Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño de un portal para una red social de recetas

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Guillem Matías Guillén

Tutores: Vicente Javier Julián Inglada,
Javier Palanca Cámara

2016-2017

Resumen

El presente proyecto consiste en el desarrollo y despliegue de una aplicación web de carácter social y dentro de un ámbito nutricional. El sistema deberá permitir la gestión de los usuarios, gestión de recetas, así como la interacción entre dichos usuarios mediante mensajes o mediante un sistema de seguidores/seguidos. La aplicación interactuará con una base de datos que cuenta con más de nueve mil recetas reales y con una lógica para la gestión de la red social que habrá que desarrollar.

Palabras clave: red social, recetas, MEAN, NodeJS, MongoDB.

Resum

Aquest projecte consiteix en el desenvolupament i llançament d'una aplicació web de caràcter social i dins d'un àmbit nutricional. El sistema deurà permetre la gestió dels usuaris, gestió de receptes, així com la interacció entre aquests usuaris per mitjà de missatges o per mitjà d'un sistema de seguidors/seguits. L'aplicació interactuarà amb una base de dades que compta amb més de nou mil receptes reals i amb una lógica per a la gestió de la xarxa social que caldrà crear.

Paraules clau: xarxa social, receptes, MEAN, NodeJS, MongoDB.

Abstract

This project involves the development and deployment of a social network within the nutritional scope. The system should allow the management of users and recipes, as well as the interaction between these users through messages or through a followers/following system. The application will interact with a database that has more than nine thousand real recipes and a logic for the management of the social network that must be developed.

Keywords: social network, recipes, MEAN, NodeJS, MongoDB.

Agradecimientos

A mis tutores Vicente Javier Julián Inglada, Javier Palanca Cámara y Stella María Heras Barberá por el apoyo prestado, sus consejos y su motivación a lo largo del desarrollo de este proyecto.

A mi madre y mi hermano por ser un pilar de apoyo muy importante y por motivarme cada día para siempre dar un poco más. A mi primo, Joan, por iniciarme en el mundo de la informática y por todos los consejos tanto a nivel profesional como académico. Para mí, un modelo a seguir.

A mis compañeros informáticos, entre ellos Danny, por ayudarme en estos cuatro años de retos y Manuel, por guiarme durante el largo camino que ha sido el aprendizaje de la tecnología que se emplea en este proyecto. A todos ellos por aguantarme.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1 Contexto y motivación

Desde el envío del primer *e-mail* en 1971, hasta el nacimiento de comunidades interactivas como MySpace (2003), Facebook (2004) y Twitter (2006) [1], las principales actividades que un usuario realiza en Internet han ido cambiando con el tiempo, pasando de simples búsquedas y navegación sobre información que ya estaba almacenada a interacciones directas con otros usuarios.

Estas comunidades, más conocidas como redes sociales, definidas mayoritariamente como sitios en la red cuya finalidad es permitir a los usuarios relacionarse, comunicarse, compartir contenido y crear comunidades, han ido evolucionando a lo largo de estos últimos años. Esta evolución no se ha dado solo en el ámbito tecnológico, sino también en cuanto a la cantidad de personas o entidades que las usan. Para entrar en contexto, en enero de 2016 la población mundial era de 7,4 mil millones de habitantes. De ese total, el número de usuarios registrados en redes sociales según un informe realizado por *We Are Social*, era de 2,3 mil millones [2]. Estos datos, junto a un informe que lanzó el portal *globalwebindex* en el que indican que los usuarios de Internet tienen una media de 5.54 cuentas en redes sociales [3], nos ayudan a visualizar lo extendido que está hoy en día el uso de estas plataformas.

Otro aspecto que ha ido evolucionando con estos portales es el uso que se les da. Las redes sociales nos ayudan tanto a estar en contacto con conocidos o familiares como a estar informados de lo que pasa a nuestro alrededor las 24 horas del día. Además, y debido a que estas comunidades son una fuente de información para terceros ya que en éstas indicamos nuestros gustos y dejamos constancia de nuestros intereses, las redes sociales se han convertido también en un punto de encuentro entre marcas y consumidores, como apuntan en el portal Web PuroMarketing [9]:

> *El 60% de los consumidores online sigue o interactúa con alguna marca a través de estos canales.*
> *Las redes sociales son asimismo una fuente útil de información para los marketers. El 91% las utiliza como herramienta de marketing de contenido.*
> *[…] A la hora de encontrar nuevos clientes, el 52% de las empresas afirma haber captado muchos de ellos a través de redes sociales*.

En cuanto a la tipología de las redes sociales, no existe un estándar o acuerdo a la hora de clasificarlas. La forma más común de dividir estos portales es la siguiente [4]:

- Horizontales: Aquellas redes sociales sin una temática definida y dirigidas a un público general. Portales como *Facebook* o *Twitter* forman parte de este grupo.
- Verticales: Tienen como objetivo agrupar, en torno a una temática concreta, a un colectivo específico. Pueden clasificarse a su vez en:
    - Profesionales: Se trata de redes sociales especializadas en establecer relaciones laborares. Se puede compartir información, experiencia, documentos…
      El ejemplo más conocido sería *LinkedIn*.
    - De ocio: Se trata de conectar a personas con unos gustos similares en alguna área como puede ser en la música, el deporte, los animales... Algunos ejemplos son *Dogster*, *Wipley*...
    - Mixtas: Tienen como objetivo unificar los subgrupos anteriores (profesional y de ocio). Un ejemplo es *Unience* o *Pinterest.*
- De contenido: En estas redes sociales las relaciones se establecen dependiendo del contenido publicado. En este grupo se encuentran redes sociales como *Instagram*, *Flickr* o *Youtube*.

De todas las tipologías anteriormente descritas, este trabajo estaría incluido en el grupo de las redes sociales verticales de ocio, ya que presentaría una temática concreta. Como se ha indicado en el apartado *Resumen*, el ámbito que nos ocupa sería el de la nutrición, dietética y cocina.

En este contexto, el grupo del **Departamento de Sistemas Informáticos y Computación**, **GTI-IA** (Grupo de Tecnología Informática Inteligencia Artificial), lleva años trabajando en sistemas inteligentes que ayuden a la predicción de alérgenos y recomendación de dietas y recetas basándose en la información de carácter nutricional de los usuarios, así como de sus gustos culinarios. Para la realización de este proyecto es necesario interactuar con usuarios reales que aporten su propia experiencia y valoren los resultados de dicha investigación. De la necesidad de disponer de este tipo de información para poder llevar a cabo el trabajo de este grupo de investigación surge el desarrollo de una plataforma social atractiva y sencilla para los usuarios (*Receteame*) que les permita compartir sus recetas y gustos y que, además, sirva como base para proporcionarles recomendaciones ajustadas a sus intereses y teniendo en cuenta cualquier tipo de intolerancia alimentaria.

1.2 Objetivos

EL objetivo principal del proyecto es el diseño, implementación y despliegue de un portal web para una red social de recetas. Para llevar a cabo dicho objetivo, lo dividiremos en los objetivos secundarios siguientes:

- Adaptar y ampliar la base de datos previamente creada por el grupo de investigación **GTI-IA** para poder mantener un registro no solo de las recetas e ingredientes, sino también de usuarios, mensajes y cualquier tipo de notificación.
- Desarrollar un servidor que consistirá en una API RESTful escrita en Javascript y que permitirá la gestión de peticiones que realicen los usuarios.
- Diseño de interfaces amigables y adaptables a cualquier dispositivo móvil (diseño *responsive*).
- Permitir al usuario la creación y consulta de recetas.
- Creación de herramientas de interacción social entre usuarios basadas en el envío y la recepción de mensajes, así como de relaciones de seguidor/seguido entre ellos.
- Proporcionar la posibilidad de crear una versión de una receta anteriormente creada, partiendo de la información de ésta, pero con la capacidad para modificar cualquier aspecto.
- Demostrar la capacidad del autor para desarrollar un proyecto de desarrollo web completo, a partir de los conocimientos adquiridos a lo largo de la titulación y de su capacidad de autoaprendizaje.
- Realizar el despliegue de la plataforma en el dominio receteame.com

2. Estado del arte

Antes de empezar con la especificación de requisitos y la descripción más detallada de los aspectos técnicos del proyecto que nos ocupa, se pretende realizar un análisis de alternativas dentro del ámbito nutricional en las redes sociales además de realizar una justificación de la tecnología que se empleará en la implementación del mismo.

2.1 Aplicaciones similares

En el mundo de las redes sociales, no es muy común encontrar comunidades centradas únicamente en temáticas nutricionales ya que no son tan populares como pueden ser portales como *Facebook*, *Instagram* o *Youtube*, que cuentan con muchos usuarios alrededor del mundo.

Para esta sección, se han seleccionado un conjunto de plataformas que implementan y ofrecen funcionalidades similares a las que se plantean en este proyecto.

COCINARIO

Cocinario es una comunidad interactiva española y gratuita. En ella, cualquier usuario puede visualizar las recetas que se publican, pero sólo los usuarios registrados pueden acceder a funcionalidades sociales y crear recetas. Presenta un área privada en la que un usuario registrado puede mantener una gestión de su información, así como de usuarios a los que sigue o le siguen. También nos ofrece la posibilidad de entrar a través de nuestra cuenta de *Facebook*.

Figura 1. Captura de pantalla de la página principal de Cocinario

ALLRECIPES

Otra opción sería la plataforma *Allrecipes*. En esta comunidad, el contenido gira en torno al usuario quienes participan activamente mediante perfiles que expresan sus historias, triunfos y fracasos culinarios, fotografías, preferencias, comentarios, calificación de recetas, hobbies y tradiciones. El usuario generará una red de amigos con intereses en común dentro de la misma comunidad, compartiendo así recetas y vivencias.

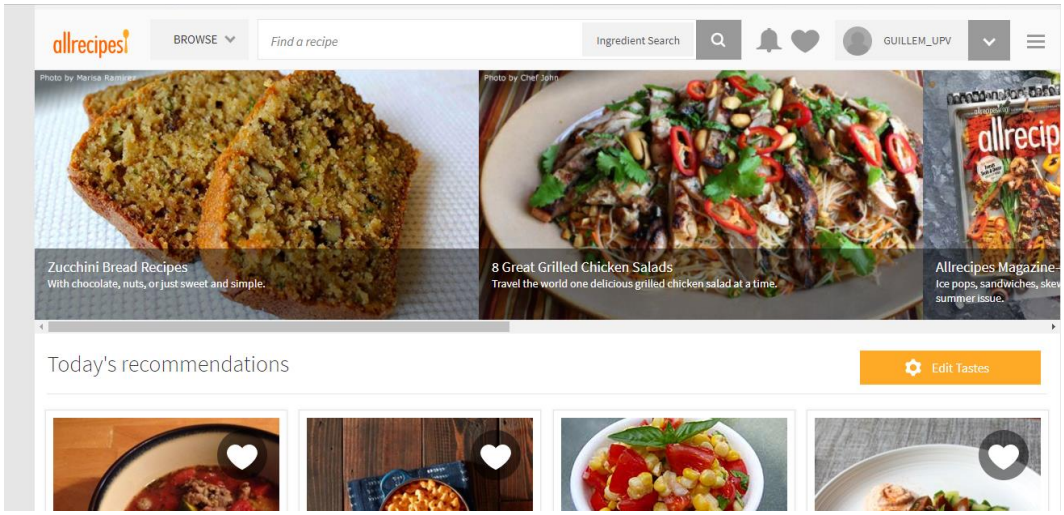

Figura 2. Captura de pantalla de la página principal de Allrecipes

QUÉ HAY EN LA NEVERA

Por último, encontramos una opción que no sólo nos permite crear recetas sino también podemos personalizar menús completos y poder valorar los menús de otros usuarios, con previo registro en la plataforma. Un usuario sin cuenta, sólo podrá ver las recetas que se cuelgan en la página.

Figura 3. Captura de pantalla de la página principal de ¿Qué hay en la nevera?

Una vez introducidas las plataformas escogidas como referencia a la hora de comparar este proyecto, pasaremos a describir, en formato de tabla comparativa, las diferencias entre todas estas y *Receteame.com*. Para ello, se escogerán características comunes en redes sociales, así como funcionalidades propias del dominio nutricional y se indicará si alguna de ellas la implementa o no.

Característica	**Cocinario**	**Qué hay en la nevera**	**Allrecipes**	**Receteame**
Creación de recetas	X	X	X	X
Filtrado de recetas	X	X	X	X
Versionado de recetas	X			X
Envío de mensajes	X			X
Información alimentaria de usuarios				X
Creación de menús		X		
Ranking		X		
Lista de favoritas	X	X	X	X
Registro con redes sociales	X	X	X	X
Perfil de usuarios	X	X	X	X
Valorar recetas	X	X	X	X

Tabla 1. Comparativa de funcionalidades entre Receteame.com y otras comunidades

Como podemos observar, *Receteame.com* implementa la mayoría de las características que encontramos en los demás portales y, además añade funcionalidad tan relevante en el contexto nutricional como la gestión de información sensible de carácter alimentario de los usuarios.

2.2 Tecnologías para el desarrollo Web

Para finalizar con el apartado de *Estado del arte*, realizaremos un repaso de las tecnologías más empleadas a la hora de desarrollar aplicaciones Web y concluiremos con una justificación de la tecnología que se ha elegido para la realización de este proyecto. Cabe puntualizar que en este apartado no se entrará en detalle en cuanto a la descripción de la tecnología empleada en *Receteame.com*. Dicha información se encontrará en el apartado *Tecnologías empleadas*.

Cuando se trata de llevar a cabo un proyecto tecnológico en el ámbito Web, una de las primeras preguntas a la que debemos responder es: *¿Qué tecnologías puedo emplear?*

El aspecto que más se suele tener en cuenta a la hora de buscar una combinación de herramientas para el desarrollo de aplicaciones Web es la popularidad, la cantidad de plataformas que están implementadas con esos lenguajes. Basándonos en esta premisa, una simple búsqueda en el índice que actualiza mensualmente la compañía TIOBE [5] y que mide la popularidad de los lenguajes de programación, nos indica que, a la hora de desarrollar la parte del servidor (*back-end*), los lenguajes más populares son: Java, Python, C#, Javascript y PHP.

Para la parte del cliente (*front-end*), la parte que visualiza el usuario, tenemos la combinación usual formada por la trinidad HTML, Javascript y CSS.

Jun 2017	Jun 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.493%	-6.30%
2	2		C	6.848%	-5.53%
3	3		C++	5.723%	-0.48%
4	4		Python	4.333%	+0.43%
5	5		C#	3.530%	-0.26%
6	9	▲	Visual Basic .NET	3.111%	+0.76%
7	7		JavaScript	3.025%	+0.44%
8	6	▼	PHP	2.774%	-0.45%

Tabla 2. Índice TIOBE con los lenguajes de programación más populares

Por otra parte, y entrando ya en el ámbito de las redes sociales, en *Wikipedia* podemos encontrar una tabla con las tecnologías que usan las Webs más populares, entre ellas *Facebook, Twitter y Google*.

Programming languages used in most popular websites*

Websites	Popularity (unique visitors per month)[1]	Front-end (Client-side)	Back-end (Server-side)	Database	Notes
Google.com[2]	1,600,000,000	JavaScript	C, C++, Go,[3] Java, Python	BigTable,[4] MariaDB[5]	The most used search engine in the world
Facebook.com	1,100,000,000	JavaScript	Hack, PHP (HHVM), Python, C++, Java, Erlang, D,[6] Xhp,[7] Haskell[8]	MariaDB, MySQL,[9] HBase Cassandra[10]	The most visited social networking site
YouTube.com	1,100,000,000	JavaScript	C, C++, Python, Java,[11] Go[12]	Vitess, BigTable, MariaDB[5][13]	The most visited video sharing site
Yahoo	750,000,000	JavaScript	PHP	MySQL, PostgreSQL[14]	Yahoo is presently[*when?*] transitioning to Node.js[15]
Amazon.com	500,000,000	JavaScript	Java, C++, Perl[16]	Oracle Database[17]	Popular internet shopping site
Wikipedia.org	475,000,000	JavaScript	PHP, Hack	MySQL[*citation needed*], MariaDB[18]	"MediaWiki" is programmed in PHP, runs on HHVM; free online encyclopedia
Twitter.com	290,000,000	JavaScript	C++, Java, Scala, Ruby[19]	MySQL[20]	140 characters social network

Tabla 3. Lenguajes de programación usados por las páginas webs más populares

Como podemos observar en las dos tablas anteriores, lo más común es emplear un lenguaje distinto para cada parte de la aplicación (*back-end/front-end*), pero para la realización de este trabajo se ha decidido emplear el mismo lenguaje tanto para la parte del cliente como para la parte del servidor, más concretamente, Javascript. Esta decisión viene motivada por dos razones:

La primera de ellas es que, a pesar de que antes este lenguaje se empleaba casi en exclusiva para el diseño del lado del cliente en una Web, hoy en día ya no es así. El uso de Javascript se ha extendido a la parte del servidor e incluso a la parte de las bases de datos, gracias al paquete MEAN (véase apartado *Tecnologías empleadas*). Una búsqueda en *Google Trends* muestra cómo las tecnologías que forman este paquete de desarrollo son cada vez más populares.

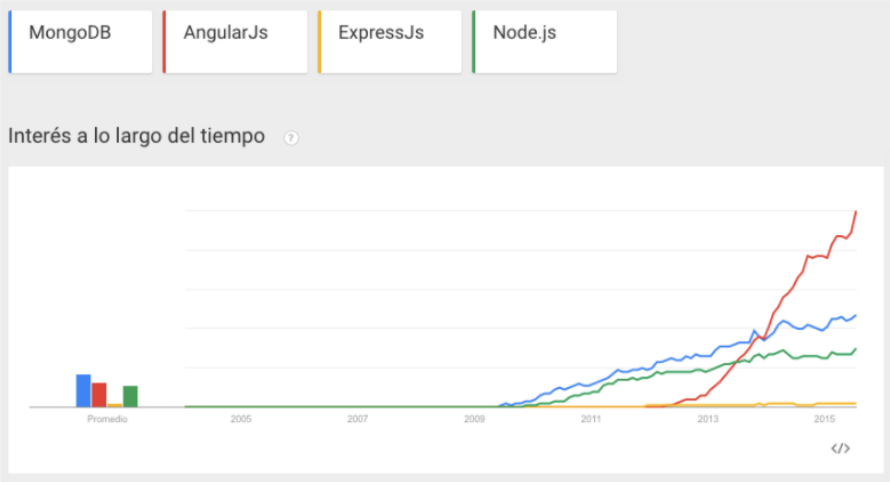

Figura 4. Gráfica de Google Trends sobre el stack MEAN

¿Y por qué este hecho representa una innovación? La razón es sencilla. Hoy en día, un desarrollador que sepa Javascript, puede desarrollar una aplicación Web desde cero y completa. A este perfil de desarrollador se le conoce con el nombre de desarrollador *full-stack* o *rock-star*.

En la actualidad, es este perfil de desarrollador el más demandado en empresas de nueva creación, pequeñas y con enorme aporte de innovación tecnológica (*startups*) debido a su enorme polivalencia.

Laurence Gellert, un desarrollador de *software*, fue el primero en llegar a una definición de este tipo de perfiles en agosto de 2012 [6]. Al comienzo del artículo, Gellert señala que, durante una convención de código abierto, un trabajador de *Facebook* le dijo que la red social sólo contrataba profesionales con dicho perfil. Según Gellert, un desarrollador *full-stack* debe tener conocimientos en:

- Servidores y redes
- Modelado de datos
- Lógica de negocio
- API RESTful
- Interfaces y experiencia de usuario

En conclusión, un desarrollador *full-stack* debe tener conocimiento no solo en cuanto a desarrollo, sino también en cuanto a aspectos más propios de la Ingeniería del Software (modelado de datos, análisis de requisitos, etc.).

Por último, la otra razón por la que se ha elegido desarrollar este proyecto usando el *stack* MEAN es que gracias al empleo del lenguaje Javascript tanto en la parte *back-end* como en el *front-end*, los tiempos de desarrollo se reducen debido al uso de un único lenguaje de programación.

3. Proceso de desarrollo *Software*

En las últimas décadas, el *software* ha superado al *hardware* como factor determinante del éxito. Para entrar un poco más en detalle, entre el 1950 y 1980, el reto era la mejora del *hardware* para el aumento del rendimiento y la capacidad de almacenamiento. A partir del 1990 y gracias a los avances que se realizaron en el campo de la electrónica, el principal desafío ha sido mejorar la **calidad del *software*** y **reducir costes** [7]. Pero, ¿qué es software de calidad?

Un producto *software* de calidad es aquel que, a pesar de cambios en el entorno, pueda adaptarse a ellos fácilmente; un producto que se adapte a nuevos entornos y que sea coherente con los requisitos establecidos previamente. Para lograr conseguir estos objetivos es importante establecer una metodología de trabajo, la cual define un proceso de desarrollo. El uso de una metodología conduce a la realización de tareas de una forma ordenada y estructurada. El conocimiento en todo momento del estado del trabajo, saber de qué fases está compuesto y cuantas de ellas quedan para finalizar el desarrollo, son algunas ventajas que ofrece seguir una metodología de desarrollo.

Para llevar a cabo la realización de este proyecto, se ha utilizado una metodología tradicional de desarrollo *software*, estructurada en las fases de: especificación de requisitos, análisis, diseño, implementación y pruebas. Además, se añade otra fase de despliegue. Cada una de las fases formará un subapartado dentro de esta sección.

3.1 Especificación de requisitos

La especificación de requisitos se podría considerar como una submemoria dentro de este trabajo. Su objetivo es la descripción detallada del producto, enumerando sus funciones y definiendo conceptos fundamentales para su planteamiento. Basándonos en el estándar IEEE 830, esta submemoria quedaría compuesta por las secciones que siguen [8]:

3.1.1 Introducción

3.1.1.1 Propósito

El propósito de este proyecto es proporcionar una plataforma que permita a los usuarios crear y compartir sus recetas, gestionar su información de contacto y alimentaria y, sobretodo, interaccionar con otros usuarios. Mediante esta especificación de requisitos, será posible el diseño de la arquitectura de la misma.

3.1.1.2 Ámbito

Receteame.com deberá cumplir con las funcionalidades que se establecen en la especificación y, además, tendrá que satisfacer los aspectos que se enumeran a continuación:

- **Despliegue:** La aplicación se encontrará disponible bajo el dominio receteame.com.
- **Escalabilidad:** La aplicación se adaptará a cualquier número de peticiones.
- **Seguridad:** Las direcciones de correo serán verificadas antes de la activación de la cuenta para evitar algún tipo de fraude.

3.1.1.3 Definiciones, acrónimos y abreviaturas

- **Receta:** Descripción ordenada de un proceso culinario. Suele incluir una lista de ingredientes y una sucesión de pasos mediante los cuales se elabora un plato.
- **API:** *Application Programming Interface.* Conjunto de funciones ofrecidas por alguna librería para ser utilizadas por otro *software*.
- **BD:** Base de datos. Es la capa que se encarga de almacenar de forma persistente los datos que va a gestionar una aplicación.
- **Cuenta:** Conjunto de información que identifica o distingue a un usuario de otro.
- **Javascript:** Lenguaje de programación interpretado por el navegador, dialecto del estándar ECMAScript. Se define como orientado a objetos, imperativo, débilmente tipado y dinámico.
- **ECMAScript:** Es el estándar que define cómo debe de ser el lenguaje Javascript.
- **JSON:** *Javascript Object Notation* (notación de objetos de Javascript). Es un formato ligero de texto y está inspirado en el lenguaje Javascript. Se basa en un objeto clave/valor.
- **MVC:** Patrón de diseño que desglosa la arquitectura de un producto *software* en tres capas: La base de datos, la lógica de negocio y la interfaz.
- **Responsive:** Cualidad de una interfaz para adaptarse a distintos tipos de dispositivos.

3.1.2 Descripción general

Este apartado describe de forma general el producto, sin entrar en detalles técnicos

3.1.2.1 Perspectiva del producto

Se pretende desarrollar un producto orientado a cualquier sistema operativo, así como a cualquier tipo de dispositivo (pc, portátil, Tablet, móvil…). Deberá funcionar pues en cualquier dispositivo que disponga de un navegador web. Esto se conseguirá a través de la aplicación de un diseño de carácter *responsive* (adaptativo); en ningún caso se desarrollarán diferentes *front-end*.

El producto no formará parte de ningún otro sistema mayor, es decir, será una aplicación autocontenida.

3.1.2.2 Funciones del producto

La plataforma *Receteame.com* deberá satisfacer las funcionalidades que se describen a continuación:

- **Gestión de usuarios:** Los usuarios podrán hacer registros, inicios de sesión y modificar su información personal e información referente a cualquier aspecto alimentario. También tendrán la posibilidad de buscar y seguir a otros usuarios.
- **Gestión de recetas:** Se permitirá la creación, búsqueda, filtrado, valoración, versionado y visualización de recetas.
- **Gestión de imágenes:** Se dará la posibilidad de relacionar una imagen al perfil del usuario y a una receta.
- **Comunicaciones internas:** Los usuarios podrán ponerse en contacto entre ellos mediante el envío de mensajes a través de su área de conversaciones.
- **Notificaciones:** Se notificará al usuario correspondiente cuando reciba un mensaje, consiga un nuevo seguidor o alguien valore alguna de sus recetas.

3.1.2.3 Características de los usuarios

Esta red social está orientada a personas con afición por la cocina, ya sean consumidores o creadores del propio contenido que se gestione en la aplicación. También está dedicada a cocineros con experiencia laboral que quieran compartir sus conocimientos o experiencias con otros usuarios.

Existirán dos roles de usuarios, que se detallan en las siguientes tablas.

Tipo	**Usuario anónimo**
Acciones permitidas	Crear una cuenta local Buscar recetas Filtrar recetas Visualizar recetas

Tabla 4. Acciones permitidas a un usuario anónimo

Tipo	**Usuario registrado**
Acciones permitidas	Iniciar sesión con cuenta local Iniciar sesión a través de *Facebook* Cerrar sesión Buscar recetas Filtrar recetas Visualizar recetas Listar recetas propias Listar recetas de otros Listar versiones de una receta Crear recetas Versionar recetas Valorar recetas Añadir recetas a “Favoritas” Eliminar recetas de “Favoritas” Listar recetas favoritas Listar recetas favoritas de otros Seguir a un usuario Dejar de seguir a un usuario Buscar usuarios Enviar mensajes a otros usuarios Ver mensajes Acceder a su perfil personal Acceder a otros perfiles Modificar su información personal Listar notificaiones Borrar notificaciones

Tabla 5. Acciones permitidas a un usuario registrado

3.1.2.4 Restricciones generales

- El producto será una aplicación web dinámica.
- El servidor deberá poder atender un número considerable de peticiones concurrentes.
- El servidor será capaz de contener tanto la base de datos como las imágenes que suban los usuarios, ya sean imágenes del perfil o de recetas.

3.1.2.5 Suposiciones y dependencias

Se espera que la plataforma se use a través de navegadores que cumplan con los estándares recomendados por el *World Wide Web Consortium*. Actualmente, la mayoría de los navegadores se mantienen actualizados de manera automática por lo que no se prevé ningún tipo de problema de compatibilidad en este aspecto.

3.1.3 Requisitos específicos

En este apartado, se describirá el producto de forma detallada y teniendo en cuenta aspectos técnicos. Esto nos permitirá conocer las funcionalidades que deberá implementar la aplicación.

3.1.3.1 Requisitos de interfaz externos

Estos requisitos son los que deberá cumplir la interfaz del producto.

- **IR-1:** Deberá ser adaptable a cualquier dispositivo. Para ello se usarán las cualidades *responsive* que contemplan los estándares pertinentes.
- **IR-**2: No deberá depender de eventos de ratón u otro tipo que no sean multiplataforma.

3.1.3.2 Requisitos funcionales

Este tipo de requisitos establecen las funcionalidades o los servicios que ofrecerá la aplicación.

- **FR-1: Crear cuenta local.** Se podrán crear cuentas de usuario indicando el nombre y apellidos, el correo electrónico, un nombre de usuario, el sexo y una contraseña.
- **FR-2: Iniciar sesión con cuenta local.** El usuario, previamente registrado y una vez verificado su correo electrónico, podrá acceder a la aplicación usando su correo y contraseña.
- **FR-3**: **Iniciar sesión a través de *Facebook*.** El usuario podrá acceder a la plataforma usando su cuenta de *Facebook*.

- **FR-4: Modificar cuenta.** El usuario, una vez haya creado su cuenta, podrá editar en cualquier momento cualquier dato personal, además podrá eliminar o añadir restricciones alimentarias si lo desea.
- **FR-5: Cerrar sesión.** Será posible cerrar una sesión previamente iniciada.
- **FR-6: Crear receta.** Debe ofrecerse la posibilidad de crear recetas a los usuarios registrados. Para ello será necesario indicar, como mínimo, un título, la lista de ingredientes que componen la receta y los pasos que se requieren para realizarla, pudiendo, además, indicar información como el número de comensales para los que está pensada la receta, una aproximación del precio que costaría realizarla, información de carácter alimentario (receta vegana, sin huevo, etc.) y alguna observación que se quiera aportar.
- **FR-7: Versionar receta.** Los usuarios registrados podrán crear versiones de una receta previamente creada, modificando la información de ésta para adaptarla a cualquier restricción alimentaria o a sus gustos o preferencias.
- **FR-8: Buscar receta.** Cualquier usuario será capaz de buscar la receta que quiera, siempre y cuando exista en la base de datos. En caso contrario, el sistema le informa de que no existe y le anima a que se registre (si no lo está) y la cree.
- **FR-9: Filtrar recetas.** Una vez realizada una búsqueda, al usuario se le proporcionará la opción de refinar su búsqueda de receta mediante el tipo de cocina.
- **FR-10: Listar recetas propias.** Un usuario podrá llevar un registro de las recetas que crea.
- **FR-11: Listar recetas de otro usuario.** Los usuarios podrán visualizar las recetas que han creado otros.
- **FR-12: Listar recetas favoritas de otro usuario.** Se permitirá visualizar las recetas que han marcado como favoritas otros usuarios.
- **FR-13: Listar versiones de una receta.** Se dará la posibilidad de visualizar, en caso de que existan, las versiones de una receta cuando se esté visualizando ésta.
- **FR-14: Valorar receta.** Un usuario autentificado, podrá valorar cualquier receta de otro usuario mediante un sistema de “Me gusta/No me gusta”. Esta acción quedará restringida a una sola vez por usuario, es decir, un usuario solo podrá dar “Me gusta” o “No me gusta” una única vez, pero podrá cambiar su valoración por la opuesta en cualquier momento.
- **FR-15: Añadir receta a favoritas.** Un usuario tendrá la opción de añadir recetas creadas por otros a una lista de recetas favoritas.
- **FR-16: Listar recetas favoritas.** El usuario podrá acceder a su registro de recetas favoritas para eliminar o visualizar en detalle cualquiera de ellas.

- **FR-17: Eliminar receta de favoritas.** En cualquier momento los usuarios podrán eliminar de esta lista cualquier receta que consideren.
- **FR-18: Seguir a un usuario.** Los usuarios estarán relacionados entre sí mediante un sistema de seguidores.
- **FR-19: Dejar de seguir a un usuario.** En cualquier momento un usuario podrá dejar de seguir a otro.
- **FR-20: Enviar mensajes**. Los usuarios podrán comunicarse entre ellos mediante el envío de mensajes.
- **FR-21: Ver mensajes.** Se dispondrá de una sección en la cual se podrá llevar un registro de los mensajes que los usuarios envían y reciben.
- **FR-22: Listar notificaciones.** Los usuarios estarán informados de las interacciones que reciben por parte de otros mediante un listado de notificaciones.
- **FR-23: Visualizar perfil propio.** Se facilitará un acceso al perfil del usuario desde donde poder modificar su información, así como crear nuevas recetas o visualizar las que haya creado y sus recetas favoritas.
- **FR-24: Visualizar perfil de otro usuario.** Este acceso permitirá a usuarios visitar el perfil de otros.

3.1.3.3 Requisitos lógicos de base de datos

- **LR-1:** La base de datos deberá ser de tipo no relacional, ya que los registros proporcionados en la base de datos creada por el grupo de investigación **GTI-IA**, son documentos en formato JSON.

3.1.3.4 Requisitos de rendimiento

- **PR-1:** El sistema deberá tener un tiempo de respuesta no superior a un segundo [13].

3.1.3.5 Restricciones de diseño

Al utilizarse un sistema de base de datos concreto será necesario utilizar un lenguaje de programación y/o *framework* que disponga de alguna API que permita hacer conexiones a dicho sistema.

3.1.3.6 Atributos

En este apartado, se citan los atributos de calidad que se han seleccionado para este proyecto:

- **Fiabilidad:** La plataforma deberá ser fiable, es decir, la probabilidad de su correcto funcionamiento deberá tender a uno. En caso de fallo, se atenderá dentro de lo posible y si no es posible su reparación, se notificará a los usuarios de forma clara y sencilla.
- **Seguridad:** Por una parte, las contraseñas de los usuarios se almacenarán en la base datos cifradas mediante el uso del algoritmo *Blowfish*. Este algoritmo permite comprobar si un valor escogido es igual a otro valor cifrado sin necesidad de conocer la semilla que se usó para cifrar. De esta forma, si un atacante consiguiese acceso a nuestra base de datos, sería computacionalmente impracticable que obtuviese alguna contraseña, ya que la semilla (aleatoria) se desecha una vez usada.
  Por otra parte, para poder realizar el inicio de sesión, la cuenta de ese usuario deberá estar verificada. Esta verificación se realiza mediante el envío de un enlace al correo que se empleó en el registro.

3.2 Análisis

En esta sección de la memoria se muestran los aspectos técnicos de la aplicación a desarrollar basándonos en la especificación de requisitos que hemos realizado en el punto anterior. Para ello, se utilizará el lenguaje de modelado UML.

3.2.1 Diagrama de clases

Un diagrama de clases tiene como objetivo mostrar la estructura estática del sistema que se pretende desarrollar, mostrando las entidades más importantes y las relaciones entre ellas.

En primer lugar, describiremos las entidades que pasarán a formar parte de nuestro diagrama de clases, para luego mostrar el esquema resultante.

- **User:** Usuario. Representa el dueño de una cuenta, la persona o entidad que entrará a formar parte de la comunidad una vez realizado el registro. Esta clase contendrá toda su información personal, así como el listado de recetas favoritas y los listados de seguidores y seguidos.
- **Notification:** Notificación. Mensaje de carácter informativo que se crea al realizarse algún tipo de interacción entre usuarios, como por ejemplo el envío de un mensaje o la valoración de una de sus recetas. Está asociada a un usuario, aunque también almacena información del usuario que origina y una fecha en la que se crea esta notificación por cuestiones de presentación.
- **Message:** Mensaje. Simboliza la forma de comunicación interna entre usuarios. Esta clase almacena información sobre el usuario origen, el destinatario y la fecha de su creación.
- **Recipe:** Receta. Esta entidad interpreta el papel del contenido que se creará en la red social. Guarda información sobre el número de comensales, el título, la secuencia de pasos para llevarla a cabo y una referencia a los ingredientes que la forman. También tiene asociada un creador y, en caso de que existan, un listado de versiones y restricciones nutricionales.
- **Ingredient:** Ingrediente. Representa una unidad básica del grupo de ingredientes que formará parte de una receta y almacena información sobre su nombre, formas comunes de escribir ese nombre y la cantidad. Además, contiene una referencia al ingrediente genérico de la base de datos americana **USDA** (*U.S. Department of Agriculture*).

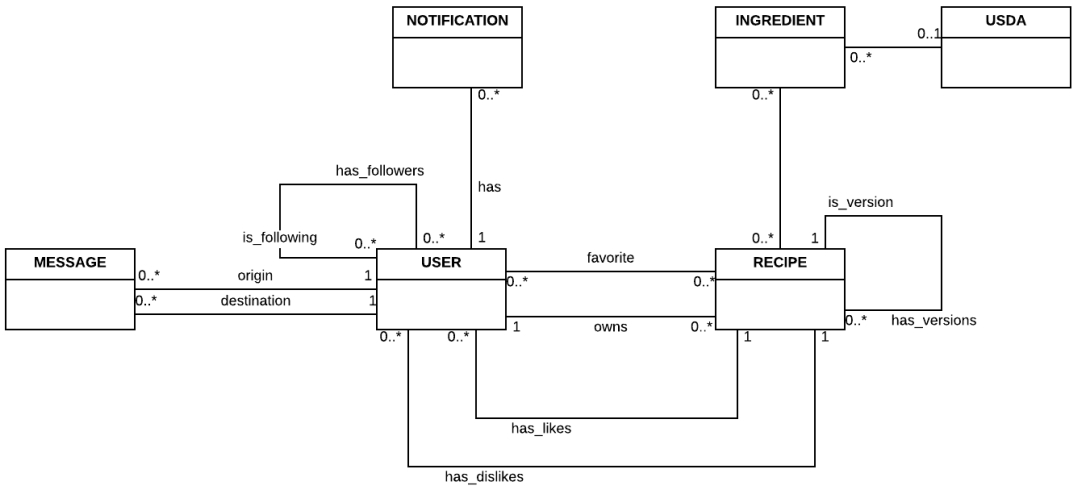

Figura 5. Diagrama de clases de la aplicación

3.2.2 Diagrama de casos de uso

El diseño de este tipo de diagramas nos ayuda a capturar la información de cómo trabaja el sistema, o de cómo se desea que trabaje. El diagrama de casos de uso se emplea para capturar los requisitos funcionales del sistema que se pretende desarrollar. Está formado por los actores (entidad que interacciona con el sistema) y por los casos de uso (funcionalidad).

Para simplificar el diagrama, se dividirán los casos de uso en distintas características o grupos de funcionalidades.

Gestión de usuarios:

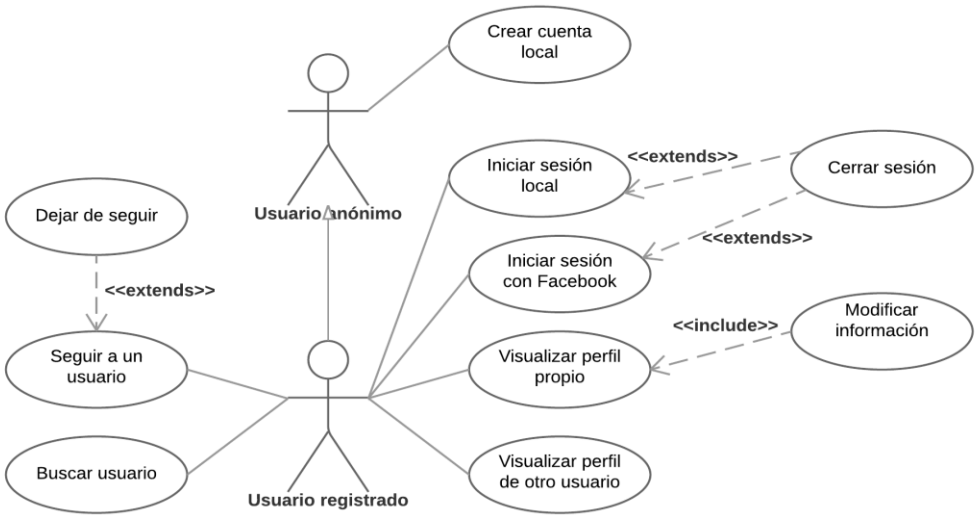

Figura 6. Casos de uso para la gestión de usuarios

Gestión de mensajes:

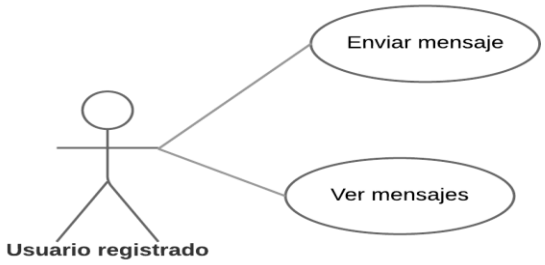

Figura 7. Casos de uso para la gestión de mensajes

Gestión de notificaciones:

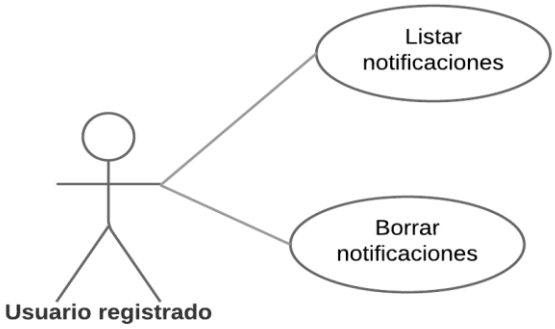

Figura 8. Casos de uso para la gestión de notificaciones

Gestión de recetas:

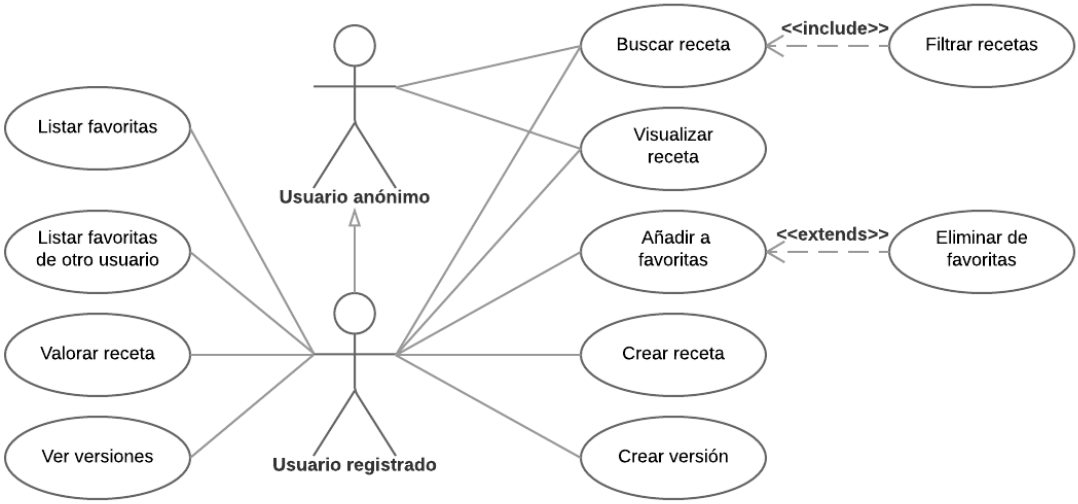

Figura 9. Casos de uso para la gestión de recetas

Para finalizar este punto de la memoria, se describirá cada caso de uso con más detalle para facilitar su comprensión, para ello se enumerarán y se expondrán en formato de tablas.

Caso de uso	**Crear cuenta local**
Actores	Usuario anónimo
Descripción	Permite a un usuario crear una cuenta nueva para acceder a las funcionalidades que ofrece la plataforma.
Secuencia principal	• El usuario solicita registrarse • El sistema le ofrece la posibilidad de introducir sus datos mediante un formulario. • Se comprueba que los datos introducidos no producen ningún conflicto. • Se registra el usuario en la base de datos y se envía un correo electrónico al usuario para que active por completo su cuenta.
Flujos alternativos	• Si ya existe una cuenta con ese correo electrónico o nombre de usuario, se informa al usuario que debe cambiar ese dato.
Requisitos	Ninguno.

Tabla 6. Caso de uso: Crear cuenta local

Caso de uso	**Iniciar sesión local**
Actores	Usuario registrado
Descripción	Permite a un usuario acceder a una cuenta previamente creada para utilizar la aplicación de forma autenticada.
Secuencia principal	• El usuario solicita iniciar sesión con su cuenta local. • El sistema le facilita un formulario donde introducir el correo electrónico y la contraseña que usó para crear la cuenta. • El sistema crea una sesión y se devuelve al usuario a la página de inicio de la plataforma.
Flujos alternativos	• Si los datos introducidos son erróneos, se muestra un mensaje de aviso para que corrija los datos.
Requisitos	Haber activado la cuenta mediante el enlace proporcionado al correo que el usuario facilita a la aplicación.

Tabla 7. Caso de uso: Iniciar sesión local

Caso de uso	**Iniciar sesión con *Facebook***
Actores	Usuario registrado
Descripción	Permite a un usuario crear y acceder a una cuenta a través de los datos que se emplearon para registrarse en *Facebook*.
Secuencia principal	• El usuario solicita acceder a la aplicación mediante su cuenta de *Facebook*. • El sistema, haciendo uso de la API que proporciona *Facebook*, solicita al usuario los datos empleados en esa red social para su posterior validación e inicio de sesión.
Flujos alternativos	• Si es la primera vez que el usuario solicita el acceso a través de la otra plataforma, se creará un registro en la base de datos de usuarios y se iniciará la sesión.
Requisitos	Disponer de una cuenta en *Facebook*.

Tabla 8. Caso de uso: Iniciar sesión con Facebook

Caso de uso	**Cerrar sesión**
Actores	Usuario registrado
Descripción	Permite a un usuario eliminar una sesión, pasando a ser un usuario anónimo.
Secuencia principal	• El usuario solicita el cierre de sesión. • El sistema invalida la sesión y, en caso de que el usuario se encuentre en una ruta que requiera autenticación, lo envía de vuelta a la vista principal.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 9.Caso de uso: Cerrar sesión

Caso de uso	**Visualizar perfil propio**
Actores	Usuario registrado
Descripción	Permite a un usuario ver su información personal.
Secuencia principal	• El usuario pulsa sobre un elemento que enlaza con su perfil. • El sistema le proporciona la vista de su perfil con su información y otros enlaces.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 10. Caso de uso: Visualizar perfil propio

Caso de uso	**Modificar información**
Actores	Usuario registrado
Descripción	Permite a un usuario modificar su información personal
Secuencia principal	• El usuario solicita acceder a su perfil. • El Sistema le facilita la posibilidad de modificar sus datos mediante un formulario. • El usuario introduce los datos que quiere modificar. • Se modifican los datos y se muestran los cambios.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 11. Caso de uso: Modificar información

Caso de uso	**Visualizar perfil de otro usuario**
Actores	Usuario registrado
Descripción	Permite a un usuario ver la información y recetas de cualquier otro usuario registrado en la plataforma.
Secuencia principal	• El usuario pulsa sobre un elemento que enlaza con el perfil de otro usuario. • El sistema le proporciona la vista del perfil con su información y otros enlaces (Sus recetas, sus recetas favoritas).
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 12. Caso de uso: Visualizar perfil de otro usuario

Caso de uso	**Buscar usuario**
Actores	Usuario registrado
Descripción	Ofrece la posibilidad de encontrar a otro usuario por su nombre.
Secuencia principal	• El usuario solicita buscar a otro • El sistema posibilita la introducción del nombre del usuario que está buscando. • El usuario introduce el nombre del usuario que está buscando. • El sistema muestra los usuarios que concuerdan con ese nombre.
Flujos alternativos	Sin introducir ningún nombre, el sistema muestra todos los usuarios que están registrados en la plataforma, paginados.
Requisitos	El usuario debe estar autenticado.

Tabla 13. Caso de uso: Buscar usuario

Caso de uso	**Seguir a un usuario**
Actores	Usuario registrado
Descripción	Permite a los usuarios añadir a otro a su lista de usuarios a los que sigue.
Secuencia principal	• El usuario presiona sobre el elemento para seguir a otro. • El sistema notifica al usuario que ha ganado un nuevo seguidor y al usuario que ha realizado la acción de seguir le muestra un mensaje diciendo que la operación se ha realizado con éxito.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 14. Caso de uso: Seguir a un usuario

Caso de uso	**Dejar de seguir**
Actores	Usuario registrado
Descripción	Permite a los usuarios eliminar a otro de su lista de usuarios a los que sigue.
Secuencia principal	• El usuario presiona sobre el elemento para dejar de seguir a otro. • El sistema notifica al usuario que ha realizado la acción con un mensaje diciendo que la operación se ha realizado con éxito.
Flujos alternativos	Ninguno.
Requisitos	El usuario que realiza la acción debe estar autenticado y debe seguir al usuario que desea eliminar de su lista de seguidos.

Tabla 15. Caso de uso: Dejar de seguir

Caso de uso	**Enviar mensaje**
Actores	Usuario registrado
Descripción	Permite enviar mensajes privados a otra cuenta.
Secuencia principal	• El usuario solicita enviar un mensaje a otro. • El sistema le facilita la posibilidad de introducir el texto que se desea enviar mediante un formulario. • Tras escribir el mensaje, el usuario presiona sobre el elemento que se encarga de hacer llegar este mensaje al usuario destino. • El sistema notifica al usuario destino de que tiene un nuevo mensaje y debe comprobar su apartado para mensajes.
Flujos alternativos	Ninguno.
Requisitos	El usuario que realiza la acción debe estar autenticado.

Tabla 16. Caso de uso: Enviar mensaje

Caso de uso	**Ver mensajes**
Actores	Usuario registrado
Descripción	Permite a los usuarios llevar un registro de los mensajes que envía o recibe.
Secuencia principal	• El usuario solicita ver sus mensajes. • El usuario selecciona una conversación con otro usuario de su lista. • El sistema carga los mensajes que ha enviado y los que ha recibido de ese otro usuario.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 17. Caso de uso: Ver mensajes

Caso de uso	**Listar notificaciones**
Actores	Usuario registrado
Descripción	Permite a los usuarios ver las notificaciones que reciben, mostrando una descripción.
Secuencia principal	• El usuario solicita ver sus notificaciones. • El sistema le muestra la información.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 18. Caso de uso: Listar notificaciones

Caso de uso	**Borrar notificaciones**
Actores	Usuario registrado
Descripción	Permite a los usuarios eliminar las notificaciones que ya se hayan leído.
Secuencia principal	• El usuario solicita ver sus notificaciones. • El sistema le muestra la información. • El usuario presiona sobre el elemento que le permite borrar las notificaciones que acaba de recibir. • El sistema las elimina.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 19. Caso de uso: Borrar notificaciones

Caso de uso	**Buscar receta**
Actores	Usuario registrado y usuario anónimo
Descripción	Permite a los usuarios encontrar las recetas que concuerden con el título que introduzcan.
Secuencia principal	• Los usuarios introducen en el buscador la receta que quieren buscar. • El sistema proporciona un listado de las recetas que concuerdan con el título introducido y pagina los resultados para no sobrecargar la vista.
Flujos alternativos	Ninguno.
Requisitos	Ninguno.

Tabla 20. Caso de uso: Buscar receta

Caso de uso	**Filtrar recetas**
Actores	Usuario registrado y usuario anónimo
Descripción	Permite a los usuarios refinar sus búsquedas, indicando el tipo de cocina de las recetas que quieren listar.
Secuencia principal	• Los usuarios seleccionan un tipo de cocina de los disponibles en la base de datos. • El sistema lista las recetas que tienen ese tipo de cocina y pagina los resultados.
Flujos alternativos	Ninguno.
Requisitos	Ninguno.

Tabla 21. Caso de uso: Filtrar recetas

Caso de uso	**Visualizar receta**
Actores	Usuario registrado y usuario anónimo
Descripción	Permite a los usuarios ver en detalle la información de una receta.
Secuencia principal	• Los usuarios seleccionan una receta. • El sistema les proporciona una vista con toda la información de la receta seleccionada.
Flujos alternativos	Ninguno.
Requisitos	Ninguno.

Tabla 22. Caso de uso: Visualizar receta

Caso de uso	**Crear receta**
Actores	Usuario registrado
Descripción	Permite a los usuarios autenticados crear recetas.
Secuencia principal	• El usuario solicita crear una nueva receta. • El sistema le facilita una vista con distintos campos a completar para añadir la información de la receta que se desea crear. • El usuario introduce los distintos datos que formaran la receta y presiona sobre el elemento que permite la creación. • El sistema crea la receta y redirecciona al usuario a la vista de detalle de esa receta y le pide que suba una imagen para completarla.
Flujos alternativos	Si el usuario elige no adjuntar ninguna imagen a la receta, se visualiza una imagen por defecto.
Requisitos	Para la creación de la receta es necesario disponer, como mínimo, del título, los ingredientes y los pasos a seguir para elaborarla. Además, el usuario debe estar autenticado

Tabla 23. Caso de uso: Crear receta

Caso de uso	**Crear versión**
Actores	Usuario registrado
Descripción	Permite a los usuarios crear versiones de recetas ya existentes para adaptarlas a sus gustos o restricciones alimentarias.
Secuencia principal	• El usuario presiona sobre el elemento que le permitirá crear la versión. • Se redirige al usuario hacia la vista en la que podrá editar la información de la receta para adaptarla. • El usuario completa los campos que desee cambiar. • El sistema crea la versión, que será visible desde la vista de la receta versionada.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 24. Caso de uso: Crear versión

Caso de uso	**Añadir a favoritas**
Actores	Usuario registrado
Descripción	Permite a los usuarios añadir recetas de otros a la lista de sus recetas favoritas.
Secuencia principal	• El usuario presiona sobre el elemento que le permite añadir una receta a su lista de favoritos. • El sistema añade esa receta al registro de recetas favoritas del usuario
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 25. Caso de uso: Añadir a favoritas

Caso de uso	**Listar favoritas**
Actores	Usuario registrado
Descripción	Permite a los usuarios visualizar las recetas que han marcado como favoritas.
Secuencia principal	• El usuario solicita visualizar sus recetas favoritas. • El sistema le muestra aquellas recetas que marcó como favoritas y pagina los resultados para no sobrecargar la vista.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 26. Caso de uso: Listar favoritas

Caso de uso	**Eliminar de favoritas**
Actores	Usuario registrado
Descripción	Un usuario será capaz de desmarcar una receta como favorita.
Secuencia principal	• El usuario presiona sobre el elemento que le permite eliminar una receta de su lista de favoritas. • El sistema la borra de su registro de recetas favoritas, dándole la oportunidad de volver a añadirla cuando quiera.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado y la receta debe de haberse añadido a las recetas favoritas previamente.

Tabla 27. Caso de uso: Eliminar de favoritas

Caso de uso	**Listar favoritas de otro usuario**
Actores	Usuario registrado
Descripción	Permite a los usuarios visualizar las recetas que han marcado como favoritas otros usuarios.
Secuencia principal	• El usuario solicita visualizar sus recetas favoritas de un usuario específico. • El sistema le muestra aquellas recetas que ese usuario seleccionado marcó como favoritas y pagina los resultados para no sobrecargar la vista.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado.

Tabla 28. Caso de uso: Listar favoritas de otro usuario

Caso de uso	**Valorar receta**
Actores	Usuario registrado
Descripción	Permite a los usuarios valorar positiva o negativamente una receta.
Secuencia principal	• El usuario solicita visualizar una receta. • El sistema le muestra toda la información de la receta que solicitó ver. • El usuario presiona sobre el elemento que le permite valorar positivamente la receta. • El sistema almacena la valoración y muestra el total de esas valoraciones positivas
Flujos alternativos	• El usuario presiona sobre el elemento que le permite valorar negativamente la receta. • El sistema almacena la valoración y muestra el total de esas valoraciones negativas
Requisitos	El usuario debe estar autenticado.

Tabla 29. Caso de uso: Valorar receta

Caso de uso	**Ver versiones**
Actores	Usuario registrado
Descripción	Permite a los usuarios visualizar las distintas versiones que se han creado de una receta.
Secuencia principal	• El usuario solicita visualizar las versiones de una receta. • El sistema le muestra las versiones y pagina el resultado.
Flujos alternativos	Ninguno.
Requisitos	El usuario debe estar autenticado y la receta debe tener versiones.

Tabla 30. Caso de uso: Ver versiones

3.3 Diseño

El propósito de la fase de diseño es describir la arquitectura de la aplicación. Esta arquitectura definirá sus componentes, interfaces y comportamientos.

Al tratarse de un proyecto de desarrollo web, es conveniente plantear la arquitectura en las dos partes en las que estará dividida la aplicación: el *back-end* y el *front-end*. En ambos casos, se planteará una arquitectura de tres capas (datos, lógica y presentación) y seguirán el patrón de diseño Modelo-vista-controlador (MVC).

La arquitectura en tres capas es un patrón de diseño cliente-servidor que divide el *software* en tres capas diferentes: **datos** (persistencia), **lógica** (capa de negocio) y **presentación**. La primera de ellas se encarga de mantener de forma persistente los datos que va a gestionar la aplicación. Debe proporcionar una interfaz a la capa de lógica para que esta pueda acceder a los datos sin importar la forma en la que estén guardados, es decir, debe encargarse de abstraer el almacenamiento de los datos.

La segunda capa, la lógica, es la encargada de controlar las diferentes funcionalidades de la aplicación. Por último, la capa de presentación se encarga de interactuar con el usuario. Muestra la información que obtiene de la capa de lógica y convierte las llamadas del usuario en llamadas a esa capa.

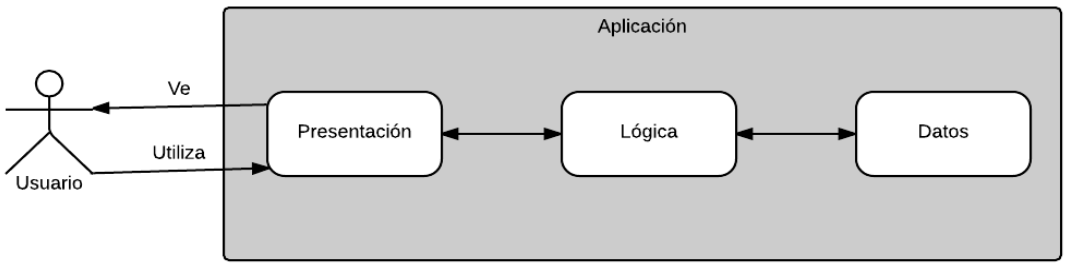

Figura 10. Flujo de interacción de la arquitectura de tres capas

Por otra parte, el patrón de diseño Modelo-vista-controlador, divide la aplicación en tres tipos de componentes:

- **Modelos:** Describen la información almacenada. Estas representaciones permiten manejar los datos y la lógica de la aplicación.
- **Vistas:** Son maneras de mostrar la información al usuario. En una aplicación web, normalmente, son archivos HTML que definen las interfaces de usuario. En nuestro caso, serán archivos *Pug* [14].
- **Controladores:** Su función es la de gestionar la interacción del usuario y transformarla en interacción con los modelos.

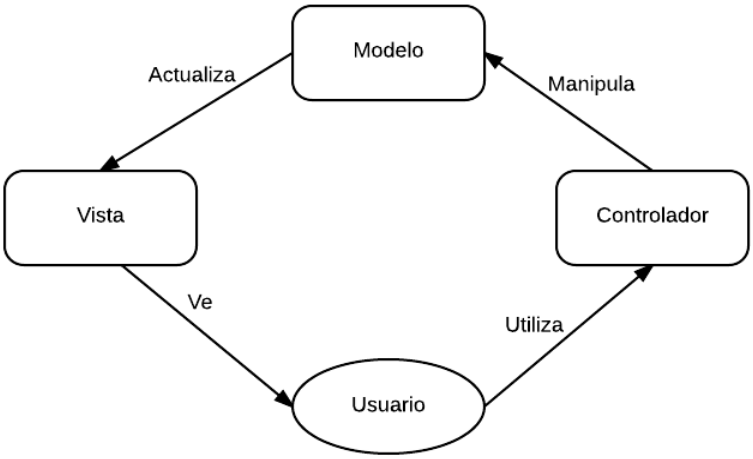

Figura 11. Flujo de interacción del patrón Modelo-vista-controlador

Por último, si combinamos ambos modelos, obtenemos la arquitectura que se plantea para el desarrollo de este proyecto.

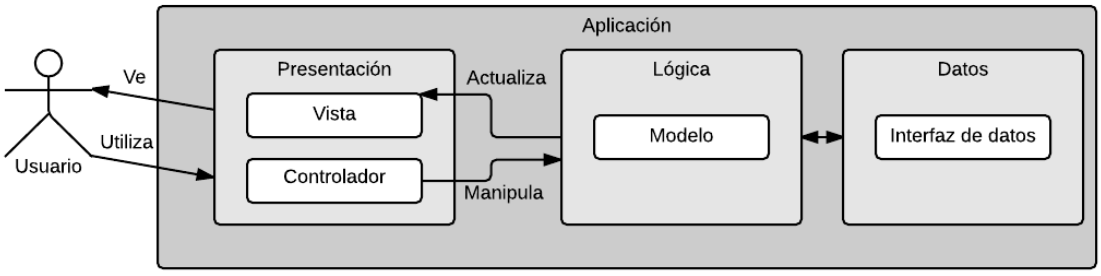

Figura 12. Flujo de interacción de la arquitectura de tres capas y el patrón MVC

Esta arquitectura planteada para la realización del proyecto nos proporciona una gran modularidad a la hora de llevar a cabo la implementación. Así, es posible reutilizar el código y hace posible la incorporación de modificaciones de forma cómoda y fácil.

3.3.1 Back-end

El *back-end* del proyecto está formado por una API RESTful escrita en Javascript, usando NodeJs [15] y ExpressJs [16]. Es la parte que se ejecuta en el servidor.

3.3.1.1 Capa de datos

Como se ha indicado anteriormente, la capa de datos es la encargada de mantener la información que gestiona la aplicación de forma permanente y hacerla accesible a la capa de lógica. En este caso se opta por usar un sistema de gestión de base de datos no relacionales, concretamente MongoDB.

Esta tecnología y las demás usadas se describirán con detalle más adelante. En este apartado, describiremos los esquemas que se han creado para cada entidad de la base de datos.

RECETAS	
_id	Identificador único del objeto.
Images	Fichero de imagen.
Ingredients	Listado de ingredientes (Se especifica en la siguiente tabla).
Instructions	Enumeración de instrucciones de la receta.
Tips	Etiquetas: dificultad, temporada, tiempo de preparación, precio, tipo de cocina…
Title	Título de la receta.
Yield	Número de comensales.
Likes	Lista de referencias a los usuarios que han valorado positivamente la receta.
Dislikes	Lista de referencias a los usuarios que han valorado negativamente la receta.
idUser	Referencia al usuario que ha creado la receta.
isVersion	Valor que indica si la receta es una versión de otro o no.
Versions	Listado de versiones de la receta.
Observations	Enumeración de observaciones sobre esa receta.

Tabla 31. Esquema para la entidad Receta

INGREDIENTS	
Amt	Cantidad. Se divide en: *qty* (cantidad numérica), *unit* (unidad) y *unit_en* (unidad en inglés).
Canonical_name	Nombre más común para el ingrediente.
Code	Referencia al ingrediente en la colección *Ingredientes* (campo _id).
Name	Nombre tal como aparece en la receta.
Name_en	Traducción del nombre al inglés.
Simplified_name	Simplificación del nombre (sin mayúsculas, plurales, etc.).

Tabla 32. Esquema para el campo ingredients de la colección Recetas

La colección de Ingredientes contiene el listado de ingredientes que los integrantes del grupo **GTI-IA** del **DSIC** han revisado manualmente y asociado con un ingrediente de la base de datos americana **USDA** (*U.S. Department of Agriculture*).

INGREDIENTE	
_Id	Identificador único del objeto.
Commons	Lista de nombres con los que aparece este ingrediente en diferentes recetas.
Curated	Booleano que indica si ha sido revisado.
Curated_at	Fecha en que fue revisado.
Curated_by	Usuario que revisó el ingrediente.
Img	URL a una imagen que representa el ingrediente.
Name	Nombre del ingrediente (Se utiliza el más repetido de los commons).
Usda	Referencia al ingrediente en la BD de USDA
Validated	Booleano que indica si ha sido validado.

Tabla 33. Esquema para la entidad Ingrediente

USUARIO	
_Id	Identificador único del objeto.
Name	Nombre completo del usuario.
Username	Alias que emplea el usuario en la plataforma.
email	Correo electrónico que usa para el inicio de sesión.
Password	Contraseña para acceder a la aplicación.
Sex	Sexo del usuario.
City	Ciudad del usuario.
Image	Imagen (Avatar).
Facebook	En caso de acceder a la plataforma usando las credenciales de *Facebook*, se almacenan en este campo.
Followers	Lista de referencias a los usuarios que tiene como seguidores.
Following	Lista de referencias a los usuarios que sigue.
Favorites	Lista de recetas que ha marcado como favoritas.
Restrictions	Enumeración de restricciones alimentarias del usuario.
Verified	Booleano que indica si el correo electrónico del usuario está verificado.
Activation_token	Token que se emplea para la activación de la cuenta.

Tabla 34. Esquema para la entidad Usuario

MENSAJE	
_Id	Identificador único del objeto.
Message	Texto que se pretende enviar.
Sender	Referencia al usuario que crea el mensaje.
Addressee	Referencia al usuario que recibe el mensaje.
Date	Fecha en la que se envía el mensaje.

Tabla 35. Esquema para la entidad Mensaje

NOTIFICACIÓN	
_Id	Identificador único del objeto.
Header	Texto de cabecera para la notificación.
Message	Texto que se muestra en el cuerpo de la notificación.
Creator	Referencia al usuario que origina la notificación.
Addressee	Referencia al usuario que recibe esa notificación.
Date	Fecha en la que se crea la notificación.
link	Enlace al apartado de la plataforma al que hace referencia la notificación.

Tabla 36. Esquema para la entidad Notificación

3.3.1.2 Capa de lógica

Tal y como se ha señalado anteriormente, la capa de lógica se encarga de procesar los datos.

Para trabajar con la capa de datos, se ha empleado una librería para NodeJs llamada *mongoose*. Esta librería nos ofrece la posibilidad de realizar cualquier tipo de consulta a una base de datos MongoDB. Con el fin de manipular las distintas colecciones de la base de datos, se han creado unos modelos los cuales consisten en objetos JSON (*Javascript Object Notation*) con los mismos atributos que hemos descrito en las tablas anteriores. De esta forma los controladores que usan las funciones de la librería *mongoose* interactúan directamente con los modelos y, sin necesidad de ninguna implementación adicional, los cambios realizados se ven reflejados de forma automática en la capa de datos.

3.3.1.3 Capa de presentación

Es normal pensar que una API RESTful no dispone de capa de presentación. Si bien es cierto que no dispone de vistas como tales, sí tiene controladores. Las vistas, en este caso, corresponderían a las respuestas HTTP cuyo cuerpo está en formato JSON y contiene los resultados de las peticiones que hacen los usuarios.

Para este proyecto se ha implementado un controlador por cada entidad que formará la base de datos.

3.3.2 Front-end

El *front-end* de la aplicación consiste en un conjunto de vistas, controladores y servicios desarrolados haciendo uso del *framework* AngularJs [17].

3.3.2.1 Capa de datos

La capa de datos del *front-end* la gestiona el navegador web. La información que se desea almacenar son los datos de sesión del usuario que inicia sesión en la aplicación. Para ello se emplea un módulo para NodeJs llamado *express-session*. Este módulo crea una sesión cada vez que el usuario entra en la plataforma y almacena sus datos.

3.3.2.2 Capa de lógica

La lógica del *front-end* es la encargada de inicializar, estructurar y gestionar el funcionamiento del mismo. Esta capa es la encargada de comunicarse con el *back-end*.

Se definirán diversas factorías [18]. Estas factorías son contenedores de código que podemos usar en nuestros sitios desarrollados con AngularJs. Son un tipo de servicio con el que podemos implementar librerías de funciones o almacenar datos. Dichas factorías se encargarán de hacer llamadas al *back-end* para realizar cualquier tipo de petición.

3.3.2.3 Capa de presentación

Esta capa es la más importante del front-end, ya que se trata del punto de entrada del usuario a la plataforma. Está formada por vistas, cada una de las cuales está asociada a un controlador y dispone de su propia plantilla. El controlador se encarga de hacer uso de las factorías, anteriormente descritas, y el framework AngularJs se encargará de presentar la información que se obtenga en las plantillas.

A continuación, se expondrán las vistas relevantes del proyecto. Como la aplicación ha sido desarrollada completamente y desplegada, no se hará uso de ningún boceto, sino que se mostrarán capturas de las interfaces finales.

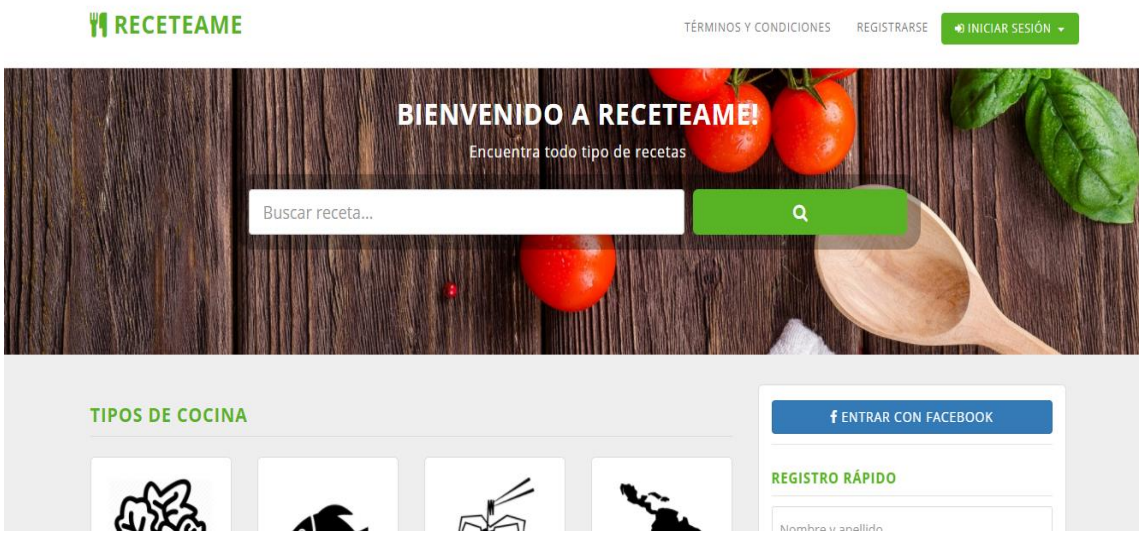

Figura 13. Vista home

Figura 14. Formulario de inicio de sesión en la barra de navegación

Esta es la vista que se muestra al acceder a la plataforma. En ella encontramos una barra de navegación en la parte superior desde la que podemos acceder al formulario de registro y al formulario para iniciar sesión. También disponemos de un campo en el que podemos introducir el título de una receta para buscarla. En la parte central, apreciamos los tipos de cocina que existen, por el momento, en la base de datos. Accediendo a alguno de ellos, podremos visualizar las recetas que formen parte de ese grupo.

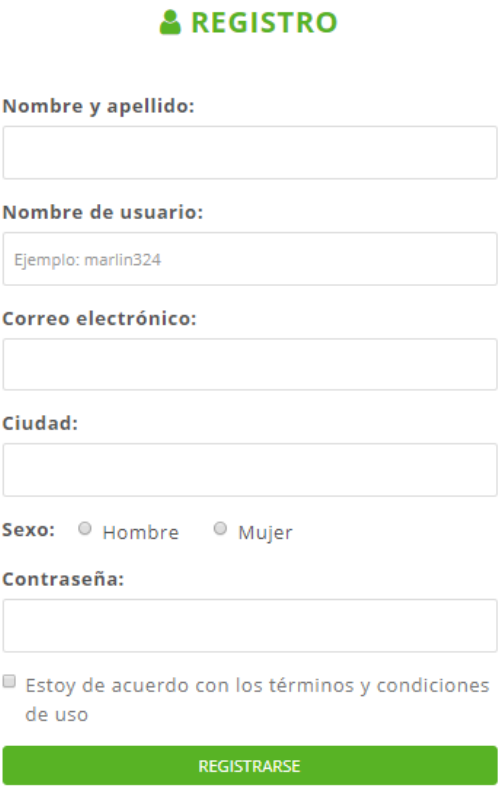

Figura 15. Formulario de registro

Esta vista corresponde al formulario que tiene que completar un usuario para poder registrarse en la comunidad.

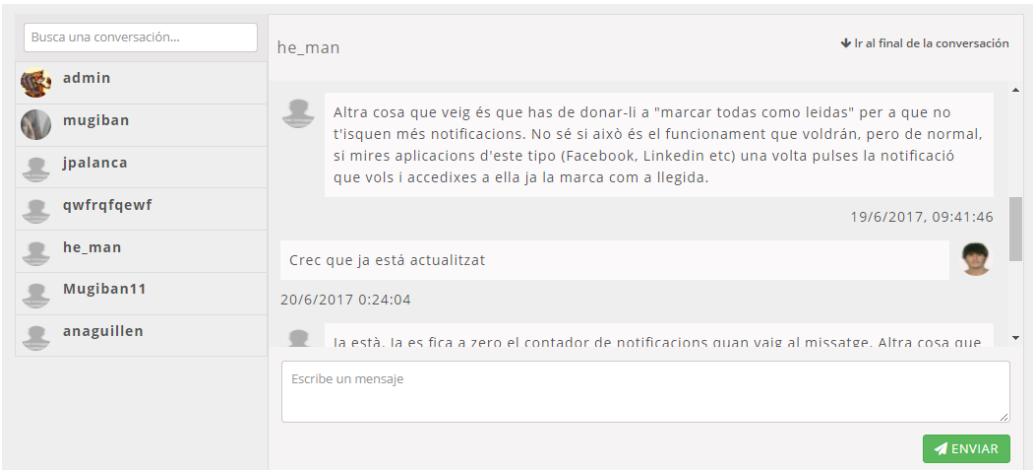

Figura 16. Vista de conversaciones y mensajes

Esta interfaz muestra las conversaciones que ha mantenido el usuario y todos los mensajes que pertenecen a dicha conversación.

Figura 17. Listado de las notificaciones en la barra de navegación

En la barra de navegación superior también podremos encontrar el listado de las notificaciones que vayamos recibiendo. Al hacer *click* en alguna, se elimina de la lista y, además, redirige al usuario a la sección de la página que corresponde con la notificación.

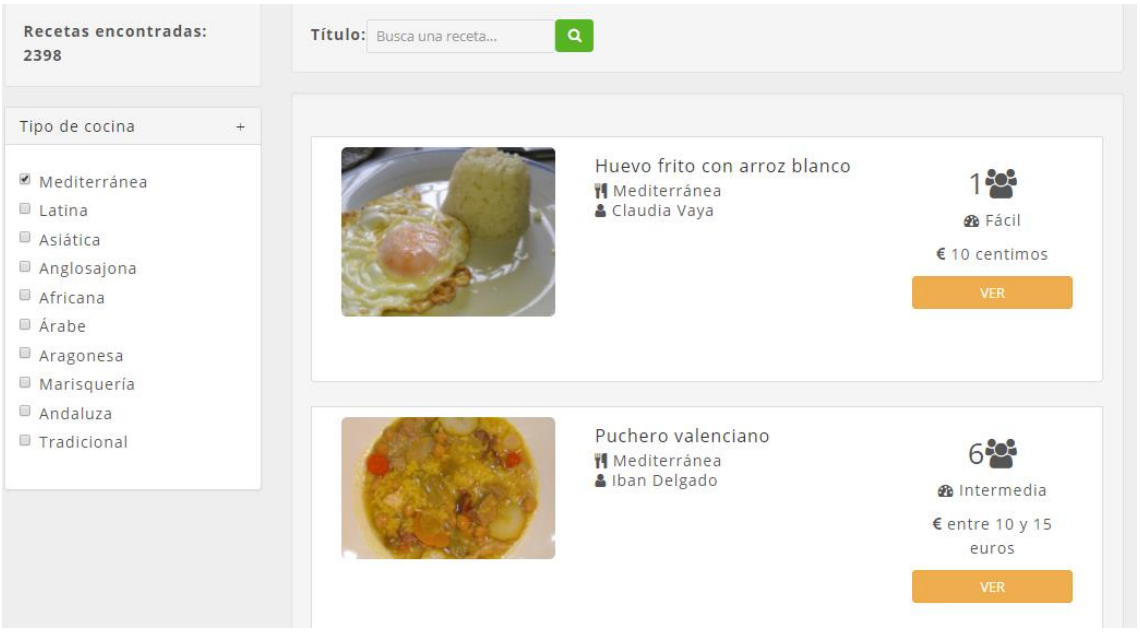

Figura 18. Vista del listado de recetas

En la vista del listado de recetas podremos visualizar el conjunto de recetas que cumpla el criterio de búsqueda que hayamos empleado. En este caso, se filtraron las recetas por el tipo de cocina *Mediterránea*.

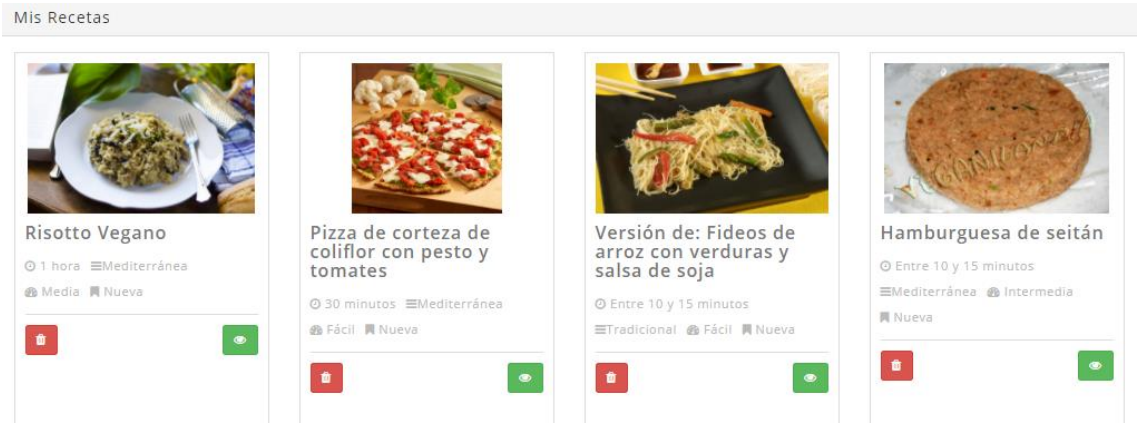

Figura 19. Vista de mis recetas

Esta vista muestra las recetas creadas por el usuario. Para no sobrecargar la página, a partir de ocho, se paginan los resultados. Esta estructura se ha reutilizado para las vistas de las recetas favoritas de los usuarios.

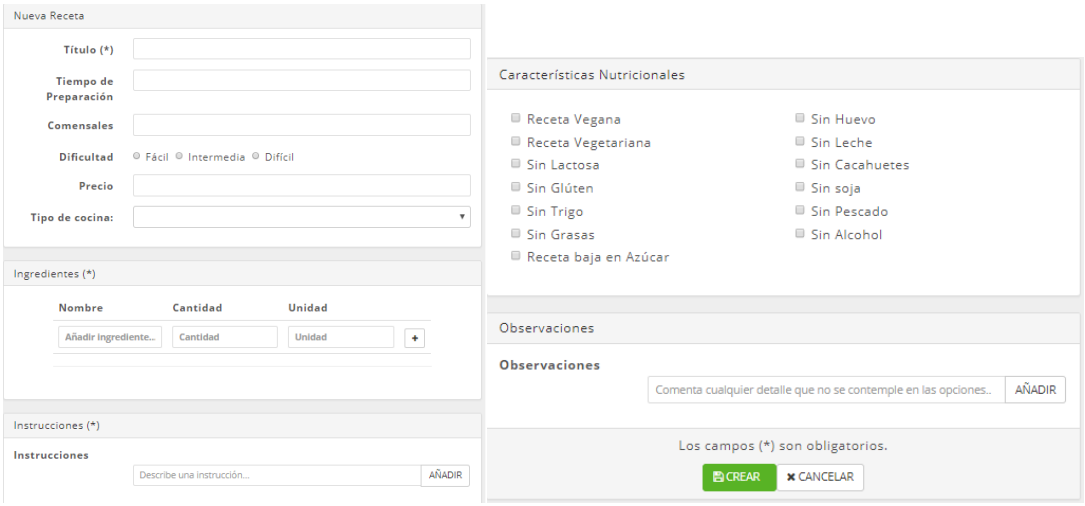

Figura 20. Vista de creación de una receta

Esta interfaz representa el formulario a completar para crear una receta. Cómo se ha indicado en apartados anteriores, una receta deberá contener, como mínimo, un título, una lista de ingredientes y los pasos para poder realizarla. La estructura de esta interfaz se ha reutilizado para crear una versión de una receta. En este caso, se cargarían en la interfaz los datos de la receta a versionar y se daría la opción de modificar o añadir cualquier dato.

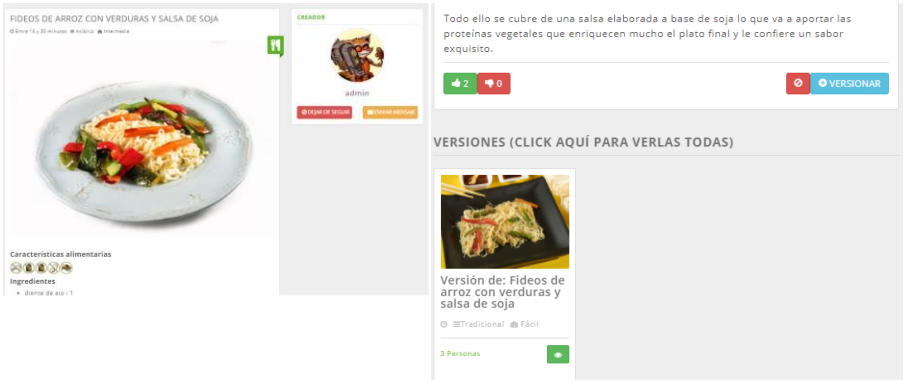

Figura 21. Vista del detalle de una receta

Esta vista muestra las recetas en detalle y nos da la opción de valorarla y poder crear versiones. También, en caso de que la receta tenga versiones creadas, se muestran tres de ellas y se proporciona un enlace a todas las demás.

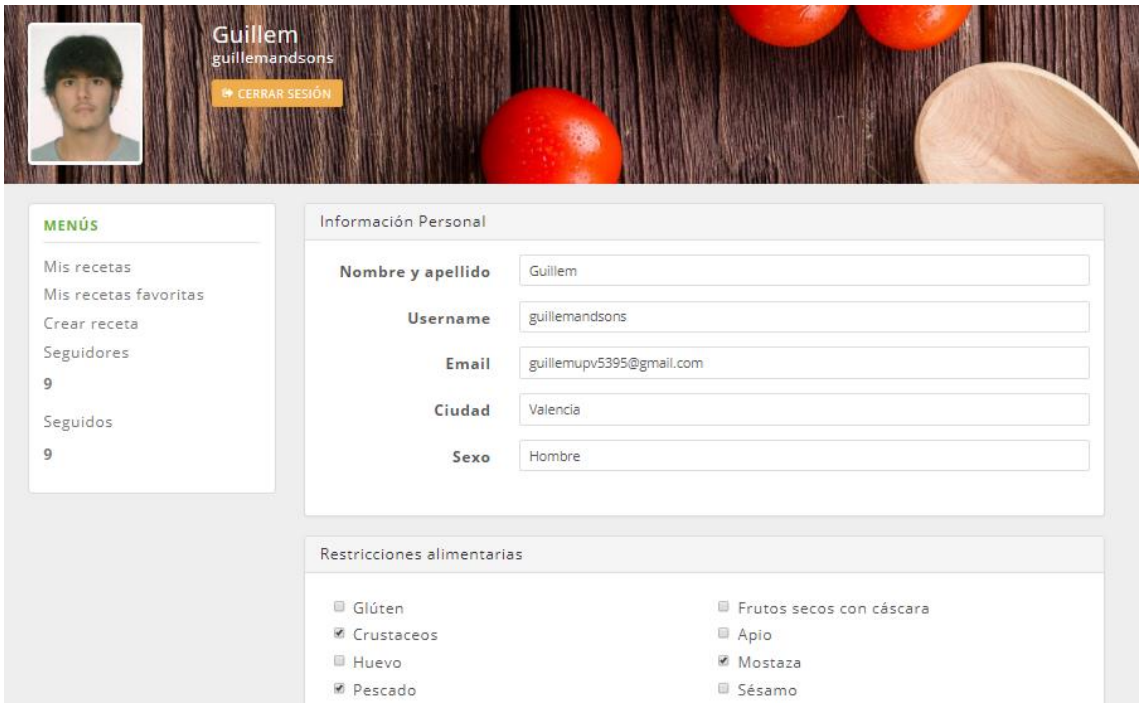

Figura 22. Vista del perfil de usuario

En la vista del perfil de usuario podemos visualizar y modificar la información personal de nuestra cuenta. Además, disponemos de enlaces a otras secciones de interés. Para la vista del perfil de otro usuario se ha seguido el mismo diseño con la diferencia de que también podemos visualizar si ese usuario nos sigue y podemos enviarle un mensaje o seguirle/dejar de seguirle.

Figura 23. Vista del buscador de usuarios

Esta interfaz nos proporciona la posibilidad en buscar a cualquier usuario introduciendo su nombre en el campo de texto. Si no se especifica ningún nombre, se muestran todos los usuarios que hay en la plataforma.

3.4 Implementación

Esta fase tiene como objetivo realizar la codificación de la aplicación, ya sea desde cero o reutilizando otros componentes. Esta fase se realiza teniendo en cuenta los requisitos (fase de análisis) y la arquitectura (fase de diseño).

3.4.1 Construcción del producto

En este apartado describiremos cómo ha sido el proceso de codificación de este proyecto, centrándonos en la construcción de los componentes más relevantes de la aplicación.

Como hemos indicado en el apartado de *Diseño*, nuestra aplicación estará basada en el patrón **Modelo-Vista-Controlador**. Esto requerirá estructurar el proyecto en las tres secciones interconectadas que describen el nombre del patrón elegido. Además, en nuestro caso, el proceso será algo más complicado pues tenemos controladores tanto a nivel de *back-end* como de *front-end*.

En primer lugar, tendremos que crear el **Modelo** para cada entidad que vaya a mantener la aplicación. Estos modelos nos serán de ayuda para manipular las colecciones que almacenemos en nuestra base de datos ya que presentarán los mismos atributos que hayamos definido en nuestras colecciones. De esta forma, los controladores interactuarán directamente con estos modelos y los cambios se verán reflejados directamente en la base de datos. Para la construcción de los esquemas se hace uso de la librería *mongoose* para NodeJS.

```
const ingredienteSchema = new Schema(
    {
        commons: [],
        curated: {type: Boolean},
        curated_at: {type: Date},
        curated_by: {type: String},
        img: {
            url: String
        },
        last_updated: Boolean,
        name: String,
        needs_revision: Boolean,
        num_recetas: Number,
        recipes: [],
        slug: String,
        usda: {
            code: String,
            url: String
        },
        validated: Boolean
    }
);
```

Figura 24. Ejemplo de esquema para la entidad Ingrediente

En segundo lugar, y antes de empezar a construir los controladores que actuarán sobre los modelos creados, definiremos las rutas para la **API RESTful**. Realizar este paso antes de seguir con el desarrollo de los demás componentes del patrón nos ayudará a plantear las consultas que deberemos realizar a la base de datos. La codificación de estas rutas se basa en la especificación de una cadena de texto (la ruta), el método por el cual se accede a dicha ruta (GET, POST, PUT, DELETE) y la función que deberá ejecutar el controlador correspondiente.

```
'use strict';

const mensaje = require('../../controllers/mensaje');

function mensajeRoute(app) {
    app.route('/api/mensajes/:idUser').get(mensaje.getByUserId);
    app.route('/api/mensaje').post(mensaje.createMensaje);
}
module.exports = mensajeRoute;
```

Figura 25. Ejemplo de API RESTful

Seguidamente, deberemos crear los **Controladores** para que interactúen con los modelos y así poder guardar y mostrar la información de la base de datos. Estos controladores serán ejecutados por el servidor NodeJS y están compuestos por funciones que hacen llamadas a métodos de los ficheros que se encargan de ejecutar consultas sobre la base de datos y devuelven, mediante respuestas HTTP, los resultados de dichas consultas.

```
'use strict';

const co = require('co');
const mensajeBD = require('../repository/mensaje');

function createMensaje(req, res) {
    co(function* () {
        const parametros = req.body;
        const mensaje = {
            mensaje: parametros.mensaje,
            origen : parametros.origen,
            destino: parametros.destino,
            fecha  : parametros.fecha
        };

        const result = yield mensajeBD.createMessage(mensaje);
        res.send({
            mensaje_creado: result
        })
    })
        .catch((err)=> {
            res.send(err);
        });
}
```

Figura 26. Ejemplo de función de un controlador

Con el fin de obtener una estructura que facilite el mantenimiento de la aplicación y futuras ampliaciones en su funcionalidad, se decidió separar las consultas a la base de datos de los controladores. Así, en caso de querer modificar o añadir alguna operación, sólo será necesario efectuar dicho cambio en un solo fichero y de esta forma quedará un código más legible. Estos ficheros se encuentran en un paquete llamado *Repository* y, como se ha indicado anteriormente, se encargan de acceder a la base de datos y realizar todo tipo de consultas.

```
//Recuperamos las recetas de un usuario
exports.findByUser = function* (userId) {
    return RecetaModel.find({ idUser: userId}).populate('idUser').lean();
}

//Recuperamos las versiones de la receta con el id buscado
exports.findVersions = function* (recetaId) {
    return RecetaModel.findOne({_id: recetaId}, {versions: 1, _id:0}).sort({_id: -1}).populate('versions').lean();
}
```

Figura 27. Ejemplo de funciones en un fichero del paquete Repository

Llegados a este punto, tendríamos construida la parte del *back-end*, con lo que quedaría realizar la codificación de la parte *front-end*. Para ello, el primer paso será el diseño de las factorías que se emplearán para conectar con la parte del servidor y obtener los datos que mostraremos en las vistas. Estas factorías son módulos desarrollados con AngularJS y que se componen de funciones que se encargan de hacer peticiones a nuestra API en la parte *back-end*.

Es importante entender que, en esta parte, no se define ninguna ruta, sino que se especifican las rutas como argumentos para los métodos de la directiva *$http* de AngularJS.

```
angular.module('mensajeService', [])
    .factory('Mensajes', function($http) {
        return {
            getByUser: function (idUser) {
                return $http.get('/api/mensajes/'+ idUser);
            },
            create: function (mensaje) {
                return $http.post('/api/mensaje', mensaje);
            }
        }
    });
```

Figura 28. Ejemplo de factoria desarrollada con AngularJS

Posteriormente y, a medida que se vayan diseñando, será necesario describir para cada interfaz la ruta mediante la cual se accederá a dicha interfaz y el controlador que tendrá asociado.

Las rutas para la parte del cliente (*front-end*) se encargarán de renderizar en cada caso la vista correspondiente y, si es necesario, realizar cualquier petición a la base de datos. A pesar de tratarse de rutas para la parte que el cliente visualizará, se definen mediante el uso del *framework* ExpressJS en la parte del proyecto que corresponde al servidor. Así, una vez se arranca el servidor, la aplicación queda a la espera de recibir alguna petición a una de estas rutas para redireccionar al usuario hacia la sección de la página correspondiente.

```
// About
app.route('/about').get(function (req, res) {
  res.render('./usuario_anonimo/about',{
    user: req.user
  });
});

// Home
app.route('/').get(        function (req, res) {
  res.render('./usuario_anonimo/index', {
    user: req.user,
  });
});
app.route('/index').get(        function (req, res) {
  res.render('./usuario_anonimo/index', {
    user: req.user,
  });
});
```

Figura 29. Ejemplo de rutas definidas con ExpressJS

Cada interfaz consistirá en un archivo en formato *jade* en el cual cargaremos el controlador AngularJS que tendrá asociado y que se encargará de estructurar la información de forma visual. En estos ficheros usaremos las potentes directivas y etiquetas que nos proporciona el *framework* Angular y que nos facilita cosas como la repetición de secciones en caso de querer mostrar listados o la ocultación de partes si no queremos mostrarlas hasta que se cumpla cierta condición.

```
.container(ng-controller='misRecetasController' ng-init='init(#{JSON.stringify(user)}, #{JSON.stringify(page)})')
  .row
    .col-sm-12
      .panel.panel-default
        .panel-heading
          h4.panel-title
            a(href='#collapseB1', data-toggle='collapse')  Mis Recetas
        .panel-body#UserProfileForm.form-horizontal
          .row
            center(ng-show='cargando')
              img.img-circle(src='/images/loader.gif')
              |  Cargando...
            .col-sm-3(ng-repeat="receta in recetas")
              .item
                .item-ads-grid
                  .item-img-grid
                    a(ng-href='/detalle-receta/{{receta._id}}')
                      img.img-responsive.img-center(src='/images/no_recipe_small.jpg' ng-show="!receta.images[0]")
                      img.img-responsive.img-center(alt='{{receta.title}}', ng-src='/images/recetas/{{receta.images[0]}}
                  .item-title
                    a(ng-href='/detalle-receta/{{receta._id}}')
                      h4 {{receta.title}}
                  .item-meta
```

Figura 30. Ejemplo de código Jade/Pug

El último paso a realizar será la construcción del controlador que queramos asociar a la interfaz diseñada previamente. Los controladores para la interfaz se encargan de recoger los datos que se deseen mostrar mediante el uso de las funciones diseñadas en las factorías para luego ser mostrados en la interfaz, además de poder realizar cualquier actualización o creación de nuevos registros y crear funciones asociadas a algún elemento de la vista (Ejemplo: función que se ejecute al hacer *click* en un botón). Todos los controladores hacen uso de una función que se ejecuta al cargarse la vista en el navegador (función *Init*).

```
receteameApp.controller('enviarMensajeController', function ($scope, $http, $window, Users, Recetas, Notificaciones, Mensajes) {

    $scope.init = function (origen, receta) {
        $scope.origen = origen;
        $scope.destino = receta.idUser._id;
    }

    $scope.enviarMensaje = function () {
        if($scope.origen){
            if($scope.mensaje){
                $scope.mensajeParaEnviar = {
                    mensaje: $scope.mensaje,
                    origen : $scope.origen._id,
                    destino: $scope.destino,
                    fecha  : new Date(Date.now()).toLocaleString()
                };

                Mensajes.create($scope.mensajeParaEnviar)
                    .success(function (data) {
                        if(!data.err){
                            toastr.success('Has enviado un mensaje a este usuario!');
                            $scope.mensaje = "";
                            Notificaciones.create({
                                cabecera: "Tienes un nuevo mensaje",
                                mensaje: $scope.origen.username + " te ha enviado un mensaje. Revisa tu conversación con " + $scop
```

Figura 31. Ejemplo de controlador para una vista

3.4.2 Tecnologías empleadas

Las tecnologías implicadas en la implementación de este proyecto son los lenguajes de programación y *frameworks* que servirán para crear tanto el *back-end* como el *front-end*. Empezaremos hablando del paquete de desarrollo MEAN [19], pues es la base para la implementación de esta plataforma y seguiremos describiendo herramientas que trabajan en conjunto y han sido necesarias para lograr el objetivo del desarrollo.

3.4.2.1 *Framework MEAN*

MEAN es un framework para el desarrollo de aplicaciones y páginas web dinámicas, que están basadas en el popular lenguaje de programación Javascript. MEAN son las siglas que aluden al cuarteto para la creación de aplicaciones web: MongoDB, ExpressJS, AngularJS y NodeJS.

- **MongoDB:** Sistema de base de datos NoSQL orientado a documentos, desarrollado bajo licencia AGPL [22] como *software* libre.
- **ExpressJS:** Proporciona la funcionalidad básica para atender y recibir peticiones en un servidor.
- **AngularJS:** Es un *framework* de Javascript de código abierto, mantenido por **Google**, especialmente indicado para crear páginas web SPA (*Single Page Application*). Empleado en nuestra aplicación para desarrollo *front-end*, que nos permite gestionar las vistas del proyecto, así como interaccionar con el servidor.
- **NodeJS:** Se trata de un proyecto de *software* libre (licencia MIT [23]) que permite la ejecución de código Javascript en el lado del servidor de forma muy eficiente.

3.4.2.2 *MongoDB*

MongoDB es un sistema de bases de datos NoSQL orientado a ficheros y desarrollado bajo licencia AGPL como *software* libre. Guarda estructuras de datos en documentos BSON (*Binary JavaScript Object Notation*) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Este sistema se caracteriza por ser: escalable, ya que posee mecanismos internos que permiten ir aumentando la potencia de procesamiento a medida que los requerimientos de datos suben; y flexible, puesto que prescinde de un esquema inicial definido. MongoDB puede considerarse un buen competidor de las bases de datos tradicionales gracias a su velocidad de cómputo. Los datos se encuentran guardados en objetos clave-valor e internamente a cada objeto le asocia un id para realizar consultas por índice por defecto, en caso de que el usuario no cree índices personalizados.

Los inconvenientes de MongoDB residen en la inexistencia de algunos mecanismos útiles de las bases de datos relacionales. Sobre una base de

datos no relacional como MongoDB cabe destacar que: no están presentes las transacciones, solo se garantizan operaciones atómicas a nivel de documento, una única edición de documento por consulta e inexistencia de métodos o llamadas *joins* para consultar datos relacionados en dos o más colecciones.

3.4.2.3 ExpressJS

Express es un *framework* por encima de Node.js que permite crear servidores web y recibir peticiones por protocolo HTTP o HTTPS de una manera sencilla. Se considera el estándar de facto de los desarrollos en NodeJs, es modular con lo que permite la conexión con otros módulos para crear mediante enlazado de módulos aplicaciones de gran tamaño. Simplifica en gran medida la creación de APIs REST, tecnología también empleada en el proyecto.

3.4.2.4 AngularJS

AngularJS es un *framework* basado en JavaScript para la parte cliente o *front-end* de una aplicación web, que respeta el paradigma Modelo-Vista-Controlador y permite la interacción de los usuarios con las vistas, sin necesidad de recargado de páginas, de manera más o menos sencilla. Facilita la interconexión con el backend. Es un proyecto mantenido por Google y que actualmente está muy en auge.

3.4.2.5 NodeJS

NodeJS es un entorno de programación en JavaScript para el *back-end* basado en el motor V8 de JavaScript del navegador Google Chrome, orientado a eventos y no bloqueante, lo que lo hace muy rápido a la hora de crear servidores web y emplear tiempo real. Fue creado en 2009 y aunque aún es joven, las últimas versiones lo hacen muy robusto, además de la gran comunidad de desarrolladores que posee. No sólo se utiliza del lado del servidor, se ha extendido tanto que se emplea en Stylus, un preprocesador CSS, en Grunt un gestor de tareas basado en JavaScript, etc.

3.4.2.6 Jade/Pug

Jade, actualmente conocido como Pug, es una herramienta de generación de código HTML a partir de ficheros *jade*, se fundamenta en una sintaxis más simple que a la que nos ha acostumbrado HTML en la que se evita el proceso de cerrar directivas o emplear corchetes. En este caso el código fuente se va escribiendo de manera secuencial y es el propio compilador el que genera el HTML a partir del código jade escrito por el desarrollador. Cada indentado equivale a una directiva nueva y cada primera palabra de cada directiva codifica que directiva es.

Hemos empleado Jade por:

- Simplicidad del código.
- Menor número de líneas.
- Facilidad de lectura.
- No admite malos indentados.
- El código que no cumple las reglas de estilo no compila.

3.4.2.7 Npm

Npm es el gestor de paquetes *back-end* de la tecnología Node. A lo largo del desarrollo de una aplicación necesitamos recurrir a funcionalidades de terceros, para evitar reinventar la rueda cada vez. Estas piezas *software* se pueden controlar mediante npm (manejador de paquetes Node) para mantenerlas actualizadas.

En el caso del desarrollo llevado a cabo en el contexto del proyecto se ha hecho uso de más de 40 paquetes, entre ellos podemos destacar:

- **Multer**: Librería para gestionar la subida de ficheros.
- **Bcrypt**: Para encriptar y desencriptar las contraseñas antes de persistirlas.
- **Bower**: Para hacer uso del gestor de paquetes frontend.
- **Co:** Para las promesas de javascript.
- **Express**: Para el uso de la tecnología ExpressJs.
- **Jade**: Para compilar a HTML los archivos jade.
- **Mongoose**: Para el uso de la base de datos.
- **Nodemailer**: Para el envío de correos.
- **Passport**: Para el login y la creación de la cookie.

- **Passport-facebook**: Para el acceso mediante las credenciales de *Facebook*.
- **Ng-table:** Para el uso de tablas con AngularJS.

3.4.2.8 Bower

Este paquete es similar a npm pero para el *front-end*. En este caso las utilidades que hemos empleado son muchas menos que las usadas con npm. Podemos destacar entre los 8 paquetes instalados: **angular**, para poder emplear las funcionalidades del *framework* AngularJS. **Toastr**, para las notificaciones emergentes o **Bootstrap**, para realizar un diseño *responsive*.

3.4.2.9 Git

Es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente.
Git ofrece la posibilidad de almacenar en un servidor o una carpeta en local una réplica de lo que se quiere versionar. Cada vez que se quiere subir una modificación, Git no sobrescribe los archivos sino que busca las diferencias entre la nueva versión y la versión anterior y las plasma asociándole a esta nueva versión un identificador único, con lo que, en caso de que se quisieran revertir, solo habría que ir al estado anterior y Git tendría conocimiento de cual es este estado.

3.4.3 Herramientas empleadas

Para la implementación del código, tanto de la parte *back-end* como de la parte *front-end*, se ha empleado **WebStorm**. Esta aplicación es el entorno de desarrollo para Javascript desarrollado por **JetBrains**.

Este entorno permite el desarrollo de la capa del cliente (*front-end*) en Javascript pero también de la capa del servidor con NodeJS. Además, está integrado con Git y proporciona soporte para trabajar con tecnologías como Angular, VueJS, React, etc.

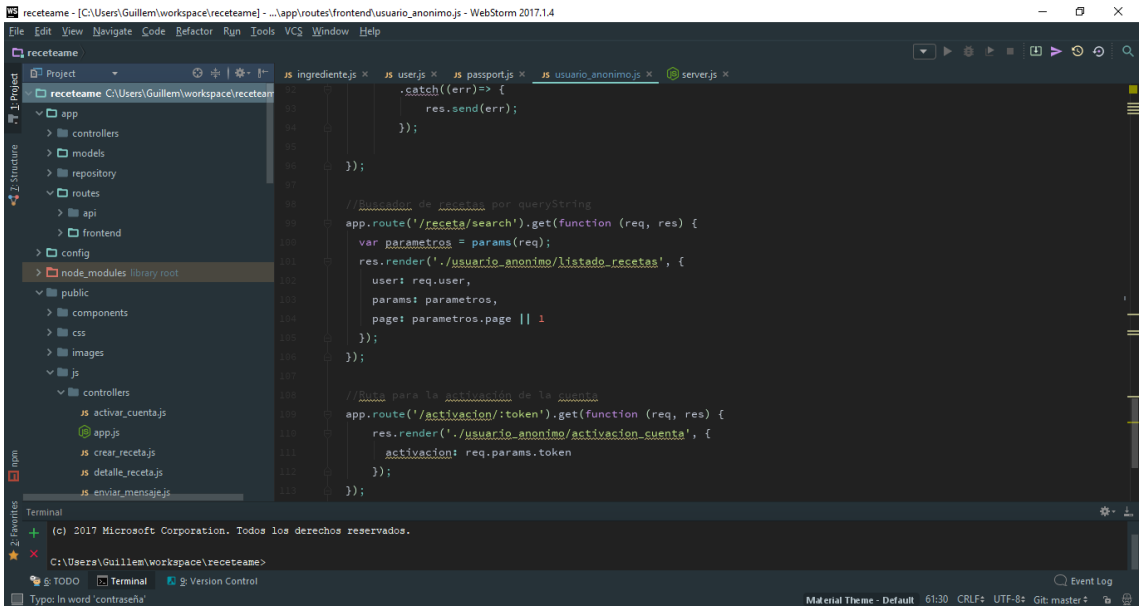

Figura 32. Entorno de desarrollo WebStorm

Para la gestión de la base de datos, se ha empleado la aplicación **Robomongo**. Esta herramienta nos permite gestionar **gráficamente** nuestras bases de datos en MongoDB. También, ofrece la posibilidad de mantener múltiples conexiones a distintas bases de datos, separadas por pestañas y permite realizar cualquier tipo de consulta sobre ellas.

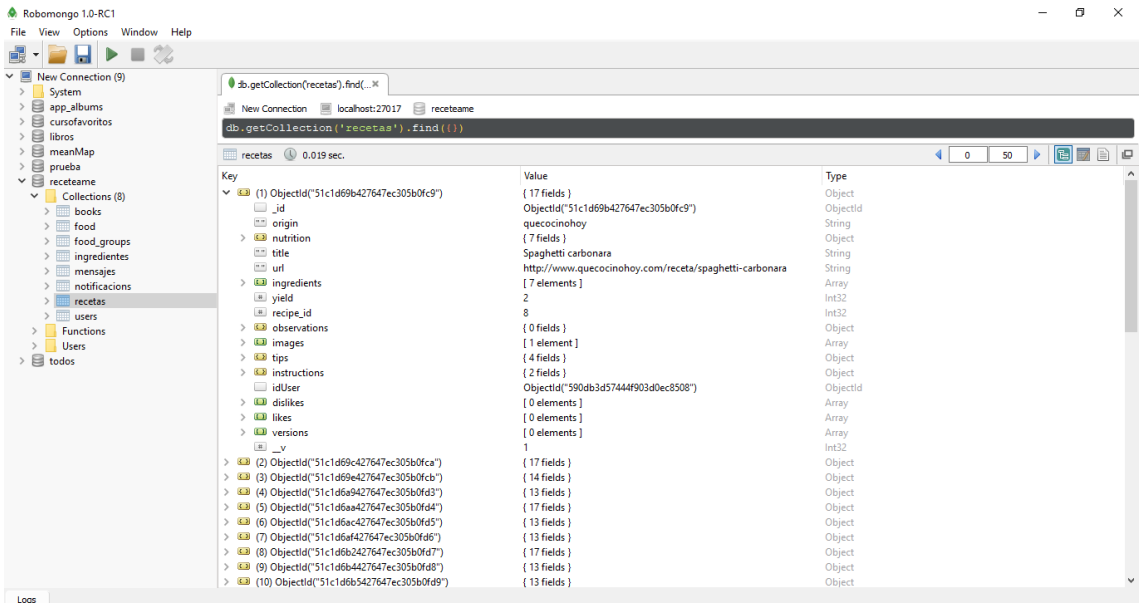

Figura 33. Gestor de bases de datos Robomongo

Durante el desarrollo, también se ha hecho uso de la aplicación **Postman**, que nos permite probar servicios web fácilmente, indicando la dirección, el método HTTP (POST, GET, PUT, DELETE) y los parámetros de la petición.

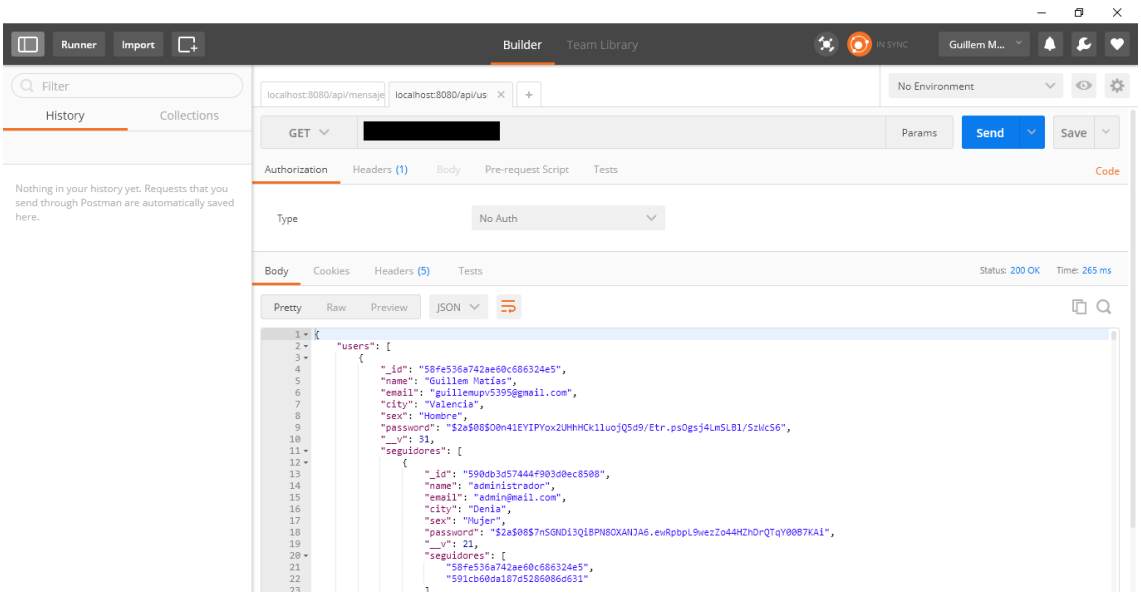

Figura 34. Aplicación Postman

3.4.4 Estructura de directorios

En esta sección, conoceremos la estructura del proyecto. Para ello mostraremos las diferentes carpetas que forman la aplicación, describiendo mínimamente la utilidad de cada directorio.

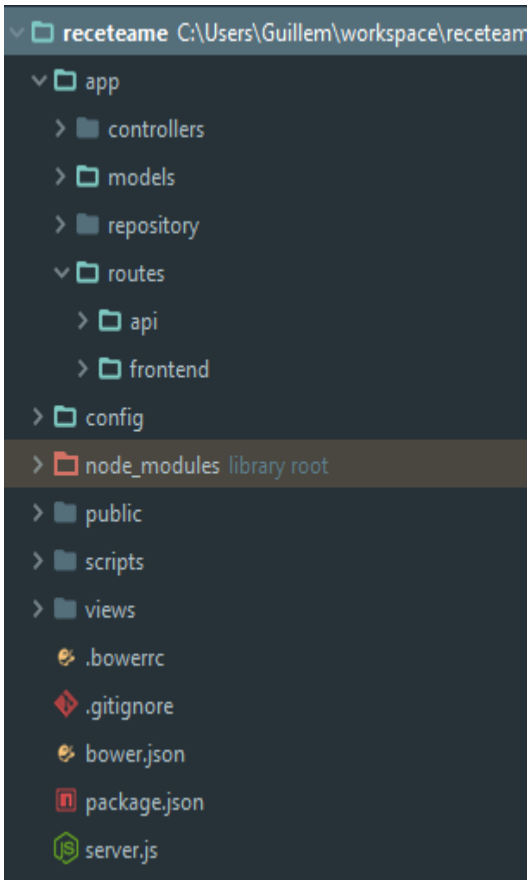

Figura 35. Estructura de directorios

Como se puede observar en la figura adjunta, el proyecto está formado por seis directorios principales y cinco archivos.

Con respecto a los ficheros que se muestran en la raíz del proeycto:

- **.bowerrc:** Fichero de configuración de la herramienta *Bower*.
- **.gitignore:** Fichero de configuración del gestor de versiones *Git*.
- **bower.json:** Fichero que contiene el historial de paquetes descargados con *Bower.*
- **package.json:** Fichero de datos relativos al proyecto y los módulos para su instalación mediante *Npm.*
- **server.js:** Este fichero se encarga de arrancar el servidor, así como de su configuración.

En relación a los directorios:

- **app:** Contiene todo lo relativo al desarrollo *back-end*, como los modelos, los ficheros encargados de gestionar las rutas y los controladores para los modelos.
- **config:** Incluye ficheros de configuración para la autenticación del usuario con *Facebook*, así como el fichero encargado de gestionar el inicio de sesión y la ruta donde escucha el servidor de la base de datos.
- **node_modules:** Es el directorio donde se instalan los módulos descargados por *Npm*.

- **public:** Contiene todo lo referente al *front-end*. En este directorio se almacenan los paquetes descargados con *Bower*, además de imágenes, ficheros CSS y todos los controladores y factorías desarrollados con AngularJS.
- **scripts:** Este directorio almacena un fichero que se usó para asociar las recetas proporcionadas por la base de datos heredada a un usuario por primera vez.
- **views:** Contiene todas las vistas que forman las diferentes interfaces de usuario.

3.5 Pruebas

Como hemos indicado en el proceso de desarrollo *software*, es necesario probar y validar el producto antes de darlo por finalizado.

El correcto funcionamiento de las distintas funcionalidades que ofrece el proyecto, han sido validadas mediante pruebas manuales durante la fase de implementación del código. Con cada cambio, la aplicación era reiniciada en el servidor web local y, a continuación, probada mediante la interfaz. Este tipo de prueba se realizó para cada caso de uso. Además, se diseñaron pruebas unitarias automatizadas para probar el correcto funcionamiento de la API desarrollada usando dos librerías para el entorno NodeJS, *mocha.js y chai.js*.

Mocha es un *framework* de pruebas para Javascript que se ejecuta en NodeJS y nos permite crear tanto test síncronos como asíncronos de una forma muy sencilla. Nos proporciona muchas utilidades, entre ellas la posibilidad de emplear cualquier librería de aserciones (*chai.js* en nuestro proyecto).

De esta forma, con *mocha.js* ya dispondríamos de un entorno para la creación de nuestros test, pero, ¿Cómo probamos de forma automatizada llamadas HTTP? Es más, ¿Cómo probamos que una petición GET nos está devolviendo el objeto JSON que esperábamos? Necesitamos una librería de aserciones, aquí es donde entra *chai.js*. Esta librería nos permite elegir el estilo de que resulte más legible a la hora de diseñar las pruebas: *should, expect, assert*.

Una vez establecido el entorno que emplearemos para realizar nuestras pruebas deberemos diseñarlas. Como la funcionalidad que ofrecen las distintas APIs desarrolladas (Recetas, Usuarios, Mensajes, etc.) son similares, describiremos un conjunto de pruebas diseñadas para la API de los usuarios.

El primer caso que se ha tenido en cuenta es la creación de un usuario. Debido a que en el diseño del modelo para la colección de usuarios especificamos que el campo *email* debía ser necesario para la creación del registro, especificaremos un caso de prueba en el cual intentaremos crear un usuario sin ese campo. El resultado esperado deberá ser que el sistema no dará de alta dicho usuario.

```
describe('/POST user', () => {
    it('No debería crear un usuario sin email', (done) => {
        let user = {
            name: "Usuario Test",
            username: "user_to_test",
            seguidores: [],
            seguidos: [],
            favoritas: []
        }
        chai.request(server)
            .post('/api/user')
            .send(user)
            .end((err, res) => {
                res.should.have.status(200);
                res.body.should.be.a('object');
                res.body.should.have.property('errors');
                res.body.errors.should.have.property('email');
                res.body.errors.email.should.have.property('kind').eql('required');
                done();
            });
```

Figura 36. Test para la creación de un usuario sin email

Como vemos en la figura 36, los diseños de los casos de prueba se realizan de manera legible. Este caso especifica que la respuesta HTTP a la petición de crear un usuario debería cumplir:

- Tener un estado 200.
- Ser un objeto JSON.
- Una de las propiedades de la respuesta debería llamarse *errors.*
- La propiedad *errors* debería contener el campo que falta (*email*) como propiedad.
- *Email* debería tener la propiedad *kind* igual a *required* y así poder destacar la razón por la que se ha obtenido una respuesta negativa del servidor.

Seguidamente, se diseña el test para la creación de un usuario con los campos requeridos.

```
it('Debería crear un usuario', (done) => {
    let user = {
        name: "Usuario Test",
        username: "user_to_test",
        email: "mail@test.com",
        seguidores: [],
        seguidos: [],
        favoritas: []
    }
    chai.request(server)
        .post('/api/user')
        .send(user)
        .end((err, res) => {
            res.should.have.status(200);
            res.body.should.be.a('object');
            res.body.should.have.property('name');
            res.body.should.have.property('username');
            res.body.should.have.property('email');
            done();
        });
```

Figura 37. Test para la creación de un usuario con los campos requeridos

Aquí podemos observar que el objeto usuario que creamos sí contiene el campo *email*, con lo cual a la hora de crearlo el sistema deberá añadir el registro a la base de datos sin ningún problema. Este caso deberá responder con un estado 200 y un objeto con los campos *name, username* y *email*, correspondientes al usuario creado.

El último caso que se pretende describir prueba el funcionamiento de obtener un usuario de la base de datos especificando su identificador.

```
describe('/GET/:id user', () => {
    it('Debería obtener el usuario con el id especificado', (done) => {
        let user = new User({ name: "Usuario test2", username: "Test2", email: "mail2@test.com"});
        user.save((err, user) => {
            chai.request(server)
                .get('/api/user/' + user._id)
                .send(user)
                .end((err, res) => {
                    res.should.have.status(200);
                    res.body.should.be.a('object');
                    res.body.user.should.have.property('name');
                    res.body.user.should.have.property('username');
                    res.body.user.should.have.property('email');
                    res.body.user.should.have.property('_id').eql(user._id.toString());
                    done();
                });
        });
```

Figura 38. Test para la obtención de un usuario especificando su id

En él, se crea un usuario en la base de datos de forma correcta y, posteriormente, se realiza la petición de obtener dicho registro mediante su identificador. El sistema debería responder correctamente y debería devolver un objeto con los datos del usuario que se creó para probar la funcionalidad.

Para todos los casos anteriormente descritos, el sistema pasa los test.

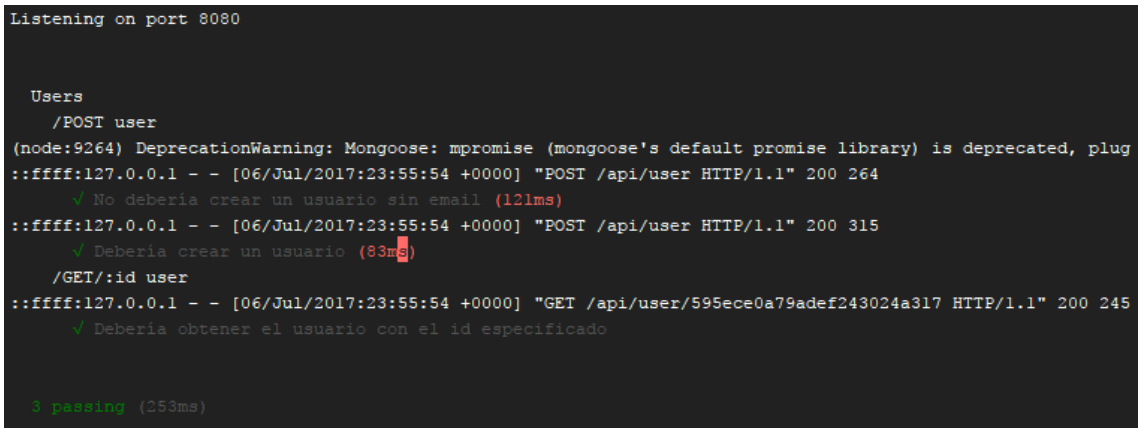

Figura 39. Resultados de las pruebas para la API de usuarios

Por otra parte, también es necesario validar las distintas interfaces que presenta la plataforma basándonos en estándares de **usabilidad** y **accesibilidad.**

Respecto a la usabilidad, debemos tener en cuenta [20] **:**

- **Objetivos del sitio web:** ¿están bien definidos? ¿Los contenidos ofrecidos se corresponden con estos objetivos?
  Sí, gracias al análisis previo y a haber seguido una metodología a la hora de llevar a cabo el desarrollo, los contenidos que ofrece la aplicación se corresponden con los planteados inicialmente.
- **URL:** ¿tiene una URL correcta, clara y fácil de recordar?
  La URL de la red social desarrollada en este proyecto corresponde a su nombre, por lo que sería fácil de recordar y totalmente clara (http://receteame.com).
- **Estructura del sitio web:** ¿está orientada al usuario?
  Sí, mediante una barra de navegación en la parte superior donde el usuario puede encontrar los enlaces a opciones de interés para su cuenta (sus recetas, recetas favoritas, perfil, mensajes) y con una zona principal en el centro donde se mostrarán los contenidos.
- **Diseño:** ¿es coherente y reconocible?
  Para el diseño de las distintas interfaces se ha seguido la misma estructura y con la misma paleta de colores.

- **Logo:** ¿es significativo y suficientemente identificable?
  El logo aparece siempre en la misma posición de las interfaces ya que la barra de navegación permanece estática durante toda la navegación. Además, presenta el color más representativo de la aplicación y un tamaño de letra que lo hace suficientemente reconocible.
- **Información del sitio:** ¿se ofrece algún tipo de información sobre el sitio web?
  Se pode a disposición del usuario una página donde se detallan los términos y condiciones de uso de la plataforma.
- **Navegación:** ¿existen páginas huérfanas?
  No, todas las páginas de la aplicación están enlazadas a otras.
- **Sobrecarga informativa:** ¿se evita?
  Sí, manteniendo la suficiente separación entre elementos para distinguirlos y separando la información en secciones.
- **Ruido visual:** ¿la interfaz es clara y limpia?
  Los contenidos mostrados son los estrictamente necesarios.
- **Mensajes informativos:** ¿se informa al usuario de lo que está ocurriendo?
  En acciones como el registro, la creación de una receta o con acciones no permitidas, se muestra una notificación emergente en la parte superior derecha de la pantalla lo suficientemente llamativa.

En cuanto a la accesibilidad, se han empleado herramientas *online* que evalúan la accesibilidad de un sitio web**.**

- Evaluador de accesibilidad web mediante Pautas de Accesibilidad para el contenido web 2.0 (WCAG 2.0): http://examinator.ws/

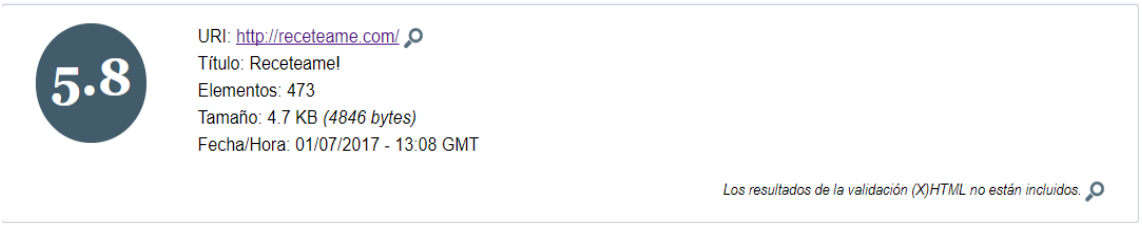

Figura 40. Nota del informe de accesibilidad

Para conseguir el aprobado en esta herramienta se han tenido que realizar acciones correctivas como por ejemplo la inclusión de texto alternativo en todas las imágenes o indicar en el etiquetado HTML el idioma de uso. Cabe destacar que la mayor parte de errores que remarca el test vienen dados por las etiquetas personalizadas del *framework* AngularJS.

Para finalizar con este apartado, cabe destacar que, al tratarse de una aplicación desplegada y completamente accesible por cualquier usuario a través de un navegador web, se han tenido en consideración todos los comentarios con sugerencias o en los que se señalaba algún error para posteriormente ser resuelto.

3.6 Despliegue

A pesar de no formar parte del proceso de desarrollo de *software*, la fase de despliegue es muy importante en una aplicación web ya que permite ponerla a disposición de los usuarios y, de esta forma, poder percibir si el producto es de su agrado o podría mejorarse cualquier aspecto. En este apartado describiremos los pasos que hemos llevado a cabo para realizar este proceso.

El grupo del DSIC, GTI-IA, dispuso un servidor bajo el dominio *receteame.com* donde poder realizar el despliegue, con una máquina Ubuntu 16.04 y un servidor web Nginx. Una vez autenticados en el servidor, el primer paso es instalar el entorno NodeJS, bajo el cual funciona la aplicación. El siguiente paso, será instalar MongoDB para la gestión de la base de datos y el gestor de paquetes para NodeJs (Npm).

Una vez instaladas esas herramientas, debemos clonar el repositorio donde hemos almacenado el proyecto al servidor para poder acceder a la aplicación. A partir de aquí, tendremos que instalar todos los paquetes que aparecen en los ficheros de módulos descritos en el apartado *Estructura de directorios* (package.json y bower.json).

Ahora que ya disponemos del proyecto y todo lo necesario para hacerlo funcionar, instalamos mediante Npm un gestor de procesos para aplicaciones NodeJS, llamado PM2. Este gestor nos proporciona una forma sencilla de gestionar y hacer que se ejecuten nuestras aplicaciones desarrolladas con NodeJS. Una vez instalado, mediante una sencilla instrucción arrancamos la aplicación en nuestro servidor local.

Llegados a este punto, sólo falta lanzar el proyecto al servidor web para que otros usuarios puedan acceder fácilmente. Para esto, tendremos que configurar el servidor Nginx como proxy hacia la aplicación NodeJS, modificando el archivo que se encuentra en la dirección */etc/nginx/sites-available/default* y añadir esta configuración [21]:

```
location / {
    proxy_pass http://localhost:8080;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_cache_bypass $http_upgrade;
}
```

Si nuestro servidor está disponible en *receteame.com*, accediendo a http://receteame.com a través de cualquier navegador web, se mandará una petición al archivo *server.js* (Véase apartado 3.4.3) que lanza el servidor en http://localhost:8080.

4. Conclusiones

En este capítulo presentaremos nuestras conclusiones agrupadas en tres secciones: Conclusiones del proyecto, trabajos futuros y valoración personal.

4.1 Conclusiones del proyecto

En cuanto a los objetivos planteados inicialmente, podemos afirmar que los hemos alcanzado.

Se ha desarrollado una aplicación completamente funcional y accesible por cualquier usuario a través de un navegador en la que podemos dar solución a la problemática planteada: crear un portal para una red social de recetas a partir de la cual poder recopilar información de carácter nutricional sobre los usuarios para, posteriormente, ser usada por el grupo **GTI-IA** en sus investigaciones. La plataforma ofrece la posibilidad de crear recetas, así como de permitir la interacción entre los distintos usuarios de la misma. Además, se han empleado tecnologías novedosas y presenta un diseño *responsive*, para su visionado en distintos tipos de dispositivos.

Como resultados del proyecto realizado podríamos destacar los siguientes:

- Se ha conseguido adaptar y ampliar la base de datos existente previamente, asociando las recetas que ya existían con un usuario y añadiendo las nuevas colecciones que se emplean a la base de datos.
- Se ha implementado un servidor y una API RESTful capaz de gestionar las peticiones que realizan los clientes.
- Se ha creado una lógica capaz de ofrecer toda la funcionalidad planteada inicialmente a los usuarios.
- Se han diseñado interfaces sencillas y amigables bajo la premisa de un diseño adaptable a distintos tipos de dispositivos (diseño *responsive*).
- Se ha llevado a cabo un proceso de despliegue que garantiza el acceso de cualquier usuario a la aplicación mediante cualquier navegador web y a través de la dirección http://receteame.com.

4.2 Trabajos futuros

Como se ha indicado a varias ocasiones, a lo largo de la memoria, la realización de este proyecto abre las puertas a la implementación de una herramienta capaz de recomendar recetas basándose en los gustos de los usuarios y teniendo en cuenta cualquier tipo de restricción alimentaria. También, en un futuro, *Receteame* será capaz de realizar predicciones de posibles intolerancias que presenten los usuarios gracias a un historial que podrá mantener dichos usuarios y en el cual llevarán un registro de todas sus comidas y cómo les han sentado.

Por otra parte, al tratarse de una aplicación desarrollada siguiendo un patrón de diseño, podrá ser ampliada con facilidad en caso de querer añadir más funcionalidades.

4.3 Valoración personal

Personalmente, el desarrollo de este proyecto, ha significado un reto que he afrontado con muchas ganas. La realización de este producto ha implicado una adquisición de conocimientos muy amplios ya que se ha desarrollado usando tecnologías propias del ámbito informático, pero no vistas en el Grado de Ingeniería Informática y también ha permitido repasar conceptos que sí se han visto durante el grado.

Por otra parte, cabe destacar que llevar a cabo el desarrollo de una aplicación web, cosa que a priori puede parecer trivial, implica profundizar en muchas áreas de conocimiento, tales como la gestión de proyectos, la ingeniería del *software*, la criptografía e incluso algunos aspectos legales. Si bien es cierto que cualquier persona podría llegar a desarrollar un producto en el ámbito web habiendo adquirido algunos conocimientos básicos, no contemplará todo aquello que debe para obtener un producto final completo, fiable y seguro. La participación de un profesional siempre es imprescindible y beneficia tanto al cliente de un producto como a todos los usuarios que puedan llegar a utilizarlo.

Bibliografía

1. etsinf. (20 de diciembre de 2011). *Blog Historia de la Informática*. Obtenido de http://histinf.blogs.upv.es/2011/12/20/redes-sociales/

2. WeAreSocial. (26 de enero de 2016). *Global Digital Snapshot*. Obtenido de https://www.slideshare.net/wearesocialsg/digital-in-2016/7-wearesocialsg_7GLOBAL_DIGITAL_SNAPSHOTINTERNETUSERSTOTALPOPULATIONACTIVE_SOCIALMEDIA

3. Mander, J. (2015). *blog.globalwebindex.net. Internet users have average of 5.54 social media accounts*. Obtenido de http://blog.globalwebindex.net/chart-of-the-day/internet-users-have-average-of-5-54-social-media-accounts/

4. Wikipedia. (13 de mayo de 2017). *Tipología de las redes sociales en Internet*. Obtenido de https://es.wikipedia.org/wiki/Red_social#Tipolog.C3.ADa_de_redes_sociales_en_Internet

5. TIOBE. (junio de 2017). *TIOBE Index for June 2017*. Obtenido de https://www.tiobe.com/tiobe-index/

6. Gellert, L. (1 de agosto de 2012). *Laurence Gellert's Blog. What is a Full stack developer?* Obtenido de http://www.laurencegellert.com/2012/08/what-is-a-full-stack-developer/

7. Apuntes de Ingenieria del software.Grado en Ingeniería Informática.Universidad Politécnica de Valencia.(Curso 2015-2016).Tema 1: Introducción a la ingeniería del software

8. Apuntes de Análisis y especificación de requisitos. Grado en Ingeniería Informática.Universidad Politécnica de Valencia.(Curso 2016-2017). Tema 4: Especificación de requisitos

9. Puromarketing. (18 de marzo de 2014). *¿Cómo influyen las redes sociales en los negocios?* Obtenido de http://www.puromarketing.com/42/19505/como-influyen-redes-sociales-negocios.html

10. Wikipedia.(9 de marzo de 2017). Interfaz de programación de aplicaciones. Obtenido de https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

11. Wikipedia.(9 de marzo de 2017). Interfaz de programación de aplicaciones. Obtenido de https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

12. Méndez, G.(22 de octubre de 2008). Especificación de requisistos según el estándar de IEEE 830. Obtenido de https://www.fdi.ucm.es/profesor/gmendez/docs/iso809/ieee830.pdf

13. Nielsen, J. (1993). Response Times: The 3 Important Limits. Nielsen Norman Group. Obtenido de https://www.nngroup.com/articles/response-times-3-important-limits/

14. Regadas, A. (12 de diciembre de 2016). Getting started with Pug template engine. Obtenido de http://www.hostilejourney.com/getting-started-with-pug-template-engine/

15. Kiessling, M. (2012). The Node beginner book.

16. Mardan, A. (2014). Express.js Guide: The comprehensive book on Express.js.

17. Basalo, A. (28 de agosto de 2014). Qué es AngularJS. Obtenido de https://desarrolloweb.com/articulos/que-es-angularjs-descripcion-framework-javascript-conceptos.html

18. Basalo, A. (15 de diciembre de 2014). Factorías (factory) en AngularJS. Obtenido de https://desarrolloweb.com/articulos/factorias-factory-angularjs.html

19. Holmes, S. (2015). Getting MEAN with Mongo, Express, Angular, and Node.

20. Hassan, Y. (30 de marzo de 2003). Guía de evaluación heurística de sitios web. Obtenido de http://www.nosolousabilidad.com/articulos/heuristica.htm

21. Bearnes, B. (1 de noviembre de 2016). How to set up a Node.js application for production on Ubuntu 16.04. Obtenido de https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-16-04#prerequisites

22. Wikipedia. (19 de marzo de 2017). GNU Affero general public license. Obtenido de https://es.wikipedia.org/wiki/GNU_Affero_General_Public_License

23. Wikipedia. (16 de junio de 2017). Licencia MIT. Obtenido de https://es.wikipedia.org/wiki/Licencia_MIT