición			
Postcondición			
Propósito	Mostrar al usuario un mayor numero de recetas.		
Resumen	El usuario puede ver de un vistazo mas recetas si solo desea guiarse por las fotos.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el modo `detalle` de búsquedas y selecciona el modo `miniatura`	
2. Fin de caso de uso.	

Cuadro 8.29: Caso de uso: Buscar receta (vista miniaturas).

Caso de uso	Ultimas noticias (Todas).	**Identificador**	C_{30}
Precondición	El usuario esta autenticado.		
Postcondición			
Propósito	Informar al usuario de todas las actualizaciones de sus suscripciones.		
Resumen	El usuario puede ver todas las actualizaciones de los usuarios a los que esta suscrito.		

Curso normal	**Curso alternativo**
1. Actor acaba de autenticarse y es automaticamente enviado.	1.1. Actor selecciona el logo superior.
2. Fin de caso de uso.	

Cuadro 8.30: Caso de uso: Ultimas noticias (Todas).

Caso de uso	Ultimas noticias (Filtro: recetas).	**Identificador**	C_{31}
Precondición	El usuario esta autenticado.		
Postcondición			
Propósito	Informar al usuario de las recetas de sus suscripciones.		
Resumen	El usuario puede elegir solo ver las recetas de los usuarios a los que esta suscrito.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en ultimas noticias	
2. Actor selecciona la pestaña `Recetas`	
3. Fin de caso de uso.	

Cuadro 8.31: Caso de uso: Ultimas noticias (Filtro: recetas).

Caso de uso	Ultimas noticias (Filtro: favoritos).	**Identificador**	C_{32}
Precondición	El usuario esta autenticado.		
Postcondición			
Propósito	Informar al usuario de los favoritos de sus suscripciones.		
Resumen	El usuario puede elegir solo ver las recetas que los usuarios a los que esta suscrito han marcado como favoritas.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en ultimas noticias	
2. Actor selecciona la pestaña `Favoritos`	
3. Fin de caso de uso.	

Cuadro 8.32: Caso de uso: Ultimas noticias (Filtro: favoritos).

Caso de uso	Ultimas noticias (Filtro: social).	**Identificador**	C_{33}
Precondición	El usuario esta autenticado.		
Postcondición			
Propósito	Informar al usuario de las actuaciones sociales de sus suscripciones.		
Resumen	El usuario puede elegir solo ver las actualizaciones sociales de sus suscripciones, tales como actualizcaiones de estado o nuevas suscripcciones.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en ultimas noticias	
2. Actor selecciona la pestaña `Social`	
3. Fin de caso de uso.	

Cuadro 8.33: Caso de uso: Ultimas noticias (Filtro: social).

Caso de uso	Visitar notificación).	**Identificador**	C_{34}
Precondición	El usuario esta autenticado en el sistema.		
Precondición	El usuario tiene una notificación pendiente.		
Postcondición	La notificación queda marcada como leída.		
Propósito	Marcar notificaciones como leídas.		
Resumen	El usuario accede a una notificación que tiene pendiente y esta queda marcadacomo leída.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en ultimas noticias	
2. Actor selecciona una notificación	
3. Fin de caso de uso.	

Cuadro 8.34: Caso de uso: Visitar notificación).

Caso de uso	Ver resumen de mensajes.	**Identificador**	C_{35}
Precondición	El usuario esta autenticado en el sistema.		
Postcondición			
Propósito	Mostar todos los hilos de mensajes del usuario.		
Resumen	Al usuario se le muestra todos los hilos de mensaje para que pueda elegir en cual desea entrar.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en cualquier página	1.1. Actor se encuentra en la pagina principal
2. Actor selecciona en la parte superior la etiqueta de mensaje	2.2. Actor selecciona en el menu izquierdo `Mensajes`
3. Fin de caso de uso.	

Cuadro 8.35: Caso de uso: Ver resumen de mensajes.

Caso de uso	Ver un mensaje.	**Identificador**	C_{36}
Precondición	El usuario esta autenticado en el sistema.		
Precondición	Tiene un mensaje en la bandeja de entrada.		
Postcondición	El mensaje queda marcado como leído.		
Propósito	El usuario lea un mensaje.		
Resumen	El usuario puede leer en cualquier momento un mensaje en el que esta como destinatario o remitente.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el panel de mensajes.	
2. Actor selecciona el mensaje deseado	
3. Fin de caso de uso.	

Cuadro 8.36: Caso de uso: Ver un mensaje.

Caso de uso	Seguidores de un usuario.	**Identificador**	C_{37}
Precondición			
Postcondición			
Propósito	Mostar los Seguidores de un usuario.		
Resumen	El usuario puede consultar por quién es seguido un usuario del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el perfil del usuario a consultar	
2. Actor selecciona la etiqueta `Seguidores`	
3. Fin de caso de uso.	

Cuadro 8.37: Caso de uso: Seguidores de un usuario.

Caso de uso	Seguidos de un usuario.	**Identificador**	C_{38}
Precondición			
Postcondición			
Propósito	Mostar a quién sigue un usuario.		
Resumen	El usuario puede consultar a quién sigue un usuario del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el perfil del usuario a consultar	
2. Actor selecciona la etiqueta `Siguiendo`	
3. Fin de caso de uso.	

Cuadro 8.38: Caso de uso: Seguidos de un usuario.

Caso de uso	Recetas de un usuario.	**Identificador**	C_{39}
Precondición			
Postcondición			
Propósito	Mostar las recetas de un usuario.		
Resumen	El usuario puede consultar las recetas que ha escrito un usuario del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el perfil del usuario a consultar	
2. Actor selecciona la etiqueta `Recetas de <nombre de usuario>`	
3. Fin de caso de uso.	

Cuadro 8.39: Caso de uso: Recetas de un usuario.

Caso de uso	Favoritos de un usuario.	**Identificador**	C_{40}
Precondición			
Postcondición			
Propósito	Mostar las recetas favoritas de un usuario.		
Resumen	El usuario puede consultar las recetas que ha marcado como favoritas un usuario del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en el perfil del usuario a consultar	
2. Actor selecciona la etiqueta `Favoritos`	
3. Fin de caso de uso.	

Cuadro 8.40: Caso de uso: Favoritos de un usuario.

Caso de uso	Ver una receta.	**Identificador**	C_{41}
Precondición			
Postcondición			
Propósito	Mostrar una receta a los usuarios.		
Resumen	El usuario puede ver una receta del sistema cuando lo desee.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en una búsqueda o en cualquier otro listado de recetas	
2. Actor selecciona una receta	
3. Fin de caso de uso.	

Cuadro 8.41: Caso de uso: Ver una receta.

Caso de uso	Ver un perfil de un usuario.	**Identificador**	C_{42}
Precondición			
Postcondición			
Propósito	Mostrar actualizaciones de un usuario.		
Resumen	El usuario puede consultar todas las actualizcaiones que son propiedad deun usuario del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en una receta	1.1. Actor se encuentra en la panel principal
2. Actor selecciona un usuario	
3. Fin de caso de uso.	

Cuadro 8.42: Caso de uso: Ver un perfil de un usuario.

Caso de uso	Ver todas las recetas.	**Identificador**	C_{43}
Precondición			
Postcondición			
Propósito	Mostart un listado de todas las recetas.		
Resumen	Un usuario puede consultar todas las recetas que esten en la base de datos del sistema.		

Curso normal	**Curso alternativo**
1. Actor se encuentra sin autenticar en la página principal	1.1. Actor se encuentra en cualquier página
2. Actor selecciona `Todas las recetas` en el centro de la página	2.2. Actor selecciona `Todas las recetas` en la barra superior
3. Fin de caso de uso.	

Cuadro 8.43: Caso de uso: Ver todas las recetas.

Caso de uso	Ver sugerencias.	**Identificador**	C_{44}
Precondición			
Postcondición			
Propósito	Consultar las sugerencias de los usuarios a la aplicación.		
Resumen	Un usuario puede ver las sugerencias que otros usuarios han aportado para mejorar la aplicación.		

Curso normal	**Curso alternativo**
1. Actor selecciona la pestaña de sugerencias	
2. Fin de caso de uso.	

Cuadro 8.44: Caso de uso: Ver sugerencias.

Caso de uso	Añadir sugerencia.	**Identificador**	C_{45}
Precondición			
Postcondición			
Propósito	Poder sugerir una mejora para la aplicación.		
Resumen	El usuario puede sugerir la mejora que crea oportuna para la aplicación.		

Curso normal	**Curso alternativo**
1. Actor se encuentra en la pestaña de sugerencias.	
2. Actor envía el formulario con la sugerencia.	
3. Fin de caso de uso.	

Cuadro 8.45: Caso de uso: Añadir sugerencia.

8.2– Diseño.

8.2.1. Base de datos.

Como uno de los objetivos de la aplicación es la escalabilidad, no usaremos una base de datos tradicional, relacional. Utilizaremos uno de los nuevos sistemas de almacenamientos, llamados *NoSQL*. La base de datos que utilizaremos en concreto pertenece a la categoría de las bases de datos documentales, que es un paradigma diferente a las tablas.

En las bases de datos documentales cada registro es un documento. Un documento es una estructura de datos que contiene un numero indefi-

nido de pares clave-valor, llamados campos. A su vez estos valores pueden contener:

- Valores numéricos.
- Cadenas de caracteres.
- Otros documentos embebidos.
- Referencias a otros documentos.
- Una lista, pudiendo ser mixta, de cualquiera de los elementos anteriores.

Estos documentos se organizan en Colecciones, que sería el equivalente a una tabla en SQL. Pero a diferencia de SQL cada elemento que contiene no debe tener la misma estructura que el anterior, sino que cada uno puede ser completamente diferente.

Aunque esto podría llevarnos a guardar todos los documentos en una sola colección, normalmente todos los documentos que se guardan en una colección tienen una estructura similar y definida previamente.

En nuestra aplicación vamos a usar las siguientes colecciones:

users Usuarios.
recipes Recetas.
updates Actualizaciones/noticias.
threads Mensajes privados.
notifications Notificaciones.

Cuadro 8.46: Colecciones.

Como hemos expuesto previamente, cada colección tendrá una estructura pactada, aunque flexible. Para el caso de los usuarios hemos decidido una estructura prácticamente fija, a excepción de la longitud de las listas que es variable.

_id	Nombre de usuario.
name	Nombre de usuario.
dname	Nombre real del usuario.
email	Correo del usuario.
pass	Resumen de la contraseña del usuario.
stamp	Fecha de registro.
following	Lista de referencias a usuarios que sigue.
followed	Lista de referencias a usuarios por los que es seguido.

Cuadro 8.47: Colección Users (Usuarios).

Para la colección de Mensajes privados utilizaremos una estructura similar a la anterior. Pero en este caso la lista no sera de referencias exclusivamente sino que tendrá los documentos embebidos de los diferentes mensajes.

Además tiene otra particularidad, en el campo usuario guardaremos una lista de longitud fija que referencia a ambos usuarios involucrados en la aplicación.

_id usuario1:usuario2 (ordenados alfabéticamente).

users $[ref.usuario1, ref.usuario2]$.

stamp Hora del ultimo mensaje.

read Lista de usuarios que lo han leído.

messages Lista de:

from ref. usuario.

to ref. usuario.

text Texto del mensaje.

stamp Fecha del mensaje.

Cuadro 8.48: Colección de threads (Mensajes privados).

A continuación se nos presenta la estructura de documento mas grande. Cabe destacar el campo data, que es un campo que contiene un único documento embebido.

La filosofía de este campo es contener una colección de valores que dan una información extra y que pueden ampliarse en un futuro dependiendo de las necesidades que puedan surgir por demanda de los usuarios. El motivo de agruparlos y no dejarlos en el nivel superior es para que no cambie la estructura principal y básica de la receta.

_id usuario:idfy(nombre de la receta).

tiny Cadena aleatoria para acortador de URL.

id idfy(nombre de la receta).

textrecipe Texto de la preparación de la receta.

stamp Fecha de la receta.

user Referencia al usuario de la receta.

commentrecipe Comentario hecho por el autor de la receta.

raw_ingredients Copia del texto de los ingredientes introducido por el usuario.

tag_ingredients Lista de ingredientes.

tag_s_ingredients Lista de ingredientes pasados por el *stemmer*.

tag_i_tags Lista de tags pasadas por idfy.

stemm Nombre de la receta pasado por el *stemmer*.

search Lista de los valores de campos para búsquedas rápidas, tales como ingredientes, tags, nombre, etc...

comment Lista de comentarios. Para consultar los tipos vaya al Cuadro 8.50 en la página 74.

ncomments Número de comentarios.

nfavs Número de favoritos.

data :

- **totaltime** Tiempo total de preparación de la receta.
- **preptime** Tiempo de preparación de la receta (pelar, cortar, rallar, amasar, etc..).
- **cooktime** Tiempo de cocinado de la receta (tiempo al fuego, horno, etc..).
- **original** URL a la receta original.
- **yield** Servicios (1, 2, 4, 6, 8 personas).
- **entry** Introducción, campo anterior a la preparación.
- **origen** Lugar de origen.
- **difficulty** Dificultad de la receta.
- **price** Precio de la receta.

Cuadro 8.49: Colección de recipes (Recetas).

Como queremos mostrar además de los comentarios usuales de los usuarios unos comentarios especiales que indiquen si un usuario ha añadido a favoritos una receta, de igual modo que ocurre en la red de Tumblr.

Tipo comentario :

type "comment".

stamp Fecha del comentario.

user Ref. usuario.

text Texto del com.entario

Tipo favorito :

type "favorited".

stamp Fecha del comentario.

user Ref. usuario.

Cuadro 8.50: Tipos de comentarios en recetas.

En el diseño de la colección de actualizaciones podemos ver el potencial del almacenamiento de documentos con una estructura flexible. Para este diseño tenemos dos partes muy diferenciadas.

- La primera es una parte común de lo que consideramos que es la parte básica de todo lo que se puede mostrar en el perfil.

- Por otra parte tenemos los datos adicionales que le dan forma a cada tipo de actualización. Gracias a esta estructura de una sola consulta podemos extraer diferentes tipos de actualizaciones y utilizar un filtro común. Y si queremos añadir nuevos tipos no necesitamos nuevas tablas/colecciones, solo un nuevo valor para la clave type.

_id Marca de tiempo única.

user Ref. *user*.

stamp Fecha de actualización.

type Tipo.

update Documento, depende del tipo.

Opciones:

following Referencia a *followed*.

register *null*.

recipe Referencia a *recipe*.

favorited :

user Ref. *user*.

recipe Ref. *recipe*.

Cuadro 8.51: Colección de updates (Actualizaciones).

La colección de notificaciones una vez vista la colección de updates tiene poca novedad. Se trata de otra estructura divida en dos similar a la anterior.

- La primera parte, una parte común, que especifica lo que ha de tener una notificación, y una clave `type`, que especifica un segundo documento embebido.

- La segunda parte, el documento especificado por la clave `type`.

user usuario.

stamp fecha de notificación.

type tipo.

notification Documento, depende del tipo.

read valor lógico que informa sobre su lectura.

Opciones:

comment_update :

user ref. user.
recipe ref. actualización.

following ref. follower.

fav_recipe :

user ref. user.
recipe ref. *recipe*.

comment_recipe :

user ref. user.
recipe ref. *recipe*.

Cuadro 8.52: Colección de notifications (Notificaciones).

8.2.2. Gestión de la petición.

El tratamiento de la petición pasará por diferentes fases.

La primera fase es el análisis de la URL. El sistema identificará cada segmento de la URL y la clasificará de la manera oportuna para su posterior procesamiento. Realizaremos también un primer redireccionamiento temprano de URLs no válidas. No permitiremos URLs que terminen en «/».

La segunda fase es la inicialización del gestor de la base de datos. Si no podemos cargar la base de datos se abortará la petición.

La tercera fase corresponde a la identificación de la sesión y el usuario. Preparamos los datos para ser usados posteriormente.

La cuarta fase es la carga de los diferentes módulos de la aplicación. Aquí cargaremos las funciones que controlan a los usuarios y las recetas.

La quinta fase es el procesado de las acciones llevada a cabo por los usuarios de la aplicación. Este procesado se delegará a los módulos cargados.

La sexta fase es la llamada fase de *enrutamiento*, a partir de los datos guardados en las fases anteriores, principalmente por la primera fase, se va a hacer una elección de una plantilla para la *vista* y una disyunción de caminos que dará lugar a un prodcesado específico de la petición.

La séptima fase será ese procesamiento individual que dependerá de la *vista*.

La octava y última fase sera el envío al usuario del *renderizado* de la plantilla previamente seleccionada.

9 Nuts.

En este capítulo desarrollaremos Nuts, el *microframework* desarrollado para este proyecto. La decisión de crear este microframework es hacer un código mas modular y reutilizable en un futuro. Que ofreciera una interfaz expresiva y concisa en el manejo de las URLs, sin necesidad de crear infinidad de archivos o configuraciones.

9.1– UML.

El modelo conceptual del núcleo es el que se recoge a continuación. Hemos optado por una unica clase que conforma el núcleo, ya que para un *microframework* no necesitaremos más.

Además del núcleo ofreceremos un auto-cargador de módulos, para mayor comodidad del usuario y eficiencia del sistema, permitiendo cargar solo los módulos necesarios.

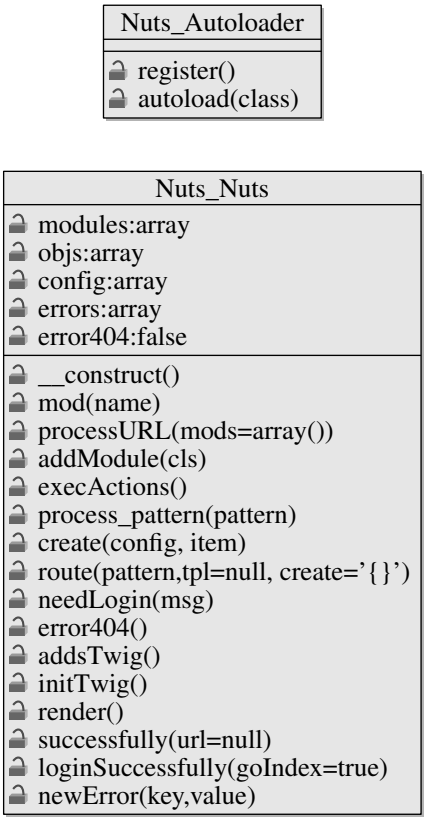

Figura 9.1: UML: núcleo de Nuts

9.2– Inicialización.

La inclusión de Nuts es tan sencillo como incluir el auto-cargador e indicar los módulos que queremos usar:

Código 9.1: Inicialización de Nuts

```
1 require('./Nuts/Autoloader.php');
2 Nuts_Autoloader::register();
3
4 $nuts = new Nuts_Nuts();
5 $nuts->processURL(array('Nuts_User', 'Nuts_Recipe'));
```

9.3– Tareas.

9.3.1. Crear id aleatorio.

Crea una cadena alfanumérica de longitud 4, con caracteres aleatorios.

9.3.2. Idfy.

Eliminar acentos y caracteres no-ASCII, deja los espacios como espacios, cambia los guiones por espacios.

9.3.3. Urlfy.

Eliminar acentos y caracteres no-ASCII, cambia los espacios por guiones, deja los guiones como guiones.

9.3.4. Stemming.

Stemming es un método para reducir una palabra a su raíz o (en inglés) a un stem o lema. Hay algunos algoritmos de stemming que ayudan en sistemas de recuperación de información. Stemming aumenta el recall que es una medida sobre el número de documentos que se pueden encontrar con una consulta. Por ejemplo, una consulta sobre "bibliotecas" también encuentra documentos en los que solo aparezca "bibliotecario" porque el stem de las dos palabras es el mismo ("bibliotec"). [Wik12]

9.3.5. Llamada asociativa.

La función call_method de Nuts provee de una interfaz intuitiva para hacer una llamada asociativa a un método mediante ReflectionObject y invokeArgs.

Código 9.2: call_method

```
call_method($obj, $method, $params)
```

9.3.6. Procesar URLs.

Se lleva a cabo mediante el método processURL y lleva a cabo las siguientes tareas:

1. Análisis de la URL
2. Redireccionamiento
3. Ejecución de métodos adecuados de los módulos

9.3.7. Suscripción de módulos.

Se lleva a cabo mediante el método processURL y lleva a cabo las siguientes tareas:

1. Análisis de la URL
2. Redireccionamiento
3. Ejecución de métodos adecuados de los módulos suscritos

9.3.8. *Enrutador* (Contexto por rutas).

Se puede utilizar mediante el método Route.

Preparara el contexto para las plantillas a partir de la ruta.

Además el *enrutador* también puede encargarse de hacer comprobaciones referentes a usuarios u otros objetos.

9.3.9. Comprobación autenticación.

Se puede utilizar mediante el método needLogin.

Comprueba si el usuario está autenticado, si no lo está lo manda a la página de autenticación

9.3.10. Render.

Realiza el renderizado de las plantillas mediante el motor de Twig, Nuts le pasa los valores que ha recolectado a lo largo de todo el procesado de la petición, además de los proporcionados por el usuario.

10 Creación aplicación de Inicio.

10.1– Appfog.

Si nos decidimos a usar AppFog como nuestro servidor, tenderemos que seguir los siguientes pasos:

1. EL primer paso sera registrarnos en AppFrog. Para ello accedemos a AppFog[1], rellenar los campos correspondientes, continuaremos pulsando `Singup`

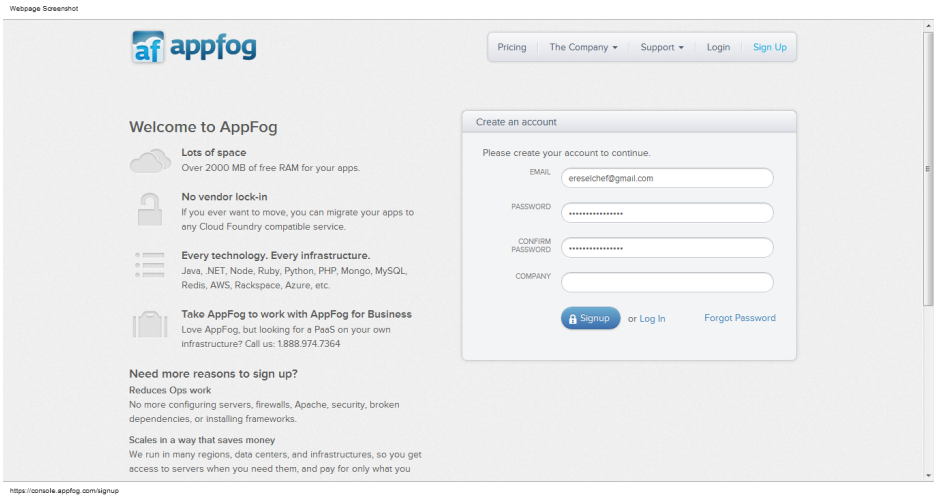

Figura 10.1: Registro en AppFog.

2. Esto nos llevará a una segunda pantalla donde seleccionaremos primero PHP y luego el cloud que deseamos, por ejemplo el cloud de

[1] **AppFog**: https://console.appfog.com/signup

Amazon del Oeste de Europa. Por último le pondremos el nombre a nuestra aplicación y pulsamos *Create App*

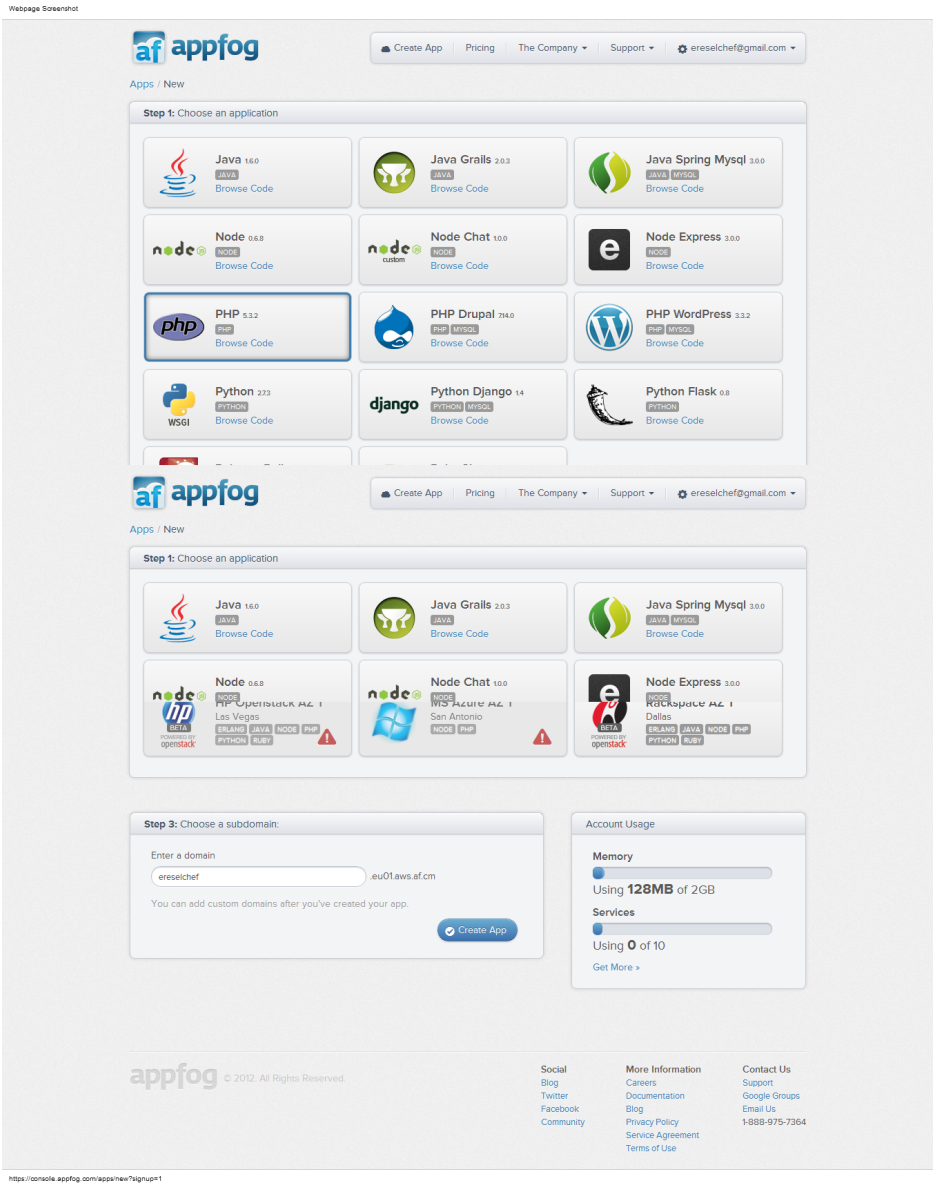

Figura 10.2: Nueva aplicación en AppFog.

3. Se nos mostrará una pantalla de espera que nos informará del progreso de nuestro aplicación, como la siguiente. Cuando termine, pulsaremos en *My Apps*.

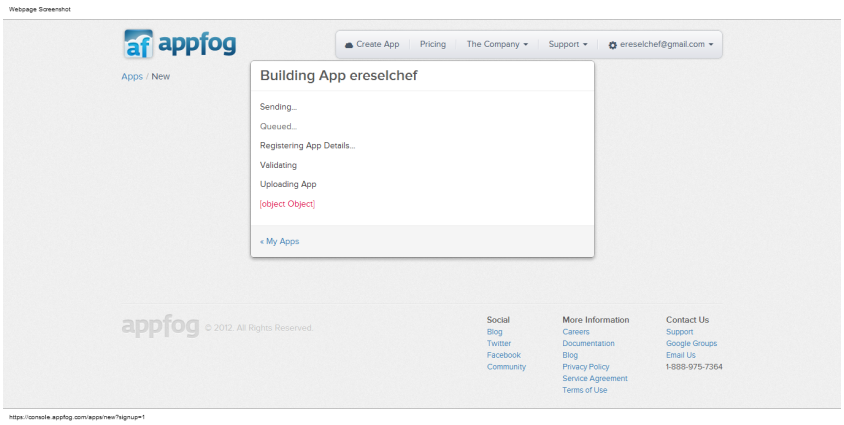

Figura 10.3: Pantalla de espera en AppFog.

4. Nos encontramos ante el panel de nuestras aplicaciones, como es nuestra primera aplicación solo se nos mostrará una. Se nos informa de su nombre, URL, memoria destinada y su estado, si esta funcionando o parada.

   Pulsaremos sobre el nombre de la aplicación para que nos lleve al panel de control.

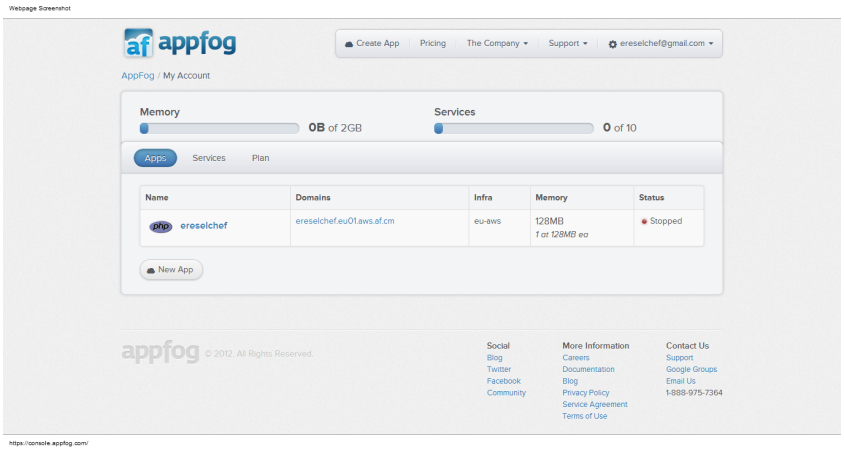

Figura 10.4: Panel de aplicaciones en AppFog.

5. Estamos ya en el panel de la aplicación, aquí podemos iniciar la aplicación si no está iniciada, a veces no se pone en marcha automáticamente. Si lo deseamos podemos aumentar el espacio dedicada a la aplicación por *AppFog*.

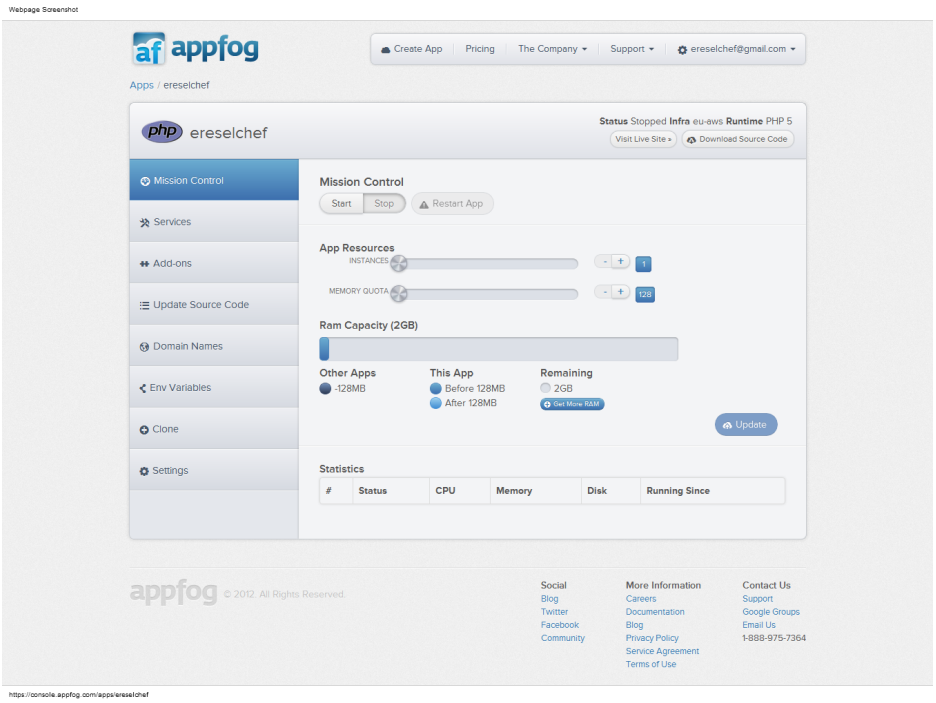

Figura 10.5: Panel de la aplicación en AppFog.

6. Vamos ahora a dar de alta el servidor MongoDB, para ello pulsaremos en la pestaña de servicios.

   Estamos ahora en la pantalla de Servicios, aquí daremos de alta el servicio MongoDB. Para ellos solo tenemos que pulsar sobre el logo de MongoDB, darle un nombre al servicio y pulsar sobre Create. Tras esto es necesario relacionar este servicio con nuestra aplicación. Para ello pulsaremos sobre Bind (aparecerá al lado de nuestro nuevo servicio) y esperaremos a que se reinicie nuestra aplicación.

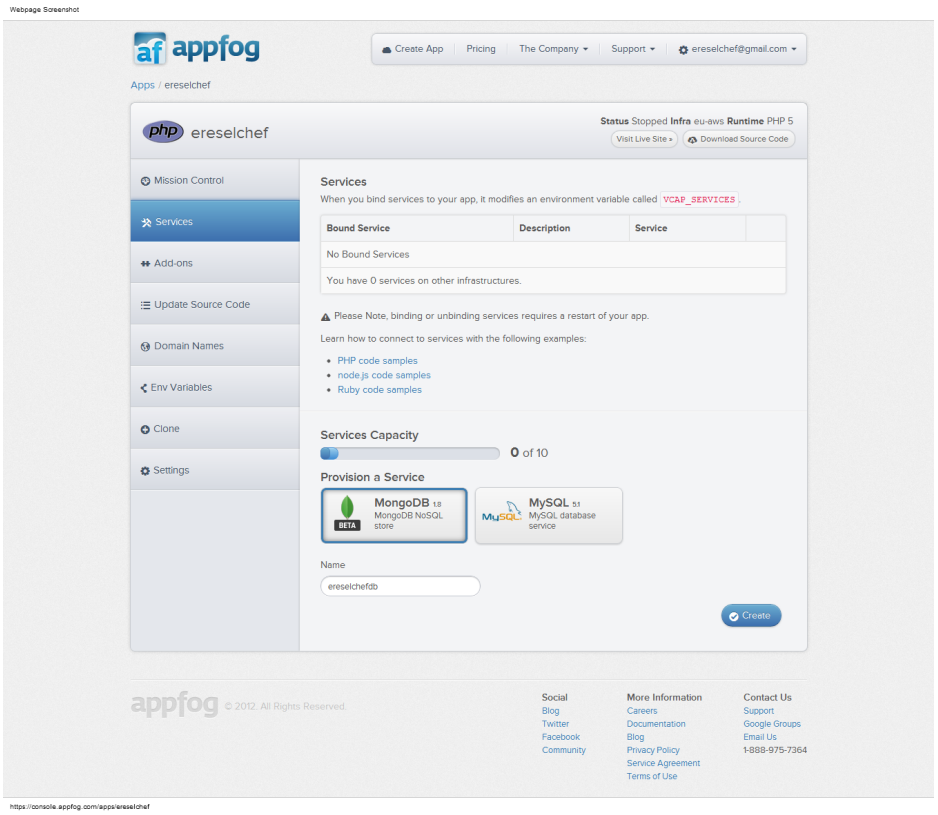

Figura 10.6: Crear Servicio en AppFog.

7. Ya tenemos lo necesario para poder crear nuestra aplicación. Solo nos falta subirla al servidor. Encontraremos las instrucciones en inglés en la pestaña `Update Source Code`.

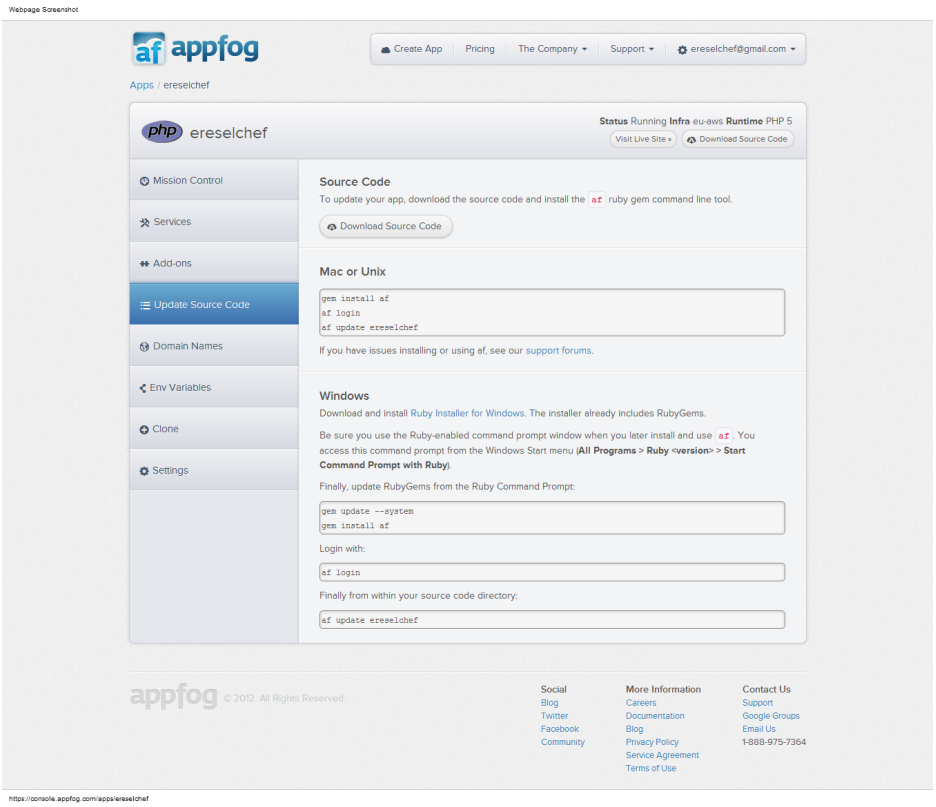

Figura 10.7: `Update source code` en AppFog.

8. Antes de subir nuestro primer código necesitaremos tener instalado Ruby (Como Sistema operativo más extendido nos centraremos en Windows) para ello accederemos a rubyinstaller[2] y descargamos el instalador pulsando en downloads. Luego seguiremos los pasos habituales en cualquier instalación de cualquier otro programa.

9. Una vez instalado accedemos al menú de inicio/Todos los programas/Ruby<*version*>/Start Command Prompt with Ruby.

10. En la terminal nueva que se nos habrá abierto introduciermos el siguiente comando:

Código 10.1: instalar appfog

```
gem update --system
gem install af
```

[2]**rubyinstaller**: http://rubyinstaller.org/

11. Luego nos autenticaremos:

Código 10.2: autenticación en appfog

```
af login
```

12. Cambiaremos al directorio donde se encuentre nuestro código que queremos subir.

13. Por último ya podemos subir nuestro código fuente con el siguiente comando:

Código 10.3: descarga del proyecto

```
af update {nombre de la aplicacion}
```

10.2– Hola mundo con Nuts y MongoDB.

Para crear una aplicación con Nuts solo necesitaremos 4 archivos.

El primero de ellos es un archivo de directivas de configuración para Apache *.htaccess*, en él le indicamos que todas las peticiones las deberá manejar el archivo *index.php*. El archivo de ejemplo podría servir para cualquier tipo de aplicación.

Código 10.4: .htaccess

```
1 RewriteEngine on
2
3 RewriteRule ^static/(.*)$ static/$1 [QSA,L,NC]
4 RewriteRule ^robots\.txt$ static/robots.txt [QSA,L,NC]
5 RewriteRule ^humans\.txt$ static/humans.txt [QSA,L,NC]
6 RewriteRule ^sitemap\.xml$ static/sitemap.xml [QSA,L,NC
    ]
7 RewriteRule ^favicon\.ico$ static/favicon.ico [QSA,L,NC
    ]
8
9 RewriteRule ^(.*)$ index.php [QSA,L,NC]
```

Es el archivo principal, en el cargaremos Nuts y lo configuraremos. En el caso de nuestra aplicación lo hemos configurado con el modulo que veremos a continuación `Nuts_Hola`.

También se encarga de la descripción de las URLs de nuestra aplicación. En el código de ejemplo contaremos con 3 declaraciones de URLs:

- La primera es el directorio raíz (o directorio barra). Utilizará la plantilla *hola(.twig)*
- La segunda manejara todas las peticiones del tipo `/hello/<variable name>`. También utilizará la plantilla *hola(.twig)*
- La tercera manejará los errores 404.

Por ultimo cargaremos el sistema de plantillas Twig y ejecutaremos la plantilla que corresponda con *render*.

Código 10.5: index.php

```
<?php
require('./Nuts/Autoloader.php');
Nuts_Autoloader::register();

$nuts = new Nuts_Nuts();
$nuts->processURL(array('Nuts_Hola'));

$nuts->config['description'] = "Hola";
$nuts->config['keywords'] = array('Hola','mundo');

if($nuts->route("","hola")):
elseif($nuts->route("hola/%name", "hola")):
else:
    $nuts->error404();
endif;

$twig = $nuts->initTwig();
$nuts->render();
?>
```

El siguiente archivo que vamos a analizar es *Hola.php*. Este se encargará de las conexiones con MongoDB. La clase `Nuts_Hola` consta de dos métodos.

El primer método es `public_addRegister`. Este método tiene un comportamiento especial porque su nombre empieza por *public_**. Estos métodos son llamados por Nuts cada vez que éste recibe un formulario por *post* y que contenga un campo `action` que tenga como valor el nombre del resto de la función, en este caso `addRegister`.

Este método se encarga de añadir el valor del campo `txt` en un documento que se introducirá en la base de datos en la colección *registros*.

El segundo método es getRegister, este método simplemente hace una consulta de todos los documentos de la colección `registros`. Sería el equivalente a un `Select * from registros;` en SQL.

Código 10.6: Nuts/Hola.php

```
<?php

class Nuts_Hola{

    function public_addRegister($txt){
        $collection = DB::$db->registros;
        $collection->insert(array("txt"=>$txt));
    }

    function getRegisters(){
        $collection = DB::$db->registros;
        return $collection->find();
    }

}
?>
```

Por último nos encontramos con la plantilla para nuestra mini-aplicación. En primer lugar vemos la simpleza con la que Nuts nos ofrece la variable `name` que hemos indicado en el patrón de las URLs de la aplicación.

En segundo lugar encontramos el formulario que está conectado con el módulo, simplemente como hemos comentado antes mediante el campo `action`. Además también podemos observar que con Nuts podemos acceder facilmente a las variables que hemos pasado por *postpost*.

En tercer lugar encontramos el recorrido de un cursor del resultado de la consulta a MongoDB. Como podemos observar en cada iteración imprimiremos el valor asociado a la clave `txt`.

Código 10.7: templates/hola.twig

```
Hola {{name|default("mundo")}}!
<br/>
<a href="/hola/Manu"> Hola Manu </a> - <a href="/hola/
    Jose"> Hola Jose </a>
<br /><br />
<form method="post">
```

```
6  <input type="hidden" name="action" value="addRegister"
       />
7  <input type="text" name="txt" value="{{post.txt|default
       ('escribe texto aqui')}}" />
8  <input type="submit"/>
9  </form>
10 {% for register in Nuts_Hola.getRegisters %}
11     {{register.txt}}
12     <hr>
13 {% endfor %}
```

Aquí tenemos un ejemplo de nuestra pequeña aplicación en funcionamiento.

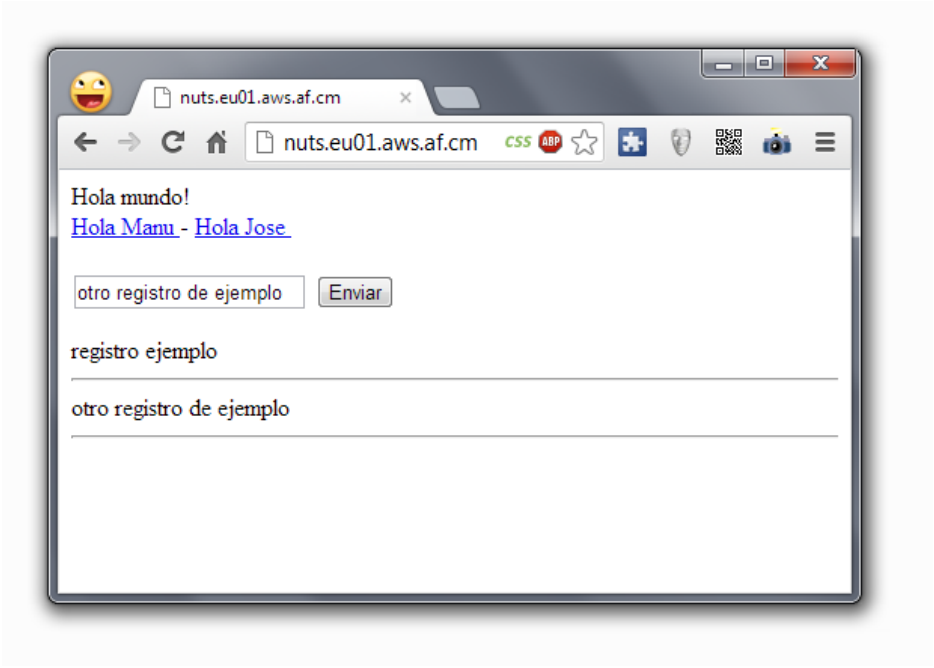

Figura 10.8: Vista primera aplicación.

11 Creación de la aplicación ereselchef.

11.1– Modelo

Gracias a Nuts la creación del modelo de la aplicación se reduce a la creación de dos módulos para Nuts. El convenio para que funcionen con el auto-cargador es que la clase auto-cargable comience con *Nuts_** y que las acciones accesibles por el usuario de la web comiencen con *public_**.

Por modularidad hemos separados los adaptadores para MongoDB de las reglas de validación y el resto de las gestiones de la aplicación, así en total tendremos 4 clases, que presentamos a continuación.

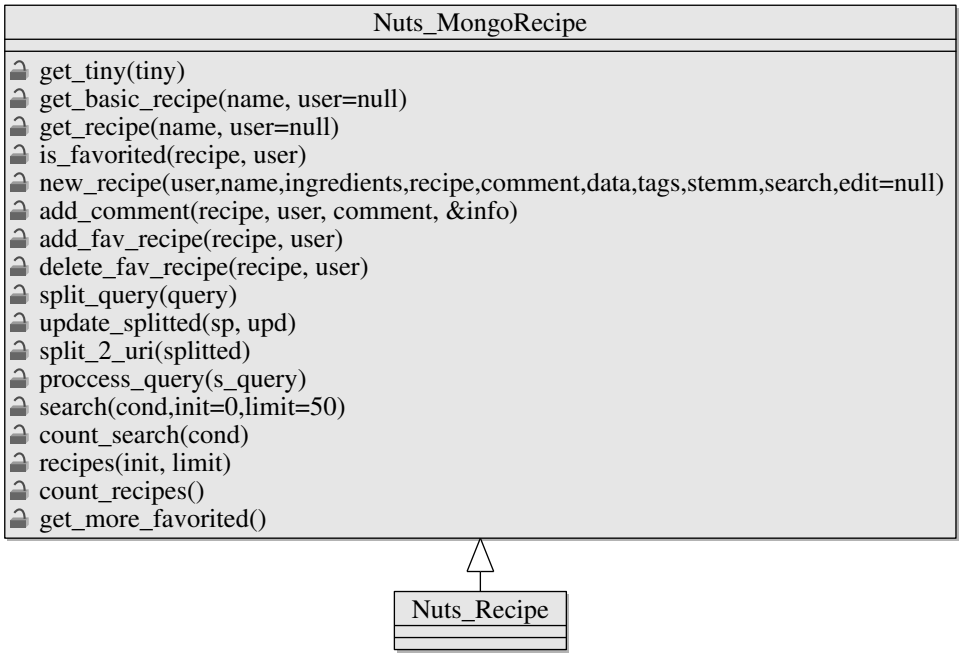

Figura 11.1: UML: Detalle de Nuts_MongoRecipe

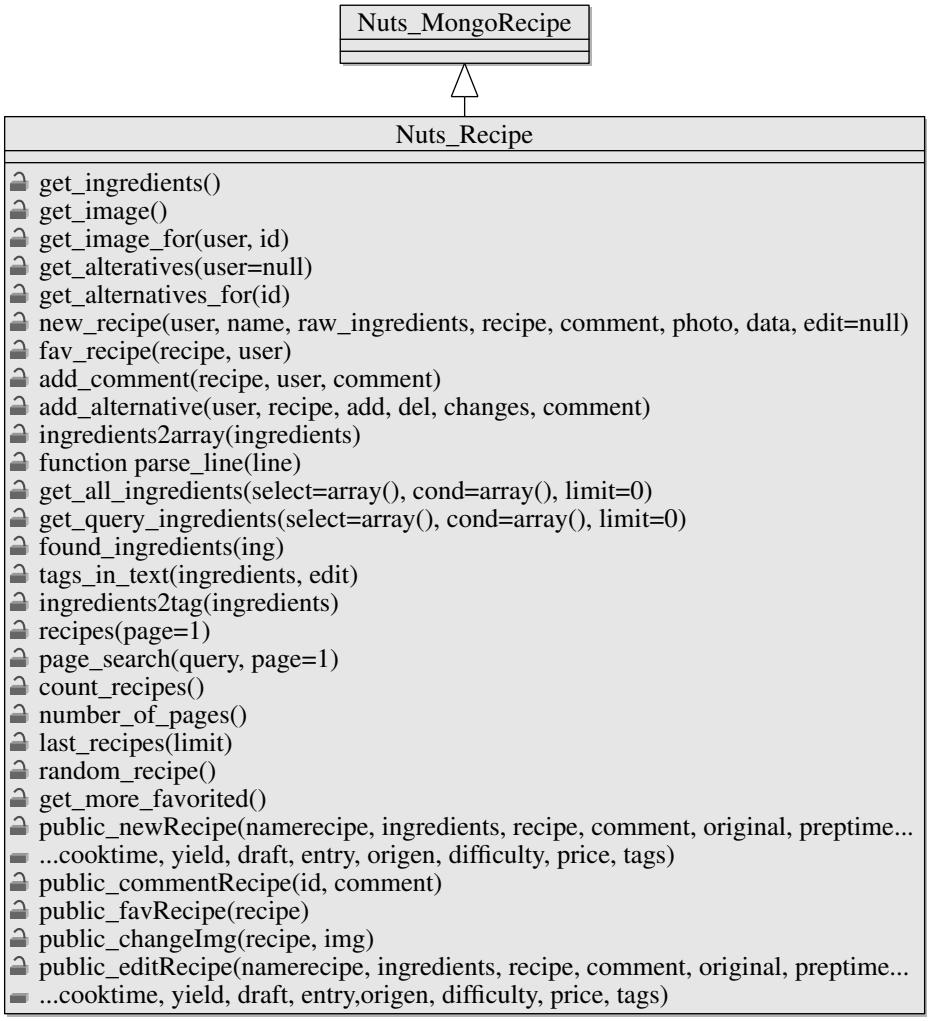

Figura 11.2: UML: Detalle de Nuts_Recipe

Figura 11.3: UML: Detalle de Nuts_MongoUser

Figura 11.4: UML: Detalle de Nuts_User

12 Alojamiento Web.

12.1– Alternativas

12.1.1. Pago vs. gratuito.

La diferencia entre unos y otros es muy grande. Los sistemas gratuitos son bastante escasos, y que además cumplan los requerimientos de nuestro proyecto son aún más escasos. Además algunos de los servicios gratuitos existentes agregan publicidad a nuestro contenido y tienen limitación de tráfico.

Entre los sistemas de pagos hay gran variedad. La ventaja del sistema de pago es que podremos controlar los detalles de nuestra máquina con mayor precisión. No agregan publicidad y la limitación de tráfico es flexible dependiendo del plan de precios.

12.1.2. Local vs. remoto.

Los servidores locales tienen como ventaja el acceso directo a la máquina y la posibilidad de hacer cambios a nuestro antojo sin tener que preocuparnos de accesos remotos. Como desventaja tiene que si tenemos una línea domestica, en la mayoría de los casos no es ni siquiera simétrica, siendo la velocidad de subida (la que necesitamos para servir nuestras páginas) mucho menor que la de bajada. Por otro lado tenemos el inconveniente de que tenemos que preparar el sistema para tenerlo 24 horas encendido y preocuparnos de que todo vaya bien.

Por su parte los servidores remotos, si bien habitualmente son más caros, nos proporcionarán lo que necesitamos realmente para una aplicación. Cuentan con el ancho de banda necesario, tienen encargados en mantener

en pie nuestro sistema y además suelen ofrecer un soporte técnico gratuito y efectivo.

12.1.3. AppFog

AppFog es el servicio que hemos elegido como recomendado para nuestra aplicación.

Combina la ventaja de los servicios de pagos con los gratuitos, ya que ofrece un plan gratuito que podramos ampliar siempre que queramos a uno de pago.

Appfog no es un servicio de alojamiento web al uso, ya que no ofrece sus propios servidores como alojamiento, sino que sirve de intermediario con los clouds más famosos como pueden ser los de Amazon, Rackspace o Microsoft Azure; entre los que nos podremos mover con total libertad.

Todo esto lo completa con unas configuraciones personalizadas para cada tipo de lenguaje de programación que queramos correr y con una fácil integración con servicios como Mongodb, PostgreSQL y MySQL, sin que por ello tengamos que preocuparnos de instalaciones.

Además ofrece balanceo de carga automática, 50GB de transferencia, posibilidad de personalizar el dominio, multi-instancias y un sorporte técnico realmente rápido, eficiente y gratuito.

13 Distribución Xampp.

Nos hemos decido por Xampp como nuestro servidor para pruebas locales porque como ya vimos el punto 6.1.7 es un servidor multiplataforma y software libre.

Xammp lo usaremos para correr nuestro Apache y código PHP. No quiere decir que estemos desaprovechando recursos, puesto que Xampp permite desactivar los servicios que no necesitemos, como es MySQL.

Como Xampp solo lo usaremos como servidor de desarrollos y pruebas no entraremos a detallar aspectos de seguridad. De hecho, aunque en la actualidad Xampp se pueda usar como servidor en produción se desaconseja su uso como tal, puesto que para un uso intensivo es recomendable una instalación más personalizada.

13.1– Instalación

Para la instalación de Xampp nos dirigiremos a los siguientes enlaces, dependiendo de nuestra plataforma:

Windows:

http://www.apachefriends.org/es/xampp-windows.html

Linux:

http://www.apachefriends.org/es/xampp-linux.html

Mac OS X:

http://www.apachefriends.org/es/xampp-macosx.html

Nos descagaremos el fichero y seguimos los pasos especificados para cada plataforma. Es importante fijarnos también en la manera de levantar el servicio de apache y php en nuestro sistema.

En windows, el sistema elegido por nosotros para el desarrollo y la documentación, el proceso sera tan simple como descargarnos el instalador y seguir los pasos proporcionados por el asistente para su instalación. Para levantar el servicio tendremos un panel de control donde se especifica cada uno de los servicios. Solo tendremos que iniciar apache para nuestro proyecto.

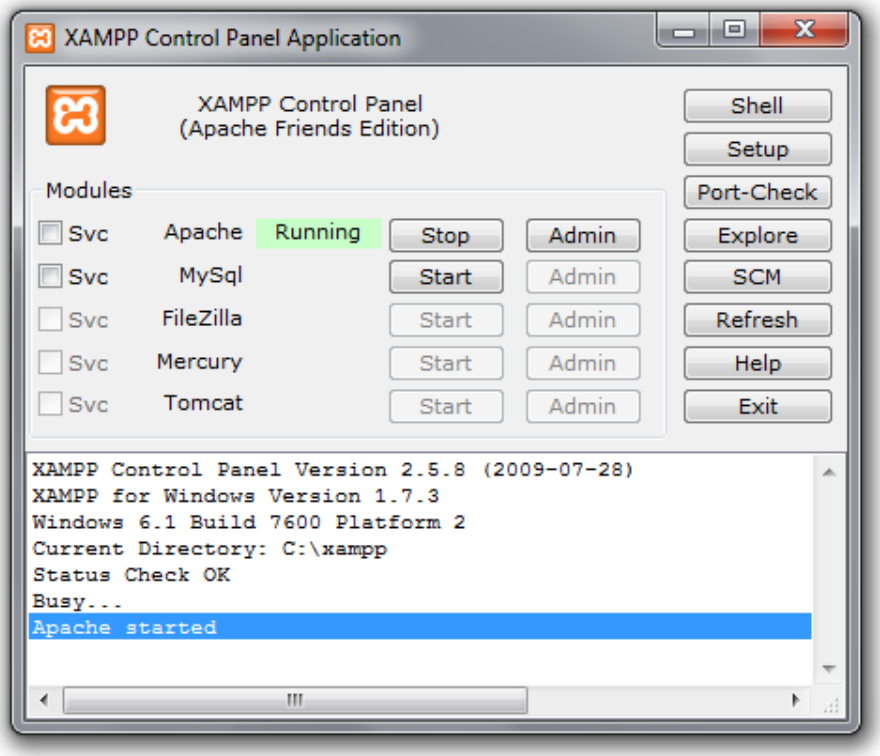

Figura 13.1: Panel de control de XAMPP

Luego tendremos que copiar nuestro proyecto a la carpeta *htdocs*, que en Windows la ruta por defecto se encuentra en *C:\xampp\htdocs*.

14 MongoDB Server.

Como su propio nombre indica MongoDB Server es la aplicación que ejecuta el servicio de la base de datos MongoDB. Para nuestro proyecto requerimos la versión 1.8 o superior.

Nota: Hay que tener en cuenta que si nuestra máquina es de 32 bits nuestra base de datos estará limitada a 2GB de datos.

14.1– Descarga e inicialización.

Para descargar MongoDB nos dirijimos a:

http://www.mongodb.org/downloads

Ahí nos descargaremos la versión adecuada para nuestro sistema operativo, lo descomprimimos, y ya está! Solo tendremos que ejecutar *mongod*, *.exe* en el caso de Windows.

Cuando ejecutemos el servidor tendremos una terminal con un *log* de las peticiones al servidor.

Nota: En caso de cierre forzado *MongoDB* deja un archivo *mongod.lock* que debe ser eliminado manualmente para volver a ser iniciado.

14.2– Consola Javascript.

MongoDB trae además una consola interactiva con la que podremos acceder a las *colecciones* que tengamos en nuestra base de datos. Para ello iniciaremos *mongo(.exe)* presente en la misma carpeta que el servidor.

14.3– Introducción a la consola.

14.3.1. Obteniendo la conexión a la base de datos.

Ahora vamos a centrarnos en la manipular la base de datos a través de la consola.

(Podríamos realizar operaciones similares desde cualquier lenguaje de programación utilizando su correspondiente *driver*. La consola es lo adecuado para usos interactivos y administrativos).

Primero inicia la Consola MongoDB Javascript. En caso de Windows abriendo *mongo.exe*.

Por defecto la consola se conecta a la base de datos `test` en `localhost`. Verás algo como:

Código 14.1: instalar AppFog

```
MongoDB shell version: <numero de la version>
url: test
connecting to: test
type \comillas{help} for help
>
```

`connecting to:` te dice el nombre de la base de datos que está usando la consola. Para cambiar de base de datos escribe:

Código 14.2: instalar appfog

```
> use mydb
switched to db mydb
```

Cambiar de base de datos con el comando `use` no crea inmediatamente una nueva base de datos, la base de datos se crea *perezosamente* la primera vez que insertamos datos. Esto significa que si tu usas `use` una base de datos por primera vez no sera mostrado por `show dbs` antes de que los datos sean insertados.

Para ver el manual de comandos escribe `help`.

Nota para los desarrolladores con experiencia en otras bases de datos:

Como puedes ver en el ejemplo anterior nosotros no creamos nunca

bases de datos o colecciones. MongoDB no necesita que tu lo hagas. Tan prongo como tu insertes algo MongoDB crea *al vuelo* las colecciones y bases de datos. Si tu consultas una colección que no existe MongoDB la trata como una colección vacía.

14.3.2. Esquema dinámico (Esquema libre).

MongoDB tiene bases de datos, colecciones e indices similares a los tradicionales [1]Sistema de gestión de bases de datos relacionales, por sus siglas en inglés: Relational DataBase Management System.. En algunos casos (bases de datos y colecciones), estos objetos pueden ser creados implicitamente, sin embargo una vez creados se guardan en el catálogo del sistema (`db.system.collections`, `db.system.indexes`).

Las colecciones contienen documentos (BSON). Dentro de estos documentos están los campos. En MongoDB no hay *predefinición* de campos (Como ocurren en los RDBMS). No hay esquema para los campos dentro de los documentos, los campos y los tipos de sus valores pueden variar. Por lo tanto no hay noción de la operación *alter table* o de añadir una *columna*. En la práctica, es muy común que para cada colección tengamos una estructura homogénea para el conjunto de todos los documentos, sin embargo esto no es un requisito. Esta flexibilidad significa que la migración y aumento del esquema es muy fácil en la práctica, raramente nosotros necesitaremos escribir *scripts* para llear a cabo las operaciones del tipo *alter table*. Además, al proporcionar una migración de esquema flexible facilitamos el desarrollo de software iterativo sobre la base de datos.

14.3.3. Insertar datos en una colección.

Vamos a crear una colceción de prueba e insertar algunos datos en ella. Crearemos dos objetos, `j` y `t`, y los guardaremos en una colección `things`.

En el siguiente ejemplo, > indica los comandos que deben ser introducidos en la consola.

Código 14.3: instalar appfog

```
j = { name : "mongo" };
{"name" : "mongo"}
```

[1]RDBMS

```
> t = { x : 3 };
{ "x" : 3  }
> db.things.save(j);
> db.things.save(t);
> db.things.find();
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
>
```

A tener en cuenta:

- No *predefinimos* la colección. La base de datos se crea automaticamente al insertar.
- Los documentos son guardados, pueden tener diferentes campos, de hecho en el ejemplo los documentos no tienen campos de datos en común. En la práctica, normalmente guardaremos documentos con la misma estructura dentro de las colecciones.
- Una vez insertados en la base de datos a los objetos de le asigna un `ObjectId` (si no tienen uno) en el campo `_id`.
- Cada vez que realizamos el ejemplo de arriba nuestro `ObjectID` tendrá un valor diferente.

Vamos a añadir algunos registros más a la colección:

Código 14.4: instalar appfog

```
> for (var i = 1; i <= 20; i++) db.things.save({x : 4, j : i});
> db.things.find();
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "x" : 4, "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
has more
```

Si miramos el ejemplo anterior nos daremos cuenta de que no todos los documentos han sido mostrados, la consola limita el número a los primeros 20. Como ya teniamos 2 documentos en la colección sólo veremos los primeros 18 documentos nuevos insertados.

Si queremos que muestre el siguiente conjunto de resultados, usaremos el acceso rapido `id`. La continuación del ejemplo anteriore sería:

Código 14.5: instalar appfog

```
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
has more
> it
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "x" : 4, "j" : 20 }
```

Tecnicamente, `find()` devuelve un objeto de tipo *cursor*. Pero en los casos como los de arriba nosotros no hemos asignado el cursor a una variable. Por eso la consola automaticamente itera sobre el cursor dandonos el conjunto inicial de resultados, permitiendonos continuar con el comando `it`.

Pero nosotros también podemos trabajar con el cursor directamente, como discutiremos en el siguiente punto.

14.3.4. Accediendo a los datos de una consulta.

Antes de que discutamos las consultas en profundidas, vamos a hablar sobre como trabajar con los resultados de una consulta, un objeto de tipo cursor. Nosotros usaremos simplemente el metódo `find()`, que devuelve toda la colección, y hablaremos sobre como crear consultas específicas mas tarde.

Para ver todos los elementos de la colección cuando usamos la consola de mongo nosotros necesitaremos usar el cursor devuelto por la operacion `find()`.

Vamos a repetir la misma consulta, pero esta vez usaremos el cursor que devuelve `find()`, y lo iteraremos dentro de un bucle:

Código 14.6: instalar appfog

```
> var cursor = db.things.find();
> while (cursor.hasNext()) printjson(cursor.next());
```

```
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "x" : 4, "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "x" : 4, "j" : 20 }
```

El ejemplo superior nos muestra la iteración al *estilo cursor*. La función `hasNext()` nos dice si hay algún documento más para devolver y la función `next()` devuelve el nuevo documento. Nosotros podemos usar tambien el método incluido `printjson()` para imprimir el documento en su formato apropiado.

Cuando trabajamos en la consola JavaScript, nosotros podemos usar las características funcionales del lenguaje, y podemos simplemente llamar a `forEach` sobre el cursor. Repitiendo el ejemplo anterior, pero usando `forEach()` directamente en el cursor en vez de el bucle while:

Código 14.7: instalar appfog

```
> db.things.find().forEach(printjson);
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "x" : 4, "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
```

```
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "x" : 4, "j" : 20 }
```

En el caso de `forEach()` nosotros podemos definir una función que sera llamada por cada documento en el cursor.

En la consola mongo podemos tratar el cursor como una lista:

Código 14.8: instalar appfog

```
> var cursor = db.things.find();
> printjson(cursor[4]);
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
```

Cuando usamos el cursor de esta manera todos los documentos anteriores al cursor[4] han sido cargados en la RAM al mismo tiempo. Este método no es apropiado para conjuntos de resultados muy grandes que se salgan de la memoria. Los cursores deben ser usados como iteradores con cualquiera consulta que devuelva un gran número de elementos.

Además de acceder *estilo array* al curosr podemos tambien convertir el cursor a un verdadero array:

Código 14.9: instalar appfog

```
> var arr = db.things.find().toArray();
> arr[5];
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
```

Pero esta característica es específica de la consola interactiva de mongo, y no se ofrece en todos los *drivers*.

Los cursores de MongoDB no son una *snapshot*[2], las operaciones realizadas por nosotros u otros usuario en la colección mientras se está consultado entre la primera y última llamada a `next()` a nuestro cursor *podrían o no* ser devueltas por el cursor. Debemos usar explicitamente el bloqueo para realizar una consulta sobre una *snapshot*.

[2] **snapshot**: Literalmente instántanea. En informática, una copia instantánea de volumen o Snapshot es una instantánea del estado de un sistema en un momento determinado. Puede referirse a una copia real del estado de un sistema o de una capacidad que ofrecen los sistemas de copia de seguridad.

14.3.5. Especificando que queremos que devuelva la consulta.

Ahora que sabemos como trabajar con el objeto cursor que nos devuelven las consultas, vamos a poner el foco en como adpatar las consultas para devolver cosas específicas.

En general, la manera de hacer esto es crear *documentos de consulta*, con estos documentos indicamos cual es el patrón de claves y valores que han de ser emparejados.

Es mas fácil demostrarlo que explicarlo. En los siguientes ejemplos nosotros daremos ejemplos de consultas SQL y demostraremos como representar los mismo usando las consultas de MongoDB vía la consola de mongo. Esta forma de especificar consultas es básica para MongoDB, encontraremos la misma facilidad en cualquier *driver* o lenguaje.

SELECT * FROM things WHERE name=“mongo”

Código 14.10: .

```
> var arr = db.things.find().toArray();
> arr[5];
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
```

SELECT * FROM things WHERE x=4

Código 14.11: .

```
> db.things.find({x:4}).forEach(printjson);
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "x" : 4, "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "x" : 4, "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "x" : 4, "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "x" : 4, "j" : 20 }
```

La expresión de la consulta es un documento en sí misma. Un documento de una consulta de la forma a:A, b:B, ... significa "where a==A and b==B and ..."

MongoDB tambien te permite devolver partes de documentos, documentos que tienen sólo un subcojunto de elementos del documento guardado en la base de datos. Para hacer esto, debemos añadir un segundo argumento a la consulta `find()`, ofreciendo un documento que liste los elementos para ser devueltos.

Para ilustrarlo, repetiremos el ejemplo `find({x:4})` con un argumento adicional que limita el documento devuelto a solo los elemetos `j`:

SELECT j FROM things WHERE x$\bar{4}$

Código 14.12: .

```
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "j" : 1 }
{ "_id" : ObjectId("4c220a42f3924d31102bd857"), "j" : 2 }
{ "_id" : ObjectId("4c220a42f3924d31102bd858"), "j" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd859"), "j" : 4 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85a"), "j" : 5 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85b"), "j" : 6 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85c"), "j" : 7 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85d"), "j" : 8 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85e"), "j" : 9 }
{ "_id" : ObjectId("4c220a42f3924d31102bd85f"), "j" : 10 }
{ "_id" : ObjectId("4c220a42f3924d31102bd860"), "j" : 11 }
{ "_id" : ObjectId("4c220a42f3924d31102bd861"), "j" : 12 }
{ "_id" : ObjectId("4c220a42f3924d31102bd862"), "j" : 13 }
{ "_id" : ObjectId("4c220a42f3924d31102bd863"), "j" : 14 }
{ "_id" : ObjectId("4c220a42f3924d31102bd864"), "j" : 15 }
{ "_id" : ObjectId("4c220a42f3924d31102bd865"), "j" : 16 }
{ "_id" : ObjectId("4c220a42f3924d31102bd866"), "j" : 17 }
{ "_id" : ObjectId("4c220a42f3924d31102bd867"), "j" : 18 }
{ "_id" : ObjectId("4c220a42f3924d31102bd868"), "j" : 19 }
{ "_id" : ObjectId("4c220a42f3924d31102bd869"), "j" : 20 }
```

Podemos observar que el campo `_id` siempre es devuelto.

14.3.6. findOne() - *Syntactic Sugar*

Por convenio, la consola mongo (y otros *drivers* nos permite sobrecargar el curosr, y perimite otener un documento por medio de la función `findOne()`. `findOne()` toma exactamente los mismos parámetros que la función `find()`, pero en vez de devolver un cursor, nos devolverá el primer documento devuelto desde la base de datos, o `null` si no se ha encontrado ninguno documento emparejado con la consulta especificada.

Como ejemplo nos permite obtener el documento con `name==\mongo"`. Hay muchas maneras de hacerlo, incluyendo simplemente llamar a `next()` sobre el cursor (después comprobar si es `null`, por supuesto), o tratar el cursr como un array y acceder al elemento número `0`.

Sin embargo, el método `findOne()` es más conveniente y eficiente que ambos.

Código 14.13: .

```
> printjson(db.things.findOne({name:"mongo"}));
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
```

Ésto es más eficiente porque el cliente pide un sólo objeto de la base de datos, por eso es menos trabajo que hacer por la base de datos y la red. Ésto es el equivalente de `find({name:\mongo").limit(1)`.

Otro ejemplo de encontrar un único documento es por `_id`:

Código 14.14: .

```
> var doc = db.things.findOne({_id:ObjectId("4
    c2209f9f3924d31102bd84a")});
> doc
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
```

14.3.7. Limitando el conjunto de resultados via `limit()`.

Nosotros podemos limitar el tamaño del conjunto de resultados de las consultas especificando el número máximo de resultados que será devuelto por medio del método `limit()`.

Esto es muy recomendable por razones de rendimiento, ya que limita el trabajo que la base de datos hace, y limita la cantidad de datos devuelto por la red. Por ejemplo:

Código 14.15: .

```
> db.things.find().limit(3);
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84b"), "x" : 3 }
{ "_id" : ObjectId("4c220a42f3924d31102bd856"), "x" : 4, "j" : 1 }
```

14.3.8. Más ayuda.

Además del comando general `help`, podemos llamar a la ayuda en la en `db` y `db.<collection>` (donde *<collection>* es el nombre de la colección) para ver el resumen de métodos disponibles.

Si queremos investigar que están haciendo las funciones, podemos escribir su nombre si `()`s y la consola imprimira el código fuente, por ejemplo:

Código 14.16: .

```
> printjson
function (x) {
    print(tojson(x));
}
```

mongo es una consola completa de Javascript, por eso cualquier Javascript, por eso cualquier función, sintaxis, o clase JavaScript puede usarse en la consola.

Podemos ver la API completa en http://api.mongodb.org/js/.

15 Análisis temporal y costes de desarrollo.

15.1– Tareas.

Dividiremos el desarrollo de nuestra aplicación en diversas tareas que describiremos a continuación.

15.1.1. Planteamiento inicial.

Esta tarea consiste en la elección de nuestro proyecto, en como fusionamos las diferentes ideas y nos marcamos los objetivos principales de nuestro proyecto y las vías para llevarlo a cabo.

15.1.2. Análisis de Redes Sociales.

Como inicio de nuestro proyecto el primer paso es buscar referentes y analizar la competencia. Investigar los antecedentes y determinar cual es nuestro hueco en la red.

15.1.3. Análisis de la estructura.

Aquí nos planteamos como ha de ser los cimientos de nuestra aplicación, que tecnologías usaremos y cual desarrollaremos nosotros mismos.

Decidimos que parte del proyecto es la que vamos a desarrollar, planteamos que parte del proyecto es generalizable y que parte es la específica.

15.1.4. Elección del modelo de Base de Datos

Es el momento de plantearnos si las bases de datos relacionales serán lo suficientemente potentes para nuestro proyecto.

Estudiaremos los diferentes motores de bases de datos fuera del ámbito relacional, los llamado noSQL.

15.1.5. Estudio de MongoDB.

Una vez elegido MongoDB como base de datos que utilizaremos en nuestro proyecto procederemos a un estudio en profundidad sobre dicha tecnología

15.1.6. Desarrollo de Nuts.

Una vez planteada la parte genérica que desarrollaremos procedemos a encapsularla dentro del framework que hemos llamado Nuts.

15.1.7. Elección del sistema de plantillas.

En este punto nos planteamos que sistéma de plantillas vamos a elegir y estudiaremos su sintaxis.

15.1.8. Primera aplicación.

Desarrollado ya el *framework* y elegido el sistema de plantillas y el motor de base de datos es el momento de desarrollar una primera aplicación que nos asegure que todo funciona.

15.1.9. Pruebas de la primera aplicación.

Es el momento de probar que todo ha ido bien, y verificar si funciona todo como esperabamos, y si podemos dar por concluido el desarrollo de nuestro framework.

15.1.10. Diseño de la aplicacón final.

Una vez tenemos claro y fijos los componentes nos toca ponernos a estudiar los detalles de nuestra aplicación, la experiencia de usuario y el diseño gráfico de nuestra web.

15.1.11. Aplicación final.

LLegados a este punto solo nos queda desarrollar nuestra aplicación sobre todo lo estudiado.

15.1.12. Pruebas de la aplicación.

Es el momento de verificar que todo esta correcto, y probar que todo va como lo propusimos.

15.1.13. Conclusión.

Tras todo el proceso y llegados a la versión final, estable y funcional, analizaremos si se han cumplido los objetivos planteados, y se han cubierto los requisitos iniciales. Analizaremos la evolución temporal del mismo y los costes de desarrollo.

15.1.14. Documentación.

Recopilamos toda la información sobre el proyecto y la estructuraremos y plasmaremos en el presente documento.

15.1.15. Orientación del tutor.

Incluye la presentación inicial del proyecto al tutor, así como los distintos encuentros mantenidos a lo largo de su desarrollo para orientar la consecución del mismo y lograr los mejores resultados posibles, estructuración del trabajo y documentación.

15.2– Análisis temporal.

En el siguiente cuadro ilustraremos la distribución de días que hemos seguido a lo largo del desarrollo del proyecto.

	Nombre	Duración	Inicio
1	Planteamiento inicial	7 días	27/2/2011
2	Analisis de Redes Sociales	7 días	9/3/2011
3	Análisis de la estructura	4 días	18/3/2011
4	Elección del modelo de Base de Datos	5 días	22/3/2011
5	Estudio de MongoDB	30 días	22/4/2011
6	Desarrollo de Nuts	12 días	22/5/2011
7	Elección del sistema de plantillas	5 días	4/5/2011
8	Primera aplicación	2 días	10/5/2011
9	Pruebas de la primera aplicación	1 días	10/5/2011
10	Diseño de la aplicacón final	45 días	11/5/2011
11	Aplicación final	150 días	1/7/2011
12	Pruebas de la aplicación	60 días	1/2/2012
13	Conclusión	5 días	10/8/2012
14	Documentación	41 días	12/7/2012
15	Orientación del tutor	300 días	7/10/2011

Cuadro 15.1: Exposición de tareas

15.3– Costes de desarrollo.

El presupuesto de esta aplicación lo hemos intentado ajustar al máximo desde el principio, sin que por ello tenga que verse mermada la calidad del producto final.

Para conseguir esto y siempre que se ha podido se ha buscado basarnos en tecnologías que estén liberadas bajo licencias *libres*. Estos productos en su mayoría proporcionan software gratuito de un alto rendimiento.

Además nuestra aplicación está desarrollada para que no suponga un coste de mantenimiento al administrador, simplemente hay que instalar la aplicación y funcionará por si sola.

En cuanto a los gastos inicales relacionados con el servidor hemos podido ahorrarnoslo gracias a AppFog, que nos ofrece hasta 2GB de RAM para nuestra aplicación y hasta 1 GB para la base de datos, lo que será su-

ficiente si para aplicación.

En caso de que crecieramos por encima de esos límites, este sería el único coste que inevitablemente se vería incrementado, pero este escollo es imposible de solventarlo si queremos crecer por encima de eso.

Las estamaciones basadas en ejemplos reales son las siguientes, 18KB por cada 7 Recetas y 4KB por cada 14 usuarios. Dejando un margen para el resto de datos que a priori no crecerían tanto como lo que podrían crecer estos podemos obtener una cota de lo que nos podemos permitir gratuitamente en AppFog.

Primero calcularemos que porcentaje ocupará cada uno de estas dos colecciones de la capacidad total:

$$Porcentaje\ ocupado\ por\ las\ recetas = \frac{18KB}{18KB + 4KB} = ,82 \qquad (15.1)$$

$$Porcentaje\ ocupado\ por\ los\ usuarios = \frac{4KB}{18KB + 4KB} = ,18 \qquad (15.2)$$

Suponiendo que dejamos libres 124MB para el resto de los datos y margen de error, nos quedarían 900MB para repartirnos entre recetas y usuarios, lo que nos da un total de:

$$MBs\ ocupado\ por\ las\ recetas = 900MB \times ,82 = 738MB \qquad (15.3)$$

$$MBs\ ocupado\ por\ los\ usuarios = 900MB \times ,18 = 162MB \qquad (15.4)$$

Lo que representaría:

$$\begin{aligned} Capacidad\ de\ Usuarios &= 162MB \times \frac{14Usuarios}{4KB \times \frac{1MB}{1024KB}} \\ &= 580.608\ Usuarios \\ &\approx \frac{1}{2}\ Mill.\ de\ Usuarios \end{aligned} \tag{15.5}$$

$$\begin{aligned} Capacidad\ de\ Recetas &= 738MB \times \frac{7Recetas}{18KB \times \frac{1MB}{1024KB}} \\ &= 293.888\ Usuarios \\ &\approx \frac{1}{4}\ Mill.\ de\ Recetas \end{aligned} \tag{15.6}$$

Estimamos que esta cota es lo suficientemente alta como para que si ocurriese nos plantearamos buscar inversores, u otro modo de que el proyecto diera beneficios y no perdidas.

16 Pruebas.

16.1– Fase de pruebas

Podemos distinguir dos fases de pruebas en nuestra aplicación.

La primera fase ha sido la de pruebas privadas, donde hemos probado intensivamente la aplicación a lo largo de todo su desarrollo.

Esta fase de prueba se realizaba mediante los scripts de instalación y desinstalación que realizaban las funciones de test unitarios. Estos scripts se encargan de llamar a los diferentes métodos de los módulos principales de la aplicación.

De este modo un diagnóstico temprano de los errores nos permite solventarlos sin demora. Lo que agiliza el desarrollo y las largas sesiones de *debug*.

La segunda fase de pruebas ha sido la de pruebas públicas, en esta fase la aplicación ya estaba terminada y era completamente funcional.

Esta fase de prueba se ha realizado con un grupo de usuarios reales, ya que se encontraba abierta al público. Que nos han ayudado a refinar la aplicación, solventar errores menores y mejorar la esperiencia de usuario.

16.2– Exploradores

Hemos probado nuestra aplicación en las últimas versiones de Chrome, Firefox e Internet Explorer. Los resultados han sido los siguientes:

Figura 16.1: Vista en Chrome.

Figura 16.2: Vista en Firefox.

Figura 16.3: Vista en Internet Explorer.

Como podemos comprobar hemo tenido unos resultados satisfactorios para todas los navegadores. Esto no quiere decir que no haya diferencias visuales, ya que podemos comprobar que Internet Explorer no *renderiza* adecuadamente bordes redondeados ni degradados. Esto no se ha tomado como un error debido a que son adornos visuales y no intervienen decisivamente en la interacción con el usuario, lo que consideramos importante es que no haya una distribución de capas erroneas y todos los elementos sean visibles y legibles.

Esta decisión si bien antes no era la que se tomaba, hoy en día es la que se prefiere pues los llamados *hacks* se consideran anti-productivos y ademas conllevan una recarga del sistema, y aumentan el volumen de la transmisión.

16.3– Acessos múltiples.

Como prueba de resistencia de nuestra aplicación hemos realizado pruebas de acceso simultaneo por varios usuarios.

Estas pruebas han ofrecido un resultado positivo, ya que el sistema no ha tenido ningún problema en responder a las solicitudes realizadas.

16.4– Reconocimiento semántico.

Este es uno de los objetivos principales de la aplicación, que los buscadores reconozcan nuestras recetas como tal, y que reconozca las diferentes secciones presentes en ella. Para ello nos remitiremos a la herramienta de Google, Rich Snippets Testing Tool[1].

El resultado de una receta *tipo* de nuestra aplicación es el siguiente:

[1] **Rich Snippets Testing Tool**: http://www.google.com/webmasters/tools/richsnippets

Figura 16.4: Vista de Rich Snippets Testing Tool.

Podemos observar que Google reconoce el formato *hrecipe* y que todos los campos se han reconocido correctamente. Además podemos observar una previsualización de como se mostraría nuestra receta en el buscador de Google.

17 Conclusiones.

17.1– Objetivos Secundarios.

17.1.1. Creación de una plataforma social.

Este objetivo lo damos por satisfecho, hemos creado un sistema de usuarios completo, con la posibilidad de las debidas interacciones sociales entre ellos.

17.1.2. Implementación de un estándar semántico.

Como hemos visto en el capítulo de pruebas en la sección 16.4 este objetivo se ha cumplido.

17.1.3. Estructuras de datos para la representación semántica

Podemos ver en el capítulo 8 en la sección 8.2 cual ha sido esta estructura, que evidentemente por el punto anterior ha cumplido su cometido.

17.1.4. Modo intuitivo de introdución de datos.

Éste quizá sea el objetivo más complicado de medir, pero gracias a nuestro grupo de usuarios de prueba podemos decir que no ha habido ningún problema y por lo tanto lo damos por cumplido.

17.1.5. Buscador de recetas.

Hemos creado un buscador que funciona conforme a las especificaciones presentadas, así que podemos dar este objetivo por satisfecho también.

17.2– Objetivo Principal.

Dada la consecución de los objetivos secundarios y la integración de los diferentes componentes entre sí, el objetivo princpal está satisfecho completamente.

La aplicación es completamente funcional, y esta disponible para ponerla en *producción*, como ocurre ya con http://www.ereselchef.com.

18 Desarrollos futuros y ampliaciones.

En este capítulo abordaremos las posibilidades de ampliaciones futuras que presenta este proyecto.

18.1– Ajustes.

Crear una pantalla de ajustes con mas posibilidades, actualmente solo existe la de eliminar cuenta en esta sección.

El usuario, por ejemplo, podría seleccionar sus preferencias sobre qué tipo de notificaciones desea recibir y cuales no.

18.2– Guardado rápido de recetas de internet.

Una posibilidad interesante sería la posibilidad de importar otras recetas te internet que esten bajo algun estandar semántico, o mediante interacción con el usuario y heurística.

Esto ayudaría a los usuarios a mantener su colección de recetas dentro de la aplicación, de la manera mas cómoda, además ayudaría a los otros usuarios seguidores a conocer las recetas recolectadas por los usuarios seguidos.

18.3– Login con diferentes redes sociales.

Esta posiblidad da un paso mas allá en la interacción con otras redes sociales, mediante el protocolo OAuth podemos permitir la autenticación de los usuarios sin necesidad del registro tradicional.

Si se aplica esta mejora, daría la posibilidad de nuevas facilidades al usuario, se podría compartir las recetas o las notificaciones con esa red social de manera transparente para el usuario. También, gracias a esta interacción podría buscar a los que son sus *amigos* en las otras redes.

18.4– Permitir pedido de recetas.

A veces no es suficiente con lo que encontramos cuando buscamos una receta, y deseamos algo que tenga ciertos ingredientes, o recetas para una ocasión especial y no las encontramos, esto se podría solventar mediante un formulario de pedido de nuevas recetas.

Estos pedidos podrían ser atendidos por otros usuarios de la red, especialmente por sus seguidores. O incluso se podría crear un servicio de chefs que atiendan las demandas de los usuarios, este servicio podría ser gratuíto o de pago.

18.5– Enciclopedia de ingredientes.

La aplicación podría contar con una enclicopedia de ingredientes que detallara información sobre estos. Esta información podría encontrarse desde una mera descripción hasta sus calorías, proteínas, lugar de procedencia, o estacionalidad si la tuviese.

Esto podría ayudar a identificar las recetas más o menos calóricas sin necesidad de ser especificado por el usuario. Además podría contar con ingredientes sustitutivos, o fotos para el reconocimiento de ingredientes.

18.6– Creador de menús.

La herramienta de creación de menú permitiría hacer una lista con una serie de platos, y poder compartirla con el resto de los usuarios. Estos platos podrían ser para una única comida, o alargarse hasta una semana.

La posibilidad de incluir cada plato en un contexto permite que creemos una lista de platos que combinen bien para una fiesta temática, o un menú para una cena completa. Además los usuarios tendrían la posiblidad de tener menús semanales, ya sea de ellos o de otros usuarios, con los que podrían agilizar la lista de la compra o hacer estudíos de lo equilibrada que es una dieta.

18.7– Sugerencias/Tips/Consejos.

Hay ciertos trucos que no se ajustan a un plato en concreto, sino que pueden servir para varios, la posibilidad de incluirlos como una identidad le daría la posibilidad de manejo por parte de la aplicación.

El lector podría ver todos los consejos asociados a un ingrediente, aunque el redactor de la receta no lo haya incluido explicitamente. El redactor por su parte no necesita incluir una lista de consejos cada vez que redacta una receta, siendo un engorro si desea dar esa posiblidad.

18.8– Enlazar recetas y restaurantes.

La posibilidad de enlazar recetas de restaurante puede dar un plus más de información a nuestra receta. Quizá alguien haya escríto una receta inspirada en aquel plato que comiste en un restaurante remoto al que fuiste en tu último viaje. O el propio restaurante decide que una buena forma de darse a conocer es enseñarnos qué y cómo lo cocinan.

Y, seamos sinceros, a veces lo que más nos apetece es que nos lo prepare otro. Con esta posibilidad lo tenemos muy sencillo, podemos elegir un restaurante por lo que hemos decidido comer y no una comida por el restaurante al que hemos decidido ir.

18.9– Recetas y chefs destacados.

Estas secciones pueden contener las recetas que decidamos que sean mas importante, se puede realizar una selección por número de interacciones, como visitas, comentarios o favoritos.

Estas secciones pueden suponer una recompensa por su labor a los usuarios destacados. Además esta secciones significan para el usuario una referencia de calidad de nuestro sitio, un ejemplo que puede imitar al escribir sus recetas.

18.10– API/App móvil.

La implementación de una API supone que no sólo nosotros podemos interactuar con otros servicios sino que otros servicios puedan interactuar con nosotros. Habitualmente asociado a una API hoy en día viene asociada la correspondiente aplicación o aplicaciones móviles.

Las aplicaciones móviles proporcionan mayor comodidad para el usuario a la hora de interactuar con la aplicación, y además reducen el tráfico, ya que solo hay que enviar y recibir los datos necesarios y no las estructuras que conforman la web.

18.11– Modelo de negocio.

Por último como ya hemos visto en el capítulo anterior a partir de cierto número de usuarios la web necesitaria un modelo de negocio para poder mantenerse activa sin que esto provoque un gasto para el administrador.

Este modelo de negocio puede estar basado básicamente en dos pilares:

Publicidad: La inclusión de publicidad puede darse tanto en la web como en la aplicación móvil si la hubiera, dependiendo del tipo de publicidad y la cantidad que estemos dispuestos a poner, esta publicidad puede simplemente ayudarnos a pagar los pagos, o ganar realmente dinero con la aplicación. Pero tiene un peligro, la inclusión de demasiada publicidad en una fase temprana podría hacer huir al usuario, con lo que no se recaudaría nada a largo plazo.

Pagos: Los pagos por servicios premiums serían tambien aplicables. Un ejemplo de oferta podría ser los perfíles de restaurantes o las páginas de chef profesionales.

19 Manual de usuario.

19.1– Registro de usuario.

- Accedemos a la aplicación[1]

Figura 19.1: Inicio de la aplicación.

- Entramos en la sección de registro, en la parte superior, y rellenamos los campos adecuadamente.

[1]: http://www.ereselchef.com

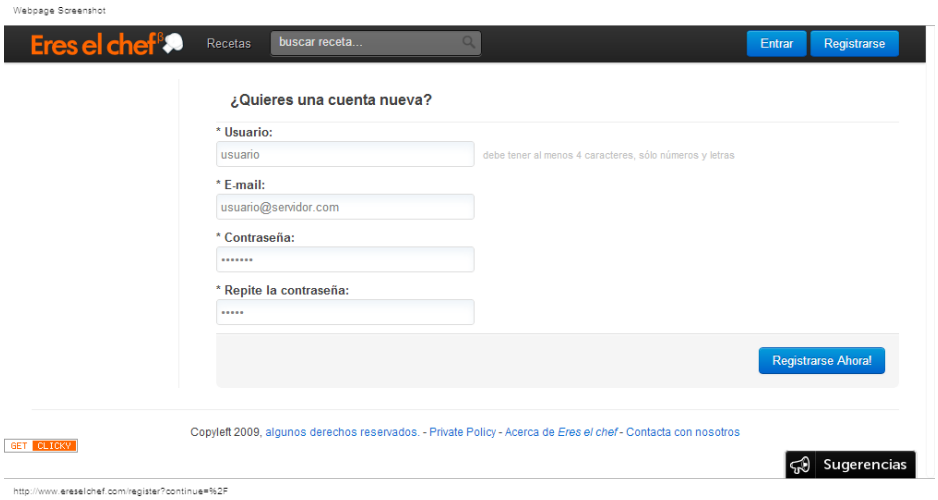

Figura 19.2: Registro de la aplicación.

- Si esto no los hemos rellenado adecuadamente no te preocupes, el sistema te informará.

Figura 19.3: Error en el registro de la aplicación.

- Automáticamente habremos iniciado la sesión (si todo se llevo a cabo correctamente) con el usuario que acabamos de registrar.

Figura 19.4: Error en el registro de la aplicación.

19.2– Autenticación.

- Accedemos a la aplicación.
- Entramos en la sección de inicio de sesión accionando el botón entrar y rellenamos los campos presentados.

Figura 19.5: Error en el registro de la aplicación.

- Si se ha realizado la operación con exito se nos mostrará la pantalla principal de usuario.

19.3– Cambiar imagen de perfil.

- Nos dirigimos a la página de perfil de nuestro usuario, pulsando en nuestro nombre de usuario. Pulsamos sobre el botón `Cambiar foto de perfil` y nos aparecera el diálogo de selección de imagen.

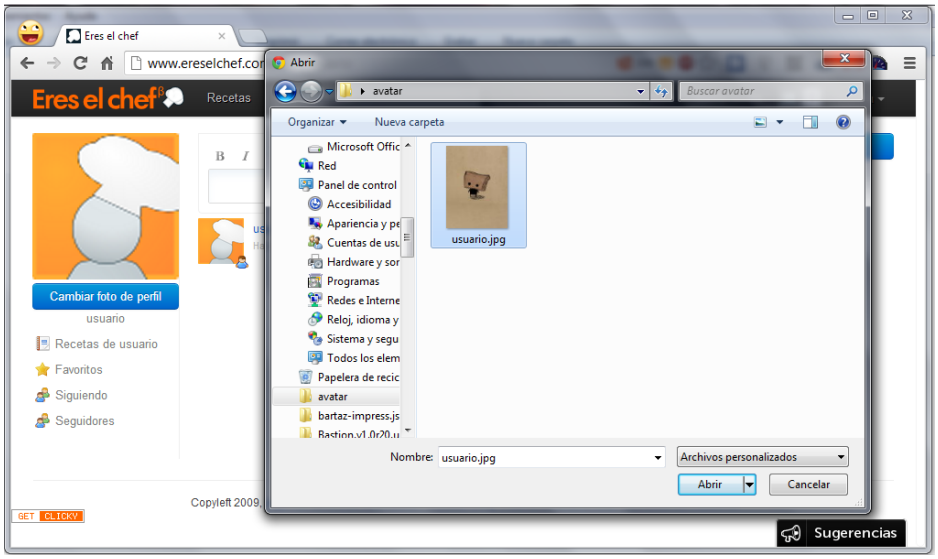

Figura 19.6: Selección de foto de perfil.

- El resto del proceso se llevara a cabo automáticamente, y se nos mostrará nuestra nueva foto de perfil.

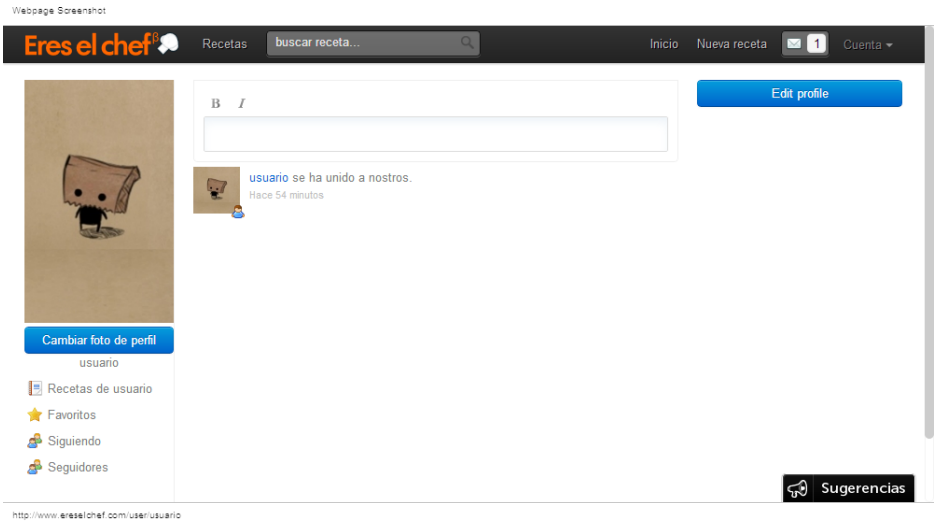

Figura 19.7: Nueva foto de perfil.

19.4– Buscando una receta.

- Nos dirigimos a la parte superior donde pone `Buscar receta...` e introducimos los terminos deseados.

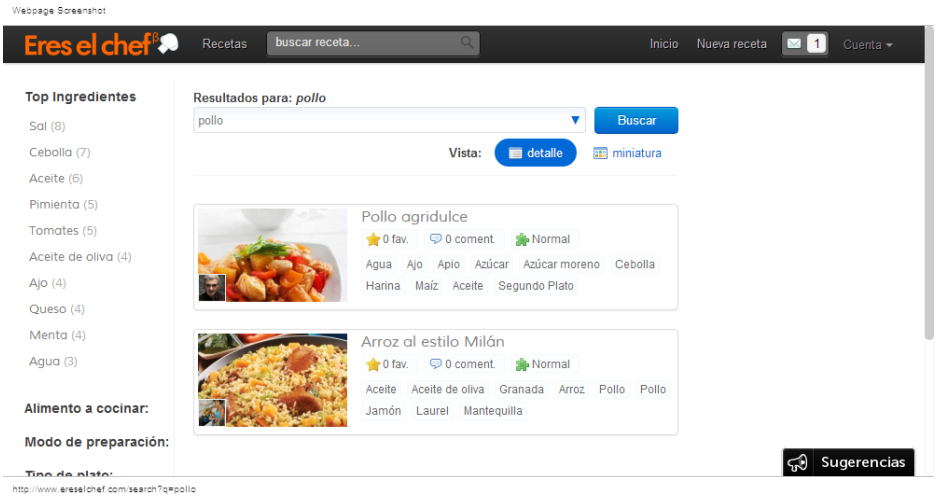

Figura 19.8: Búsqueda: Pollo.

- Podemos realizar, búsquedas más complejas, utilizando diferentes comandos.

Expresión	Descripción
`ingrediente:(`<*nombre*>`)`	Busca una receta **con** un ingrediente determinado, los paréntesis se pueden omitir si el ingrediente sólo contiene una palabra.
`-ingrediente:(`<*nombre*>`)`	Busca una receta **sin** un ingrediente determinado, los paréntesis se pueden omitir si el ingrediente sólo contiene una palabra.
`etiqueta:(`<*nombre*>`)`	Busca una receta con una etiqueta determinada, los paréntesis se pueden omitir si la etiqueta sólo contiene una palabra.

Cuadro 19.1: Opciones avanzadas de búsqueda.

O sus atajos:

Atajo	Expresion equivalente
`i:(`<*nombre*>`)`	`ingrediente:(`<*nombre*>`)`
`-i:(`<*nombre*>`)`	`-ingrediente:(`<*nombre*>`)`
`e:(`<*nombre*>`)`	`etiqueta:(`<*nombre*>`)`

Cuadro 19.2: Atajos de las opciones avanzadas de búsqueda.

Aunque las etiquetas no estan restrigidas, ofrecemos una serie de sugerencias que te pueden resultar útil. Estas sugerencias las podrás ver más tarde en el Cuadro 19.3.

- Si no deseamos utilizar los *comandos* manualmente, podemos usar el formulario para la búsqueda avanzada y rellenando los campos conforme a sus etiquetas. Para desplegarlo pulsaremos sobre la flecha triangular en el campo de texto de la pantalla de búsqueda.

- Tenemos además una vista alternativa para los resultados de las búquedas centrada en la fotografía del plato.

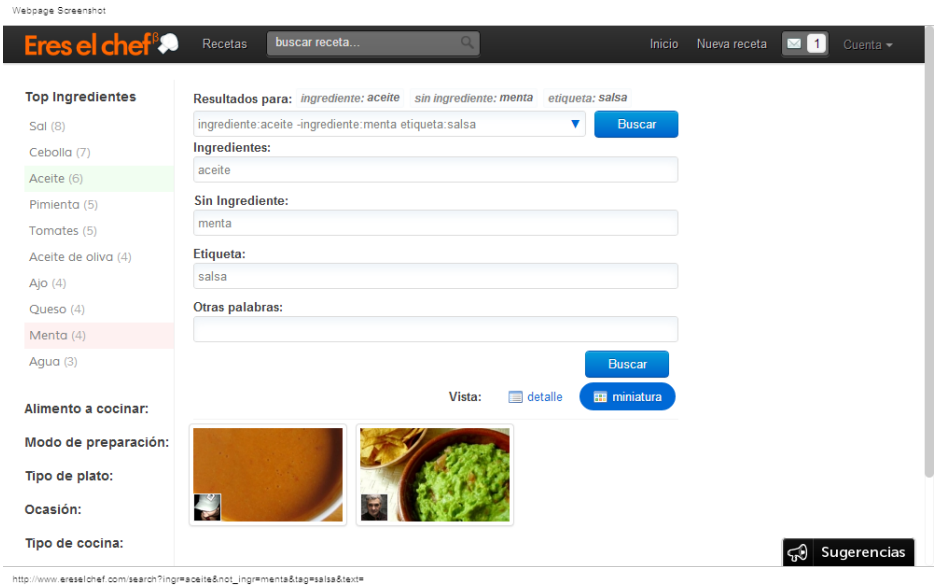

Figura 19.9: Búsqueda: i:aceite -i:menta e:salsa.

19.5– Siguiendo Usuarios.

- Entra en el perfil de otro usuario pulsando en su nombre, o entra en una receta del usuario al que queremos seguir.

- Pulsa sobre la etiqueta `Seguir` situada en el menu izquierdo de la pantalla.

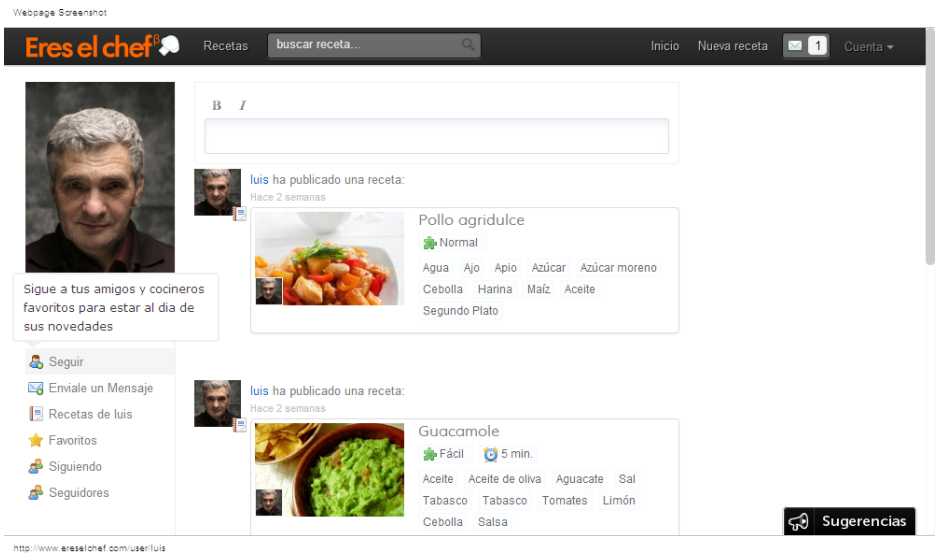

Figura 19.10: Seguir.

- La etiqueta cambirá su texto a `Dejar de seguir`, asi que si queremos cancelar la acción en cualquier momento sólo tendremos que volver y pulsar de nuevo.

19.6– Enviar Mensajes

- Vamos a enviarle un mensaje a otro usuario, para ello situados en el perfil del usuario utilizaremos la opción `Enviale un mensaje`. Si es la primera vez que nos comunicamos en este usuario nos encontraremos una pantalla como la siguiente.

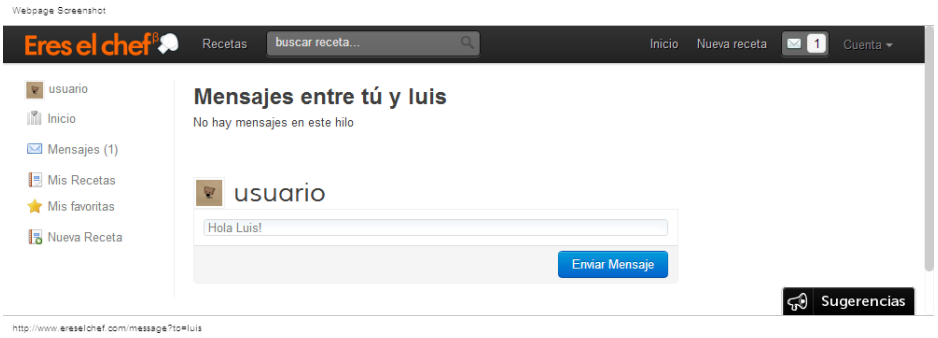

Figura 19.11: Enviando un mensaje.

19.7– Leer Mensajes.

- Como vemos en la parte superior derecha, nada mas registrarnos ya tendremos un mensaje esperandonos. Haremos click sobre el sobre para verlo.

- Nos encontraremos con todas las conversaciones, y solo tendremos que hacer click sobre el mensaje destacado.

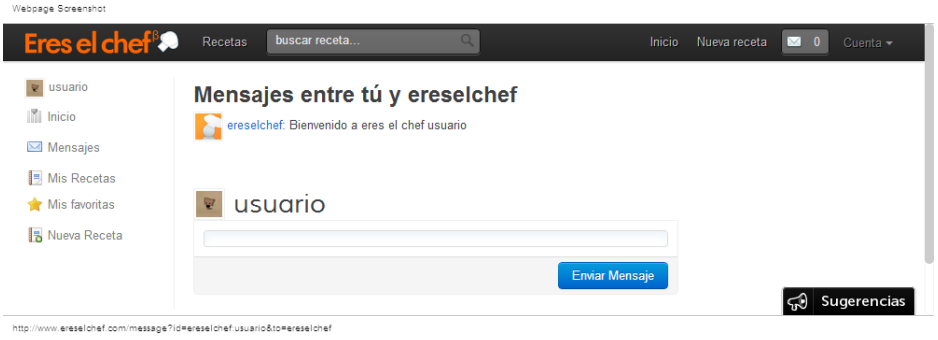

Figura 19.12: Leyendo un mensaje.

19.8– Añadir una actulización de estado.

- Nos dirigimos a la pantalla principal, por ejemplo pulsando sobre el logo.

- En la parte superior nos encontramos un campo de texto sobre el cual haremos foco para que aparezca el botón de compartir.

- Además, tenemos la posiblidad de añadir negritas y cursivas a nuestro mensaje, para ello utilizaremos los controles superiores al cuadro de texto.

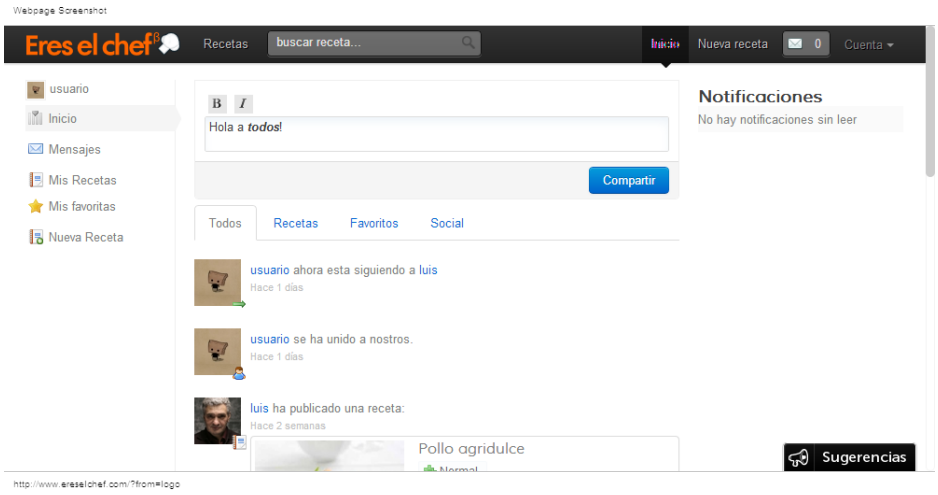

Figura 19.13: Añadiendo una actualización de estado.

- por ultimo, solo nos queda accionar el botón de compartir.

Figura 19.14: Estado actualizado.

19.9– Añadir un comentario en un estado.

- Bajo cada comentario podemos ver un campo de texto para introducir comentarios. Sólo tendremos que introducirlo y se enviarán con la tecla `intro`.

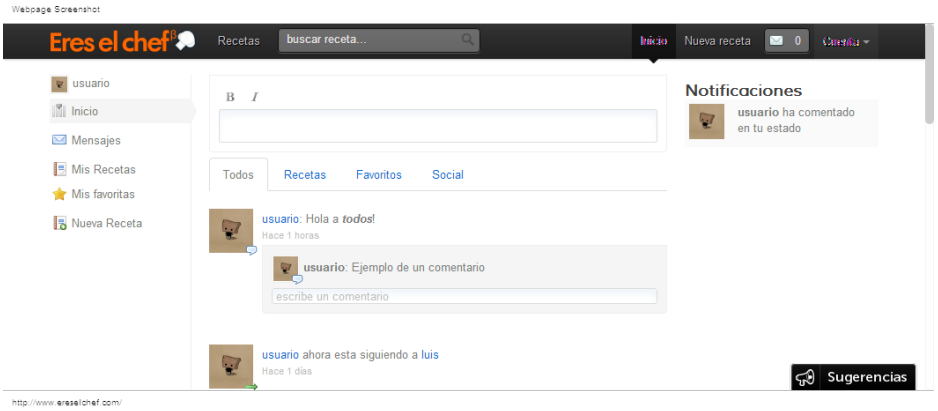

Figura 19.15: Comentario en una actualización.

19.10– Actualizar datos del perfil.

- Nos dirigimos hacia nuestro perfil de usuario. Y seleccionamos `Editar perfil` para que se nos despliguen los campos de datos.

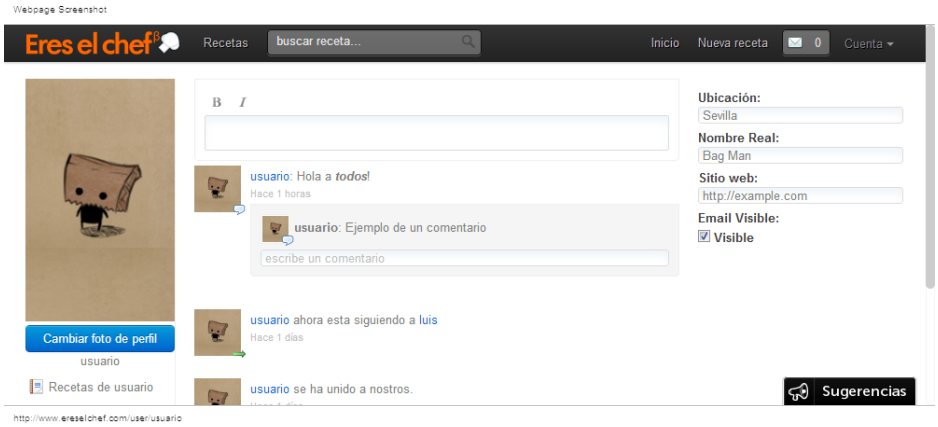

Figura 19.16: Editando el perfil de usuario.

- La actualización se lleva a cabo en segundo plano sin necesidad de confirmar, asi que cuando volvamos a entrar encontraremos los datos actualizados.

Figura 19.17: Nuevos datos del perfil.

19.11– Nueva receta.

- Pulsaremos sobre la parte superior en `Nueva Receta` para acceder al formulario de creación de recetas.

- Nos encontraremos con el formulario simplificado que se muestra mas abajo.

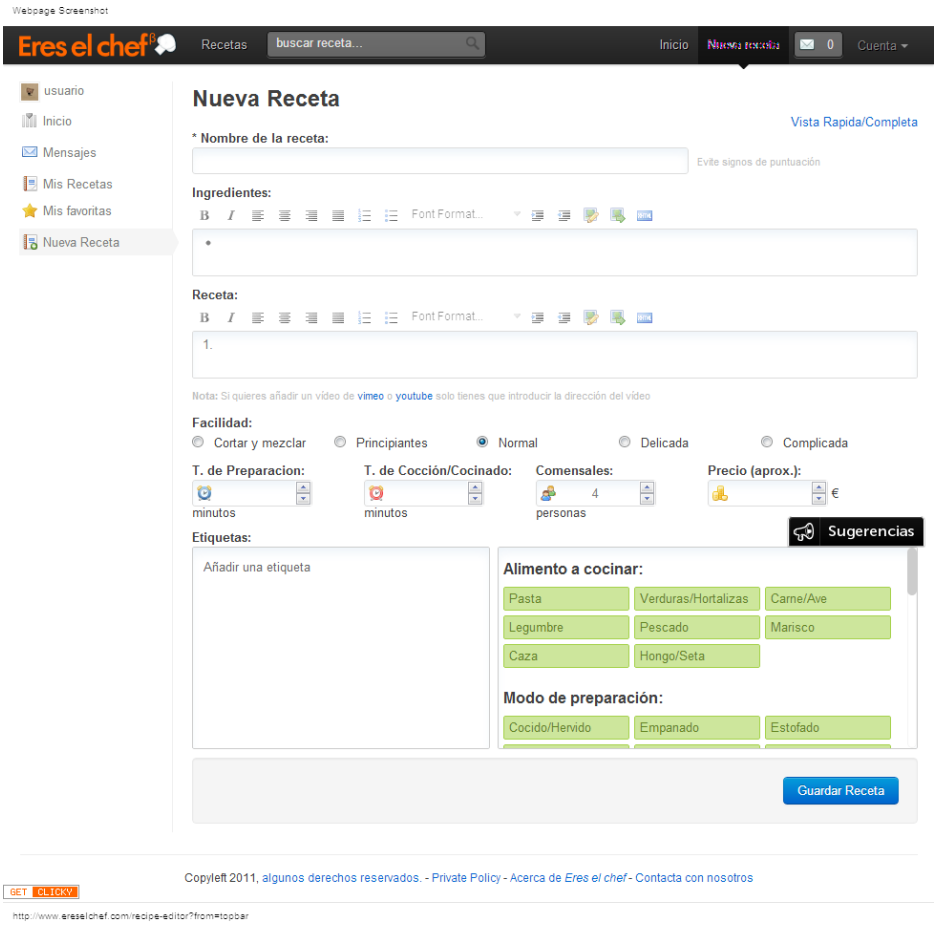

Figura 19.18: Nueva receta.

- Si queremos podemos desplegar el formulario más completo pulsando sobre `Vista Rápida/Completa`, se convertirá en el siguiente formulario.

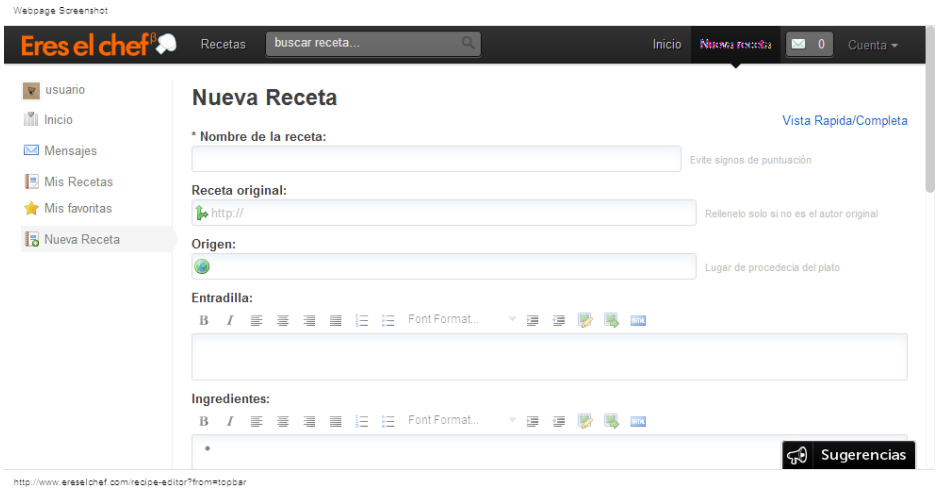

Figura 19.19: Nueva receta extendido.

- Rellenaremos los campos deseados de nuestra nueva receta.

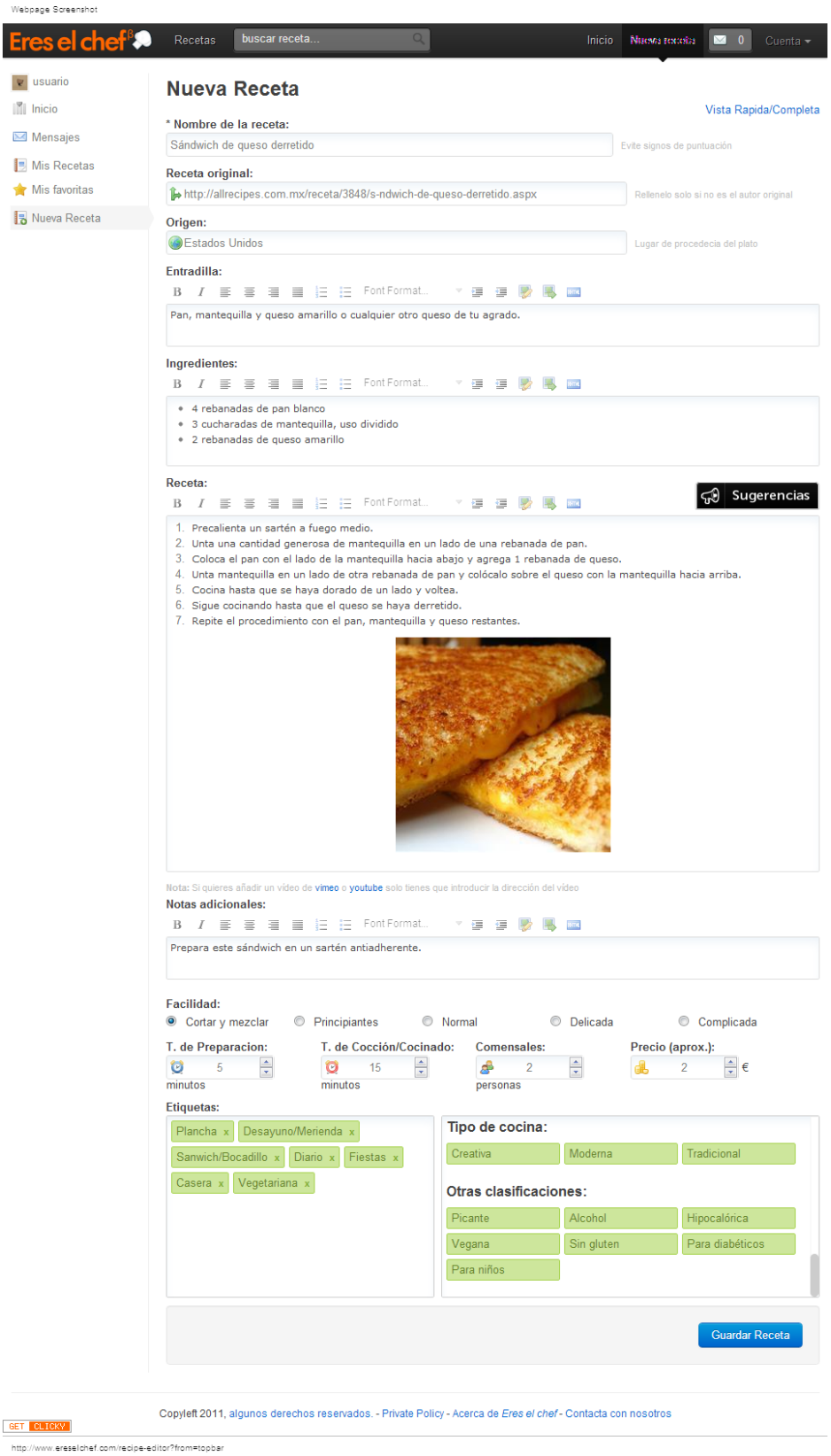

Figura 19.20: Nueva receta rellenada.

- Podemos observar que para las etiquetas tenemos las siguientes sugerencias.

Modo de preparación
Cocido/Hervido
Empanado
Estofado
Frito
Guiso
Gratinado
Horno/Asado
Microondas
Olla a presión
Parrilla/Brasa
Plancha
Rebozado
Thermomix
Vapor
Crudo
Salteado
Macerado
Enfriado
Congelado
Baño maría
Tipo de plato
Bebida/Cocktail
Aperitivo/Entrante
Ensalada
Salsa
Sopa Fría
Tarta
Mermelada/Conserva
Helado
Sanwich/Bocadillo
Sopa Crema
Dulce
Pan/Bollería
Tapa
Postre
Desayuno/Merienda
Plato Principal

Ocasión
Diario
Cumpleaños
Navidad
Semana Santa
Halloween
San Valentín
Todos los santos
Fiestas
Picnic
Invierno
Primavera
Otoño
Verano
Otras clasificaciones
Picante
Alcohol
Hipocalórica
Vegetariana
Vegana
Sin gluten
Para diabéticos
Para niños
Alimento a cocinar
Pasta
Verduras/Hortalizas
Carne/Ave
Legumbre
Pescado
Marisco
Caza
Hongo/Seta
Tipo de cocina
Creativa
Moderna
Tradicional
Casera

Cuadro 19.3: Sugerencias para etiquetas

- Una vez enviada nuestra receta visualizaremos su resultado

Figura 19.21: Vista recetas.

19.12– Añadir una receta a favoritos y compartirla.

- Accedemos a la receta que deseamos agregar.
- Si deseamos agregarla a favoritos pulsamos en la parte superior derecha en `Guardar como favorito`.
- Si deseamos compartirla en Twitter, Google+ o Facebook, pulsaremos sobre los botones identificados para tal efecto también en la parte superior derecha.
- También podemos compartirla a través de los codigos QR.

19.13– Añadir una alternativa.

- Accedemos a la receta que queramos versionar.

- Arriba a la derecha se nos presenta la opción `Crear alternativa` a la que debemos acceder.
- Se nos presentará el formulario de edición con los campos rellenos como la receta a versionar. Además el campo `Receta original` tiene un enlace hacia la receta original.

19.14– De vuelta la página de inicio.

- Ya sabemos como funciona la aplicación y seguimos a otros, es hora de volver a la página de inico. Para ello haremos click sobre el logo de la web.

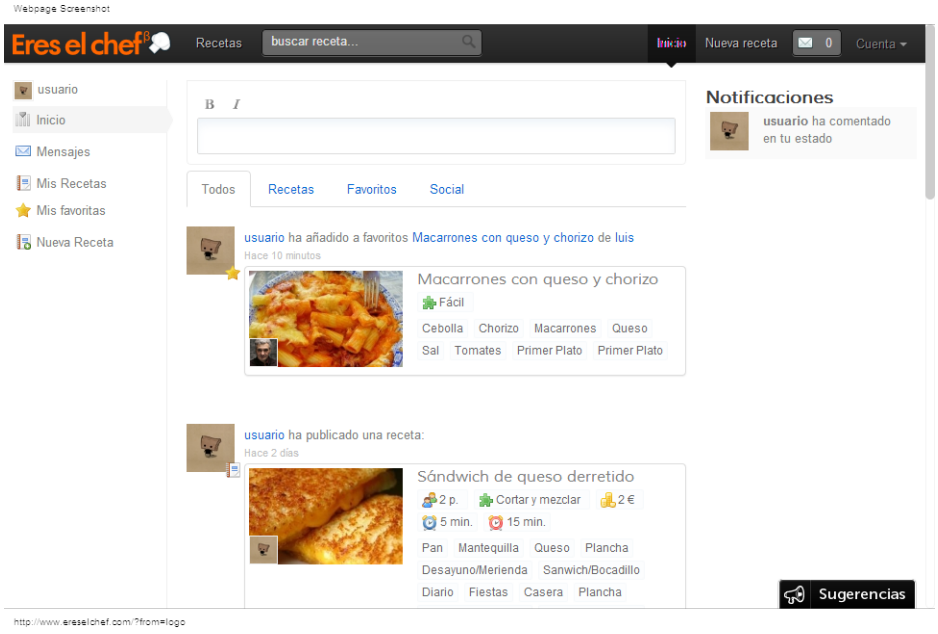

Figura 19.22: Página principal.

- Si nos fijasmos tenemos diferentes pestañas, que conforman diferentes filtros para nuestras noticias.
  - La primera pestaña `Todos` no hace filtraje ninguno.
  - La segunda pestaña `Recetas` mostrará todas las recetas publicada por nosotros y nuestras subscripciones.

  - La tereceta pestaña `Favvoritos` mostrará todas las recetas que han sido agregadas a favoritos por nosotros o nuestras subscripciones.
  - La cuarta pestaña `social` mostrará la fecha de registro de aquellos a los que seguimos, las actualizaciones referentes a *quién esta suscrito a quién* y las actualizaciones de estado.

- Arriba a la derecha encontraremos las notificaciones que pueden ser eliminadas visitandolas o en la cruz que aparecerá cuando se posa el puntero sobre él.

Bibliografía

[Goo12] Google. http://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml?showone=Document_type#Document_type, Abril 2012.

[Mon11] MongoDB. http://www.mongodb.org/display/docs/philosophy, Diciembre 2011.

[Mon12] MongoDB. http://www.mongodb.org/display/docs/sql+to+mongo+mapping+chart, Febrero 2012.

[Rod12] Txema Rodriguez. http://www.genbetadev.com/frameworks/bootstrap, Junio 2012.

[W3C07] W3C. http://www.w3c.es/Prensa/2007/nota070911_grddl.html, Septiembre 2007.

[W3C12] W3C. http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica, Marzo 2012.

[Wik12] Wikipedia. http://es.wikipedia.org/wiki/Stemming, Mayo 2012.