
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA - INGENIERÍA DEL SOFTWARE**

# LA GLORIA

**Realizado por**

MARÍA DOLORES COSTA ZAFRA (31007772-T)

JUAN MANUEL LÓPEZ PAZOS (29499481-A)

**Dirigido por**

MARINA RAMOS SERRANO

PABLO FERNÁNDEZ MONTES

**Departamentos**

CREATIVIDAD Y NUEVAS TECNOLOGÍAS

LENGUAJES Y SISTEMAS INFORMÁTICOS

Sevilla, septiembre de 2014

# Tabla de contenidos

# Índice de tablas

# Índice de ilustraciones

# BLOQUE I: INTRODUCCIÓN

# RESUMEN

Este documento contiene la documentación del trabajo de fin de grado conjunto titulado **La Gloria**, el cual ha sido desarrollado conjuntamente por María Dolores Costa Zafra (alumna de publicidad) y Juan Manuel López Pazos (alumno de informática) y bajo la supervisión de Marina Ramos Serrano y Pablo Fernández Montes.

Este proyecto nace dentro de **SINERGIA**, una iniciativa que busca innovar en el concepto de trabajo de fin de grado, de forma que alumnos de distintas disciplinas trabajan juntos por un objetivo común.

Este proyecto tiene como objetivo desarrollar una aplicación web para la empresa de caramelos **La Gloria S.L.**, la cual dispone de una web pero necesita una nueva que sea adecuada a las tecnologías actuales.

En cuanto a metodología se usará **Scrum**, aunque de una forma modificada para que todos los miembros del grupo de trabajo que no la hayan usado anteriormente se puedan adaptar de la forma más rápida posible.

Esta metodología ágil será de mucha utilidad, ya que **La Gloria S.L.** es una empresa real y, por tanto, nuestros clientes cambiarán de opinión con mucha frecuencia. Esto supone un alto número de cambios en la elicitación de requisitos.

Además, la documentación generada se estructurará en bloques para facilitar la búsqueda de la misma, ordenada según la fase del proyecto en la que se haya generado.

Para diseñar una aplicación web muy dinámica se usarán tecnologías muy recientes como **Node.js** o **AngularJS**, de modo que la aplicación web estará desarrollada casi al completo en **JavaScript**.

La aplicación resultante estará basada en la filosofía **REST**, la cual está basada en el concepto de Recurso. Estos recursos serán manipulados mediante una **API REST**.

La base de datos usada para almacenar la información será **MongoDB**, ya que todas las tecnologías operan a la perfección con ficheros en formato **JSON**.

Para conseguir un diseño muy estético y que se adapte a cualquier dispositivo se usará **Bootstrap**.

Dicho todo esto, el reto de este proyecto es innovar desde todos los puntos de vista.

# 1. INTRODUCCIÓN

**La Gloria** es un proyecto que surge de **SINERGIA**, una iniciativa que ha nacido este año con el fin de innovar en el concepto de trabajo de fin de grado.

El objetivo de **SINERGIA** es que alumnos de distintas disciplinas y que no se conocen de nada trabajen juntos en un proyecto ambicioso con el único objetivo de conseguir las metas propuestas.

En este caso serán dos alumnos, uno de Publicidad y otro de Ingeniería del Software, son los que tendrán que trabajar en equipo para tener éxito en sus trabajos de fin de grado.

En mi caso, la innovación comienza por las tecnologías usadas. Estas han sido:

- **MongoDB**: base de datos no relacional orientada a documentos.
- **Node.js**: lenguaje JavaScript para el servidor.
- **AngularJS**: un potente framework JavaScript desarrollado por Google, bastante reciente y cuya popularidad sube por día.
- **JADE**: un motor de plantillas HTML que sirve para agilizar la implementación de vistas de una aplicación web.

Ninguna de estas tecnologías (salvo **MongoDB** ligeramente) se estudia en la titulación que estoy cursando, por lo que el mérito de elegirlas es notorio y la ambición importante, justo lo que se busca en **SINERGIA**.

Como metodología de desarrollo se usará **Scrum**, una metodología de desarrollo ágil con la que sí estoy familiarizado, aunque se adaptará un poco a las necesidades del equipo de trabajo, ya que algunos no la han usado y este proyecto es de una tipología muy marcada.

A pesar de usar **Scrum**, la estructura de la documentación generada que se muestra en este documento está ordenada por bloques según la fase del desarrollo en la que ha sido generada. Esto es una característica de las metodologías tradicionales, lo cual facilitará la lectura de este documento.

Los bloques que se definen en este documento son:

- **Introducción**: bloque en el que se pone en contexto el trabajo que aquí se desarrolla.

- **Metodología**: donde se detallan las metodologías analizadas, la metodología de trabajo usada, cómo se ha aplicado y la planificación realizada en el proyecto.
- **Análisis**: en el que se realiza la elicitación de requisitos al completo, además de realizar un estudio de las tecnologías usadas con el fin de comprender para qué sirven y cómo se usan.
- **Diseño**: para entender cómo está representada la información en el sistema a desarrollar.
- **Implementación**: por un lado se detalla la arquitectura, donde se especifican las herramientas que se han usado para la implementación del sistema y algunos conceptos básicos, y por otro lado se realizan una serie de pruebas para verificar el correcto funcionamiento de la aplicación desarrollada.
- **Manuales**: algo muy útil tanto para futuros usuarios de la aplicación como para futuros desarrolladores. A los últimos se les proporciona mucha información sobre la configuración del sistema para facilitar futuras extensiones en la web.
- **Conclusiones**: finalmente se especifican una serie de conclusiones desde distintos puntos de vista con el fin de analizar el transcurso del proyecto.

La siguiente imagen se corresponde a una de las vistas de la aplicación web desarrollada:

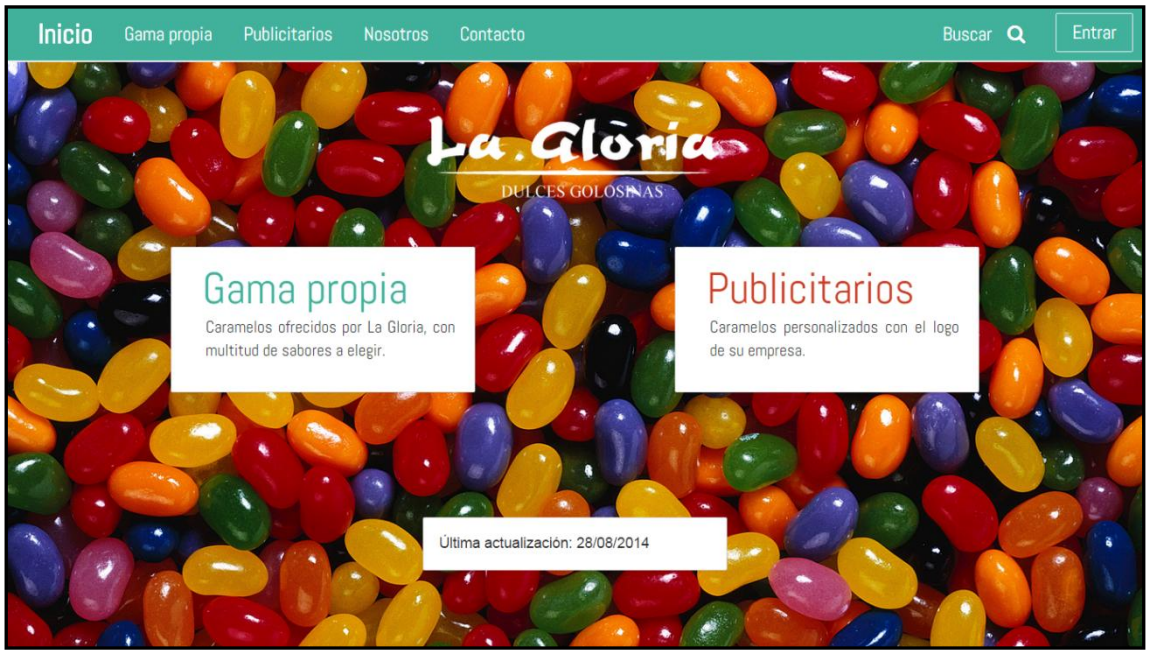



***Ilustración 1:*** *Captura de la página de inicio*

La aplicación web desarrollada está disponible en la siguiente dirección:

**http://caramelosla gloria.herokuapp.com**

# 1.1. Alcance

Con este proyecto se pretende desarrollar una aplicación web para **La Gloria S.L.**, de modo que le sirva a la empresa para mejorar su imagen, hacerse un hueco en Internet y ganar usuarios para aumentar sus ventas.

La web que resulte del trabajo realizado estará destinada a usuarios anónimos, a proveedores que quieran comprar productos de **La Gloria S.L.** y a uno o varios administradores de **La Gloria S.L.**, para que puedan moderar los contenidos de la aplicación web.

# 1.2. Objetivos

En este proyecto existen muchos objetivos, además de muchos puntos de vista. Por un lado vamos a analizar los objetivos del cliente, **La Gloria S.L.**, por otro lado analizaremos mis objetivos personales y para finalizar comentaremos los objetivos de este proyecto.

En cuanto al cliente, **La Gloria S.L.** tiene la necesidad de incrementar sus ventas, las cuales han decaído en los últimos años. Esa necesidad se traduce en tener un hueco Internet.

En la actualidad, nuestro cliente tiene una página web. Sin embargo, la página fue desarrollada en el año 2006 e implementada con Flash. Por tanto, la imagen de la empresa en Internet está muy obsoleta. Por este motivo, nuestro cliente necesita una nueva página web con la que llamar la atención de sus clientes y aumentar sus ventas.

En cuanto a mis objetivos personales yo quería realizar una página web para mi trabajo de fin de grado. En principio no tenía ninguna preferencia por ningún lenguaje, pero pensé que una buena opción sería **PHP**, ya que es el lenguaje de programación web que menos he utilizado en mi titulación.

Sin embargo, mi tutor (Pablo Fernández Montes) me planteó la posibilidad de utilizar tecnologías mucho más recientes, así como **Node.js** en el lado del servidor y **AngularJS** en el lado del cliente.

Después de realizar algunas pruebas con estas tecnologías acordamos que sería una buena apuesta y un reto de gran valor acorde a un trabajo de fin de grado, más aún dentro del proyecto **SINERGIA** donde prima la innovación.

Dicho esto, los objetivos de este proyecto no son más que realizar de forma conjunta una página web para una empresa real, innovando tanto en la forma de trabajar como en las tecnologías usadas para tal fin. De este modo, los alumnos tendremos la opción de trabajar con personas de distintas disciplinas y con las que no habíamos tenido ningún trato previo a la realización de este proyecto.

# 1.3. Justificación

Antes de decidirme a realizar este proyecto lo único que yo tenía claro era que mi trabajo de fin de grado iba a ser la realización de una página web. Las tecnologías que se usaran no eran un tema que me preocupara, pero sí tenía especial interés en implementar la página web con **PHP**, un lenguaje de programación web muy poco estudiado en la titulación del grado de ingeniería informática – ingeniería del software.

Sin embargo, a comienzos del curso académico 2013/2014, Mª del Carmen Romero Ternero y Pablo Fernández Montes – el que sería mi futuro tutor de trabajo de fin de grado – asistieron a una de mis clases de cuarto curso para hablar a todos los alumnos sobre el proyecto **SINERGIA**, el cual consistía en realizar trabajos de fin de grado de forma conjunta entre dos ó más alumnos de distintas titulaciones.

Después de informarme sobre todos los detalles me pareció una idea bastante original, y en cuanto vi la propuesta de este trabajo (la realización de una página web en **PHP** para una empresa de caramelos) decidí apostar por esta iniciativa. Además, la idea de realizar mi **TFG** de forma conjunta era algo que siempre había tenido en mente, por lo que el proyecto **SINERGIA** se presentaba como la ocasión ideal para ello.

Una vez asignados los trabajos y conformados los grupos de trabajo se llevaron a cabo las primeras reuniones, tanto entre los alumnos y sus tutores como todos juntos. En la primera reunión con mi tutor acordamos realizar la página web con lenguajes más recientes como **Node.js** para el servidor y **AngularJS** para el cliente. Eran tecnologías que yo no conocía ni había oído hablar de ellas. Por eso, tras realizar distintas pruebas durante una semana, decidimos apostar por estas tecnologías, ya que la innovación es lo que más se premia en los trabajos realizados en **SINERGIA**.

# 2. ESTRUCTURA DEL DOCUMENTO

En este capítulo hay una única sección en la que se realizarán una serie de recomendaciones para que, según el perfil del lector de esta memoria, la lectura resulte lo más amena posible. De este modo, según las necesidades de cada lector, podrá resultar de más interés un bloque u otro.

# 2.1. Cómo leer este documento

A continuación se realizan una serie de recomendaciones de lectura según el perfil del lector de este documento:

- **Desarrollador avanzado de aplicaciones web**: en el caso de un desarrollador con años de experiencia en el desarrollo web se recomienda la lectura de los bloques III, especialmente el estudio de tecnologías, IV y V. En este último bloque se detalla la arquitectura del proyecto, con lo que no debería tener problemas a la hora de entenderlo.
- **Desarrollador novel**: en este caso, un desarrollador con conocimientos medios aunque con escasa experiencia, se le recomienda leer los mismos bloque que a un desarrollador avanzado, además del bloque II, en el que se explica la metodología usada, lo cual es fundamental para entender el problema que se trata.
- **Usuario potencial de la aplicación**: para aquellas personas que vayan a usar la aplicación, independientemente del rol que vayan a tener, se recomienda altamente el bloque VI, en el que se encuentran los manuales de usuario, detallados según el rol del usuario que utilice la aplicación web.

# BLOQUE II: METODOLOGÍA

# 3. ESTUDIO DE METODOLOGÍAS

Por todos es sabido que el desarrollo de software no es una tarea nada sencilla y que, con el paso de los años, se complica cada vez más conforme avanzan las tecnologías. Cada vez se requieren mejores programas y de mejor calidad. Por tanto se requiere de constantes mejoras en el proceso de desarrollo de software.

En este capítulo se realiza una comparativa entre las metodologías tradicionales y las metodologías ágiles. Además, justificaremos por qué es mejor en nuestro caso utilizar una metodología ágil como **Scrum**.

# 3.1. Metodologías tradicionales frente a metodologías ágiles

Hace algunas décadas, el desarrollo de software era totalmente artesanal y se deba rienda suelta a la inspiración. No existían normativas, procesos definidos ni guía que sirviera a los desarrolladores para organizarse en sus proyectos.

Con el tiempo surgieron las primeras metodología (denominadas a partir de ahora 'metodologías tradicionales'). Estas metodologías eran muy estrictas y estaban fuertemente documentadas. Se definían una serie de fases, a saber: especificación de requisitos, diseño, construcción, pruebas, despliegue e implantación. La idea era realizar estas fases en cascada, lo cual a priori podía parece algo óptimo.

Sin embargo, hoy en día los clientes cambian mucho de idea. Incluso en ocasiones no saben explicarse, no tienen claro lo que realmente necesitan o no hay entendimiento entre el cliente y el equipo de trabajo. Esto implica cambios a lo largo del proyecto y, muy a menudo, volver a una fase anterior. Por tanto, estos contratiempos implican retrasos e incumplimiento de plazos.

Las situaciones en las que puede ser beneficioso utilizar una metodología tradicional pueden ser aquellos en los que:

- Los requisitos son firmes y no cambiarán bajo ningún concepto. Para esto, el cliente debe tener claro desde el comienzo qué es lo que quiere, además de que estén suficientemente detallados como para que no aparezcan nuevos requisitos.
- Los desarrolladores están familiarizados con el proyecto, la filosofía de trabajo o la tecnología, entre otros.
- El coste de realizar un cambio en los requisitos es alto, ya sea en tiempo, material o en personal.
- El proyecto no supone ningún riesgo de ningún tipo.
- El proyecto tiene un alcance limitado.

Normalmente, las metodologías tradicionales son usadas en proyectos 'cortos' o que están muy bien definidos. Como ejemplos, encontramos proyectos relacionados con desarrollo de drivers, aplicaciones de software empotrado, etc.

En las metodologías tradicionales destacan: RUP, MSF o Iconix.

En el otro extremo tenemos las metodologías ágiles. Estas metodologías surgieron como una importante alternativa.

La filosofía de las metodologías se define por una serie de ideas fundamentales:

- En lugar de crear el entorno, definir tecnologías y contratar personal se empieza formando el equipo. Una vez que el equipo se conoce son los propios miembros quienes toman las decisiones según las necesidades del proyecto.
- Es prioritario obtener un software que funcione antes que una buena documentación. Además, la documentación se debe generar única y exclusivamente cuando sea requerida, de forma que los documentos sean cortos y centrarse en el tema que se documenta.
- El cliente toma gran protagonismo, ya que participa de forma muy activa en el proyecto. De este modo, hay una constante comunicación entre el equipo de desarrollo y el cliente, de forma que el cliente siempre podrá transmitir sus sensaciones acerca del trabajo desarrollado hasta la fecha.
- Una rápida y eficaz respuesta a los cambios, bien sean requisitos, cuestiones tecnológicas o de otra índole. Por tanto, la planificación debe ser más flexible que en las metodologías tradicionales. Ésta es quizás la idea más importante de las metodologías ágiles.

Las metodologías ágiles son recomendables para proyectos que no están completamente definidos desde el inicio, proyectos muy largos o proyectos con requisitos que puedan cambiar de forma previsible.

Como ejemplos de proyectos que usan metodologías ágiles pueden ser proyectos de tipo empresarial, como por ejemplo: proyectos de desarrollo y diseño de aplicaciones web, aplicaciones de gestión de cualquier índole, etc.

Entres las metodologías ágiles podemos encontrar: Scrum, Kanban o XP (Extreme Programming).

Para visualizar de forma más rápida las características de los dos tipos de metodologías se presenta una tabla en la que ambas son comparadas:

| Metodologías tradicionales | Metodologías ágiles |
|---|---|
| Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo. | Basadas en heurísticas provenientes de prácticas de producción de código. |
| Impuestas externamente | Impuestas internamente (por el equipo de trabajo). |
| Proceso mucho más controlado, con numerosas políticas/normas. | Proceso menos controlado. |
| Existe un contrato prefijado. | No existe un contrato tradicional o al menos es bastante flexible. |
| El cliente interactúa con el equipo de desarrollo mediante reuniones. | El cliente es parte del equipo de desarrollo. |
| Grupos grandes y posiblemente distribuidos. | Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio. |
| Se genera mucho material y mucha documentación. | Se genera poca documentación. |
| Se definen muchos roles. | Se definen pocos roles. |
| La arquitectura del software es esencial y se expresa mediante modelos. | Menos énfasis en la arquitectura del software. |

***Tabla 1:*** *Estudio de metodologías - Comparación de metodologías*

En el caso de **La Gloria**, en el que hay que desarrollar y diseñar una página web, es previsible que se hagan cambios constantemente. Por tanto, la mejor opción es usar una metodología ágil.

La metodología a usar será **Scrum**, ya que la he estudiado y utilizado durante mis estudios. Por tanto, tengo experiencia con esta metodología.

# 3.2. Scrum

Esta metodología es un modelo de desarrollo ágil, en la que se adopta una estrategia de desarrollo incremental en lugar de planificar el proceso completo.

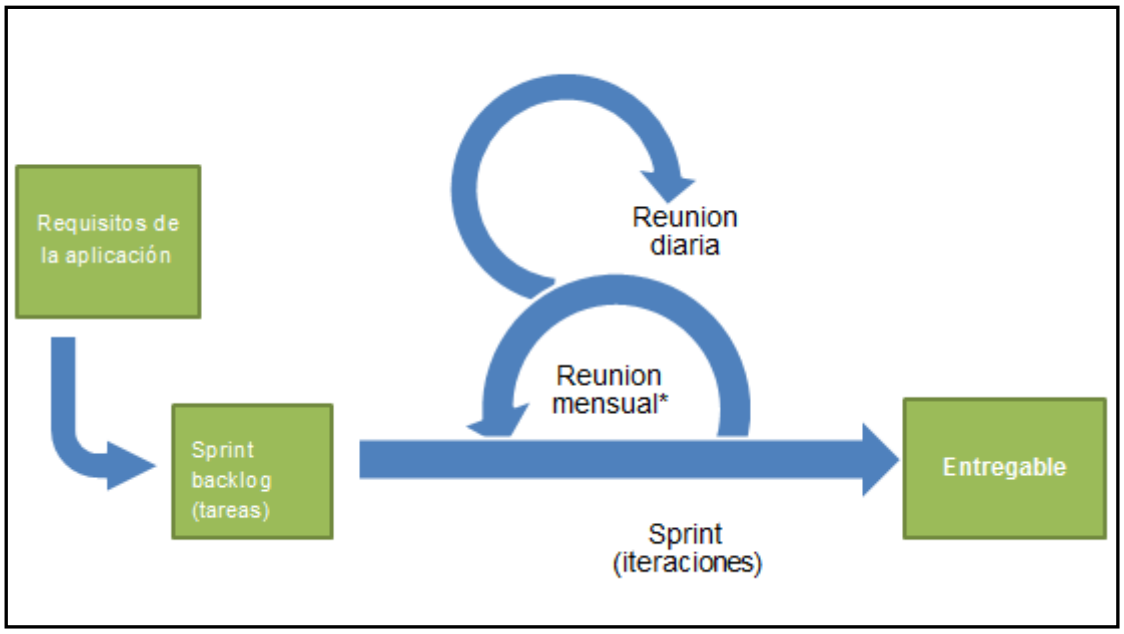



***Ilustración 2:*** *Proceso Scrum*

## Sprints

Antes de comenzar la planificación, es importante saber qué objetivos se quieren lograr y el orden en el que se van a satisfacer, aún sabiendo que los requisitos del cliente pueden cambiar (y de hecho, lo harán). De esa forma, se suelen definir varios periodos de tiempo en los que se desarrollan tareas de una misma tipología. A cada uno de estos periodos se le denomina **Sprint**.

No hay un número exacto de **Sprints** para un proyecto, pero se recomienda que sean de la misma duración y que, al final de cada **Sprint**, se obtenga un entregable funcional. Esto se traduce en que, aplicado al desarrollo de una aplicación web, en cada **Sprint** se deben implementar pocas funcionalidades pero que funcionen correctamente.

Aunque se haga una 'pre-planificación' de los **Sprints**, las tareas a desarrollar en los mismos no se deben definir hasta justo antes de comenzarlo. Esto se debe a que los requisitos de los clientes son cambiantes, por lo que siempre habrá que incorporar tareas de corrección o repetir tareas de alguna otra forma.

# Roles

En **Scrum** no se define un líder de proyecto ni ninguna jerarquía en la que haya algún miembro con más poder que otro, ya que son los propios miembros los que se auto-organizan. Sin embargo, se definen una serie de roles con ciertas responsabilidades. Por un lado hay roles principales y por otro roles secundarios. Entre los roles principales se encuentran:

- **Product owner**: *"representa la voz del cliente. Se asegura de que el equipo Scrum trabaje de forma adecuada desde la perspectiva del negocio"*[1]. Además, se encarga de definir historias de usuario (resultado de subdividir una tarea el número máximo de veces), ordenarlas según su prioridad y de colocarlas en el **Product Backlog**, el cual se explicará más adelante.
- **ScrumMaster:** su *"trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El **ScrumMaster** no es el líder del equipo... sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga"*[1]. Además se encarga de verificar que el proceso **Scrum** se utiliza correctamente y que los miembros del equipo de trabajo cumplen las reglas.
- **Equipo de desarrollo:** el equipo de desarrollo tiene la responsabilidad de elaborar y entregar el trabajo solicitado. Suele estar comprendido entre 3 y 9 personas *"con las habilidades transversales necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc)"*[1].

Los roles secundarios los poseen aquellas personas que no están directamente involucradas en el proceso Scrum, pero que han de ser tenidas en cuenta. En cuanto a los roles secundarios se encuentran:

- **Stakeholders:** son todos aquellos interesados que hacen posible el proyecto. Este rol lo suelen ocupan clientes, vendedores o proveedores, entre otros.
- **Administradores:** personal que *"establece el ambiente para el desarrollo del proyecto"*[1].

# Reuniones

Una de las particularidades más destacables de **Scrum** son las reuniones que se realizan a lo largo del proceso.

---

[1] http:// es.wikipedia.org/wiki/Scrum

### Daily Scrum

*"Cada día de un Sprint se realiza la reunión sobre el estado de un proyecto. Esto se llama* ***Daily Scrum*** *o* ***Stand-up Meeting***"[2]. Existe una serie de pautas a seguir en estas reuniones:

- La reunión comienza siempre a la misma hora.
- Puede asistir cualquier persona, pero sólo las involucradas en el proyecto pueden hablar.
- La reunión dura necesariamente 15 minutos, independientemente del número de miembros del equipo de trabajo o del trabajo realizado.
- La reunión debe celebrarse en el mismo sitio y hora todos los días.

En el **Daily Scrum** deben contestarse las siguientes preguntas:

- ¿Qué has hecho desde el último **Daily Scrum**?
- ¿Qué harás hasta el próximo **Daily Scrum**?
- ¿Has tenido algún obstáculo para cumplir tus objetivos?

En el caso de que alguno o más miembros hayan tenido obstáculos, es tarea del **ScrumMaster** anotar estos impedimentos para preverlos en un futuro.

### Scrum de Scrum

Es una reunión que tiene lugar, habitualmente, después del **Daily Scrum**. Estas reuniones sirven para que los equipos definidos puedan discutir especialmente sobre su trabajo, y no de forma global como en el **Daily Scrum**. Sin embargo, a estas reuniones acude un único miembro por equipo.

Las preguntas que se plantean son las mismas que en el **Daily Scrum**, pero añadiendo las siguientes:

- ¿Qué ha hecho tu equipo desde la última reunión?
- ¿Qué hará tu equipo hasta la próxima reunión?
- ¿Existe algún impedimento para tu equipo?
- ¿Es posible que entorpezcas a otro equipo?

---

[2] http:// es.wikipedia.org/wiki/Scrum

### Sprint Planning Meeting

Al inicio de cada **Sprint** se debe celebrar una reunión de planificación del Sprint, lo que se conoce como **Sprint Planning Meeting**. Los aspectos que han de ser tenidos en cuenta para esta reunión son los siguientes:

- Se tiene que seleccionar el trabajo que se realizará.
- Con el equipo completo, se prepara el **Sprint Backlog**, donde *"se detalla el tiempo que tomará hacer el trabajo"*[3].
- Identificar y notificar el trabajo que será posible tener realizado al final del actual **Sprint**.
- El tiempo máximo de reunión es de 8 horas.

### Sprint Review Meeting

Al final de cada **Sprint** se realiza una reunión a la que asisten los **Stakeholders** del proyecto. En esta reunión se revisa tanto el trabajo realizado como el trabajo que no se ha completado, de modo que esta reunión sirve de 'demo' para los **Stakeholders** que acudan a la reunión. En ningún caso el trabajo incompleto puede ser demostrado, y al igual que en el **Sprint Planning Meeting**, se define un tiempo máximo de reunión, aunque en esa ocasión es de 4 horas.

### Sprint Retrospective

Del mismo modo, también se realiza una reunión al final de cada **Sprint** a la que sólo acuden los miembros del equipo de trabajo, con el único fin de dejar las impresiones sobre el **Sprint** recién finalizado.

# Documentos

Durante esta sección hemos hablado de cosas como el **Product Backlog** o el **Sprint Backlog**, pero aún no los hemos definido.

El **Product Backlog** es *"un documento de alto nivel para todo el proyecto. Contiene descripciones genéricas de todos los requisitos, funcionalidades deseables, etc priorizadas según su retorno sobre la inversión. Es el qué va a ser construido. Es abierto y solo puede ser modificado por el product owner"*[3].

---

[3] http:// es.wikipedia.org/wiki/Scrum

Por otro lado tenemos el **Sprint Backlog**, el cual es *"un documento detallado donde se describe el cómo el equipo va a implementar los requisitos durante el siguiente **Sprint**. Las tareas se dividen en horas pero ninguna tarea con una duración superior a 16 horas. Si una tarea es mayor de 16 horas, deberá ser dividida en otras menores. Las tareas en el **Sprint Backlog** nunca son asignadas, son tomadas por los miembros del equipo del modo que les parezca oportuno"*[4].

[4] http:// es.wikipedia.org/wiki/Scrum

# 4. METODOLOGÍA DE TRABAJO

En este capítulo se comentan los aspectos de **Scrum** que se han utilizado a la hora de realizar la planificación de este trabajo y de qué forma se ha adaptado según nuestras necesidades.

Algunas cosas han tenido que ser adaptadas, ya que en este proyecto sólo hay un desarrollador (un servidor), por lo que habrá roles y/o conceptos que se suprimirán o se simplificarán, ya que carecerá de sentido aplicarlos tal cual se define en **Scrum**.

# 4.1. Adaptación a partir de Scrum

A la hora de realizar la planificación hemos intentado organizar al grupo de la forma más sencilla.

Hay que tener en cuenta que mi compañera de trabajo, al no estudiar materias técnicas como la ingeniería del software, no está familiarizada con este tipo de metodologías. Por tanto, hay muchos aspectos que tienen que ser modificados para facilitar su adaptación.

## Roles

Los roles utilizados en este proyecto son:

- Product owner.
- ScrumMaster.
- Equipo de desarrollo.

El rol de **Product owner** lo desempeñarán ambos tutores del proyecto, a saber Marina Ramos Serrano, profesora del **Departamento de Creatividad y Nuevas Tecnologías**, y Pablo Fernández Montes, profesor del **Departamento de Lenguajes y Sistemas Informáticos**.

El **ScrumMaster** del proyecto será un servidor, ya que soy el único de los dos alumnos que ha usado previamente esta metodología, por lo que mi experiencia es mayor que la de mi compañera.

Para finalizar, en cuanto al equipo de desarrollo lo compondremos los dos alumnos. A pesar de que el único desarrollador soy yo, mi compañera me ayudará en tareas de diseño, a demás de proporcionarme el material necesario, así como las imágenes de los productos ofrecidos y toda la información asociada a los mismos, desde las categorías de los productos a los precios por Kg.

## Reuniones

Del mismo modo, para facilitar a mi compañera la adaptación a esta metodología, se decidió reducir los tipos de reuniones y la frecuencia de celebraciones de las mismas.

En cuanto a los alumnos del proyecto nos reuniremos 1 vez por semana, preferiblemente en persona. Si alguna semana no pudiera ser, entonces esta reunión tendría que hacerse a distancia mediante **Skype** o **Google Hangouts**.

También se realizará una reunión con todos los miembros del proyecto al inicio de cada **Sprint**. Sin embargo, al inicio del proyecto se llegarán a programar dos reuniones por **Sprint**, hasta que el proyecto esté encaminado y el equipo de desarrollo tenga una perspectiva precisa de los requisitos a satisfacer.

# Documentos

En el inicio del proyecto se generarán una serie de documentos cortos y precisos (tal y como propone **Scrum**). Estos documentos surgirán en la etapa de análisis, en la que se estudiarán herramientas tecnológicas, diseños de páginas web, elicitación de requisitos, etc. De este modo, en el momento que se comience a redactar la memoria técnica del proyecto, se tendrá gran parte de la información necesaria, teniendo únicamente que estructurarla debidamente.

# 4.2. Definición de Sprints

Una vez analizado el sistema y haberse celebrado una reunión con el cliente, se decide dividir el desarrollo del proyecto en los **Sprints** que se presentan a continuación, de forma que cada **Sprint** se corresponde con un considerable incremento de la aplicación web en lo que a funcionalidades se refiere, de modo que en cada **Sprint** se obtiene un producto potencialmente entregable, tal y como se define en **Scrum**.

| Nombre | Descripción |
|---|---|
| Estructura de la aplicación web | Implementación de las plantillas principales en **JADE** y **CSS** |
| Inventario de productos | Implementación de la **API REST** para poner a manipular la información de los productos mediante peticiones **HTTP** |
| Gestión de usuarios | Implementación de la gestión de usuarios en la aplicación web |
| Aspectos sociales | Implementación del sistema de comentarios |
| Implantación | Despliegue de la aplicación web en **Heroku** y **Compose** (anteriormente **MongoHQ**) |
| Documentación del proyecto | Generación de la documentación técnica del proyecto |

***Tabla 2:*** *Definición de* ***Sprints***

# 5. HERRAMIENTAS DE GESTIÓN

En este apartado hablaremos de las herramientas que hemos utilizado para planificar nuestras tareas. Las herramientas utilizadas han sido **ProjETSII** y **Trello**, ambas propuestas por Pablo Fernández Montes.

Las herramientas de gestión sirven para mantener el control del desarrollo del proyecto, por lo que son muy útiles para proyectos software. Con herramientas de este tipo, habitualmente podemos: definir tareas, asignarlas a un miembro del equipo de trabajo, estimar la duración del proyecto, generar informes de trabajo, etc.

# 5.1. ProjETSII

Esta herramienta es muy usada en algunas asignaturas de las titulaciones impartidas en la **ETSII**. Con esta herramienta se pueden definir tareas y subtareas, asignar miembros a cada una de las tareas, estimar la duración y los plazos de entrega o generar informes con el trabajo realizado por parte de cada uno de los integrantes del equipo de trabajo.

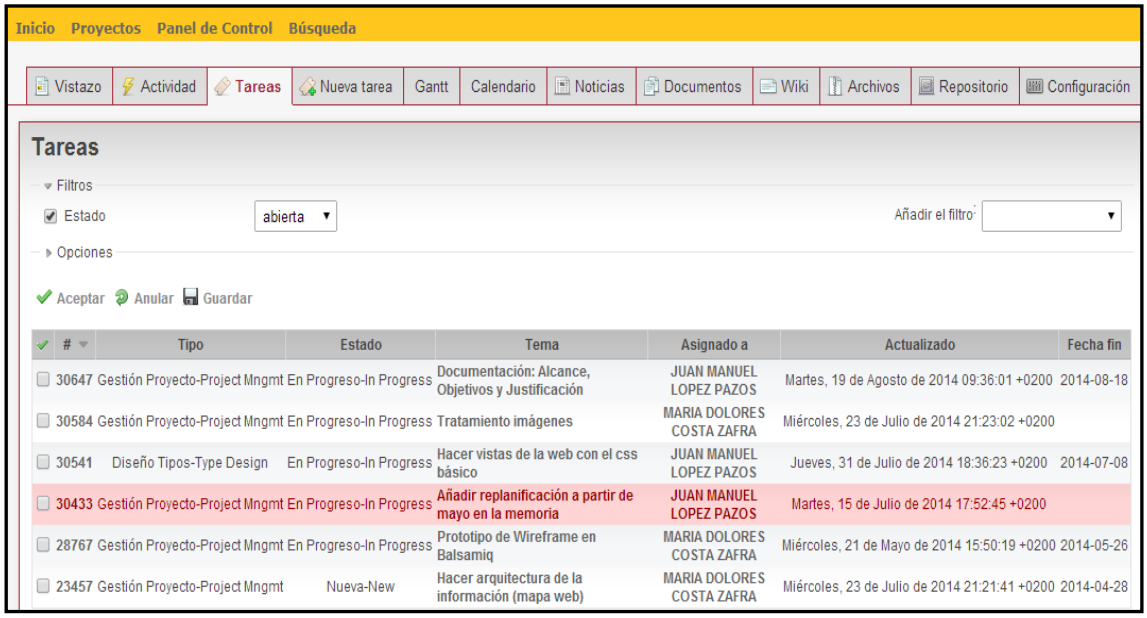



***Ilustración 3:*** *Herramientas de gestión - Vista general de* ***ProjETSII***

# 5.2. Trello

Por otra parte tenemos que hablar de **Trello**, una herramienta menos completa que **ProjETSII** pero mucho más visual. En **Trello** se pueden ver todas las tareas en una misma pantalla, tanto si están pendientes de realizarse como si se están llevando a cabo, están pendientes de revisión o incluso si están finalizadas.

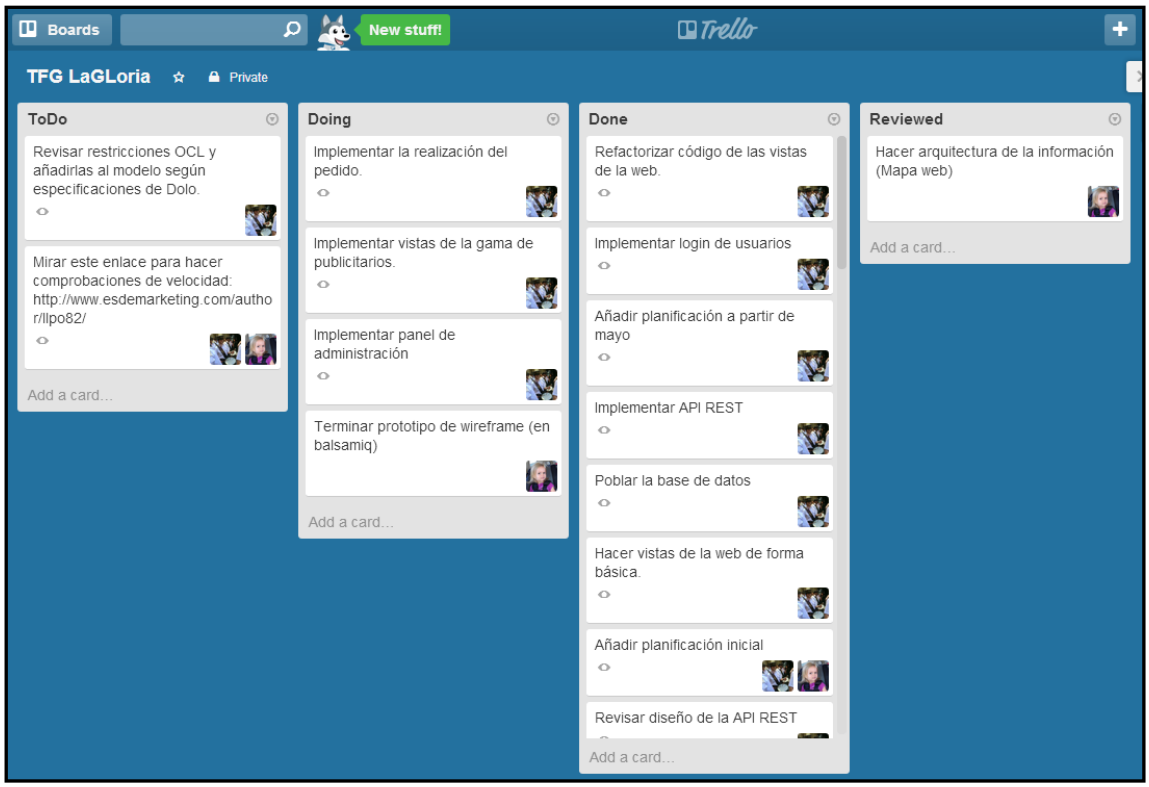



***Ilustración 4:*** *Herramientas de gestión – Vista general de* ***Trello***

Así pues, decidimos usar ambas herramientas. Usando **ProjETSII** obtendríamos unos informes muy completos, mientras que con **Trello** sabríamos en todo momento qué tareas se han realizado y cuáles están pendientes. Combinando ambas herramientas podríamos mejorar a la hora de planificar nuestras tareas.

# 6. PLANIFICACIÓN

En este capítulo se trata la planificación de este proyecto. Por un lado se estima la planificación temporal del proyecto, donde se presenta la planificación de cada uno de los 6 **Sprints** definidos para el desarrollo del proyecto.

A continuación se presenta el presupuesto inicial del proyecto, el cual se ha realizado en base a las horas estimadas, los roles identificados y los datos proporcionados por el **INE** (Instituto Nacional de Estadística).

# 6.1. Concepto de planificación

La planificación es una fase muy importante en un proyecto de desarrollo software, ya que sirve para estimar la duración de un proyecto y, con ello, los costes asociados al mismo.

La planificación suele sufrir cambios, por lo que es recomendable estimar al alza por si se producen retrasos.

Gracias a que se utilizará la metodología **Scrum** se minimizarán estos impedimentos, ya que los **Sprints** se planifican a poco tiempo de empezar, pudiendo ser incorporadas nuevas funcionalidades o tareas que se han quedado atrás.

# 6.2. Planificación estimada

En este apartado se detalla la planificación de los **Sprints** que fueron definidos en el capítulo 5. Para realizar la planificación se ha usado Microsoft Project 2010. Empezaremos mostrando la planificación global y luego iremos detallando la planificación de cada **Sprint**.

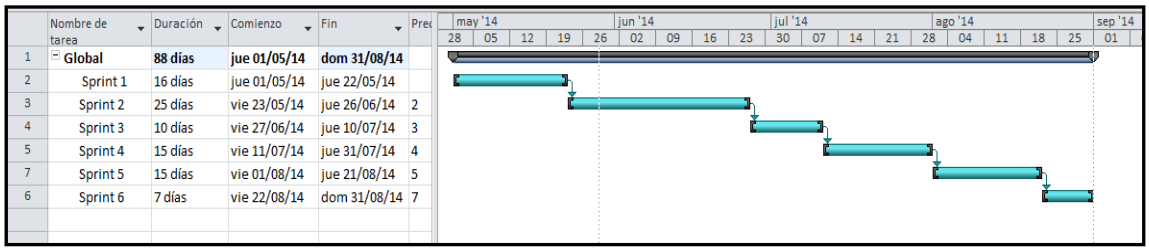



***Ilustración 5:*** *Planificación - Planificación global*

Esta imagen se corresponde con la planificación global de los **Sprints** definidos. La fecha de comienzo fue el 1 de mayo de 2014 y la fecha de fin fue el 31 de agosto de 2014.

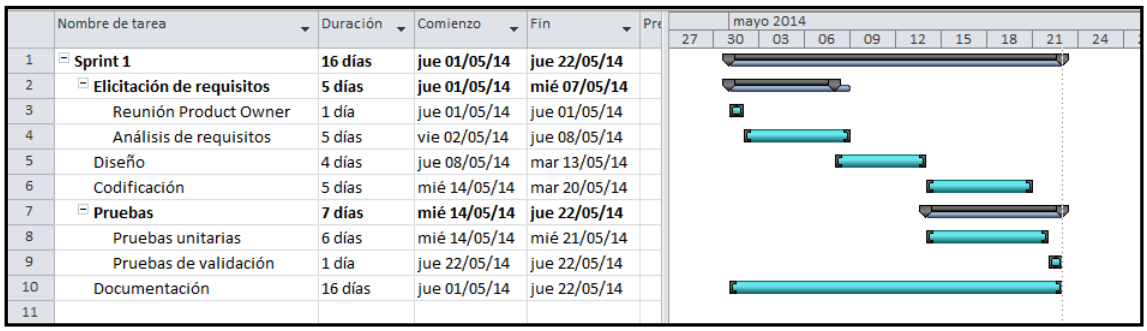
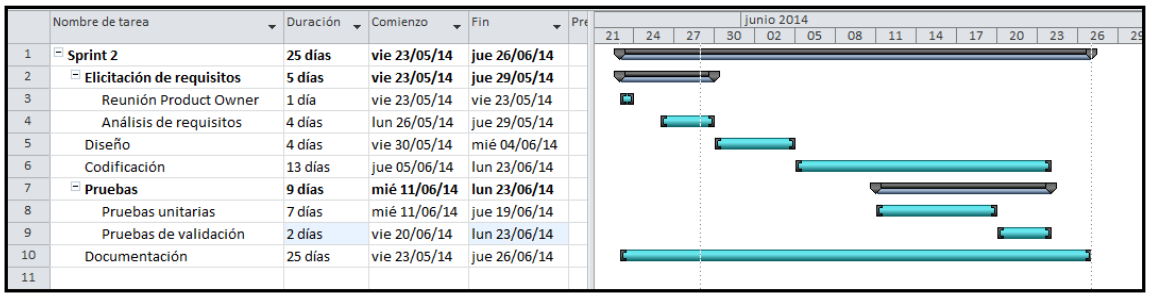



***Ilustración 6:*** *Planificación - Planificación del* ***Sprint*** *1*

Esta imagen es de la planificación del **Sprint** 1, en el que se pretende desarrollar la estructura básica de la web.

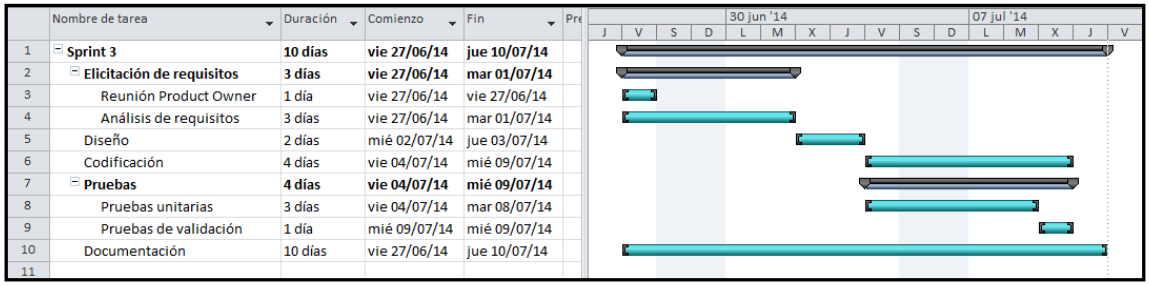


| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Sprint 2** | **25 días** | **vie 23/05/14** | **jue 26/06/14** |
| 2 | **Elicitación de requisitos** | **5 días** | **vie 23/05/14** | **jue 29/05/14** |
| 3 | Reunión Product Owner | 1 día | vie 23/05/14 | vie 23/05/14 |
| 4 | Análisis de requisitos | 4 días | lun 26/05/14 | jue 29/05/14 |
| 5 | Diseño | 4 días | vie 30/05/14 | mié 04/06/14 |
| 6 | Codificación | 13 días | jue 05/06/14 | lun 23/06/14 |
| 7 | **Pruebas** | **9 días** | **mié 11/06/14** | **lun 23/06/14** |
| 8 | Pruebas unitarias | 7 días | mié 11/06/14 | jue 19/06/14 |
| 9 | Pruebas de validación | 2 días | vie 20/06/14 | lun 23/06/14 |
| 10 | Documentación | 25 días | vie 23/05/14 | jue 26/06/14 |
| 11 | | | | |



***Ilustración 7:*** *Planificación - Planificación del* ***Sprint*** *2*

En este **Sprint** se tiene como objetivo realizar el inventario de contenidos de la web, clasificando todos los productos y añadirlos a la web.

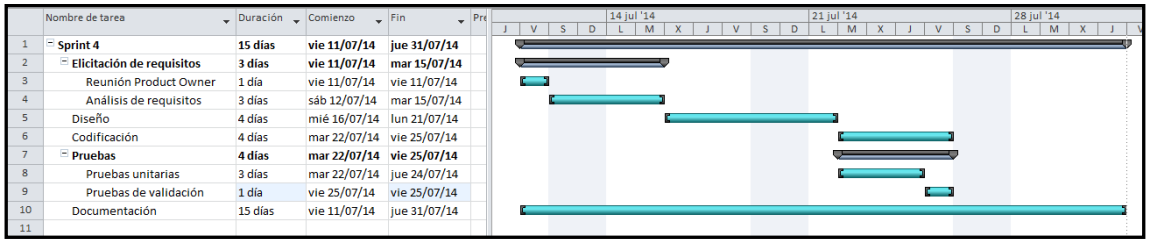


| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Sprint 3** | **10 días** | **vie 27/06/14** | **jue 10/07/14** |
| 2 | **Elicitación de requisitos** | **3 días** | **vie 27/06/14** | **mar 01/07/14** |
| 3 | Reunión Product Owner | 1 día | vie 27/06/14 | vie 27/06/14 |
| 4 | Análisis de requisitos | 3 días | vie 27/06/14 | mar 01/07/14 |
| 5 | Diseño | 2 días | mié 02/07/14 | jue 03/07/14 |
| 6 | Codificación | 4 días | vie 04/07/14 | mié 09/07/14 |
| 7 | **Pruebas** | **4 días** | **vie 04/07/14** | **mié 09/07/14** |
| 8 | Pruebas unitarias | 3 días | vie 04/07/14 | mar 08/07/14 |
| 9 | Pruebas de validación | 1 día | mié 09/07/14 | mié 09/07/14 |
| 10 | Documentación | 10 días | vie 27/06/14 | jue 10/07/14 |
| 11 | | | | |



***Ilustración 8:*** *Planificación - Planificación del* ***Sprint*** *3*

Aquí se muestra la planificación del **Sprint** 3, en el que se implementará la gestión de usuarios.

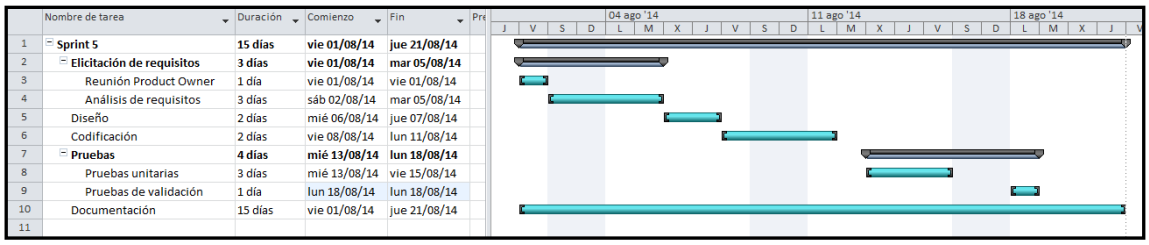


| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Sprint 4** | **15 días** | **vie 11/07/14** | **jue 31/07/14** |
| 2 | **Elicitación de requisitos** | **3 días** | **vie 11/07/14** | **mar 15/07/14** |
| 3 | Reunión Product Owner | 1 día | vie 11/07/14 | vie 11/07/14 |
| 4 | Análisis de requisitos | 3 días | sáb 12/07/14 | mar 15/07/14 |
| 5 | Diseño | 4 días | mié 16/07/14 | lun 21/07/14 |
| 6 | Codificación | 4 días | mar 22/07/14 | vie 25/07/14 |
| 7 | **Pruebas** | **4 días** | **mar 22/07/14** | **vie 25/07/14** |
| 8 | Pruebas unitarias | 3 días | mar 22/07/14 | jue 24/07/14 |
| 9 | Pruebas de validación | 1 día | vie 25/07/14 | vie 25/07/14 |
| 10 | Documentación | 15 días | vie 11/07/14 | jue 31/07/14 |
| 11 | | | | |



***Ilustración 9:*** *Planificación - Planificación del* ***Sprint*** *4*

En el **Sprint** anterior se pretende implementar los aspectos sociales.

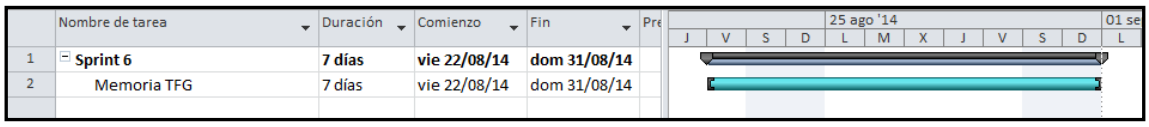


| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Sprint 5** | **15 días** | **vie 01/08/14** | **jue 21/08/14** |
| 2 | **Elicitación de requisitos** | **3 días** | **vie 01/08/14** | **mar 05/08/14** |
| 3 | Reunión Product Owner | 1 día | vie 01/08/14 | vie 01/08/14 |
| 4 | Análisis de requisitos | 3 días | sáb 02/08/14 | mar 05/08/14 |
| 5 | Diseño | 2 días | mié 06/08/14 | jue 07/08/14 |
| 6 | Codificación | 2 días | vie 08/08/14 | lun 11/08/14 |
| 7 | **Pruebas** | **4 días** | **mié 13/08/14** | **lun 18/08/14** |
| 8 | Pruebas unitarias | 3 días | mié 13/08/14 | vie 15/08/14 |
| 9 | Pruebas de validación | 1 día | lun 18/08/14 | lun 18/08/14 |
| 10 | Documentación | 15 días | vie 01/08/14 | jue 21/08/14 |
| 11 | | | | |



***Ilustración 10:*** *Planificación - Planificación del* ***Sprint*** *5*

En este **Sprint** se pretende implantar el sistema y realizar los últimos retoques necesarios.



| | Nombre de tarea | Duración | Comienzo | Fin |
|---|---|---|---|---|
| 1 | **Sprint 6** | **7 días** | **vie 22/08/14** | **dom 31/08/14** |
| 2 | Memoria TFG | 7 días | vie 22/08/14 | dom 31/08/14 |



***Ilustración 11:*** *Planificación - Planificación del* ***Sprint*** *6*

En este último **Sprint** se pretende agrupar los documentos generados y documentar una única memoria formal técnica, la que actualmente está leyendo.

# 6.3. Estimación de costes

A continuación de muestra la estimación de costes realizada en este trabajo.

Primero se estimarán los costes de personal, los cuales se corresponden con el salario de los miembros de este trabajo. En segundo lugar, se indican los costes previstos en lo que a software se refiere.

## Costes de personal

Para estimar los costes de personal se han utilizado los datos que proporcionó el **INE**[5] (Instituto Nacional de Estadística) en el año 2011.

(Nota: los datos utilizados están extraídos de gráficas, por lo que son cifras aproximadas)

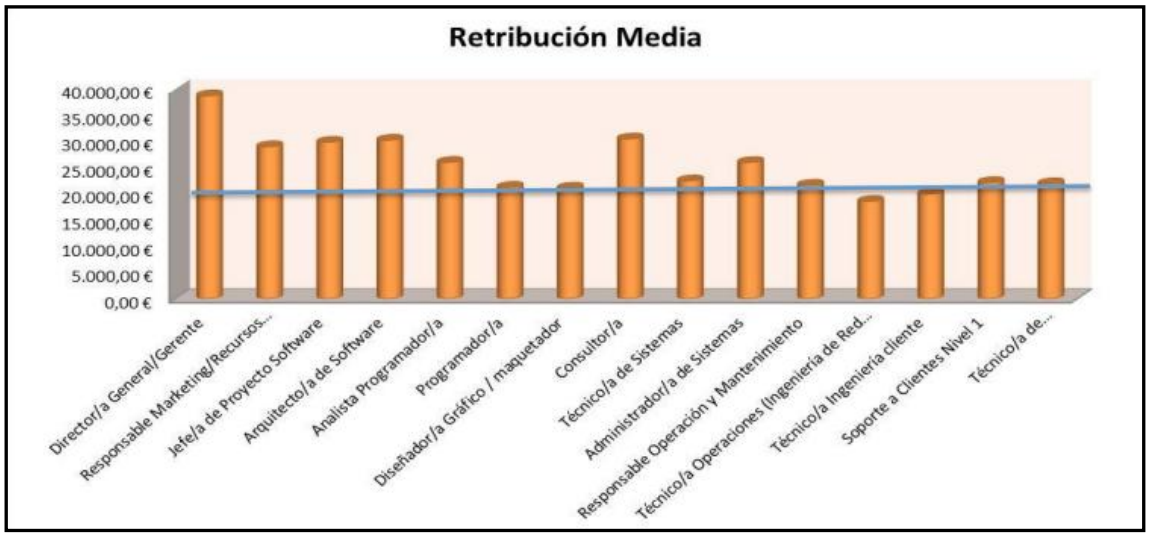



***Ilustración 12:*** *Estimación de costes - Retribución media según el INE*

Una vez dicho esto, el salario base de cada trabajador será:

- Jefe de proyecto: 32.000,00 € / 1.800 h = 17,78 € / h.
- Analista: 27.500,00 € / 1.800 h = 15,28 € / h.
- Programador: 22.790, 20 € / 1.800 h = 12,66 € / h.

[5] http://www.ine.es/

Estas cifras se corresponden con los salarios netos de cada trabajador por hora trabajada. Sin embargo, hay que añadir una serie de impuestos como la seguridad social, lo cual supone un 30 % de incremento:

- Jefe de proyecto: 17,78 € / h * 1,3 = 23,11 € / h.
- Analista: 15,28 € / h * 1,3 = 19,86 € / h.
- Programador: 12,66 € / h * 1.3 = 16,46 € / h.

Estos últimos costes son los costes brutos de los trabajadores por hora trabajada.

A continuación se desglosan las distintas fases del proyecto junto con las horas estimadas:

| Tarea | Duración (horas) | Rol | Coste (€ / rol) | Coste (€) |
|---|---|---|---|---|
| **Estructura de la aplicación web** | | | | |
| Elicitación de requisitos | 11 | Jefe de proyecto | 23,11 | 254,21 |
| Diseño | 14 | Analista | 19,86 | 278,04 |
| Codificación | 16 | Programador | 16,46 | 263,36 |
| Pruebas unitarias | 7 | Programador | 16,46 | 115,22 |
| Pruebas de validación | 5 | Jefe de proyecto | 23,11 | 115,55 |
| Documentación | 21 | Jefe de proyecto, Analista y Programador | 19,81 | 416,01 |
| **Inventario de productos** | | | | |
| Elicitación de requisitos | 11 | Jefe de proyecto | 23,11 | 254,21 |
| Diseño | 14 | Analista | 19,86 | 278,04 |
| Codificación | 50 | Programador | 16,46 | 823 |
| Pruebas unitarias | 9 | Programador | 16,46 | 148,14 |

| | | | | |
|---|---|---|---|---|
| Pruebas de validación | 5 | Jefe de proyecto | 23,11 | 115,55 |
| Documentación | 7 | Jefe de proyecto, Analista y Programador | 19,81 | 138,67 |
| **Gestión de usuarios** | | | | |
| Elicitación de requisitos | 11 | Jefe de proyecto | 23,11 | 254,21 |
| Diseño | 11 | Analista | 19,86 | 218,46 |
| Codificación | 30 | Programador | 16,46 | 493,8 |
| Pruebas unitarias | 9 | Programador | 16,46 | 148,14 |
| Pruebas de validación | 5 | Jefe de proyecto | 23,11 | 115,55 |
| Documentación | 7 | Jefe de proyecto, Analista y Programador | 19,81 | 138,67 |
| **Aspectos sociales** | | | | |
| Elicitación de requisitos | 16 | Jefe de proyecto | 23,11 | 369,76 |
| Diseño | 17 | Analista | 19,86 | 337,62 |
| Codificación | 58 | Programador | 16,46 | 954,68 |
| Pruebas unitarias | 14 | Programador | 16,46 | 230,44 |
| Pruebas de validación | 7 | Jefe de proyecto | 23,11 | 161,77 |
| Documentación | 17 | Jefe de proyecto, Analista y Programador | 19,81 | 336,77 |
| **Implantación** | | | | |
| Elicitación de requisitos | 7 | Jefe de proyecto | 23,11 | 161,77 |
| Diseño | 9 | Analista | 19,86 | 178,74 |
| Codificación | 9 | Programador | 16,46 | 148,14 |
| Pruebas unitarias | 14 | Programador | 16,46 | 230,44 |
| Pruebas de validación | 14 | Jefe de proyecto | 23,11 | 323,54 |

| | | | | |
|---|---|---|---|---|
| Documentación | 7 | Jefe de proyecto, Analista y Programador | 19,81 | 138,67 |
| **Documentación del proyecto** | | | | |
| Memoria del TFG | 80 | Jefe de proyecto, Analista y Programador | 19,81 | 1584,8 |
| **Total** | | | | 9725,97 |

**Tabla 3:** Estimación de costes - Coste de personal

Los costes brutos asociados a los gastos de personal ascienden a un total de 9725,97 €.

# Costes de software

También se estiman algunos gastos en los que a software se refiere. Se requerirá la obtención de la siguiente licencia:

| | |
|---|---|
| Licencia de JetBrains WebStorm 7 | 121,60 € |
| **Total** | 121,60 € |

# Costes totales

Una vez que han sido detallados todos los costes que se prevén en el proyecto, se tiene que el presupuesto inicial es:

| | |
|---|---|
| Costes de personal | 9725,97 € |
| Costes de software | 121,60 € |
| **Total** | **9847,57 €** |

# BLOQUE III: ANÁLISIS

# 7. ELICITACIÓN DE REQUISITOS

La elicitación de requisitos es la parte más importante de todo proyecto de ingeniería del software, ya que tiene lugar en el inicio del proyecto.

El objetivo de la elicitación de requisitos es reunir los requisitos solicitados por el cliente, los cuales han de ser satisfechos durante el desarrollo del proyecto, además de conseguir que éstos sean lo más precisos y completos posible, persiguiendo la minimización de futuros cambios a lo largo del proyecto.

Para ello es necesario conocer el dominio del problema, teniendo en cuenta las necesidades del cliente y de los usuarios potenciales de la aplicación web a desarrollar.

Para clasificar los requisitos se ha usado la clasificación propuesta en **MADEJA** [6]para documentos de requisitos del sistema.

Así pues, en este capítulo se detallan los requisitos obtenidos en el proceso de elicitación, así como los objetivos, los requisitos no funcionales y el glosario de términos.

[6] http://www.juntadeandalucia.es/servicios/madeja/

# Conceptos básicos

Antes de mostrar la elicitación de requisitos se indican en este apartado una serie de conceptos, los cuales han sido utilizados en la elicitación y es conveniente entenderlos.

- Estado: indica el estado del objetivo, requisito o caso de uso en lo que a desarrollo se refiere. Los posibles valores son:
  - En construcción: se está elaborando.
  - Pendiente de validación: existe algún conflicto que se tiene que resolver.
  - Pendiente de verificación: no existen conflictos pero ha de ser verificado.
  - Validado: ha sido validado por clientes y/o usuarios.
- Importancia: indica la importancia de la satisfacción del objetivo, requisito o caso de uso por parte de los clientes y usuarios. Los valores posibles son:
  - Vital: su cumplimiento es imprescindible.
  - Importante: se cumplimiento es recomendable.
  - Aceptable: es algo accesorio pero se permite.
- Estabilidad: indica la resistencia a los cambios de un objetivo, requisito o caso de uso. Puede ser:
  - Baja: alta probabilidad de que cambie.
  - Media: probabilidad media de que cambie.
  - Alta: ninguna probabilidad de que cambie.

# 7.1. Objetivos del sistema

En esta sección se describen los objetivos propuestos para el desarrollo de la aplicación web para **La Gloria S.L**. El objetivo principal es desarrollar una aplicación web estética y funcional que mejore a la anterior para, así, atraer nuevos clientes.

| **OBJ - 001** | **Estructura de la aplicación web** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá mostrar el diseño de la aplicación web, así como las gamas principales existentes en **La Gloria** |
| **Importancia** | Vital |

***Tabla 4:*** *Objetivo - Estructura de la aplicación web*

| **OBJ - 002** | **Gestión de usuarios** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá gestionar los usuarios registrados en el sistema |
| **Importancia** | Vital |

***Tabla 5:*** *Objetivo - Gestión de usuarios*

| OBJ - 003 | Aspectos sociales |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá ofrecer las funcionalidades sociales más comunes en la actualidad. |
| **Importancia** | Vital |

***Tabla 6:*** *Objetivo - Aspectos sociales*

| OBJ - 004 | Diseño web La Gloria |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá tener un aspecto visual totalmente propio y seguir el estilo predefinido de **La Gloria**. |
| **Importancia** | Importante |

***Tabla 7:*** *Objetivo - Diseño web* ***La Gloria***

| OBJ - 005 | Gestión de contenidos |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |

| | |
|---|---|
| **Descripción** | El sistema deberá contener toda la información necesaria sobre los productos de **La Gloria S.L.**. |
| **Importancia** | Importante |

***Tabla 8:*** *Objetivo - Gestión de contenidos*

| **OBJ - 006** | **Gestión de emails** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá permitir gestionar toda la información relacionada con los emails enviados por los usuarios. |
| **Importancia** | Importante |

**Tabla 9:** Objetivo - Gestión de emails

| **OBJ - 007** | **Gestión de pedidos** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Descripción** | El sistema deberá permitir gestionar toda la información relacionada con los pedidos realizados por los proveedores. |
| **Importancia** | Importante |

***Tabla 10:*** *Objetivo - Gestión de pedidos*

# 7.2. Catálogo de Requisitos del sistema

En esta sección se tratan los requisitos del sistema. Según el estándar definido por **IEEE**, se tiene que un requisito es una *"condición o capacidad que un usuario necesita para poder resolver un problema o lograr un objetivo"*[7]. Tal y como se ha comentado anteriormente, se han clasificado los requisitos siguiendo una de las propuestas que hace **MADEJA** para documentos de requisitos del sistema.

**MADEJA** clasifica los requisitos en 3 grandes grupos:

- Requisitos de información.
- Requisitos funcionales del sistema.
- Requisitos no funcionales.

[7] https://www.ieee.org

# 7.2.1. Requisitos de información

En este apartado se detallan los requisitos de información y los requisitos de reglas de negocio y restricciones.

Los requisitos de información (**IRQ**) especifican la información que debe almacenar el sistema para cumplir los objetivos especificados anteriormente.

Los requisitos de reglas de negocio y restricciones (**CRQ**) son aquellos que sirven para especificar el funcionamiento del negocio, el cual ha de ser respetado en la aplicación web a desarrollar.

En primer lugar se detallan los distintos requisitos de información que han sido identificados en la fase de elicitación:

| **IRQ - 001** | **Almacenar información de los caramelos** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | María Dolores Costa Zafra<br>Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes<br>Marina Ramos Serrano |
| **Dependencias** | • [IRQ-004] Almacenar información de los pedidos.<br>• [CRQ-005] Unicidad de producto en el carrito de compra.<br>• [UC-001] Ver los productos de una categoría.<br>• [UC-002] Ver los detalles de un producto en una categoría.<br>• [UC-003] Buscar producto en el sistema.<br>• [UC-004] Comprar producto.<br>• [UC-005] Crear producto.<br>• [UC-006] Editar producto.<br>• [UC-007] Eliminar producto.<br>• [UC-014] Comentar producto.<br>• [UC-015] Valorar producto.<br>• [UC-016] Editar comentario realizado de un producto.<br>• [UC-017] Eliminar comentario realizado de un producto.<br>• [UC-021] Realizar pedido.<br>• [UC-022] Ver los pedidos realizados.<br>• [UC-023] Editar un pedido realizado.<br>• [UC-024] Eliminar un pedido realizado. |

| | |
|---|---|
| **Descripción** | El sistema deberá almacenar información sobre los productos ofrecidos por **La Gloria S.L.** |
| **Datos específicos** | • Gama<br>• Categoría<br>• Tipo<br>• Modelo<br>• Sabor<br>• Precio por Kg<br>• Pedido mínimo |
| **Importancia** | Vital |

***Tabla 11:*** *Requisito de información - Almacenar información de los caramelos [IRQ-001]*

| **IRQ – 002** | **Almacenar información de los emails** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [UC-018] Enviar email.<br>• [UC-019] Ver los emails recibidos.<br>• [UC-020] Eliminar email. |
| **Descripción** | El sistema deberá almacenar los emails que hayan sido enviados por los usuarios al buzón de correo de **La Gloria S.L.** |
| **Datos específicos** | • Nombre del remitente<br>• Fecha<br>• Mensaje |
| **Importancia** | Media |

***Tabla 12:*** *Requisito de información - Almacenar información de los emails [IRQ-002]*

| IRQ - 003 | Almacenar información de los usuarios |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [OBJ-002] Gestión de usuarios.<br>• [OBJ-003] Aspectos sociales.<br>• [IRQ-004] Almacenar información de los pedidos.<br>• [CRQ-001] Unicidad de usuarios.<br>• [CRQ-002] Control de acceso sobre usuario anónimo.<br>• [CRQ-003] Control de acceso sobre usuario proveedor.<br>• [CRQ-004] Login de usuarios.<br>• [UC-008] Registrarse como usuario.<br>• [UC-009] Hacer login en el sistema.<br>• [UC-010] Hacer logout en el sistema.<br>• [UC-011] Listar usuarios registrados en el sistema.<br>• [UC-012] Editar usuario registrado en el sistema.<br>• [UC-013] Eliminar usuario registrado en el sistema. |
| **Descripción** | El sistema deberá almacenar los datos correspondientes a los usuarios registrados en el sistema. |
| **Datos específicos** | • Nick del usuario<br>• Contraseña<br>• Fecha de registro<br>• Rol en el sistema |
| **Importancia** | Importante |

***Tabla 13:*** *Requisito de información - Almacenar información de los usuarios [IRQ-003]*

| IRQ - 004 | Almacenar información de los pedidos |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |

| | |
|---|---|
| **Dependencias** | • [IRQ-001] Almacenar información de los caramelos.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [CRQ-005] Unicidad de producto en el carrito de compra.<br>• [UC-002] Ver los detalles de un producto en una categoría.<br>• [UC-004] Comprar producto.<br>• [UC-021] Realizar pedido.<br>• [UC-022] Ver los pedidos realizados.<br>• [UC-023] Editar un pedido realizado.<br>• [UC-024] Eliminar un pedido realizado. |
| **Descripción** | El sistema deberá almacenar los datos correspondientes a los usuarios que han realizado pedidos en el sistema. |
| **Datos específicos** | • Productos comprados<br>• Fecha de compra<br>• Dirección de entrega<br>• Identificador |
| **Importancia** | Importante |

***Tabla 14:*** *Requisito de información - Almacenar información de los pedidos [IRQ-004]*

| **IRQ - 005** | **Almacenar información de los comentarios** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [IRQ-001] Almacenar información de los caramelos.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [UC-002] Ver los detalles de un producto en una categoría. |
| **Descripción** | El sistema deberá almacenar los comentarios sobre los productos realizados por los usuarios en el sistema. |

| | |
|---|---|
| **Datos específicos** | • Productos comprados<br>• Fecha de compra<br>• Dirección de entrega<br>• Identificador |
| **Importancia** | Importante |

***Tabla 15:*** *Requisito de información - Almacenar información de los comentarios*

| **IRQ - 006** | **Almacenar información de las valoraciones** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [IRQ-001] Almacenar información de los caramelos.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [UC-002] Ver los detalles de un producto en una categoría. |
| **Descripción** | El sistema deberá almacenar las valoraciones de los productos realizadas por los usuarios en el sistema. |
| **Datos específicos** | • Productos comprados<br>• Fecha de compra<br>• Dirección de entrega<br>• Identificador |
| **Importancia** | Importante |

***Tabla 16:*** *Requisito de información - Almacenar información de las valoraciones*

Una vez que se han detallado los requisitos de información procedamos con los requisitos de reglas de negocio y las restricciones:

| CRQ - 001 | Unicidad de usuarios |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Descripción** | La información almacenada en el sistema deberá satisfacer la siguiente condición: *Los usuarios deben ser únicos en el sistema según su nombre de usuario*. |
| **Importancia** | Importante |

***Tabla 17:*** *Requisito de reglas de negocio y restricciones - Unicidad de usuarios [CRQ-001]*

| CRQ - 002 | Control de acceso sobre usuario anónimo |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Descripción** | La información almacenada en el sistema deberá satisfacer la siguiente condición: *Los usuarios anónimos no deben tener acceso a funcionalidades para usuarios con rol de proveedor o rol de administrador*. |
| **Importancia** | Importante |

***Tabla 18:*** *Requisito de reglas de negocio y restricciones - Control de acceso sobre usuario anónimo [CRQ-002]*

| CRQ - 003 | Control de acceso sobre usuario proveedor |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Descripción** | La información almacenada en el sistema deberá satisfacer la siguiente condición: *Los usuarios con rol de proveedor no deben tener acceso a funcionalidades para usuarios con rol de administrador*. |
| **Importancia** | Importante |

***Tabla 19:*** *Requisito de reglas de negocio y restricciones - Control de acceso sobre usuario proveedor [CRQ-003]*

| CRQ - 004 | Login de usuarios |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Descripción** | La información almacenada en el sistema deberá satisfacer la siguiente condición: *Un usuarios registrado no podrá hacer login en la aplicación hasta que un administrador active su cuenta de usuario para, así, verificar que es un proveedor*. |
| **Importancia** | Importante |

***Tabla 20:*** *Requisito de reglas de negocio y restricciones - Login de usuarios [CRQ-004]*

| CRQ - 005 | Unicidad de producto en el carrito de compra |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Descripción** | La información almacenada en el sistema deberá satisfacer la siguiente condición: *Un producto sólo puede ser añadido una vez al carrito de la compra, pudiéndose modificar solamente la cantidad del mismo*. |
| **Importancia** | Importante |

***Tabla 21:*** *Requisito de reglas de negocio y restricciones - Unicidad de producto en el carrito de compra [CRQ-005]*

## 7.2.2. Requisitos funcionales del sistema

Los requisitos funcionales del sistema sirven para definir las funcionalidades con las que cuenta el sistema a desarrollar, con el único fin de alcanzar los objetivos propuestos anteriormente.

Últimamente, la forma más habitual de especificar estos requisitos es mediante casos de uso.

### 7.2.2.1. Definición de actores del sistema

En este apartado se definen los actores que han sido identificados en las posibles interacciones del sistema. Dichos actores son los siguientes:

| **ACT - 001** | **Usuario anónimo** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | Este actor representa a cualquier usuario del sistema que no está logueado. Las posibles tareas a realizar por este actor las puede realizar el resto de actores. |

***Tabla 22:*** *Actor - Usuario anónimo [ACT-001]*

| **ACT - 002** | **Usuario proveedor** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | Este actor representa a un usuario cualquiera logueado en el sistema, cuyo rol por defecto es “proveedor” . |

***Tabla 23:*** *Actor - Usuario proveedor [ACT-002]*

| ACT - 003 | Usuario administrador |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | Este actor representa a un administrador del sistema. Dicho rol sólo lo posee un usuario en la aplicación . |

***Tabla 24:*** *Actor - Usuario administrador [ACT-003]*

#### 7.2.2.2. Diagramas de casos de uso

Para desarrollar la aplicación web que tiene como objetivo este proyecto, se definen una serie de subsistemas a desarrollar, con el único fin de facilitar dicha tarea.

Dichos subsistemas son los que se presentan a continuación:

**Diagrama de subsistemas**

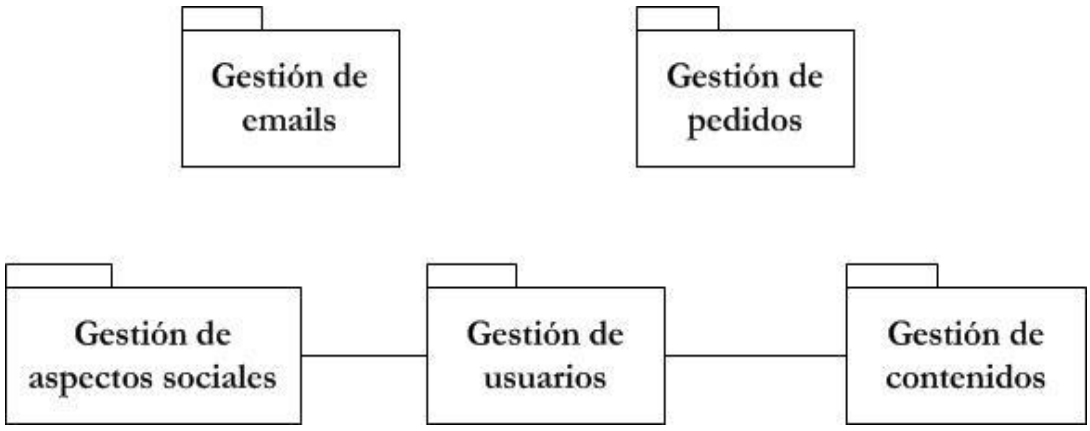



***Ilustración 13:*** *Diagrama de subsistemas*

Empezaremos por los subsistemas que están aislados, los cuales tratan aspectos o funcionalidades muy concretas y directas, y luego pasaremos a los que están íntimamente relacionados, subsistemas algo más complejos.

**Gestión de emails**

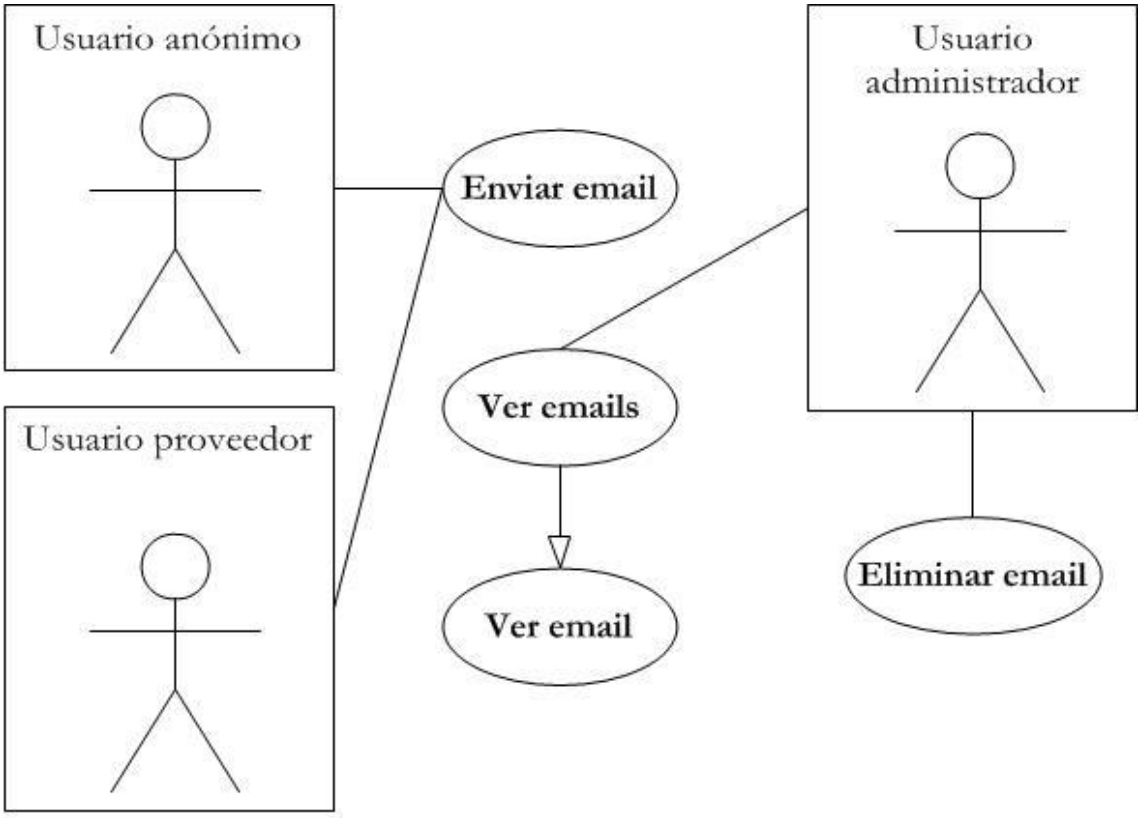



***Ilustración 14:*** *Subsistema de gestión de emails*

En el subsistema de gestión de emails interactúan los 3 actores. En primer lugar, es un usuario anónimo o un usuario proveedor quien envía un email.

Posteriormente, el administrador podrá ver la lista con todos los emails enviados y abrir el que desee.

**Gestión de pedidos**

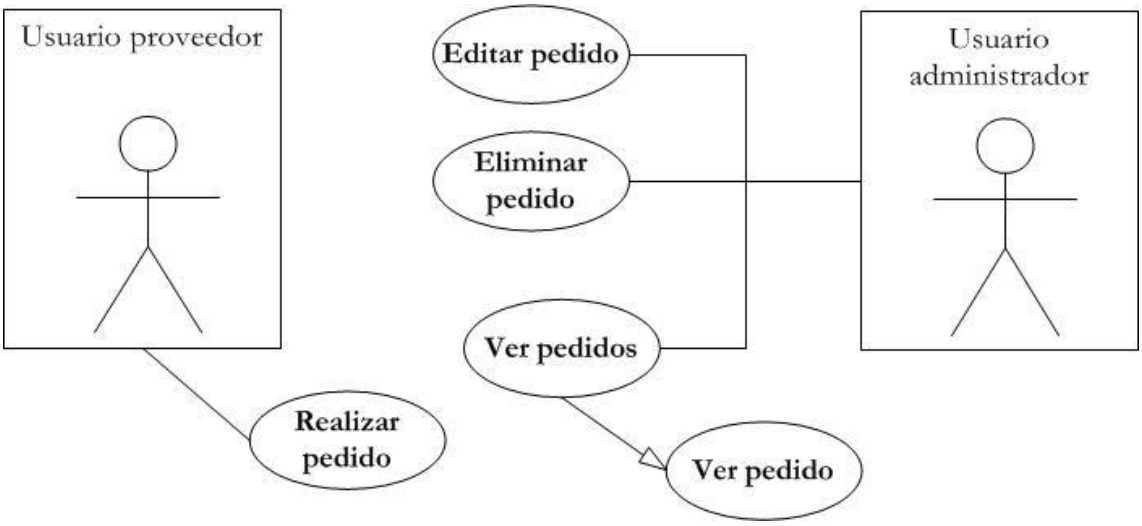



***Ilustración 15:*** *Subsistema de gestión de pedidos*

En el subsistema de gestión de pedidos sólo interactúan los actores Usuario proveedor y Usuario administrador.

En primer lugar, un proveedor realiza un pedido con una serie de productos. Después, el administrador consulta los pedidos y selecciona el pedido que ha sido realizado por el proveedor.

**Gestión de aspectos sociales**

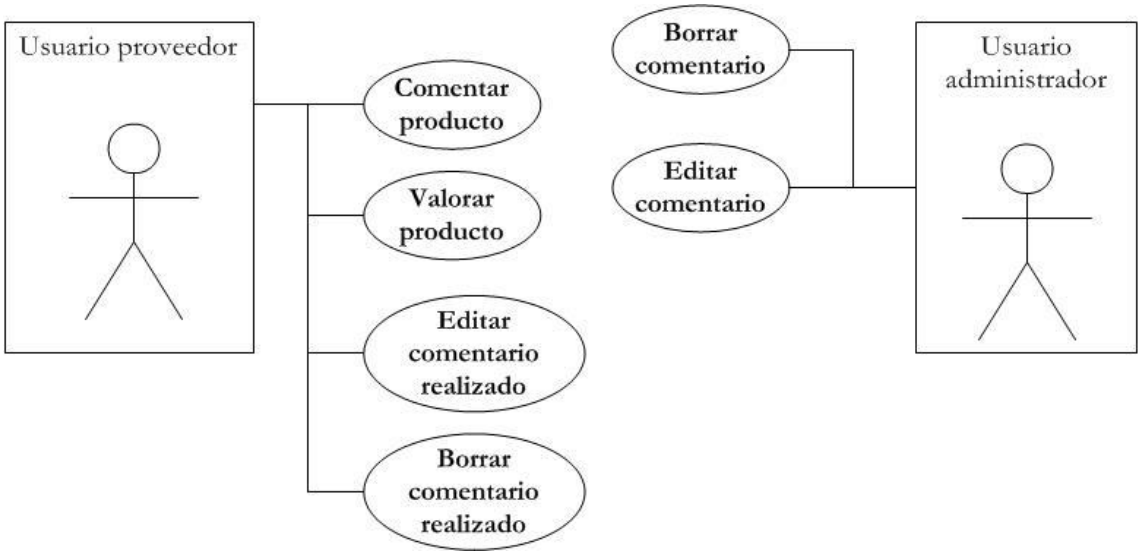



***Ilustración 16:*** *Subsistema de gestión de aspectos sociales*

En el subsistema de gestión de aspectos sociales intervienen sólo 2 actores: el Usuario proveedor y el Usuario administrador.

Un usuario proveedor puede valorar un producto, realizar un comentario, editarlo posteriormente o incluso borrarlo.

En cuanto al Usuario administrador, éste puede editar cualquier comentario o incluso borrarlo, cumpliendo un papel de moderación.

**Gestión de usuarios**

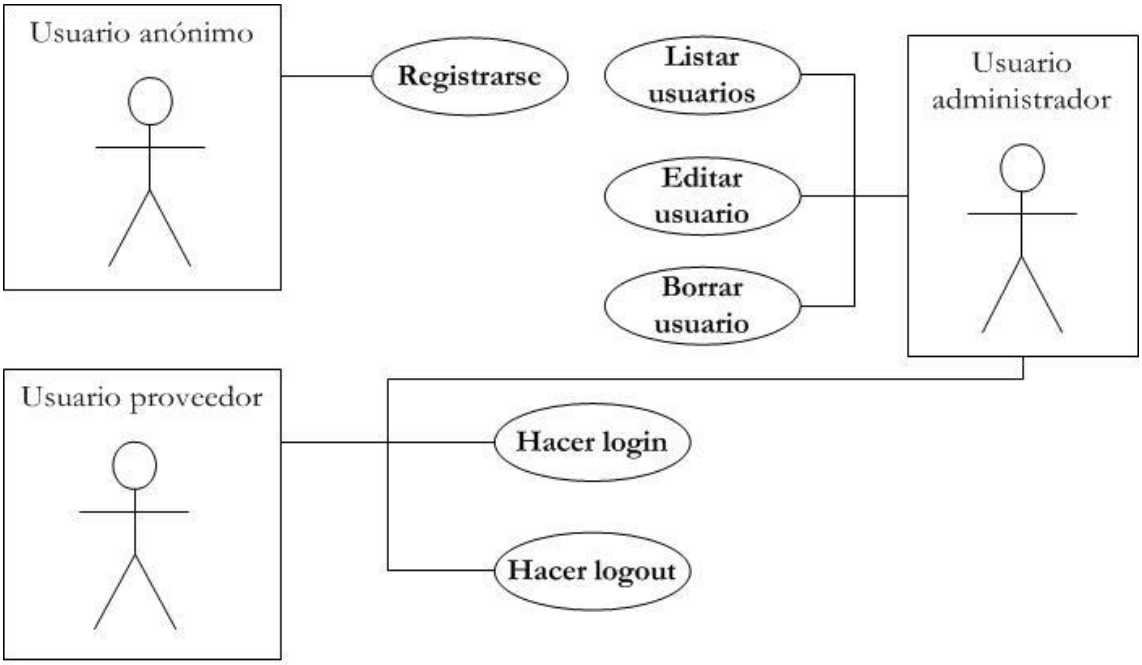



***Ilustración 17:*** *Subsistema de gestión de usuarios*

En el subsistema de gestión de usuarios interactúan todos los actores. Un usuario anónimo sólo puede registrarse ya que, en cuanto lo haga y su cuenta esté activa, pasará a ser un proveedor.

Un Usuario proveedor puede hacer login y hacer logout.

Para finalizar, un usuario administrador puede hacer las dos acciones del usuario proveedor además de: listar usuarios, editar un usuario y borrar un usuario.

**Gestión de contenidos**

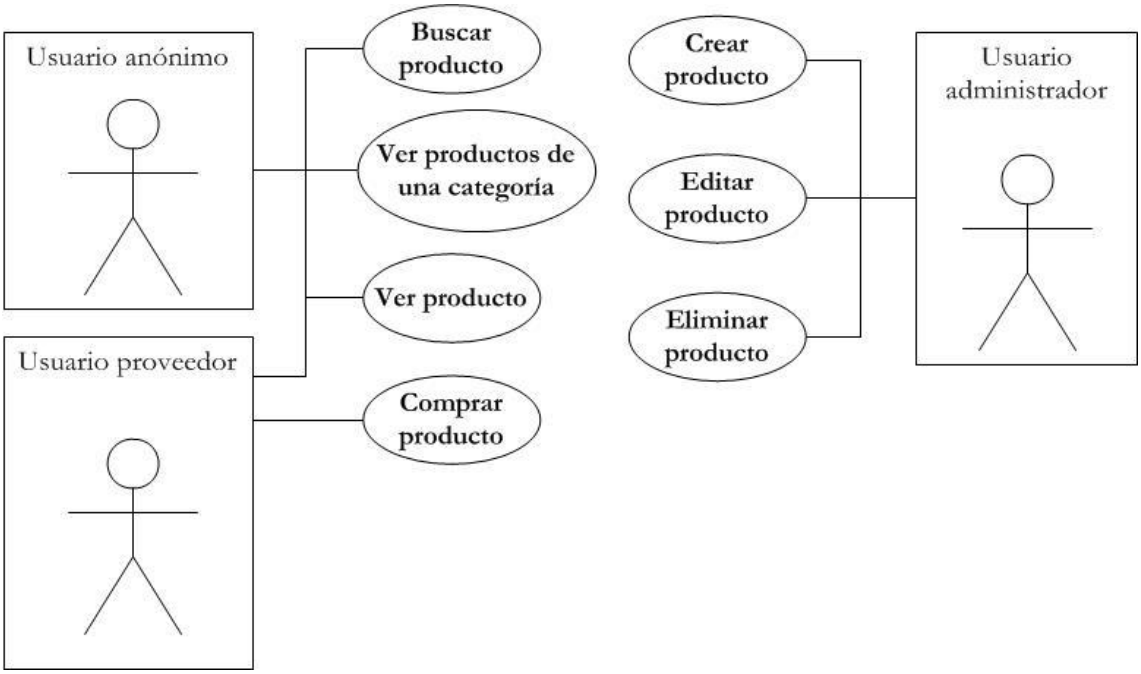



***Ilustración 18:*** *Subsistema de gestión de contenidos*

(Nota: por simplicidad, he relacionado las tareas del actor Usuario anónimo con el actor Usuario proveedor. Esto se debe a que el actor Usuario proveedor puede realizar todas las tareas del actor Usuario anónimo.)

En el subsistema de gestión de contenidos intervienen todos los actores del sistema.

Por un lado, un usuario anónimo puede buscar productos y ver la información del mismo, mientras que por otro lado un usuario proveedor, además de poder realizar esas acciones, puede comprar y valorar un producto.

En cuanto al usuario administrador puede crear, editar o eliminar productos.

### 7.2.2.3. Casos de uso del sistema

Después de haber definido los distintos actores del sistema y los diagramas en los que se detallan las posibles acciones a realizar, se especifican los casos de uso con el máximo nivel de detalle, ordenador por el subsistema al que pertenecen.

Dichos casos de uso son los siguientes:

**Gestión de contenidos**

| **UC - 001** | **Ver los productos de una categoría** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario anónimo (ACT-0001)** desee ver los productos asociados a una categoría. | |
| **Precondición** | La categoría a visualizar debe existir. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario anónimo (ACT-0001)** selecciona la gama en la que se encuentra la categoría del producto. |
| | 2 | El actor **Usuario anónimo (ACT-0001)** selecciona la categoría que quiere visualizar. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 25:*** *Caso de uso – Ver los productos de una categoría [UC-001]*

| **UC – 002** | **Ver los detalles de un producto en una categoría** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos.<br>• [IRQ-004] Almacenar información de los pedidos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario anónimo (ACT-0001)** desee ver la información de un producto asociados a una categoría. | |
| **Precondición** | La categoría y el producto a visualizar deben existir. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario anónimo (ACT-0001)** selecciona la gama en la que se encuentra la categoría del producto. |
| | 2 | El actor **Usuario anónimo (ACT-0001)** selecciona la categoría que quiere visualizar. |
| | 3 | El actor **Usuario anónimo (ACT-0001)** selecciona el producto cuyos detalles quiere conocer. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 26:*** *Caso de uso - Ver los detalles de un producto en una categoría [UC-002]*

| **UC - 003** | **Buscar producto en el sistema** |
|---|---|
| **Versión** | 1.0 (23/05/2014) |

| | | |
|---|---|---|
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario anónimo (ACT-0001)** desee buscar algún producto existente en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario anónimo (ACT-0001)** selecciona 'buscar'. |
| | 2 | El actor **Usuario anónimo (ACT-0001)** selecciona la gama en la que desea buscar. |
| | 3 | El actor **Usuario anónimo (ACT-0001)** selecciona el criterio de búsqueda. |
| | 4 | El actor **Usuario anónimo (ACT-0001)** introduce el nombre del producto o selecciona una categoría, dependiendo del criterio escogido en el paso 3. |
| **Importancia** | Media | |
| **Estado** | Validado | |
| **Estabilidad** | Media | |

***Tabla 27:*** *Caso de uso - Buscar producto en el sistema [UC-003]*

| **UC - 004** | **Comprar producto** |
|---|---|
| **Versión** | 1.0 (23/05/2014) |
| **Autores** | Juan Manuel López Pazos |

<table>
<tr><td>Fuentes</td><td colspan="2">Pablo Fernández Montes</td></tr>
<tr><td>Dependencias</td><td colspan="2">• [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos.<br>• [IRQ-004] Almacenar información de los pedidos.<br>• [CRQ-005] Unicidad de productos en el carrito de compra.</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor Usuario proveedor (ACT-0002) desee comprar algún producto existente en el sistema.</td></tr>
<tr><td>Precondición</td><td colspan="2">El producto debe existir en el sistema.<br>El actor Usuario proveedor (ACT-0002) debe estar logueado en el sistema.</td></tr>
<tr><td rowspan="3">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Se realiza el caso de uso Ver los detalles de un producto en una categoría (UC-0002) .</td></tr>
<tr><td>2</td><td>El actor Usuario proveedor (ACT-0002) selecciona ‘comprar’ .</td></tr>
<tr><td>Postcondición</td><td colspan="2">El producto comprado queda añadido en el carrito de la compra.</td></tr>
<tr><td rowspan="2">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>2</td><td>Si el actor Usuario proveedor (ACT-0002) ya ha comprado anteriormente el producto y éste se encuentra en el carrito, no se le permite volver a comprarlo.</td></tr>
<tr><td>Importancia</td><td colspan="2">Media</td></tr>
<tr><td>Estado</td><td colspan="2">Validado</td></tr>
<tr><td>Estabilidad</td><td colspan="2">Media</td></tr>
</table>

***Tabla 28:*** *Caso de uso - Comprar producto [UC-004]*

<table>
<tr><th>UC - 005</th><th colspan="2">Crear producto</th></tr>
<tr><td>Versión</td><td colspan="2">1.0 (26/02/2014)</td></tr>
<tr><td>Autores</td><td colspan="2">Juan Manuel López Pazos</td></tr>
<tr><td>Fuentes</td><td colspan="2">Pablo Fernández Montes</td></tr>
<tr><td>Dependencias</td><td colspan="2">• [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos.</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor Usuario administrador (ACT-0003) desee crear un producto en el sistema.</td></tr>
<tr><td>Precondición</td><td colspan="2">El producto no debe existir en el sistema.<br>El actor Usuario administrador (ACT-0003) debe estar logueado en la aplicación.</td></tr>
<tr><td rowspan="5">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El actor Usuario administrador (ACT-0003) selecciona la categoría en la que quiere añadir un producto.</td></tr>
<tr><td>2</td><td>El actor Usuario administrador (ACT-0003) selecciona ‘crear producto’.</td></tr>
<tr><td>3</td><td>El actor Usuario administrador (ACT-003) introduce la información del producto.</td></tr>
<tr><td>4</td><td>El actor Usuario administrador (ACT-003) selecciona ‘finalizar’.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El producto queda registrado en el sistema</td></tr>
<tr><td rowspan="2">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>4</td><td>Si el producto a añadir por el actor Usuario administrador (ACT-0003) ya existe en el sistema, no se le permite añadirlo de nuevo.</td></tr>
<tr><td>Importancia</td><td colspan="2">Media</td></tr>
</table>

| | |
|---|---|
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 29:*** *Caso de uso - Crear producto [UC-005]*

| **UC - 006** | **Editar producto** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee editar un producto en el sistema. | |
| **Precondición** | El producto debe existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona la categoría en la que quiere editar un producto. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona ‘editar producto’. |
| | 3 | El actor **Usuario administrador (ACT-003)** modifica la información del producto. |
| | 4 | El actor **Usuario administrador (ACT-003)** selecciona ‘finalizar’. |
| **Postcondición** | El producto queda modificado en el sistema | |
| **Excepciones** | **Paso** | **Acción** |

| | | |
|---|---|---|
| | 4 | Si el código de identificación del producto a añadir por el actor **Usuario administrador (ACT-0003)** ya existe en el sistema, no se le permite modificarlo. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 30:*** *Caso de uso – Editar producto [UC-006]*

| **UC – 007** | **Eliminar producto** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-005] Gestión de contenidos.<br>• [IRQ-001] Almacenar información de los caramelos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee eliminar un producto en el sistema. | |
| **Precondición** | El producto debe existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona la categoría de la que quiere eliminar un producto. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona ‘eliminar producto’. |
| | 3 | El actor **Usuario administrador (ACT-003)** confirma la eliminación del producto. |

| | | |
|---|---|---|
| **Postcondición** | El producto queda eliminado del el sistema | |
| **Excepciones** | **Paso** | **Acción** |
| | 3 | Si el producto a eliminar por el actor **Usuario administrador (ACT-0003)** no pertenecía a la categoría seleccionada, el caso de uso queda sin efecto. |
| **Importancia** | Media | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 31:*** *Caso de uso – Eliminar producto [UC-007]*

**Gestión de usuarios**

| **UC – 008** | **Registrarse como usuario** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-002] Gestión de usuarios.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [CRQ-001] Unicidad de usuarios.<br>• [CRQ-002] Control de acceso sobre usuario anónimo. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario anónimo (ACT-0001)** desee registrarse en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario anónimo (ACT-0001)** selecciona ‘entrar’ en el sistema. |

| | | |
|---|---|---|
| | 2 | El actor **Usuario anónimo (ACT-0001)** selecciona ‘registrarse’ en el sistema. |
| | 3 | El actor **Usuario anónimo (ACT-0001)** introduce sus credenciales. |
| | 4 | El actor **Usuario anónimo (ACT-0001)** selecciona ‘aceptar’. |
| **Postcondición** | El actor **Usuario anónimo (ACT-0001)** queda registrado en el sistema. | |
| **Excepciones** | **Paso** | **Acción** |
| | 4 | Si el usuario introducido por el actor **Usuario anónimo (ACT-0001)** está en uso se le informa y, a continuación, este caso de uso queda sin efecto. |
| | 4 | Si el actor **Usuario anónimo (ACT-0001)** no introduce algún dato se le informa y, a continuación, este caso de uso queda sin efecto. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 32:*** *Caso de uso - Registrarse como usuario [UC-008]*

| **UC – 009** | **Hacer login en el sistema** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |

<table>
<tr><td>Dependencias</td><td colspan="2"><ul><li>[OBJ-001] Estructura de la aplicación web.</li><li>[OBJ-002] Gestión de usuarios.</li><li>[IRQ-003] Almacenar información de los usuarios.</li><li>[CRQ-002] Control de acceso sobre usuario anónimo.</li><li>[CRQ-004] Login de usuarios.</li></ul></td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor Usuario proveedor (ACT-0002) desee hacer login en el sistema.</td></tr>
<tr><td>Precondición</td><td colspan="2">El actor Usuario proveedor (ACT-0002) debe estar registrado en el sistema.</td></tr>
<tr><td rowspan="4">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El actor Usuario proveedor (ACT-0002) selecciona ‘entrar’ en el sistema.</td></tr>
<tr><td>2</td><td>El actor Usuario proveedor (ACT-0002) introduce sus credenciales.</td></tr>
<tr><td>3</td><td>El actor Usuario proveedor (ACT-0002) selecciona ‘aceptar’.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El actor Usuario proveedor (ACT-0002) queda logueado en el sistema.</td></tr>
<tr><td rowspan="4">Excepciones</td><td>Paso</td><td>Acción</td></tr>
<tr><td>3</td><td>Si el usuario introducido por el actor Usuario proveedor (ACT-0002) no existe en el sistema se le informa y, a continuación, este caso de uso queda sin efecto.</td></tr>
<tr><td>3</td><td>Si el actor Usuario proveedor (ACT-0002) introduce una contraseña que no se corresponde con la del usuario indicado y registrado en el sistema se le informa y, a continuación, este caso de uso queda sin efecto.</td></tr>
<tr><td>3</td><td>Si el actor Usuario proveedor (ACT-0002) no introduce algún dato se le informa y, a continuación, este caso de uso queda sin efecto.</td></tr>
</table>

| | |
|---|---|
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 33:*** *Caso de uso - Hacer login en el sistema [UC-009]*

| **UC - 010** | **Hacer logout en el sistema** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-002] Gestión de usuarios.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [CRQ-003] Control de acceso sobre usuario proveedor. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee hacer logout en el sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario proveedor (ACT-0002)** selecciona ‘salir’ del sistema. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** selecciona ‘aceptar’. |
| **Postcondición** | El actor **Usuario proveedor (ACT-0002)** ya no está logueado en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |

| | |
|---|---|
| **Estabilidad** | Alta |

***Tabla 34:*** *Caso de uso - Hacer logout en el sistema [UC-010]*

| **UC - 011** | **Listar usuarios registrados en el sistema** | |
|---|---|---|
| **Versión** | 1.0 (26/02/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-002] Gestión de usuarios.<br>• [IRQ-003] Almacenar información de los usuarios. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee listar todos los usuarios registrados en el sistema. | |
| **Precondición** | El actor **Usuario administrador (ACT-0002)** debe estar logueado en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0002)** selecciona ‘usuarios’ en el sistema. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 35:*** *Caso de uso - Listar usuarios registrados en el sistema [UC-011]*

| **UC - 012** | **Editar usuario registrado en el sistema** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |

| | | |
|---|---|---|
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-002] Gestión de usuarios.<br>• [IRQ-003] Almacenar información de los usuarios.<br>• [CRQ-001] Unicidad de usuarios. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee editar un usuario registrado en el sistema. | |
| **Precondición** | El actor **Usuario administrador (ACT-0003)** debe estar logueado en el sistema.<br>El usuario a editar debe estar registrado en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona ‘usuarios’ en el sistema. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona ‘editar’ usuario en el sistema. |
| | 3 | El actor **Usuario administrador (ACT-0003)** modifica la información del usuario del sistema. |
| | 4 | El actor **Usuario administrador (ACT-0003)** finaliza la edición del usuario. |
| **Postcondición** | El usuario queda editado en el sistema. | |
| **Excepciones** | **Paso** | **Acción** |
| | 4 | Si el nuevo nombre de usuario está ocupado por otro usuario en el sistema se informa al actor **Usuario administrador (ACT-0003)** y, a continuación, el caso de uso queda sin efecto. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 36:*** *Caso de uso - Editar usuario registrado en el sistema [UC-012]*

<table>
<tr><th>UC - 013</th><th colspan="2">Eliminar usuario registrado en el sistema</th></tr>
<tr><td>Versión</td><td colspan="2">1.0 (26/02/2014)</td></tr>
<tr><td>Autores</td><td colspan="2">Juan Manuel López Pazos</td></tr>
<tr><td>Fuentes</td><td colspan="2">Pablo Fernández Montes</td></tr>
<tr><td>Dependencias</td><td colspan="2">• [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-002] Gestión de usuarios.<br>• [IRQ-003] Almacenar información de los usuarios.</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor Usuario administrador (ACT-0003) desee eliminar un usuario registrado en el sistema.</td></tr>
<tr><td>Precondición</td><td colspan="2">El actor Usuario administrador (ACT-0003) debe estar logueado en el sistema.<br>El usuario a eliminar debe estar registrado en el sistema.</td></tr>
<tr><td rowspan="5">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El actor Usuario administrador (ACT-0003) selecciona ‘usuarios’ en el sistema.</td></tr>
<tr><td>2</td><td>El actor Usuario administrador (ACT-0003) selecciona ‘editar’ usuario en el sistema.</td></tr>
<tr><td>3</td><td>El actor Usuario administrador (ACT-0003) modifica la información del usuario del sistema.</td></tr>
<tr><td>4</td><td>El actor Usuario administrador (ACT-0003) finaliza la edición del usuario.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El usuario queda editado en el sistema.</td></tr>
<tr><td>Importancia</td><td colspan="2">Vital</td></tr>
<tr><td>Estado</td><td colspan="2">Validado</td></tr>
<tr><td>Estabilidad</td><td colspan="2">Alta</td></tr>
</table>

***Tabla 37:*** *Caso de uso - Eliminar usuario registrado en el sistema [UC-013]*

**Gestión de aspectos sociales**

| **UC - 014** | **Comentar producto** | |
|---|---|---|
| **Versión** | 1.0 (11/07/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-003] Aspectos sociales.<br>• [IRQ-005] Almacenar información de los comentarios. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee comentar un producto del sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado en el sistema.<br>El producto a comentar debe estar registrado en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Ver los detalles de un producto en una categoría (UC-0002)**. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** selecciona ‘comentar’ el producto registrado en el sistema. |
| | 3 | El actor **Usuario proveedor (ACT-0002)** acepta finaliza el comentario del producto registrado en el sistema. |
| **Postcondición** | El comentario del producto queda registrado en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Media | |

***Tabla 38:*** *Caso de uso – Comentar producto [UC-014]*

| **UC – 015** | **Valorar producto** | |
|---|---|---|
| **Versión** | 1.0 (11/07/2014) | |
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-003] Aspectos sociales.<br>• [IRQ-006] Almacenar información de las valoraciones. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee valorar un producto del sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado en el sistema.<br>El producto a valorar debe existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Ver los detalles de un producto en una categoría (UC-0002)**. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** valorar el producto existente en el sistema. |
| **Postcondición** | La valoración del producto queda registrada en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Baja | |

***Tabla 39:*** *Caso de uso - Valorar producto [UC-015]*

| **UC - 016** | **Editar comentario realizado de un producto** |
|---|---|
| **Versión** | 1.0 (11/07/2014) |

| | | |
|---|---|---|
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-003] Aspectos sociales.<br>• [IRQ-005] Almacenar información de los comentarios. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee editar el comentario realizado sobre un producto del sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado en el sistema.<br>El comentario y el producto a editar deben existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Ver los detalles de un producto en una categoría (UC-0002)**. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** selecciona ‘editar’ el comentario realizado del producto existente en el sistema. |
| | 3 | El actor **Usuario proveedor (ACT-0002)** finaliza la edición del comentario. |
| **Postcondición** | El comentario del producto queda editado en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Media | |

***Tabla 40:*** *Caso de uso - Editar comentario realizado de un producto [UC-016]*

| **UC - 017** | **Eliminar comentario realizado de un producto** |
|---|---|
| **Versión** | 1.0 (11/07/2014) |
| **Autores** | Juan Manuel López Pazos |

| | | |
|---|---|---|
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-003] Aspectos sociales.<br>• [IRQ-005] Almacenar información de los comentarios. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee eliminar el comentario realizado sobre un producto del sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado en el sistema.<br>El comentario y el producto a eliminar deben existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Ver los detalles de un producto en una categoría (UC-0002)**. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** selecciona 'eliminar' el comentario realizado del producto existente en el sistema. |
| | 3 | El actor **Usuario proveedor (ACT-0002)** confirma la eliminación del comentario. |
| **Postcondición** | El comentario del producto queda eliminado en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Baja | |

***Tabla 41:*** *Caso de uso - Eliminar comentario realizado de un producto [UC-017]*

**Gestión de emails**

| **UC - 018** | **Enviar email** |
|---|---|
| **Versión** | 1.0 (11/07/2014) |

| | | |
|---|---|---|
| **Autores** | Juan Manuel López Pazos | |
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-006] Gestión de emails.<br>• [IRQ-002] Almacenar información de los emails. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario anónimo (ACT-0001)** desee enviar un email al sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario anónimo (ACT-0001)** selecciona ‘contacto’. |
| | 2 | El actor **Usuario anónimo (ACT-0001)** rellena el formulario con los datos solicitados para enviar el email. |
| | 3 | El actor **Usuario anónimo (ACT-0001)** envía el email al sistema. |
| **Postcondición** | El email queda registrado en el sistema. | |
| **Excepciones** | **Paso** | **Acción** |
| | 3 | Si el usuario introduce algún campo vacío o incorrecto se le informa y, a continuación, el caso de uso queda sin efecto. |
| **Importancia** | Importante | |
| **Estado** | Validado | |
| **Estabilidad** | Media | |

***Tabla 42:*** *Caso de uso - Enviar email [UC-018]*

| **UC - 019** | **Ver los emails recibidos** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |

| | | |
|---|---|---|
| **Fuentes** | Pablo Fernández Montes | |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-006] Gestión de emails.<br>• [IRQ-002] Almacenar información de los emails. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee leer los emails enviados al sistema. | |
| **Precondición** | El actor **Usuario administrador (ACT-0003)** debe estar logueado en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona ‘emails’. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona el email que quiera leer. |
| **Postcondición** | El email se marca en el sistema como ‘leído’. | |
| **Importancia** | Importante | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 43:*** *Caso de uso - Ver los emails recibidos [UC-019]*

| **UC - 020** | **Eliminar email** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-006] Gestión de emails.<br>• [IRQ-002] Almacenar información de los emails. |

| | | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee eliminar un email enviado al sistema. | |
| **Precondición** | El actor **Usuario administrador (ACT-0003)** debe estar logueado en el sistema.<br>El email tiene que existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Ver los emails recibidos (UC-0018)**. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona 'eliminar' el email que quiera borrar. |
| **Postcondición** | El email queda eliminado en el sistema. | |
| **Excepciones** | **Paso** | **Acción** |
| | 2 | Si el email a eliminar no existe se informa al actor **Usuario administrador (ACT-0003)** y, a continuación, el caso de uso queda sin efecto. |
| **Importancia** | Importante | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 44:*** *Caso de uso - Eliminar email [UC-020]*

**Gestión de pedidos**

| **UC - 021** | **Realizar pedido** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |

| | | |
|---|---|---|
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-007] Gestión de pedidos.<br>• [IRQ-004] Almacenar información de los pedidos. | |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario proveedor (ACT-0002)** desee realizar un pedido en el sistema. | |
| **Precondición** | El actor **Usuario proveedor (ACT-0002)** debe estar logueado. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | Se realiza el caso de uso **Comprar producto (UC-0004)**. |
| | 2 | El actor **Usuario proveedor (ACT-0002)** selecciona el 'carrito'. |
| | 3 | El actor **Usuario proveedor (ACT-0002)** confirma el pedido. |
| **Postcondición** | El pedido queda registrado en el sistema. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 45:*** *Caso de uso - Realizar pedido [UC-021]*

| **UC - 022** | **Ver los pedidos realizados** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-007] Gestión de pedidos.<br>• [IRQ-004] Almacenar información de los pedidos. |

| | | |
|---|---|---|
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee ver los pedidos registrados en el sistema. | |
| **Precondición** | El actor **Usuario administrador (ACT-0003)** debe estar logueado. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona ‘pedidos’. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona el pedido que quiera observar. |
| **Postcondición** | El pedido se marca en el sistema como ‘leído’. | |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 46:*** *Caso de uso - Ver los pedidos realizados [UC-022]*

| **UC - 023** | **Editar un pedido realizado** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Dependencias** | • [OBJ-001] Estructura de la aplicación web.<br>• [OBJ-007] Gestión de pedidos.<br>• [IRQ-004] Almacenar información de los pedidos. |
| **Descripción** | El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor **Usuario administrador (ACT-0003)** desee editar un pedido registrado en el sistema. |

| | | |
|---|---|---|
| **Precondición** | El actor **Usuario administrador (ACT-0003)** debe estar logueado.<br>El pedido debe existir en el sistema. | |
| **Secuencia normal** | **Paso** | **Acción** |
| | 1 | El actor **Usuario administrador (ACT-0003)** selecciona ‘pedidos’. |
| | 2 | El actor **Usuario administrador (ACT-0003)** selecciona ‘editar’ el pedido. |
| | 3 | El actor **Usuario administrador (ACT-0003)** edita el pedido. |
| | 4 | El actor **Usuario administrador (ACT-0003)** confirma la edición del pedido. |
| **Postcondición** | El pedido queda editado en el sistema. | |
| **Excepciones** | **Paso** | **Acción** |
| | **4** | Si ya existe un pedido con el identificador de pedido modificado se informa al actor **Usuario administrador (ACT-0003)** y, a continuación, el caso de uso queda sin efecto. |
| **Importancia** | Vital | |
| **Estado** | Validado | |
| **Estabilidad** | Alta | |

***Tabla 47:*** *Caso de uso - Editar un pedido realizado [023]*

| **UC - 024** | **Eliminar un pedido realizado** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |

<table>
<tr><td>Dependencias</td><td colspan="2"><ul><li>[OBJ-001] Estructura de la aplicación web.</li><li>[OBJ-007] Gestión de pedidos.</li><li>[IRQ-004] Almacenar información de los pedidos.</li></ul></td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema deberá comportarse tal cual se describe en el siguiente caso de uso cuando el actor Usuario administrador (ACT-0003) desee eliminar un pedido registrado en el sistema.</td></tr>
<tr><td>Precondición</td><td colspan="2">El actor Usuario administrador (ACT-0003) debe estar logueado.<br>El pedido debe existir en el sistema.</td></tr>
<tr><td rowspan="4">Secuencia normal</td><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El actor Usuario administrador (ACT-0003) selecciona ‘pedidos’.</td></tr>
<tr><td>2</td><td>El actor Usuario administrador (ACT-0003) selecciona ‘eliminar’ el pedido.</td></tr>
<tr><td>3</td><td>El actor Usuario administrador (ACT-0003) confirma la eliminación del pedido.</td></tr>
<tr><td>Postcondición</td><td colspan="2">El pedido queda eliminado del el sistema.</td></tr>
<tr><td>Importancia</td><td colspan="2">Vital</td></tr>
<tr><td>Estado</td><td colspan="2">Validado</td></tr>
<tr><td>Estabilidad</td><td colspan="2">Alta</td></tr>
</table>

***Tabla 48:*** *Caso de uso - Eliminar un pedido realizado [UC-024]*

## 7.2.3. Requisitos no funcionales

Los requisitos no funcionales están compuestos por todas aquellas características y condiciones que deben ser tenidas en cuenta a la hora de desarrollar la aplicación web, a saber: seguridad, usabilidad, rendimiento, disponibilidad, etc.

Dicho esto, a continuación se presentan los requisitos funcionales del sistema, tanto desde el punto de vista del servidor como de la aplicación.

| **NFR - 001** | **Sistemas operativos** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá funcionar sobre la mayor cantidad posible de sistemas operativos, pero es fundamental que funcione en los sistemas operativos Windows. |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 49:*** *Requisito no funcional - Sistemas operativos [NFR-001]*

| **NFR - 002** | **Servidor web** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá funcionar sobre un servidor web típico que opere con el protocolo HTTP. |
| **Importancia** | Vital |

| | |
|---|---|
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 50:*** *Requisito no funcional - Servidor web [NFR-002]*

| **NFR - 003** | **Tecnología web** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá instalarse con todas las dependencias de **Node.js**, las cuales se instalan mediante el comando "npm install", y con el framework **AngularJS**. |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 51:*** *Requisito no funcional - Tecnología web [NFR-003]*

| **NFR – 004** | **Sistema gestor de Base de Datos** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá disponer de **MongoDB 2.6.1** (http://www.mongodb.org/). |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Media |

***Tabla 52:*** *Requisito no funcional - Sistema gestor de Base de Datos [NFR-004]*

| **NFR – 005** | **Accesibilidad y usabilidad** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá ofrecer una interfaz amigable y simple, además de mantener los colores utilizados en **La Gloria S.L**. |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Media |

***Tabla 53:*** *Requisito no funcional - Accesibilidad y usabilidad [NFR-005]*

| **NFR – 006** | **Calidad y seguridad** |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá cumplir los requisitos mínimos de calidad y seguridad. De este modo, la aplicación no supondrá ningún riesgo o amenaza para los usuarios. |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Alta |

***Tabla 54:*** *Requisito no funcional - Calidad y seguridad [NFR-006]*

| NFR – 007 | Entorno de explotación |
|---|---|
| **Versión** | 1.0 (26/02/2014) |
| **Autores** | Juan Manuel López Pazos |
| **Fuentes** | Pablo Fernández Montes |
| **Descripción** | El sistema deberá ser compatible con el mayor número posible de navegadores web, pero siendo de vital importancia que funcione en **Chrome 36.0.1985.143 m**. |
| **Importancia** | Vital |
| **Estado** | Validado |
| **Estabilidad** | Media |

***Tabla 55:*** *Requisito no funcional - Entorno de explotación [NFR-007]*

# 8. ESTUDIO DE TECNOLOGÍAS

En este capítulo se realiza un estudio de las diferentes tecnologías que se han contemplado para desarrollar la aplicación web.

En la primera sección se hablará de la base de datos utilizada, la cual ha sido **MongoDB**, una base de datos no relacional orientada a documentos.

En la segunda sección se hablará acerca de **Node.js**, el lenguaje **JavaScript** que se ha empleado para la implementación del servidor.

En la tercera sección se explicará con bastantes detalles en qué consiste **AngularJS**, un framework **JavaScript** desarrollado por Google, tecnología que quizás sea la más importante de este trabajo.

En la siguiente sección se explicará **JADE**, un motor de plantillas **HTML** que ofrece muchas prestaciones interesantes enfocadas a la reutilización de código.

Y para finalizar se hablará de **Bootstrap** y de **Fontawesome**. **Bootstrap** es una librería css muy famosa, ya que es la que se utiliza en Twitter, y que cuenta con “responsive design”, lo cual nos es de gran utilidad para diseñar una web que se adapte a cualquier dispositivo. También se introducirá un poco **Fontawesome**, que no es más que una pequeña librería css conocida por sus famosos iconos.

# 8.1 MongoDB


***Ilustración 19:*** *Tecnologías –* ***MongoDB***

**MongoDB** es una base de datos **NoSQL** (no relacional) orientada a documentos. Los datos se guardan en colecciones con formato **JSON**, aunque internamente se almacenan en formato **BSON**, una variante de **JSON** con más detalles utilizada por **MongoDB**.

Esta base de datos está basada en el teorema **CAP**, según el cual se define que, de las 3 propiedades existentes (Consistencia, Disponibilidad y Tolerancia a fallos) sólo se pueden tener 2.

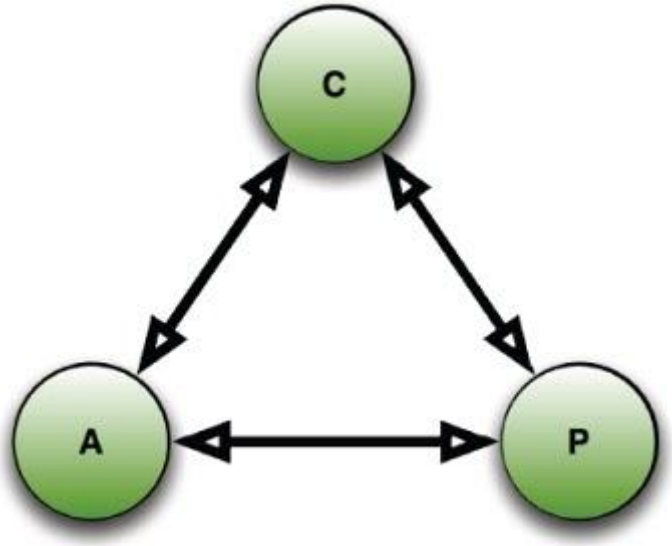



***Ilustración 20:*** *Tecnologías - Teorema CAP*

En el caso de **MongoDB** se dispone por defecto de Consistencia y Tolerancia a fallos, pero se puede cambiar la configuración del nivel de consistencia, de modo que podemos sacrificar consistencia para ganar disponibilidad.

Una particularidad altamente destacable de **MongoDB** es que los datos almacenados, al guardarse en formato **JSON**, son considerados como Strings salvo contadas excepciones como sucede con las fechas (guardadas como ISODate) o los ids (guardados como ObjectId), entre otros.

# Comparación entre MongoDB y MySQL

Para terminar de entender los conceptos básicos de **MongoDB** se presenta a continuación una comparativa con una base de datos relacional como es **MySQL**, de modo que podremos ver las principales diferencias que supondría desarrollar una aplicación web cambiando **MongoDB** por una base de datos relacional.

| | **MongoDB** | **MySQL** |
|---|---|---|
| Plataformas | Windows, Linux, OS X y Solaris | Windows, Linux, Mac, Solaris y muchos más |
| Implementación | C++ | C y C++ |
| Sistema de almacenamiento de datos | Sistema de archivos | Transaccionales y no transaccionales |
| Formato de los datos | Colecciones > Documentos > Campos | Tablas > Filas > Columnas |
| Replicación | Sí | Sí, pero con muchas limitaciones |
| Balanceo de carga | Escalación horizontal mediante shards | Mediante extensiones software (HAProxy) o hardware |
| Instalación | Fácil (mediante el terminal) | Dificultad media (mediante asistente de instalación) |
| Lenguajes soportados | C, C++, Java, PHP, Python, Node.js, Ruby, y muchos más | C, C++, Java, PHP, Ruby y algunos más |
| Búsquedas full-text | No soportadas (disponibles en versión beta) | Sí |

***Tabla 56:*** *Tecnologías - Comparación entre* ***MongoDB*** *y* ***MySQL***

**MongoDB** soporta diversas plataformas como **Windows** (a partir de Windows 7), Linux, Mac y Solaris, pero **MySQL** soporta muchísimas más plataformas.

Ambas bases de datos están implementadas en el lenguaje **C++**, pero **MySQL** también tiene algunas implementaciones realizadas en el lenguaje **C**.

A la hora de clasificar estas bases de datos podemos decir que **MongoDB** es una base de datos basada en sistemas de archivos, mientras que **MySQL** puede estar basadas en procesos transaccionales o en procesos no transaccionales.

En caso de que usáramos **MySQL** con procesos transaccionales, en caso de conflicto en la base de datos se volvería a un estado estable anterior, lo que se denomina rollback. Sin embargo, eso no se puede realizar con **MongoDB**. No obstante, según la documentación oficial de **MongoDB**, es posible simular una transacción. Sin embargo, hay que realizar a mano una serie de colecciones y configurarlas de una forma determinada. A pesar de todo, pienso que esa solución no es demasiado técnica.

En **MongoDB** la información se almacena en ficheros, mientras que en **MySQL** se guarda en tablas. Podemos decir que una colección en **MongoDB** es equivalente a una tabla en **MySQL**, que un documento en **MongoDB** es equivalente a una fila en **MySQL** y que un campo de **MongoDB** es equivalente a una columna en **MySQL**.

En la siguiente imagen del workbench de **MySQL** podemos ver en la práctica cómo se almacena la información:

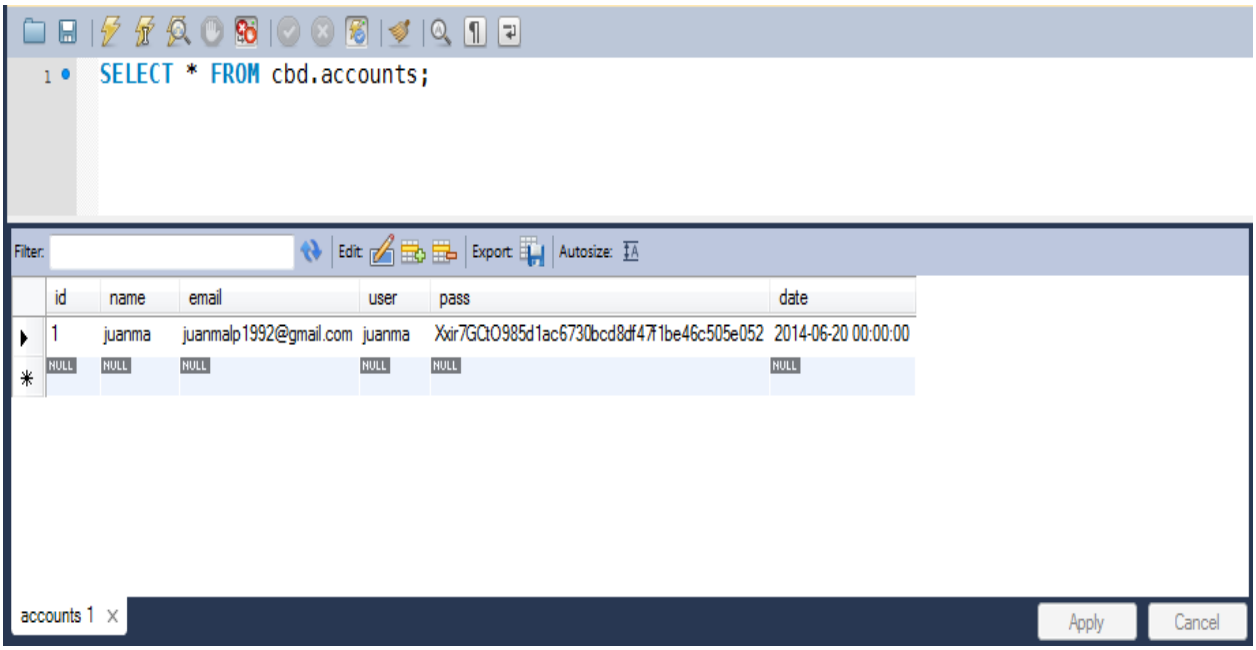



***Ilustración 21:*** *Tecnologías - Estructura de la información en* ***MySQL***

Sin embargo, en esta otra imagen de la interfaz de **Robomongo** (una herramienta para visualización y edición rápida para **MongoDB**) podemos ver cómo se almacena en **MongoDB**:

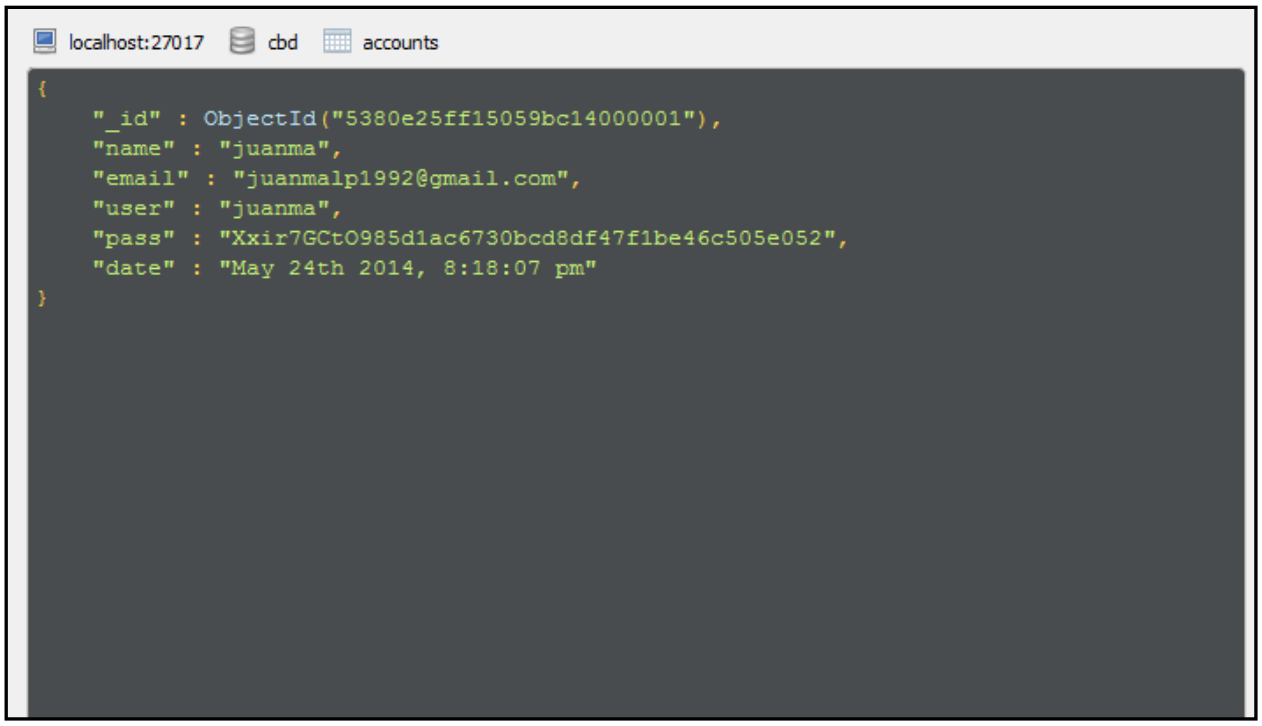



***Ilustración 22:*** *Tecnologías - Estructura de la información en* ***MongoDB***

La replicación se contempla en **MongoDB** como una relación Maestro-Esclavo. En **MySQL** también se puede replicar la información, pero es muy costoso debido a la forma en que se almacena y distribuye la información en los servidores.

**MongoDB** se puede escalar de forma horizontal, de modo que se puede repartir la información de un documento entre distintos servidores. Esto mismo no se puede hacer de forma tan sencilla con **MySQL**, ya que las limitaciones tecnológicas son grandes.

A la hora de instalar **MongoDB** no tenemos más que seguir la documentación oficial. La instalación se realiza mediante el terminal y en pocos pasos. En cuanto a **MySQL**, la instalación no llega a ser compleja ya que se realiza mediante un wizzard de instalación. Sin embargo, hay que realizar muchas configuraciones, por lo que para algún principiante puede convertirse en una tarea tediosa.

En cuanto a los lenguajes soportados, **MongoDB** cobra ventaja ya que consta de muchos drivers oficiales para los lenguajes mencionados en la tabla y otros como **Scala**, **JavaScript** o **Haskell**, entre otros. **MySQL** también soporta diversos lenguajes, pero en menor cantidad que **MongoDB**.

Otro aspecto que me ha parecido interesante analizar ha sido las búsquedas full-text. **MongoDB** no soporta las búsquedas full-text en las versiones oficiales (aunque sí las soporta en una versión beta). Sin embargo, esta característica sí la proporciona **MySQL**.

Dicho todo esto, podemos concluir que **MongoDB** es una base de datos menos estricta que las bases de datos relacionales, ya que no hay control del tipo de los datos que se almacenan, los documentos pueden tener distintos campos dentro de una misma colección, etc.

Esto último es uno de los motivos por los que la base de datos seleccionada ha sido **MongoDB**, ya que la caramelos catalogados en **La Gloria S.L.** presentan una clasificación única para cada categoría, por lo que usar una base de datos como **MongoDB** facilitaría mucho la inserción y el acceso a los datos.

# 8.2. Node.js

***Ilustración 23:*** *Tecnologías -* ***Node.js***

Este lenguaje nació en 2009, por lo que es bastante reciente. Se trata de un lenguaje **JavaScript** asíncrono y orientado a eventos para el lado del servidor.

*"**Node.js** creado por Ryan Dahl en ... 2009, este servidor trabaja con el lenguaje de programación **JavaScript** y como núcleo en su arquitectura cuenta con el motor **JavaScript V8".***[8]

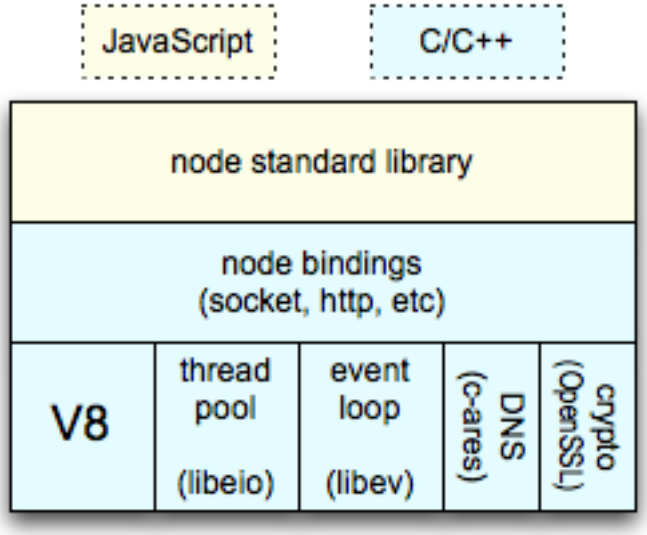



***Ilustración 24:*** *Tecnologías - Arquitectura de* ***Node.js***

Está formado por módulos básicos para su funcionamiento, pero hay muchísimos otros oficiales o de terceros que no están incluidos. Estos módulos se pueden instalar mediante el NPM (Node Package Manager), el gestor de

[8] http://www.nodehispano.com/2011/11/que-es-node-js-nodejs/

paquetes para **Node.js**. A diferencia de otros lenguajes de servidor, utiliza un único hilo común para todas las peticiones. Esto implica mayor escalabilidad pero también que sólo se pueda utilizar una única CPU.

Este lenguaje es ideal para implementar una **API REST**. Por lo tanto, como nosotros teníamos en mente hacer uso de la filosofía **REST** a la hora de usar los recursos en nuestra web, este lenguaje era una de las mejores opciones de las que disponíamos.

A continuación se muestran una serie de ejemplos para entender las particularidades de **Node.js**. Primero se muestra cómo configurar un servidor típico de **Node.js**, luego mostraremos cómo implementar las rutas de nuestra aplicación, bien sean parte de una **API REST** o devuelvan una vista o algún otro objeto, y para finalizar se muestra cómo se realizan las consultas en **Node.js** a una base de datos como **MongoDB**, esto último gracias a uno de los muchos drivers que existen para **Node.js**.

Por seguir unas buenas prácticas, se suele crear un archivo para la configuración y ejecución del servidor y otro archivo para las rutas de la aplicación, pero se podría hacer todo en un mismo fichero.

Esto también dependerá en gran medida de la aplicación que queramos desarrollar. En este caso, previendo el tamaño que iba a tener, era bastante recomendable seguir el mayor número de buenas prácticas posible, lo cual facilitará su trabajo a futuros administradores de la aplicación.

```
33  var express     = require('express');
34  var app         = express();
35  var path        = require("path");
36
37  var port = 8888;
38
39  app.configure(function() {
40      app.set('port', port);
41      app.set('views', path.join(__dirname + '/app/server/views'));
42      app.set('view engine', 'jade');
43      app.use(express.bodyParser());
44      app.use(express.session({ secret: 'super-duper-secret-secret' }));
45      app.use(express.favicon(path.join(__dirname, '/app/public/img/logo.png')));
46      app.use(express.static(__dirname + '/app/public'));
47      app.use('/img', express.static(__dirname + '/app/public/img'));
48      app.use('/js', express.static(__dirname + '/app/public/js'));
49  });
50
51  require('./app/server/router')(app);
52
53  app.listen(port, function() {
54      console.log('App listening on port ' + port);
55  });
56
```

***Ilustración 25:*** *Tecnologías - Archivo principal de* ***Node.js***

Si configuramos el servidor en un archivo “app.js”, tendría un aspecto similar al de la imagen anterior, donde se muestra una posible configuración de un servidor **HTTP** provisto por **express**, una framework **JavaScript** para **Node.js**.

De forma habitual, las configuraciones que se realizan son: el puerto de escucha, la ubicación de los archivos estáticos, la renderización de las vistas (**HTML**, **JADE** u otro lenguaje), etc.

Como se puede observar, al final se indica dónde están las rutas de nuestra aplicación, las cuales han sido definidas en un archivo “router.js”.

Por ejemplo, en la aplicación web que he desarrollado una de las rutas es la siguiente:

```
18      app.get('/', function(req, res) {
19          funcionesComunes(req);
20          if(req.session.user == null){
21              res.render('index');
22          }else{
23              if(req.session.user.role == "admin"){
24                  res.render('admin/dashboard');
25              }else{
26                  res.render('index');
27              }
28          }
29      });
30
```

***Ilustración 26:*** *Tecnologías - Petición de la página de inicio al servidor*

En este método se solicita la vista principal de la aplicación. Sin embargo, esta vista dependerá de si estamos o no logueados y del rol que tengamos como usuario.

Para un usuario anónimo o un usuario proveedor la página principal es la misma salvo pequeños detalles.

Sin embargo, para un administrador, la pantalla que se muestra es el panel de administración, el cual no tiene nada que ver con la aplicación en sí.

En caso de que el usuario esté logueado, el rol que posee es comprobado consultando el usuario almacenado en una variable de sesión, la cual se inicializa en el momento del login del usuario.

Como ejemplo de método de una **API REST** tenemos el siguiente:

```
664     app.get('/api/emails', function(req, res){
665         if(req.session.user == null){
666             res.send('unauthorized',400);
667         }else{
668             if(req.session.user.role == "admin"){
669                 DBM.getAllEmails(function(err, mails){
670                     if(err){
671                         console.log(err);
672                     }else{
673                         res.send(mails, 200);
674                     }
675                 });
676             }else{
677                 res.send('unauthorized', 400);
678             }
679         }
680     });
681
```

***Ilustración 27:*** *Tecnologías – Petición de los emails del sistema al servidor*

En este método “get” de la **API REST** definida en la aplicación web desarrollada se consultan todos los emails almacenados en la base de datos, los cuales han sido enviados por usuarios anónimos o proveedores.

Comprobando una vez más el rol del usuario logueado en la variable de session “user”, realizamos o no la consulta dependiendo del rol del usuario que está solicitando el recurso, el cual está sólo disponible para un usuario con permisos de administración.

En ese caso, se realiza la consulta mediante el método “getAllEmails”, el cual está definido en el archivo “data-base-manager.js”, el cual hará de intermediario entre el servidor y la base de datos.

```
453 exports.getAllEmails = function(callback){
454     mails.find({ $query: {}, $orderby: {fecha:1}}).toArray(
455         function(e, res) {
456             if (e){
457                 callback(e);
458             }else{
459                 callback(null, res);
460             }
461         });
462 }
463
```

***Ilustración 28:*** *Tecnologías - Consulta de todos los emails con driver para* ***Node.js*** *y* ***MongoDB***

Aquí se puede ver cómo se consultas todos los emails ordenados por fecha que se encuentran en la colección “mails”.

Ésta es una buena ocasión para explicar lo que es el “callback”. Como ya se ha comentado, **Node.js** está orientado a eventos. Por tanto, si hacemos un método en el que el código es secuencial y se realiza una consulta, en caso de demorarse, las variables que dependan de esa consulta serán “undefined”. Esto se debe a que el resto del código sigue ejecutándose, ya que es la filosofía de **Node.js**.

Esto se soluciona con el “callback”, el cual se encarga de enviar el resultado de la consulta y de actualizar el valor de una variable en caso de que sea almacenado en una.

Para que se entienda mejor, es algo similar a la inyección de independencia, sólo que a un nivel mucho más bajo de abstracción.

Después de haber visto ejemplos de uso de **Node.js** vamos a comentar algunas ventajas y desventajas que han de ser tenidas en cuenta a la hora de desarrollar una aplicación web.

**Node.js**

| Ventajas | Desventajas |
|---|---|
| Es un lenguaje **JavaScript** | La documentación oficial no está clara |
| Con pocas líneas se puede crear un servidor **HTTP** | Por sí sólo no es seguro. Necesita de librerías |
| Es capaz de manipular datos **JSON** sin conversiones intermedias | No existe un IDE específico |
| Existe una comunidad muy activa | Tiene las mismas limitaciones que **JavaScript** |
| Es fácil conectarse a una base de datos | No tiene frameworks que simplifiquen las implementaciones |
| Las librerías se instalan con un solo comando en el terminal | No hay un patrón común en la estructura de los proyectos |

***Tabla 57:*** *Tecnologías - Ventajas y desventajas de* ***Node.js***

**Node.js** es un lenguaje **JavaScript** para servidor, por tanto podríamos desarrollar una aplicación sólo con **JavaScript** o casi al completo, como sucede con la que yo he desarrollado.

Poner en funcionamiento un proyecto **Node.js** es muy sencillo y rápido. Se puede realizar un "¡Hola, mundo!" en pocos minutos.

Otro punto fuerte de **Node.js** es que es capaz de trabajar con archivos **JSON** directamente sin necesidad de realizar conversiones para tratar los datos.

También es reseñable la comunidad tan activa acerca de **Node.js**. En páginas como **GitHub** podemos encontrar una gran cantidad de proyectos muy bien documentados y muy útiles para aprender.

A la hora de conectar el servidor con una base de datos como **MongoDB** también tenemos muchas facilidades, ya que es una tarea muy sencilla y se puede realizar de muchas formas. En el proyecto que he usado como base, la gestión de la conexión a la base de datos se realiza mediante el driver de **MongoDB**, pero también se podría haber realizado con el framework **Mongoose**.

No podemos olvidar la forma de instalar las librerías. Todas y cada una de las librerías se instalan mediante **NPM** (Node Package Manager), el gestor oficial de paquetes de **Noje.js**. Basta con escribir en el terminal "npm install" seguido del nombre exacto de la librería a instalar, y ésta quedará instalada en la carpeta "node_modules" de nuestro proyecto.

Sin embargo, también existen una serie de desventajas. La primera es que la documentación oficial de **Node.js** no está clasificada de una forma muy clara. No está pensada para que la lea un principiante en **Node.js**.

Otra desventaja es que **Node.js**, por sí sólo, no es seguro. Debido a que la implementación del servidor es totalmente manual, debemos cuidar cada uno de los detalles. Otra opción es usar alguna de las diversas librería que hay para proteger nuestras aplicaciones en **Node.js**. Sin embargo, es difícil conseguir proteger al máximo nuestra aplicación si no tenemos conocimientos previos.

También hay que mencionar que no existe un **IDE** específico para **Node.js**. En mi caso he utilizado el **IDE WebStorm**, el cual sirve para aplicaciones desarrolladas con **HTML** y **JavaScript** y, desde hace poco, cuenta con un intérprete estupendo para **AngularJS**. Sin embargo, no está pensado para **Node.js** ya que, a pesar de ser un lenguaje **JavaScript**, tiene muchas características y funcionalidades propias de **Node.js**.

Podemos deducir que si **Node.js** es un lenguaje **JavaScript** para servidor, tendrá prácticamente las mismas limitaciones que **JavaScript**. Por ejemplo, en **JavaScript** no existen clases como sí ocurre en **Java**.

Otro inconveniente muy destacable es que no existe ningún framework para **Node.js** que proporcione una plantilla ya implementada con las funciones básicas, como sí sucede con otros lenguajes como **Java**, que cuenta con el framework **Spring**.

Un último inconveniente es que no existe un patrón común en los proyectos **Node.js**. Si exploramos un poco los proyectos alojados en **GitHub**, veremos que en cada proyecto se clasifican los proyectos de una forma, los nombres de las carpetas cambian, etc. Por tanto, eso dificulta el aprendizaje de este lenguaje.

# 8.3. JADE


***Ilustración 29:*** *Tecnologías –* ***JADE***

*"**JADE** es un lenguaje de plantillas desarrollado por el creador de **Express** para simplificar la sintaxis de **HTML** y acelerar el proceso de desarrollo."*[9]

*"**JADE** está fuertemente inspirado en **Haml**, un lenguaje de marcado ligero que fue concebido para solucionar los problemas más comunes que se obtienen de la utilización de motores de plantillas tradicionales".*[10]

Este motor de plantillas está diseñado especialmente para **Node.js**, por lo que usarlo es una opción muy interesante.

## Comparación con HTML

Como mejor se puede explicar **JADE** es observando la siguiente imagen, en la que se ven dos plantillas: una implementada en **JADE** y otra en **HTML**. No son muchas las diferencias entre **JADE** y **HTML**, pero sí son significativas.

---

[9] http://codehero.co/node-y-express-jade-js/
[10] http://www.kabytes.com/programacion/sistema-de-plantillas-para-node-js/

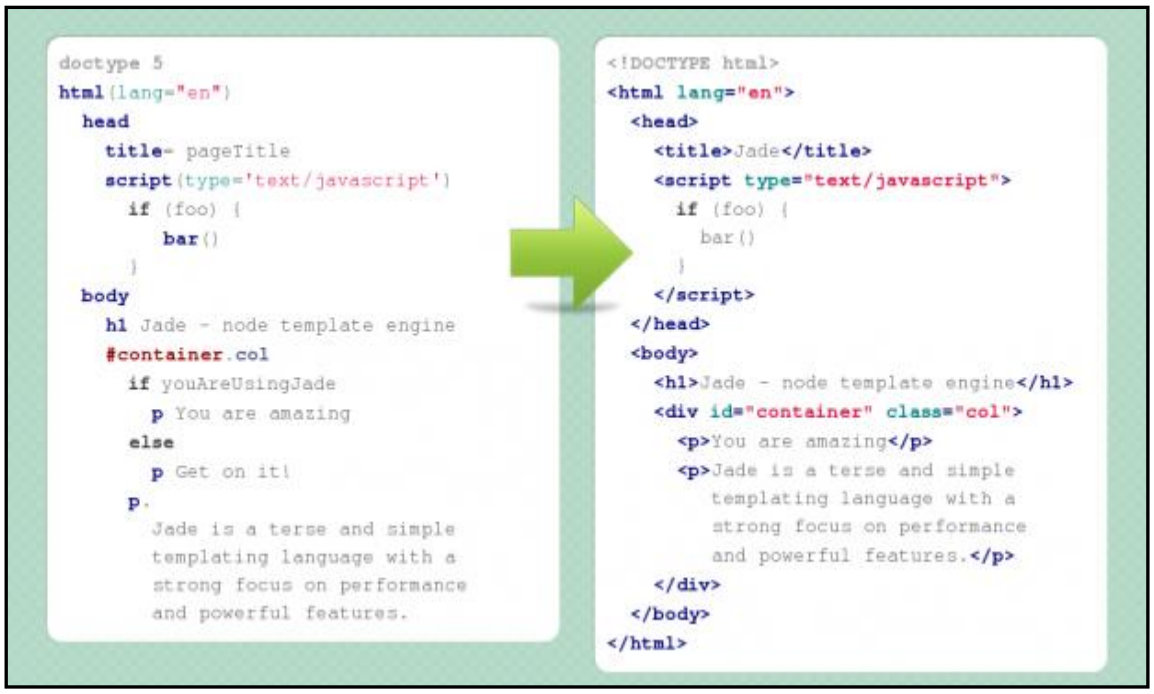



***Ilustración 30:*** *Tecnologías - Comparación entre* ***JADE*** *y* ***HTML***

En **JADE** se programa mediante indentación, tal y como sucede en **Python**, ya que las etiquetas se abren pero no se cierran; los atributos como los id, class, etc se indican pegados al nombre de la etiqueta con el selector css correspondiente (“**#**” para los ids, “.” para los class, etc), los values se indican con un espacio respecto a la etiqueta y el resto de atributos se indican entre paréntesis y separados por “,”.

La principal ventaja de **JADE** sobre **HTML** es que se pueden reutilizar plantillas. Por ejemplo, si queremos usar una barra de navegación en nuestra aplicación web, en lugar de escribir el código en todas las plantillas, podemos implementar una plantilla que contenga sólo la barra de navegación e importarla en cada plantilla.

Sin embargo, hay varias formas de reutilizar código y cada una tiene su matiz. Podemos usar la cláusula “extends” al inicio de una plantilla, de modo que el código de esa plantilla se insertará en el inicio de la plantilla que se está definiendo. El resultado es ampliar el código de la plantilla que estamos reutilizando.

Otra opción es usar la cláusula “include”, la cual se puede incluir en cualquier parte de la vista. Con esta cláusula no se amplía la plantilla anterior, sino que se inserta ese código donde ubiquemos la cláusula “include”.

Para definir un “layout” se suele utilizar la cláusula “extends” ya que las cabeceras en **HTML** siempre van al principio del documento, mientras que la

cláusula “include” se suele utilizar para añadir código común como puede ser una barra de navegación.

Esto no se puede realizar en **HTML** por sí sólo, por lo que al definir una barra de navegación en **HTML** implica copiar el código asociado en cada una de las vistas que se definan, contratiempo que podemos evitar perfectamente gracias **JADE**.

# 8.4. AngularJS

***Ilustración 31:*** *Tecnologías –* ***AngularJS***

*"**AngularJS** es un framework de JavaScript de código abierto mantenido por Google que ayuda con la gestión de lo que se conoce como aplicaciones de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC)."*[11]

Este framework tiene como uno de sus pilares a **jQuery**, por lo que todo lo que se puede hacer con **jQuery** también se puede hacer con **AngularJS**, con la diferencia de que todo se realiza con mayor nivel de abstracción, tal y como veremos más adelante.

**AngularJS**, en mi opinión, es difícil de aprender porque abarca muchos conceptos, no es complejo en sí mismo. Dicho esto vamos a explicar los conceptos fundamentales para tener una idea clara de lo que **AngularJS** supone.

Primero tenemos que hablar de los Scopes, los cuales se pueden definir como *"contextos de ejecución sobre los que trabajan las expresiones de **AngularJS**, por ejemplo, cuando referenciamos un atributo del modelo mediante la directiva "ng-model", no estamos sino apuntando a un atributo que contiene el scope sobre el que se está trabajando."*[12]

También tenemos los controladores, los cuales se encargan de manipular tanto la información provista por el servidor como la información de la vista y el modelo, esto último de forma muy sencilla mediante el Scope al que esté asociado. En la siguiente imagen podemos ver la cabecera de un controlador:

---

[11] http://es.wikipedia.org/wiki/AngularJS
[12] http://pablolazarodev.blogspot.com.es/

```
1   var app = angular.module('lagloria');
2
3   app.controller('IndexController', function ($scope) {
4
5
6   });
7
```

***Ilustración 32:*** *Tecnologías - Controlador de* ***AngularJS***

**AngularJS** tiene un sistema de plantillas muy distinto al de otros frameworks. Generalmente es el servidor el encargado de generar las plantillas con la información necesaria y de enviarlas al cliente. En este caso, el servidor sólo se encarga de enviar las plantillas y los datos (de forma separada como hemos visto antes), de modo que es **AngularJS** quien se encarga de introducir los datos en la plantilla y generando la vista.

La forma que tiene **AngularJS** de inyectar los datos en la plantilla es mediante los data bindings haciendo uso de directivas. Las directivas sirven mayormente para unir algún elemento **HTML** con alguna variable del Scope. Los data bindings sirven para acceder a dicha variable o a alguna propiedad de la variable, ya que puede ser un String, un array, un objeto o una etiqueta **HTML** completa, entre otras muchas posibilidades.

En **AngularJS**, los data bindings no sólo se pueden usar para acceder a variables definidas en el Scope, sino que también se pueden usar para evaluar expresiones.

En los sistemas de plantillas clásicos, una vez se tenía la plantilla y los datos, estos se mezclaban una única vez y se generaba la vista. Esto no sucede con **AngularJS**.

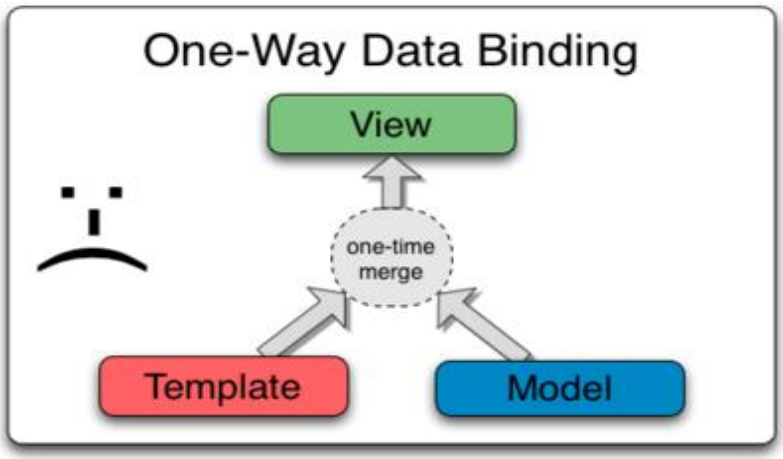



***Ilustración 33:*** *Tecnologías - One-Way Data Binding*

En **AngularJS** los data bindings tienen un comportamiento bidireccional. Esto significa que un cambio en una variable en el modelo implicará el mismo cambio en el elemento de la vista que esté asociado a esa variable mediante la directiva “ng-model”. Y lo mismo sucede a la inversa, lo que supone una instantánea sincronización que facilita muchísimo la tarea del programador. A esto se le conoce como “Two-Way Data Binding”.

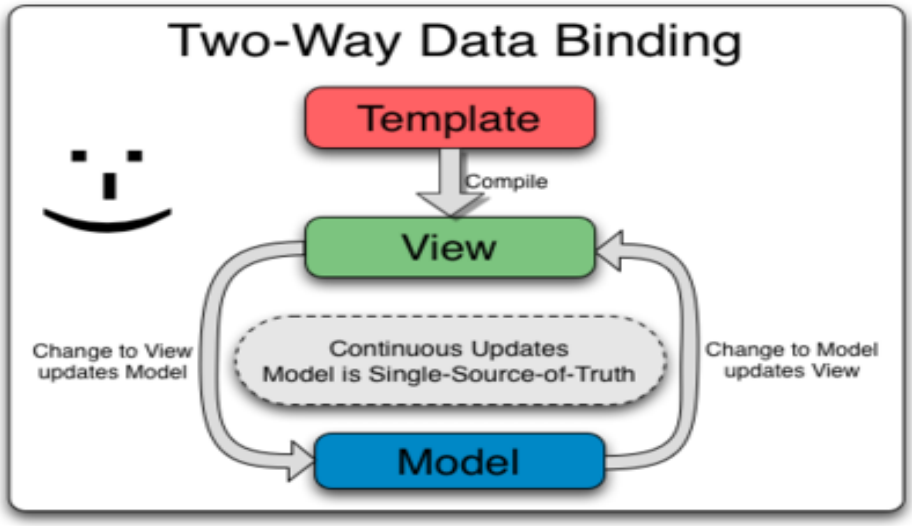



***Ilustración 34:*** *Tecnologías – **AngularJS** Two-Way Data Binding*

Otra característica muy interesante son los filters, los cuales permiten dar forma a los datos que vamos a representar, por ejemplo una fecha. Se usan directamente dentro de los data bindings y se comienzan con el carácter “|”. Un ejemplo podría ser el siguiente:

```
31
32              div.col-md-12
33                  div(ng-bind-template="{{email.fecha | date:('dd/MM/yyyy')}}")
34
```

***Ilustración 35:*** *Tecnologías - Filtro para data-binding de **AngularJS***

Un concepto muy importante de **AngularJS** que tampoco podemos olvidar es el de los servicios. **AngularJS** provee a los desarrolladores de una serie de servicios, entre los que destacan: $scope, $http, $window o $resources, entre otros muchos.

El servicio $http se utiliza para realizar peticiones **AJAX**, otra característica de **AngularJS** heredada de **jQuery**, la cual veremos más adelante.

Todos los servicios suministrados por **AngularJS** empiezan por el carácter “$”. Sin embargo, nosotros podemos crear nuestros propios servicios para definir métodos comunes que vayan a ser utilizados desde distintos controladores. De esta forma, no sólo se reutiliza bastante código sino que además se realiza una única implementación y diversas llamadas a ese servicio. Esto supone que, en caso de que cambie la implementación del método, los controladores que utilizan el servicio en cuestión no tienen por qué verse afectados necesariamente.

En la siguiente imagen vemos un controlador en el que se hace uso de los servicios $scope y $http para realizar una petición **AJAX** al servidor:

```
1  var app = angular.module('lagloria');
2
3  app.controller('ToffeesController', function($scope, $http) {
4      $scope.toffees = {};
5
6      $http.get('/api/toffees')
7          .success(function(data) {
8              if(data.message){
9                  $scope.toffees = {};
10             }else{
11                 $scope.toffees = data;
12             }
13         })
14         .error(function(data) {
15             alert(data);
16         });
17
18 });
19
```

***Ilustración 36:*** *Tecnologías - Servicios $scope y $http en un controlador de* ***AngularJS***

En la siguiente imagen se puede ver cómo se definen dos servicios propios con una petición “get” cada uno:

```
3   services
4       .service('UserService', ['$http', function($http){
5
6           this.solicitarUsuario = function(){
7               return $http.get('/api/user');
8           };
9
10      }])
11      .service('LoginService', ['$http', function($http){
12
13          this.hacerLogin = function(form){
14              return $http.post('/api/login', form);
15          };
16
17      }]);
18
19
```

***Ilustración 37:*** *Tecnologías - Definición de servicios propios en* ***AngularJS***

Como se ha dicho anteriormente, con **AngularJS** podemos realizar prácticamente las mismas cosas, con la salvedad de que el nivel de abstracción es mucho mayor. Por ello, vamos a comparar ambas tecnologías para, así, concluir qué ventajas nos aporta **AngularJS** frente a una potente librería **JavaScript** como **jQuery**.

## Comparación entre AngularJS y jQuery

Las principales ventajas de **AngularJS** son: la inyección de dependencia entre sus componentes (bastante similar al funcionamiento de **Spring**), la posibilidad de definir los controladores o servicios como módulos distintos o el alto nivel de abstracción frente a otros frameworks **JavaScript** basados en el modelo **MVC**.

Por tanto, para la aplicación que queremos desarrollar, considero **AngularJS** como la mejor opción teniendo en cuenta todas sus ventajas.

Todas las tecnologías que he seleccionado son muy recientes, de grandes prestaciones y se están haciendo un hueco en el mundo del software.

**AngularJS**

| Ventajas | Inconvenientes |
|---|---|
| Basado en el MVC + Inyección de dependencia | La documentación no está pensada para principiantes |
| Conexión entre controlador y plantillas mediante directivas | Usado junto con **jQuery** puede provocar conflictos |
| Data-binding y evaluación de expresiones en plantillas **HTML** | Es difícil seguir la traza de una aplicación |
| Mayor nivel de abstracción | Hay muchas diferencias entre versiones |
| Ofrece funcionalidades que permiten diseñar una web estéticamente impecable | Angular incorpora una versión de **Bootstrap** internamente |

***Tabla 58:*** *Tecnologías - Ventajas y desventajas de* ***AngularJS***

Una de las principales ventajas de **AngularJS** es que está basado en el patrón **MVC** (Modelo-Vista-Controlador), de modo que las vistas son totalmente independientes de la aplicación y son utilizadas por el controlador asociado en el momento de mostrar dicha vista. Esto se consigue mediante inyección de dependencia, de forma muy similar a la utilizada en el framework para **Java, Spring**.

Otra ventaja es la conexión entre las vistas y los controladores, lo cual se hace mediante directivas. Éstas están muy bien documentadas en la página web oficial de **AngularJS** y tienen una sintaxis muy parecida a **JavaScript** y **Java**.

La siguiente ventaja va muy unida a la anterior. En las plantillas **HTML** podemos evaluar expresiones o mostrar datos después de iterar sobre una colección. Esto se puede hacer gracias a los data bindings, tal y como ya hemos explicado anteriormente.

**AngularJS** posee un nivel de abstracción muy alto, ya que realiza tareas de muy bajo nivel de las que el programador no es plenamente consciente a la hora de desarrollar, por ejemplo: las peticiones **AJAX** al servidor.

Como última ventaja podemos destacar la estética que se puede llegar a conseguir en nuestra aplicación. Un ejemplo de ello es la directiva ng-bind-template.

Dicho todo esto, pasemos con los inconvenientes. La documentación de **AngularJS** está muy bien detallada. Sin embargo, es fácil de leer cuando ya se tiene cierta experiencia. Mi forma de aprender **AngularJS** no fue leyendo la documentación, ya que hace 3 meses no entendía nada. Yo conseguí aprender explorando una gran cantidad de proyectos en **GitHub**, al igual que hice con **Node.js**.

Otro inconveniente es que usar **AngularJS** y **jQuery** simultáneamente puede provocar conflictos, ya que AngularJS usa una versión de **jQuery** internamente. Además de esto, los posibles fallos se deben a que en ciertas ocasiones los cambios que hagamos en el **DOM** mediante **jQuery** no son detectados por **AngularJS**. Esto se puede solucionar usando el método **$apply** del **$scope** que tengamos definido en el controlador, sin embargo hay que saber cómo funciona dicha función para evitar estos conflictos.

También hay que mencionar que no es fácil seguir la traza de la aplicación, ya que determinados fallos no son notificados en la consola de los navegadores. Por tanto, si cometemos algún fallo como no importar una librería o escribir incorrectamente el nombre de una variable podemos perder muchas horas hasta encontrar el fallo.

El siguiente inconveniente también es de suma importancia, y es que las versiones de **AngularJS** distan mucho entre sí, ya que cambian nombres de métodos, directivas, implementaciones, etc. Por tanto, hay que tener cuidado a la hora de elegir la versión de **AngularJS**, ya que puede ser muy determinante.

Lo último puede ser un inconveniente si no se es consciente de ello. **AngularJS** utiliza por defecto cierta versión de la librería **Bootstrap**, una librería para diseño y maquetación web muy famosa. El problema puede surgir si la versión que incorpora es antigua y queremos añadir una nueva. Hay que tener nuevamente mucho cuidado porque añadir nuevas funcionalidades de **Bootstrap** puede implicar que dejen de funcionar otras de la versión antigua.

A continuación vamos a realizar una comparativa similar para **jQuery**. Como **AngularJS** usa **jQuery** de fondo, las diferencias fundamentales van a radicar en el nivel de abstracción de una librería y otra.

**jQuery**

| Ventajas | Inconvenientes |
|---|---|
| Selección de elementos DOM | Alta frecuencia de publicación de nuevas versiones |
| Manipulación de hojas de estilo | Librería de mucho tamaño |
| Peticiones **AJAX** al servidor | Ciertas funcionalidades son de muy bajo nivel |
| Efectos y animaciones | |

***Tabla 59:*** *Tecnologías - Ventajas y desventajas de* ***jQuery***

Empezando por las ventajas, podemos decir que **jQuery** permite la selección de elementos DOM mediante id, class o name de las etiquetas.

También se pueden manipular las hojas de estilo de forma dinámica según el comportamiento de nuestra web. Yo no sabía que se podía realizar esto hasta que examiné el proyecto base que he usado para mi aplicación en la que se volvían a añadir ciertas líneas en el css aunque las hubiera comentado previamente.

Una de las características más importantes es la capacidad de realización de peticiones **AJAX** a un servidor. Estas mismas peticiones se realizan con **AngularJS**, sólo que a más alto nivel.

Y la característica que sea quizás más importe es el hecho de poder realizar efectos y animaciones. Hoy en día se pueden hacer auténticas florituras en una página web.

En cuanto a inconvenientes son pocos los que tiene **jQuery**. Uno de ellos es que se publican muchas versiones, algo similar a lo que sucede con **AngularJS**, y eso puede causar muchos despistes y conflictos.

Un segundo inconveniente es el tamaño de la librería, ya que ocupa alrededor de unos 85 KB de carga. Por tanto, en ordenadores antiguos o con conexión a Internet lenta, es posible que las características asociadas a **jQuery** no funcionen del todo bien.

Y un último inconveniente según mi criterio, ya que para otra persona podría ser una ventaja, es que hay características que se usan a un nivel muy bajo. Por ejemplo, las peticiones **AJAX** hay que realizarlas con todo detalle,

mientras que en **AngularJS** se pueden realizar con una única línea gracias al servicio $http.

# 8.5. Bootstrap, Eathub theme y Fontawesome


***Ilustración 38:*** *Tecnologías -* ***Bootstrap*** *y* ***Fontawesome***

La base del diseño de la web será, por una parte, la famosa librería de CSS **Bootstrap**, la cual es desarrollada y utilizada en **Twitter**, mientras que por otra parte se utilizará el 'starter-template' de **Eathub**, una aplicación web que ha sido publicada recientemente en la que tengo el orgullo de ser desarrollador.

***Tabla 60:*** *Tecnologías -* ***Eathub*** *theme*

Para los cambios necesarios se desarrollará una plantilla **CSS** específica para **La Gloria S.L.**, la cual sobrescribirá cuantas propiedades sean necesarias.

Como complemento a todas librerías, se hará uso de **Fontawesome**, una librería **CSS** muy famosa por sus iconos, los cuales están basados en caracteres.

Dejando al margen los fantásticos diseños que se pueden obtener usando **Bootstrap**, la principal característica es lo que se denomina 'responsive design', lo cual es muy útil para visualizar correctamente una web en cualquier dispositivo desde el que se está visualizando.

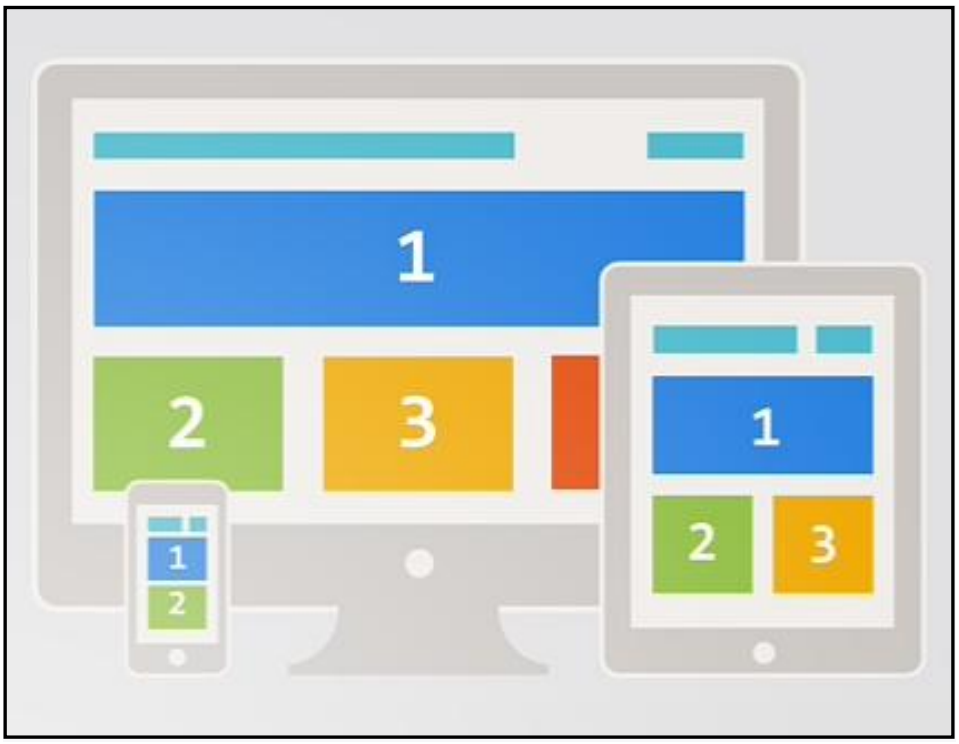



***Ilustración 39:*** *Tecnologías - Responsive design*

Gracias a **Bootstrap** solucionaremos dos problemas de una sola vez: por un lado podremos hacer un diseño estupendo para visualizar la web desde un ordenador, mientras que por otro lado, la misma vista que se ha diseñado, se mostrará en otros dispositivos, cambiando algunos elementos de tamaño y/o ubicación.

Esto se consigue gracias a las clases **CSS** llamadas 'col-' definidas por **Bootstrap**. El formato que siguen estas clases es el siguiente: 'col-xx-nn'. El fragmento 'xx' se corresponde con el ancho de la pantalla del dispositivo en el que se visualiza la web, cuyos valores pueden ser: 'lg' ( > de 1199 px), 'md' (> de 991 px), 'sm' (> 767 px) y 'xs' (el resto de dispositivos).

Todos los tamaños son abreviaturas en inglés. Tal y como se puede deducir significan ‘Large devices’, ‘Medium devices’, ‘Small devices’ y ‘Extra small devices’, respectivamente.

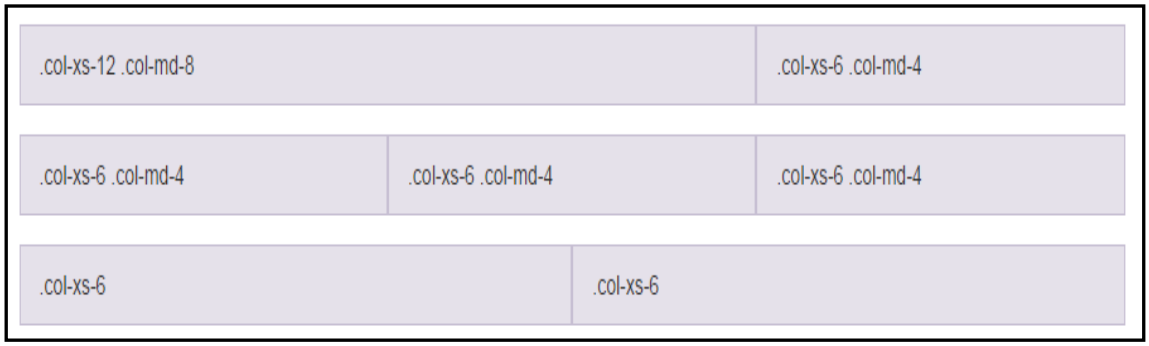



***Ilustración 40:*** *Tecnologías - Clases css col de* ***Bootstrap*** *en PC*

Aquí se puede ver un ejemplo de la aplicación de las clases ‘col’. El código para que las columnas se vean de ese modo es el siguiente:


***Ilustración 41:*** *Tecnologías - Código de las clases col de* ***Bootstrap***

A pesar de que no se han definido todos los ‘col’ de todos los tamaños, **Bootstrap** usa el ‘col’ añadido para cualquier dispositivo. Por ejemplo, en la última fila, sólo se han definido los ‘col’ para dispositivos móviles.

Sin embargo, cuando vemos esas columnas en un dispositivo de mayores dimensiones, el ancho es proporcional. Por tanto, si sólo se define un col-xs-6, al visualiza la columna en un PC se interpretará como un ‘col-md-6’.

Esto mismo sucede para todas las relaciones posibles. No obstante, si se hace al revés (definir un ‘col-md-6’ y visualizarlo desde un móvil), en lugar de mantener las proporciones se expande ocupando todo el ancho de la pantalla.

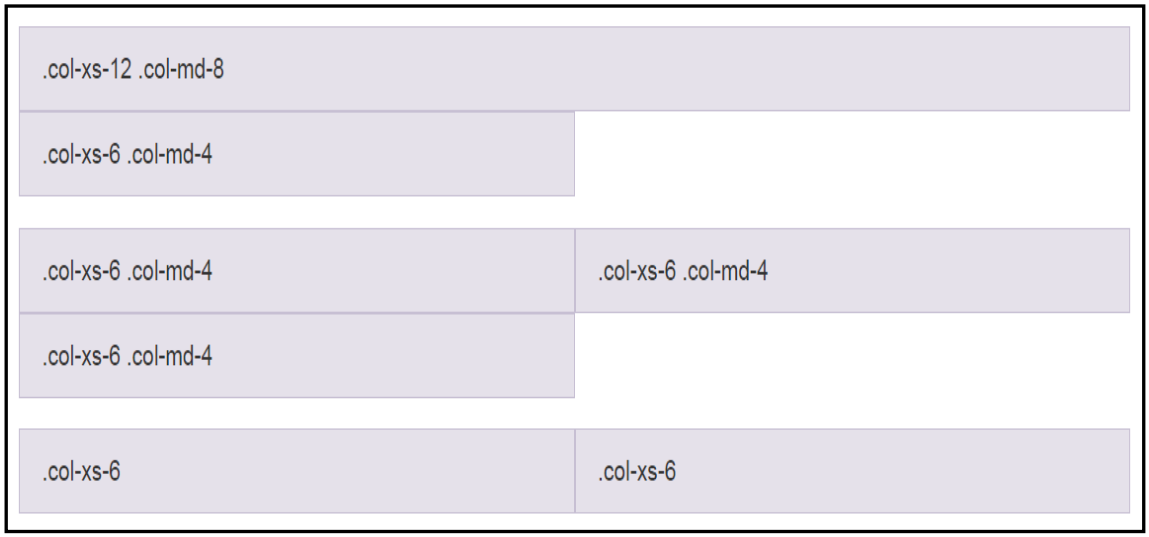



***Ilustración 42:*** *Tecnologías - Clases css col de* ***Bootstrap*** *en tablet*

Esta imagen es una prueba de lo último que he explicado. La imagen se corresponde con la vista de las columnas en una tablet (‘col-sm-...’).

Si nos fijamos en la primera columna, no tiene ningún ‘col’ específico para dispositivos de ese tamaño. Como el menor es ‘col-xs-12’, éste se traducirá en ‘col-sm-12’. Sin embargo, si sólo estuviera definido el ‘col-md-8’entonces la columna se expandiría ocupando todo el ancho del div que lo contiene.

En cuanto a las animaciones de Bootstrap podemos destacar el carousel y los modals. Son interacciones muy simples pero que resultan llamativas para los usuarios.


***Ilustración 43:*** *Tecnologías - Carousel de* ***Bootstrap***

El carousel de **Bootstrap** es un slider en el que se pueden visualizar imágenes, las cuales cambian automáticamente mediante transiciones.

```
<div id="carousel-example-generic" class="carousel slide" data-ride="carousel">
  <!-- Indicators -->
  <ol class="carousel-indicators">
    <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
    <li data-target="#carousel-example-generic" data-slide-to="1"></li>
    <li data-target="#carousel-example-generic" data-slide-to="2"></li>
  </ol>

  <!-- Wrapper for slides -->
  <div class="carousel-inner">
    <div class="item active">
      <img src="..." alt="...">
      <div class="carousel-caption">
        ...
      </div>
    </div>
    <div class="item">
      <img src="..." alt="...">
      <div class="carousel-caption">
        ...
      </div>
    </div>
    ...
  </div>

  <!-- Controls -->
  <a class="left carousel-control" href="#carousel-example-generic" role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left"></span>
  </a>
  <a class="right carousel-control" href="#carousel-example-generic" role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right"></span>
  </a>
</div>
```

***Ilustración 44:*** *Tecnologías - Código del carousel de* ***Bootstrap***

Éste es el código para mostrar un carousel en una vista **HTML**. Como se puede intuir, a mayor número de páginas mayor código en el carousel.

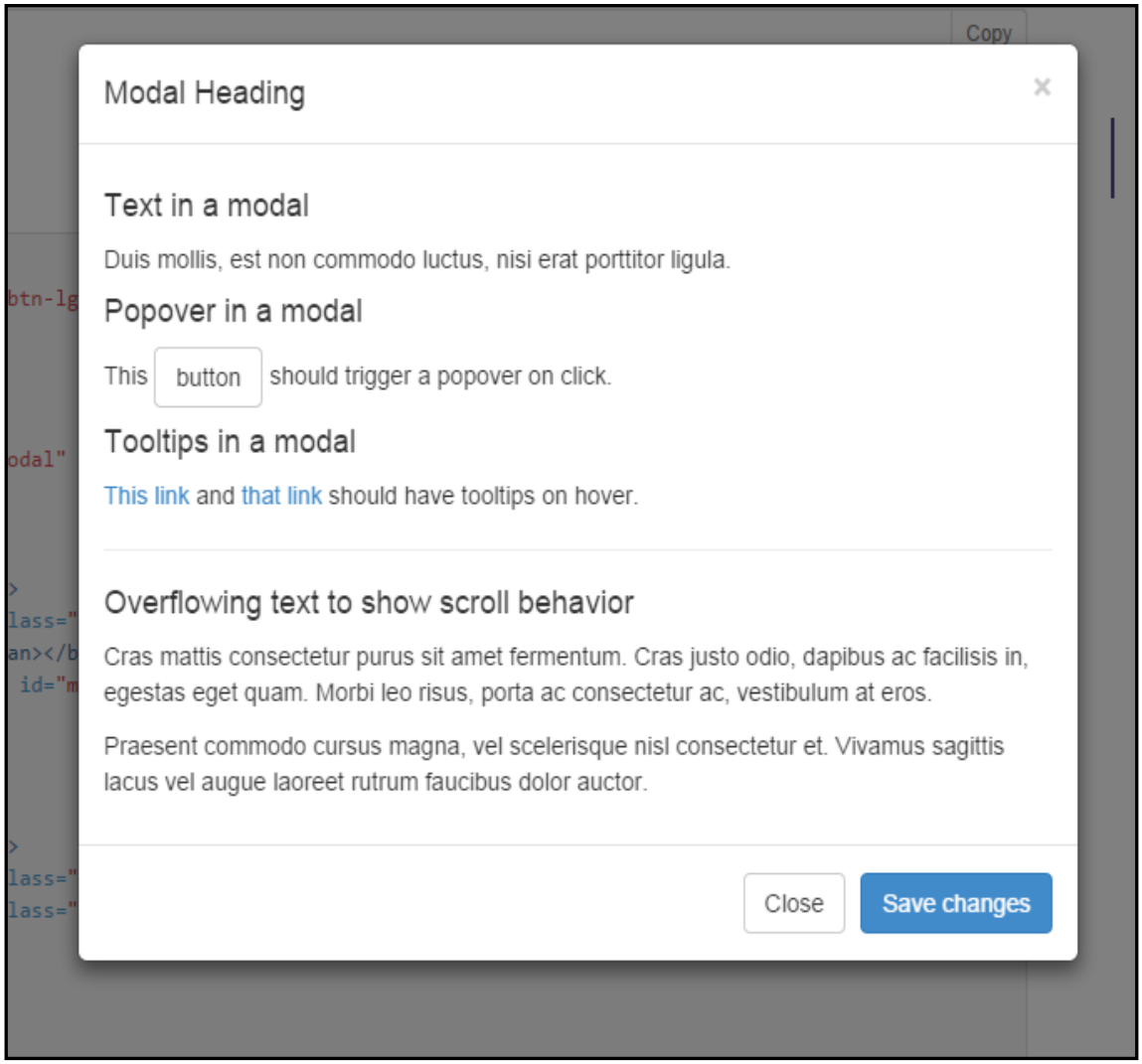



***Ilustración 45:*** *Tecnologías - Modal de* ***Bootstrap***

El cuanto al modal es una buena interacción que podemos usar de **Bootstrap** para llamar la atención del usuario, ya que el resto de la página se oscurece para resaltar el contenido del modal.

```
<!-- Button trigger modal -->
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myModalLabel" aria-
hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal"><span aria-hidden="true">&times;</span>
<span class="sr-only">Close</span></button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        ...
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

***Ilustración 46:*** *Tecnologías - Código del modal de* ***Bootstrap***

En cuanto a **Eathub**, es una aplicación web cuya interfaz de usuario está basada en **Bootstrap**, a pesar de que es una versión modificada según las necesidades de dicha aplicación.

Esto es sólo un pequeño resumen de las muchísimas cosas que ofrece Bootstrap. Sin embargo, se usarán otras herramientas para personalizar estas interacciones.

Una de ellas es el ‘starter-template’ de **EatHub**.


***Ilustración 47:*** *Tecnologías - Receta de* ***Eathub***

Como ya he comentado anteriormente, soy desarrollador en dicho proyecto y, por tanto, tengo bastantes conocimientos de las decisiones que han sido tomadas para realizar el diseño de dicha aplicación web.

Así pues, estimo oportuno seguir la misma filosofía para diseñar la página web para **La Gloria S.L.**: partir de **Bootstrap** y modificar las propiedades que necesite, además de las nuevas que defina. La única diferencia entre el diseño de los proyectos **Eathub** y **La Gloria** será que **La Gloria** hará uso de la hoja de estilos definida por **Eathub**.

En lo que a **Fontawesome** se refiere, a continuación presento una cantidad reducida de iconos, ya que sólo se muestran 33 iconos cuando podemos encontrar 439.

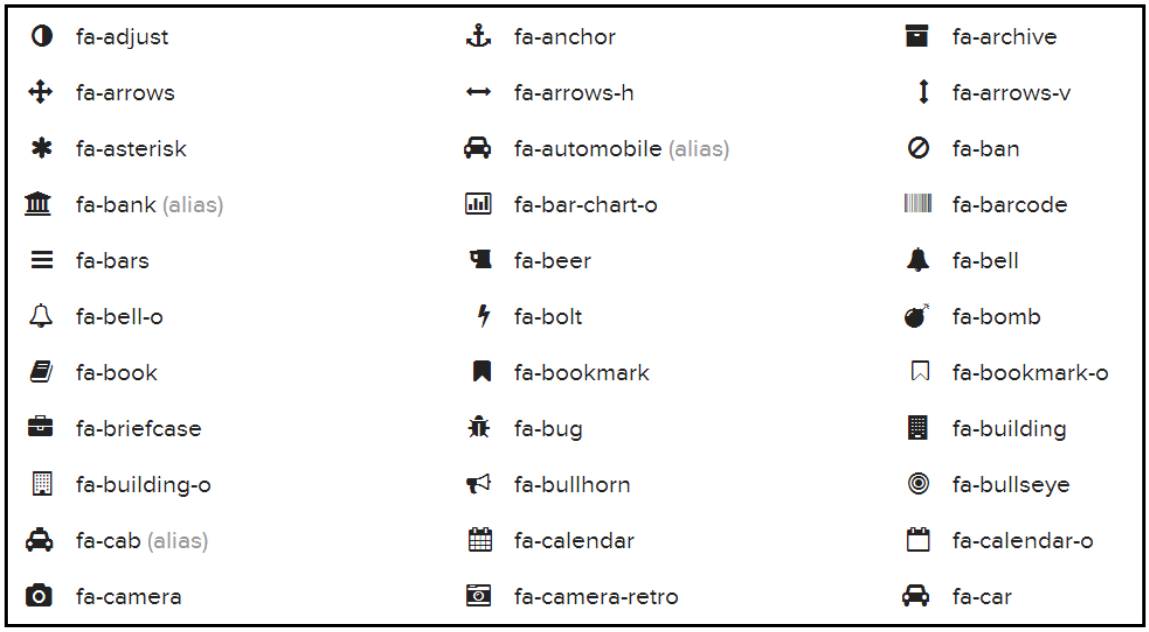



***Ilustración 48:*** *Tecnologías - Subconjunto de iconos de* ***Fontawesome***

# BLOQUE IV: DISEÑO

# 9. MODELO CONCEPTUAL

En este capítulo se presenta el modelo de clase realizado para entender el problema que hay que solucionar en este proyecto.

El modelo de conceptual no es más que una representación formal de la información que debe almacenar el sistema, así como las clases, atributos y relaciones entre ellos. Los diagramas de clases muy a menudo durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema (habitualmente mediante modelos **UML**), y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

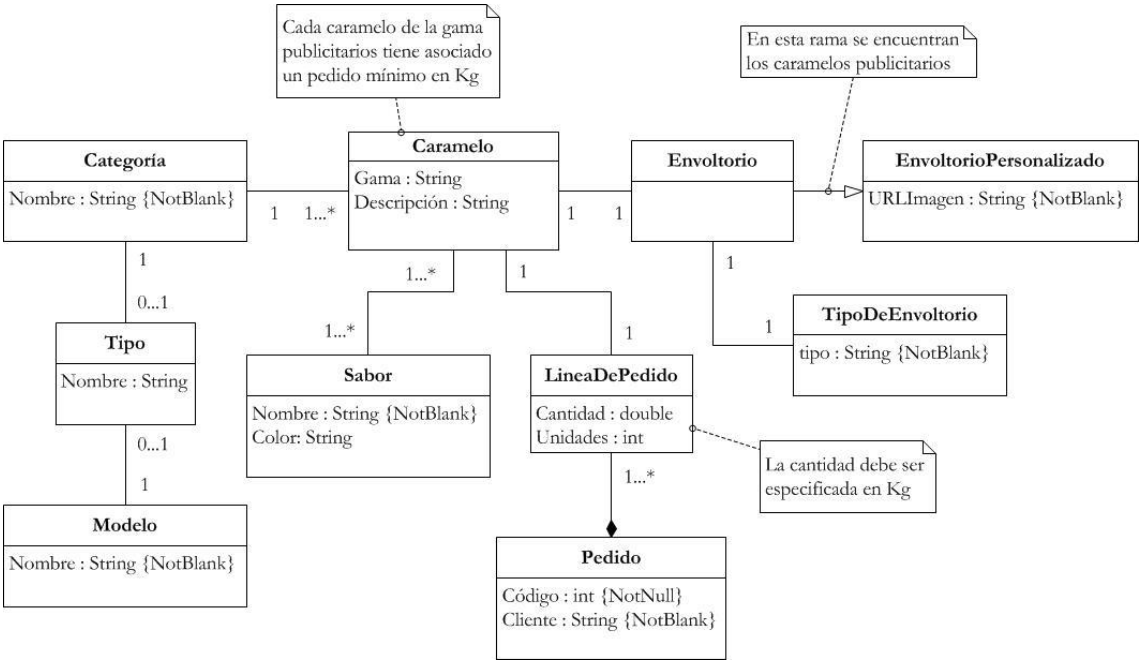



***Ilustración 49:*** *Modelo conceptual*

A continuación vamos a proceder a comentar por partes este modelo conceptual, para aclarar lo máximo posible el dominio del problema.

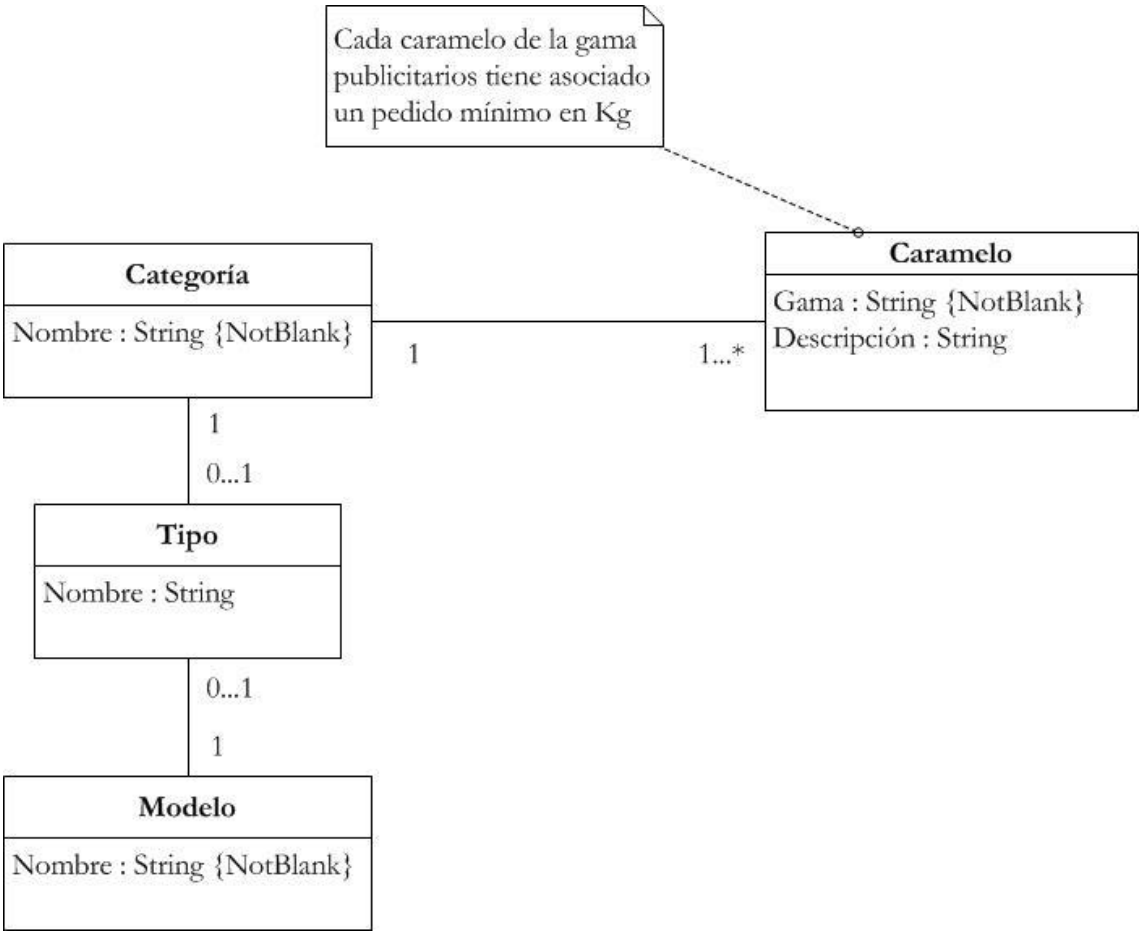



***Ilustración 50:*** *Modelo conceptual - Categoría, tipo y modelo de caramelo*

En primer lugar detallaremos el concepto de caramelo. Los caramelos pertenecerán a una gama, la cual puede ser 'Gama propia' o 'Publicitarios'. En el caso de que el caramelo sea de la gama 'Publicitarios' debe tener obligatoriamente un pedido mínimo expresado en Kg. También podrán tener una descripción, aunque no es obligatoria.

Los caramelos serán clasificados principalmente por categorías. Estas pueden ser: 'Toffees y Masticables', 'Duros', 'Grageados' y 'Con palo'.

Dentro de cada categoría puede haber más 'categorías'. Para evitar la ambigüedad, a estas subcategorías se las ha denominado 'Tipo'. Sin embargo, no es obligatorio que cada categoría tenga subcategorías, por lo que el 'Tipo' del caramelo es opcional.

Acabamos con el modelo, lo cual se corresponde con el nombre específico del caramelo. No identifica a cada caramelo dentro de todos los disponibles en el catálogo, pero sí suele identificar al caramelo dentro de la categoría.

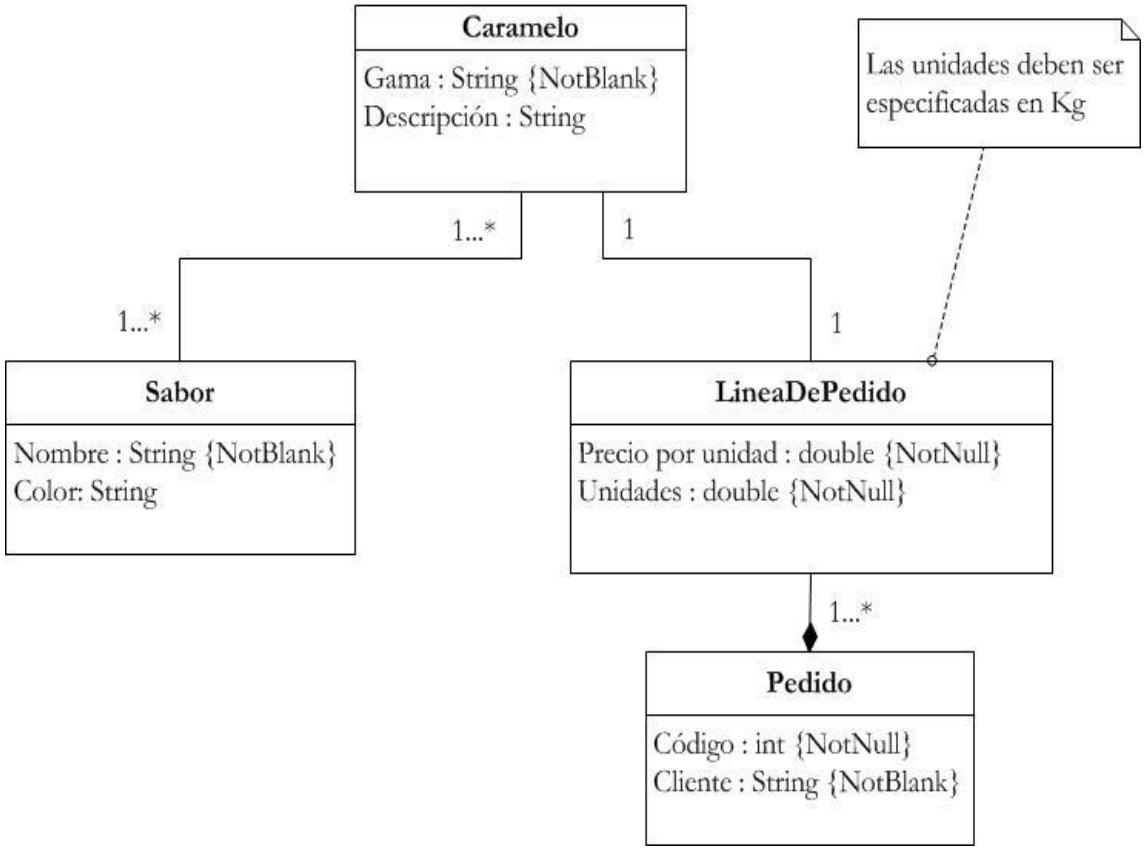



***Ilustración 51:*** *Modelo conceptual - Sabor y pedido de caramelos*

Cada caramelo tendrá asociado 1 ó más sabores. Dicho sabor o sabores tendrá un nombre y, de ser único, un color. Sin embargo son muchos los sabores con variedad de colores, por lo que no es una propiedad obligatoria.

Cada pedido tendrá asociado un código y un cliente, los cuales deben existir ya que son registrados en el sistema.

Además, cada pedido constará de una línea de pedido en la que sólo habrá un producto, teniendo una cantidad elegida (en Kg) y el número de unidades a comprar de dicho producto.

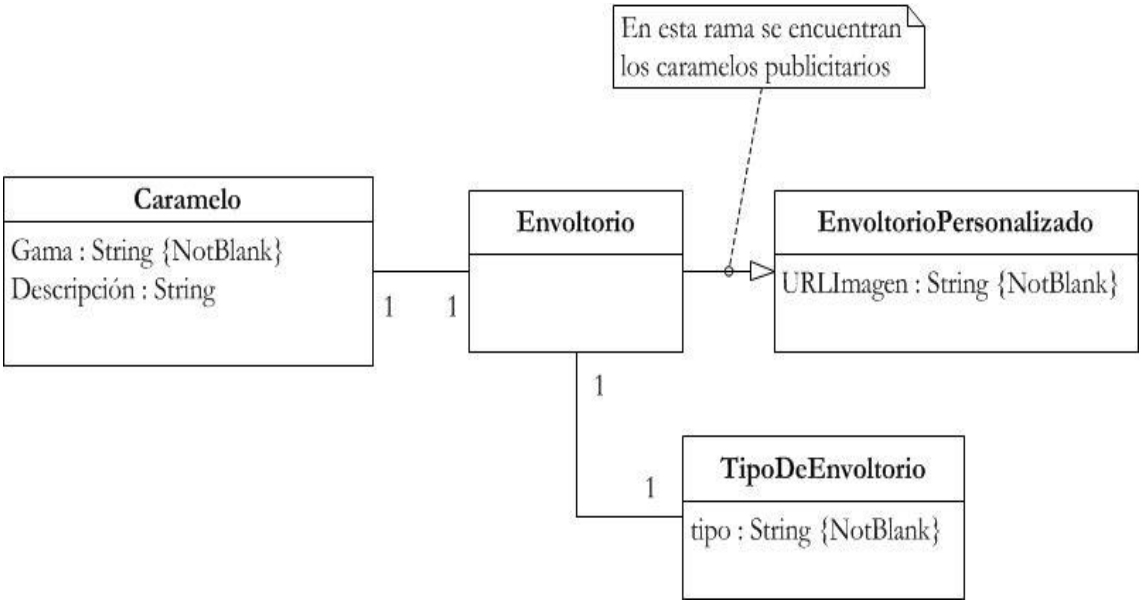



***Ilustración 52:*** *Modelo conceptual - Envoltorios de los caramelos*

Para finalizar tenemos los envoltorios. Todos los caramelos tienen un envoltorio, los cuales pueden ser de varios tipos: 'Flowpack', 'Dos lazos, 'Un lazo', etc. Sin embargo, los envoltorios de los caramelos dentro de la 'Gama propia' son fijados por la empresa, mientras que los caramelos de la 'Gama publicitarios' se pueden personalizar. Para ser más exactos, se pueden encargar caramelos con una imagen suministrada por el cliente.

En ese caso, el cliente debe indicar una URL donde se encuentre alojada dicha imagen. Por tanto, dicha URL debe ser pública.

No obstante, si fuera posible, se puede implementar alguna funcionalidad para añadir la foto online para más exactitud, aunque no es un requisito explícito del cliente.

# BLOQUE V: IMPLEMENTACIÓN

# 10. ARQUITECTURA

En este capítulo se habla sobre la arquitectura de la aplicación web y de todo lo íntimamente relacionado con las tecnologías.

Leer este capítulo a altamente recomendable para futuras extensiones o futuros desarrolladores de esta aplicación web. Aquí se explicará la metodología de trabajo usada en el proyecto y las herramientas usadas para el control del mismo, así como la interacción de las distintas tecnologías en la aplicación.

El capítulo consta de 4 secciones. En la primera se habla de la arquitectura tecnológica de la web en general, sobre la interacción de las tecnologías y cuál es el paradigma bajo el que se usan.

En la segunda se habla del entorno de gestión utilizado, **GitHub**. Esta herramienta tiene una perfecta integración con **Git** y es muy útil para alojar el código de nuestra aplicación en un repositorio.

En la tercera sección se explican los entornos de desarrollo utilizados: **WebStorm**, para implementación de código, y **Robomongo**, una aplicación con interfaz gráfica para manipular la información de una base de datos en **MongoDB**.

Para finalizar, en la última sección se explica cuál es la estructura del proyecto **Node.js** y por qué se ha estructurado así y no de otra forma.

# 10.1. Arquitectura tecnológica

Como ya se ha comentado, en primer lugar vamos a comentar de forma general las tecnologías usadas en la aplicación web, qué papel ocupan y cómo interactúan entre sí.

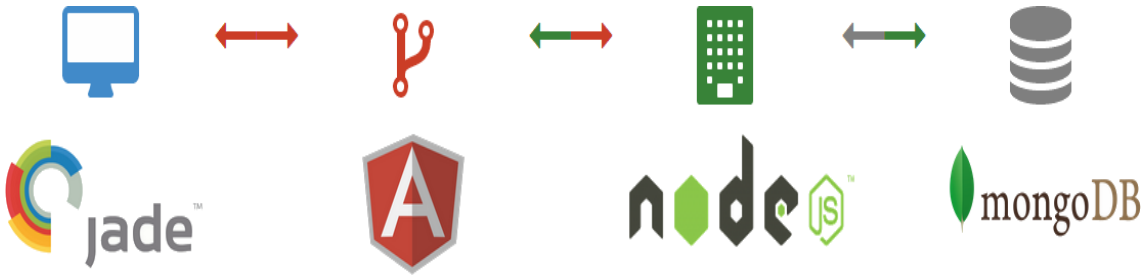



***Ilustración 53:*** *Arquitectura - Arquitectura tecnológica de la aplicación web*

Con esta imagen tan intuitiva podremos explicar muy fácilmente la arquitectura de la aplicación web. Empezaremos por la base de datos, pasando por el lado del servidor, dando lugar al framework **AngularJS** y acabando con las plantillas en el lado del cliente.

En nuestra base de datos **MongoDB** se encontrará toda la información necesaria. Dicha información será consultada desde el servidor, programado con **Node*.js***, con la ayuda del driver oficial para **MongoDB**. Dicha implementación ya venía realizada en el proyecto que he tomado como base ( el cual se puede consultar en el capítulo de la webgrafía). Sólo he tenido que modificarla y ampliarla según las necesidades de mi aplicación. Además, la implementación de dicho proyecto se hace de otra forma de la que yo conocía, por lo que he comprobado el gran número de alternativas que tenemos con **Node.js**.

A continuación viene, en mi opinión, una de las partes más importantes de este proyecto, si no la más importante. **AngularJS**, un framework de **JavaScript** de código abierto, hace uso del **MVC** (Modelo Vista Controlador) y permite la inyección de dependencias al igual que **Spring**, un framework para **Java** que estudiamos los alumnos de mi titulación en el tercer curso en la asignatura **DP** (Diseño y Pruebas). De esta forma, **AngularJS** proporciona a las plantillas **HTML** la información necesaria en el momento de mostrarla, de modo que **AngularJS** hace de intermediario constantemente entre el navegador y el servidor con un gran dinamismo.

Para finalizar tenemos las vistas, las cuales están diseñadas mediante plantillas en lenguaje **JADE**, un motor de plantillas para **HTML** que permite reducir el número de líneas de código, facilitando su implementación, y reutilizar las plantillas, entre otras cosas.

# Consultas desde Node.js a MongoDB

Las consultas se realizan gracias al driver de **Node.js** para **MongoDB**. Dichas consultas están definidas en un archivo del proyecto llamado “data-base-manager.js”. En él se definen todas y cada una de ellas, las cuales son llamadas desde el archivo “router.js”, donde están todas las rutas de la aplicación.

# Interacción entre AngularJS y Node.js

Como ya se ha dicho en reiteradas ocasiones, en la aplicación web desarrollada se hace uso de la filosofía REST, la cual está basada en los métodos **CRUD** del estándar que define el protocolo **HTTP**.

Dicho esto, vamos a explicar cómo suministra **Node.js** a **AngularJS** la información que ha consultado previamente.

Para cualquier vista que solicite el cliente, **Node.js** siempre la devuelve sin necesidad de que **AngularJS** actúe.

A continuación se muestra un ejemplo en el que se solicita una vista estática, en este caso la vista de los emails en la que un administrador lee los emails enviados por los usuarios.

```
163
164     app.get('/emails', function(req, res){
165         if(req.session.user == null){
166             res.render('error',{
167                 message : 'No puede acceder a los emails enviados a La Gloria S.L. porque no' +
168                     ' tiene permisos de administración.'
169             });
170         }else{
171             if(req.session.user.role == 'admin'){
172                 res.render('admin/emails', 200);
173             }else{
174                 res.render('error',{
175                     message : 'No puede acceder a los emails enviados a La Gloria S.L. porque ' +
176                         'no tiene permisos de administración.'
177                 });
178             }
179         }
180     });
```

***Ilustración 54:*** *Arquitectura - Petición de la vista de los emails*

En esta imagen podemos ver que, en caso de que el solicitante esté logueado y sea administrador, se envía al cliente la plantilla con para visualizar los emails.

En el siguiente paso es donde **AngularJS** entra en juego. Una vez que se muestra la plantilla al completo en el navegador del cliente, **AngularJS** se encarga de hacer una petición **AJAX** con el método **GET** a la **API REST**, de modo que solicita todos los emails registrados en el sistema.

```
3   app.controller('EmailsController', function ($scope, $http){
4
5       $scope.emails = {};
6       $scope.emailAEliminar = {};
7
8
9
10      $http.get('/api/emails')
11          .success(function(data){
12              $scope.emails = data;
13          })
14          .error(function(data){
15              alert("Ha sucedido algún error. Recargue la página.");
16          });
17
```

***Ilustración 55:*** *Arquitectura - Petición **AJAX** con **AngularJS***

Dicha petición se realiza desde el controlador asociado a la vista de los emails. Si la petición es exitosa (devuelve un código 200) se guardan los emails en la variable '$scope.emails'. De lo contrario (si devuelve un código 400), se muestra un mensaje de error.

Nuevamente, en el servidor se comprueban las credenciales y permisos del usuario. En caso de satisfacerse, se realiza la consulta mediante el **DBM** (Data Base Manager).

```
664     app.get('/api/emails', function(req, res){
665         if(req.session.user == null){
666             res.send('unauthorized',400);
667         }else{
668             if(req.session.user.role == "admin"){
669                 DBM.getAllEmails(function(err, mails){
670                     if(err){
671                         console.log(err);
672                     }else{
673                         res.send(mails, 200);
674                     }
675                 });
676             }else{
677                 res.send('unauthorized', 400);
678             }
679         }
680     });
681
```

***Ilustración 56:*** *Arquitectura - Respuesta con los emails registrados en el sistema*

Si la consulta es exitosa, se enviará al cliente y será tratado por **AngularJS** como acabamos de comentar.

# Información suministrada a JADE por AngularJS

Ya sólo nos queda la última interacción, la que existe entre **JADE** y **AngularJS**.

**JADE** es simplemente un motor de plantillas **HTML** que facilita la implementación de las vistas. Por tanto, sigue teniendo el carácter estático de **HTML**. Es **AngularJS** quien se encarga de todo el dinamismo de la web.

Esto es posible gracias a las directivas de **AngularJS**, las cuales se usan a modo de ancla en las plantillas, esto último con el fin de definir los distintos contextos en los que trabajará este framework.

```
div(ng-app="dashboard")
    div
        include ../adminPartials/navbar
    div.container-fluid
        div.row
            include ../adminPartials/sidebar
            div.col-sm-9.col-sm-offset-3.col-md-10.col-md-offset-2.main
                h1.page-header Correos electrónicos
                div.col-md-12.col-sm-12(ng-controller="EmailsController")
                    table.table
                        thead
                            tr
                                th Eliminar
                                th Remitente
                                th Fecha
                                th Mensaje
                        tbody(ng-repeat="email in emails")
                            tr(ng-show="email.leido == false").verEmail
                                td(style="font-weight: bold")
                                    span(ng-click="eliminarEmail(email._id)")
                                        i.fa.fa-trash-o
                                td(style="font-weight: bold", ng-click="verEmail(email._id)")
                                td(style="font-weight: bold", ng-click="verEmail(email._id)")
                                td(style="font-weight: bold; text-overflow: hidden !important"
                            tr(ng-show="email.leido == true").verEmail
```

***Ilustración 57:*** *Arquitectura - Plantilla **JADE** de la vista de los emails*

Aquí se puede observar un fragmento de la vista “emails”, en la que se muestran todos los emails registrados en el sistema.

En primer lugar vemos la directiva ‘ng-app’, mediante la cual definimos el módulo sobre el que se expanden los distintos componentes de **AngularJS**, a saber: controladores, servicios, etc. Existen 2 ng-app en el proyecto: ‘lagloria’ y ‘dashboard’. El primero está asociado a todas las vistas para usuarios anónimos y proveedores, mientras que la segunda está asociada a las vistas relacionadas con el administrador.

Luego podemos observar la directiva ‘ng-controller’, donde se indica el controlador que gestionará la información contenida en ese div definiendo, así, un contexto. Dicho controlador es el que se ha mostrado anteriormente.

Para finalizar podemos ver otras directivas como ‘ng-repeat’, mediante la que se itera sobre los emails recibidos desde el servidor, y los bindings asociados a propiedades de cada email sobre el que se itera.

Cualquier función que se defina en un controlador de **AngularJS** debe definirse sobre el ‘$scope’, un servicio de **AngularJS** con meta información del controlador que podemos equiparar al ‘this’ de **Java** o al ‘self’ de **Python**. De lo contrario, estaremos implementando una función de **JavaScript** y no de **AngularJS**.

# 10.2. Entorno de Gestión: GitHub

La herramienta usada para alojar el código de la aplicación web es **GitHub**. Se puede denominar como una forja para proyectos de software, la cual está basada en el sistema de control de versiones **Git**.

Se puede crear repositorios de forma totalmente gratuita. La diferencia entre las tarifas de los repositorios de **GitHub** reside en el número de contribuidores y en la visibilidad de dicho repositorio. En mi caso, en el que hay sólo un desarrollador y en el que este proyecto es académico, es más que suficiente tener un repositorio público en **GitHub**.

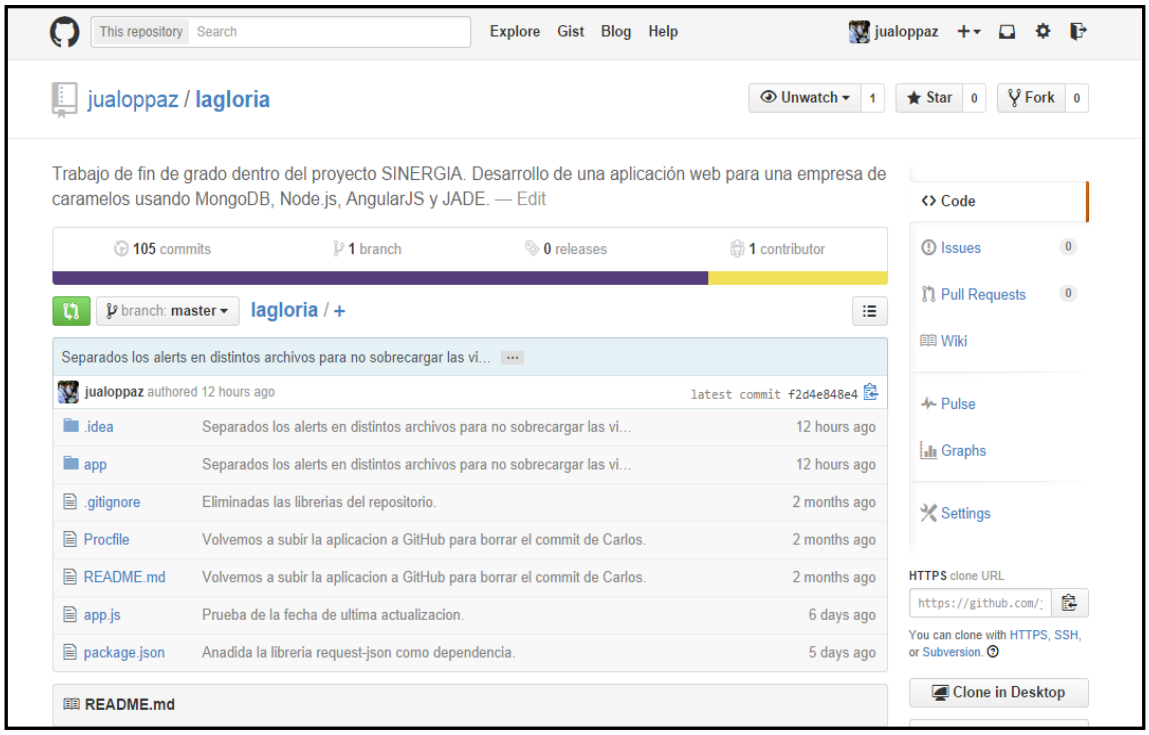



***Ilustración 58:*** *Arquitectura - Captura de la página del repositorio en* ***GitHub***

Aquí se puede observar una imagen de la página principal del repositorio “lagloria”, repositorio de mi propiedad en el que se aloja el código de la aplicación. Si en el futuro se unen otros desarrolladores, para que pueda trabajar sobre la aplicación no tiene más que ejecutar el comando “git clone” con la URL especificada abajo a la derecha. Otra opción más sencilla es seleccionar “Clone in desktop”, aunque para ello hay que tener instalada una pequeña aplicación de escritorio de **GitHub**.

Como en cualquier otra forja se pueden definir tareas, asociarlas a miembros del equipo, realizar estimaciones. Sin embargo, esta herramienta sólo ha sido usada como repositorio de código.

Una vez que hemos descargado la carpeta con el proyecto, si queremos realizar un cambio y subirlo al repositorio no tenemos más que seguir los pasos habituales de **Git**: “git init”, “git add .”, “git commit –m” seguido del mensaje del commit y “git push origin master”. Con estos comandos los cambios realizados quedan reflejados en el repositorio.

## 10.3. Entorno de Desarrollo: WebStorm y Robomongo

A día de hoy no existe un entorno de desarrollo específico para **Node.js**. Sin embargo, sí existen muchos IDE para proyectos **JavaScript**, y nosotros estamos realizando uno de este tipo.

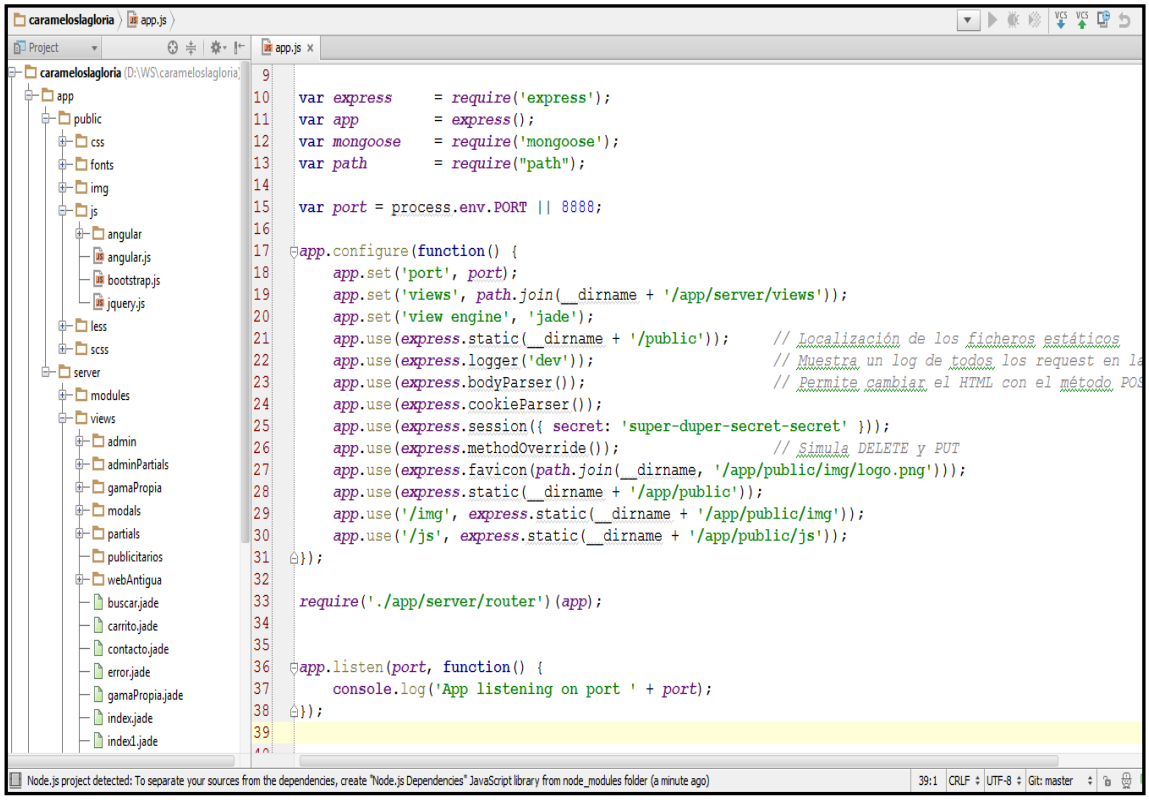



***Ilustración 59:*** *Arquitectura -* ***WebStorm 7***

El entorno de desarrollo utilizado ha sido **WebStorm 7**, el cual no compila los archivos **JavaScript** de **Node.js**, pero sí compila los archivos de **AngularJS**, ya que cuenta con un plugin para este framework. Esto último es lo que decantó la balanza a favor de **WebStorm**.

La interfaz gráfica de **WebStorm** es muy simple y sigue la misma línea que todos los IDE desarrollados por **JetBrains**.

No hay que realizar ninguna configuración especial para este entorno. Lo único que es aconsejable modificar es el tamaño de la fuente, el cual es muy pequeño por defecto. Para ello no tenemos más que dirigirnos a: 'File' > 'Settings' > 'Editor' > 'Colors & Fonts'. En esa pantalla podremos modificar el tamaño de la fuente o, incluso, crear nuestro estilo predefinido.

Otra herramienta que ha sido bastante útil ha sido **Robomongo 0.8.4**. Este herramienta no es más que una forma de consultar, añadir, editar o eliminar información de nuestra base de datos **MongoDB** de forma visual, y no mediante la consola, lo cual puede resultar en ocasiones más sencillo pero cuesta imaginar la estructura de la información en la base de datos.

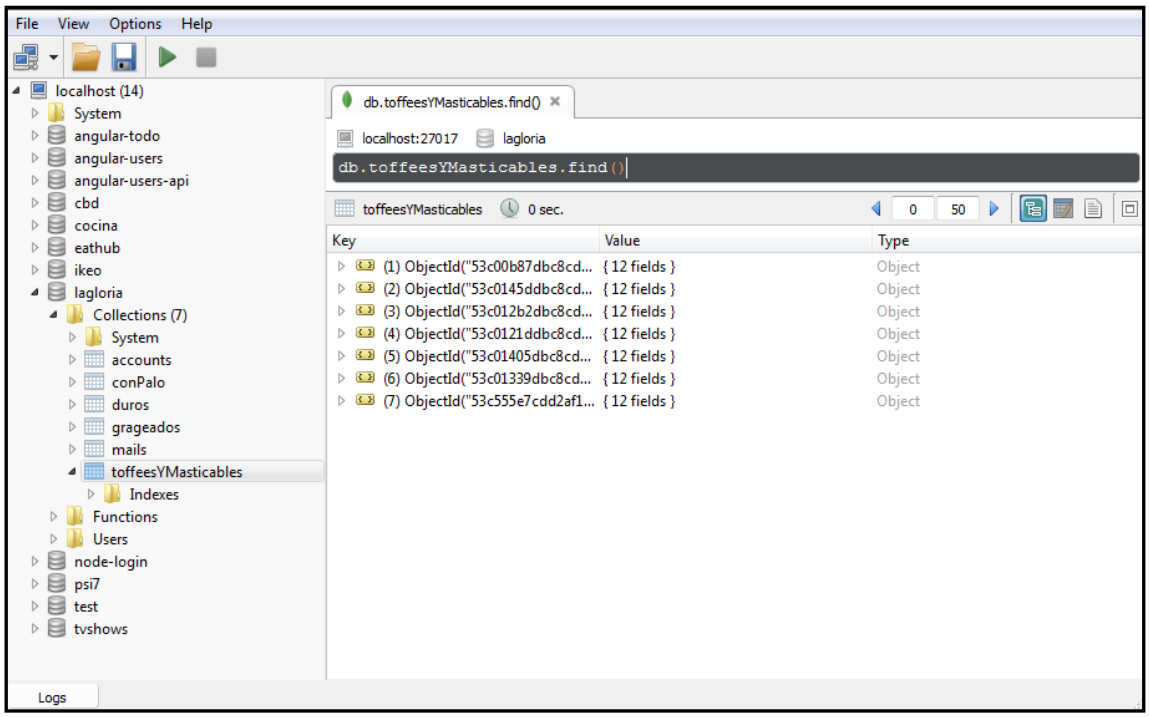



***Ilustración 60:*** *Arquitectura - Consulta realizada en* ***Robomongo***

Una de las prestaciones más interesantes de **Robomongo** es el hecho de poder editar la información de la base de datos directa y visualmente, sin necesidad de consultas en el terminal de comandos.

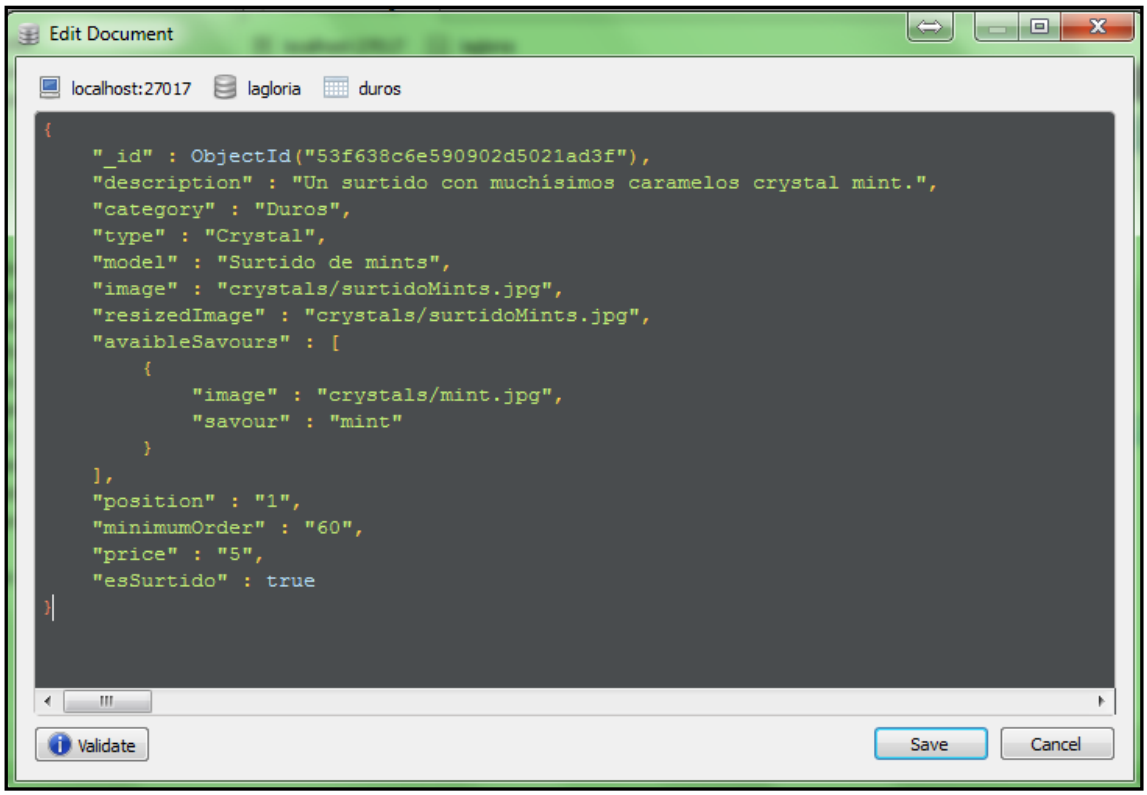



***Ilustración 61:*** *Arquitectura - Edición de documento en* ***Robomongo***

Es una herramienta muy fácil de instalar. Basta con que descarguemos el ejecutable de la página http://robomongo.org .

# 10.4. Estructura del proyecto

En esta sección se hablará de la estructura del proyecto **Node.js** en **WebStorm**, detallándose cada directorio para localizar cada archivo y conocer la utilidad y el papel que ocupan en la aplicación web.

En primer lugar se mostrará una captura de la vista general del proyecto, donde se comentarán las principales carpetas a grandes rasgos, y en segundo lugar se irá detallando de forma específica cada sección.

Esta sección es ideal para identificar cada fichero de la aplicación en caso de que se necesite realizar alguna modificación o implementación nueva.

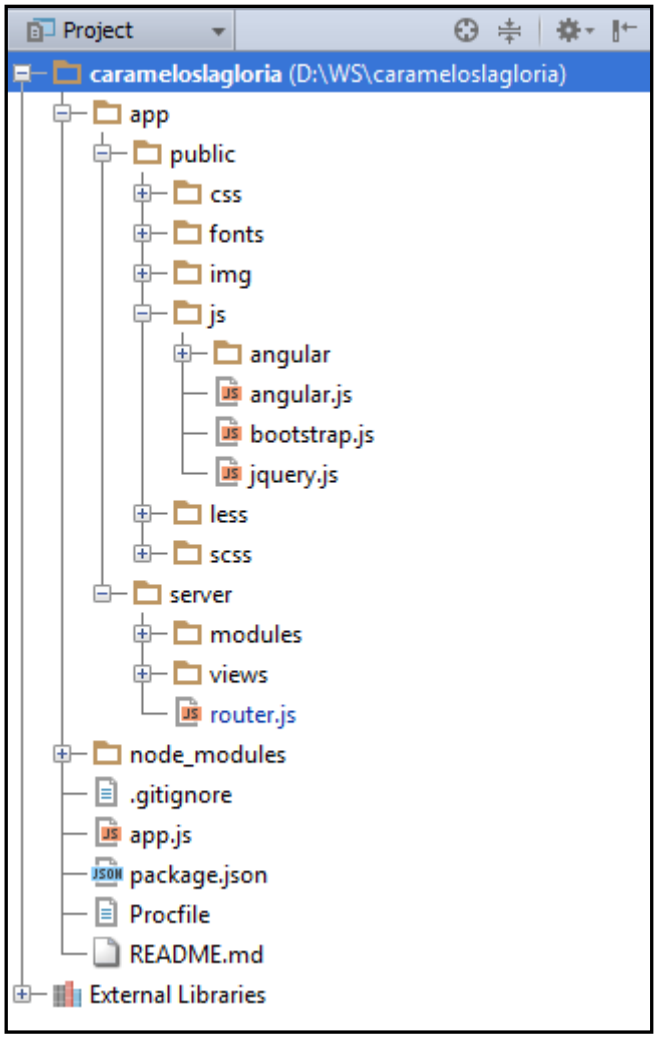



***Ilustración 62:*** *Arquitectura - Estructura del proyecto en* ***WebStorm***

En esta imagen se puede ver la estructura principal del proyecto, el cual tiene como raíz un directorio llamado “carameloslagloria”. En este directorio podemos encontrar lo siguiente:

- ‘app’: directorio en el que ubicaremos el código de la aplicación web, tanto el lado del servidor como del cliente.
- ‘node-modules’: esta carpeta contiene las dependencias de **Node.js** del proyecto, las cuales se instalan desde el terminal de comandos escribiendo ‘npm install’.
- ‘app.js’: este archivo contiene la configuración básica del servidor, así como la declaración de los archivos necesarios para el funcionamiento de la aplicación web, como es el caso del archivo ‘router.js’, el cual se explicará más adelante.
- ‘packaje.json’: en este archivo se encuentran declaradas las dependencias de nuestro proyecto, usado por **Heroku** para identificar las librerías. Para que este archivo se escriba automáticamente tenemos que añadir el parámetro ‘--save’ en el comando ‘npm install’. De lo contrario tendremos que añadirlas manualmente.
- ‘Procfile’: es un archivo sin extensión utilizado exclusivamente por **Heroku** para identificar el fichero base de la aplicación (el ‘app.js’). Tiene que tener exactamente ese contenido y el fichero ese nombre. Si el fichero tiene un nombre distinto, aunque cambie sólo un carácter, **Heroku** no lo reconocerá.

A continuación, como he dicho anteriormente, veremos de forma detallada los directorios que encontramos en el proyecto.

Primer tenemos que hablar del directorio ‘app’. En él hay 2 carpetas: ‘public’ y ‘server’. En la primera se ubican los archivos estáticos que son usados en el cliente, mientras que en el servidor se encuentran archivos a los que sólo puede tener acceso el servidor y las vistas, para que se envíen al cliente sólo cuando sean requeridas.

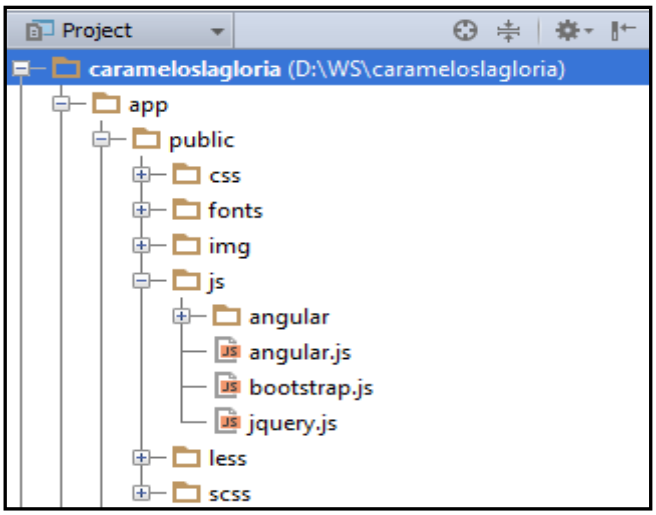



***Ilustración 63:*** *Arquitectura - Carpeta public del proyecto*

En la carpeta 'public' podemos encontrar las siguientes subcarpetas:

- 'css': aquí se encuentran las hojas de estilo utilizadas en el proyecto, a saber: **Bootstrap**, **Eathub**, **Fontawesome** y la propia de **La Gloria**.
- 'fonts': en esta carpeta se encuentran las fuentes del proyecto, así como la tipografía Abel de **Google Fonts** o archivos necesarios para **Fontawesome**.
- 'img': carpeta que contiene todas las imágenes de la aplicación web, la gran mayoría de productos de **La Gloria S.L**.
- 'js': aquí se encuentra el código **JavaScript** que se ejecutará en el navegador (en el lado del cliente). La inmensa mayoría de los ficheros **JavaScript** contenidos en esta carpeta se corresponden con ficheros de **AngularJS**, aunque también se puede ver un archivo necesario para las interacciones de **Bootstrap**, como es 'bootstrap.js'.
- Carpetas 'less' y 'scss': son carpetas utilizadas por **Fontawesome**. No es necesario realizar ningún tipo de modificación.

Procedamos ahora con la estructura de la aplicación en el lado del servidor.

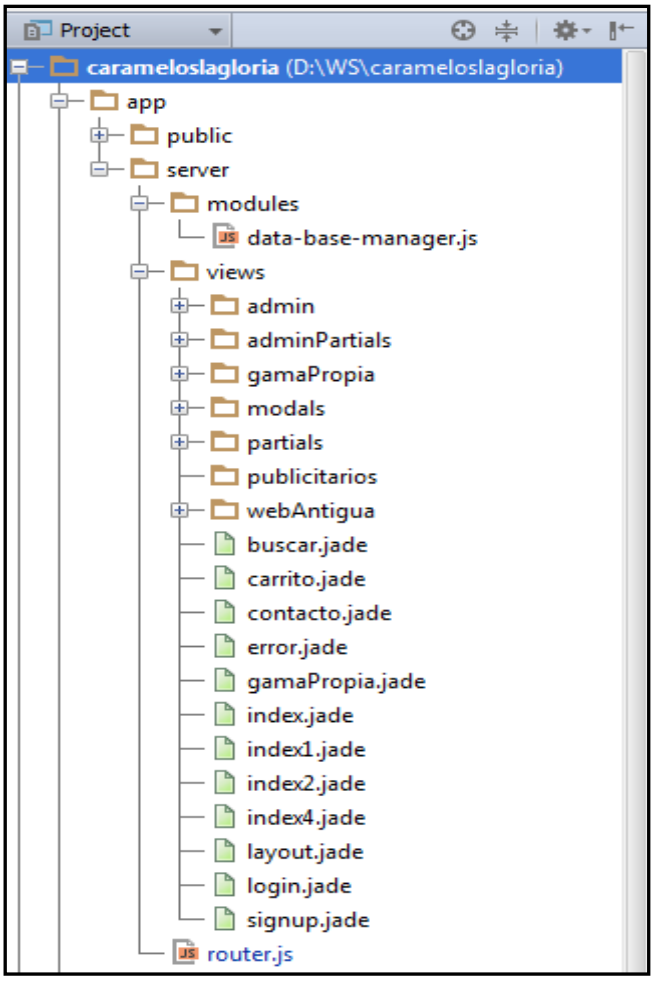



***Ilustración 64:*** *Arquitectura - Carpeta server del proyecto*

En el directorio 'server' podemos encontrar lo siguiente:

- 'modules': aquí se ubicarán los distintos módulos usados por el servidor. Actualmente sólo existe el gestor de conexión con la base de datos, el **DBM** (Data Base Manager), donde se ubican las consultas que son realizadas en la aplicación web. Este fichero es llamado desde el 'router.js', el cual se explica a continuación.
- 'views': en este directorio se definen las vistas utilizadas en la aplicación. De forma general podemos ver todas las vistas, mientras que las vistas del administrador están en una carpeta única. Así mismo, existen carpetas como 'partials' donde se definen fragmentos de código comunes en la aplicación, como por ejemplo la barra de navegación, los alerts o el carousel usado para mostrar los caramelos.
- 'router.js': fichero en el que se detallan las rutas de la aplicación web y que es llamado desde el archivo principal de la aplicación 'app.js'. El hecho de separar las rutas de la configuración del servidor es por

comodidad y por seguir buenas prácticas. Para una aplicación pequeña podrían estar ambas cosas en un mismo fichero.

Una vez dicho todo esto hemos explicado lo fundamental de la estructura del proyecto para entender la utilidad de cada fichero y por qué está estructurado así y no de otra forma.

# 11. PRUEBAS

En este capítulo se encuentran las distintas pruebas realizadas sobre la aplicación web.

El primer tipo de pruebas son las pruebas de renderizado. Se ha probado la aplicación web en varias combinaciones de dispositivos, sistemas operativos y navegadores para comprobar que se daba soporte a la mayor combinación posible, tal y como se definía en uno de los requisitos no funcionales en el bloque de análisis.

# 11.1. Pruebas de renderizado

En estas pruebas se ha probado la aplicación web en diversas combinaciones de sistemas operativos, dispositivos y navegadores, con el fin de verificar que la aplicación web se puede utilizar en la mayor cantidad de plataformas posible.

Los dispositivos usados para las pruebas han sido un PC, un iPad y un Smartphone, mientras que los navegadores usados han sido Chrome, Firefox y Safari.

Los sistemas operativos utilizados han sido: Windows 7, iOS 7.1.2 y Android 4.1.2.

Para finalizar, los navegadores utilizados han sido: Chrome 37.0.2062.94 m, Mozilla Firefox 31.0 y Safari 7.0.

Las capturas van a ser mostradas ordenadas por dispositivos, de modo que irán variando los navegadores sobre los que se visualiza la aplicación.

A continuación se muestran todas las capturas realizadas, detallando el dispositivo, plataforma y navegador usado en la misma:

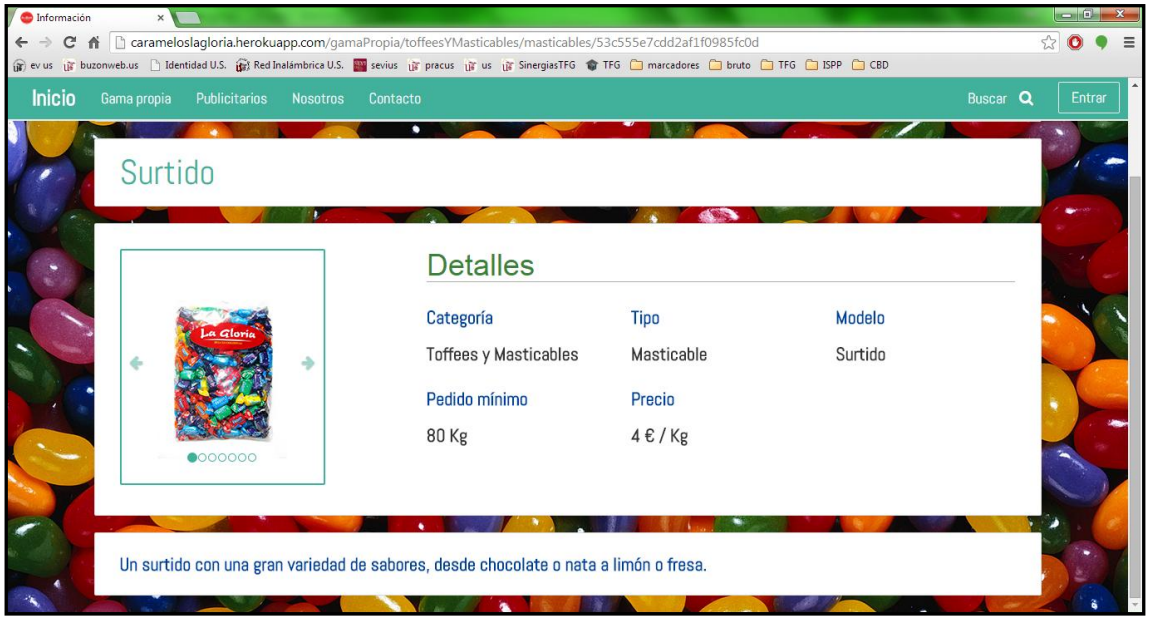



***Ilustración 65:*** *Pruebas de renderizado - Chrome 37.0.2062.94 m – Windows 7*

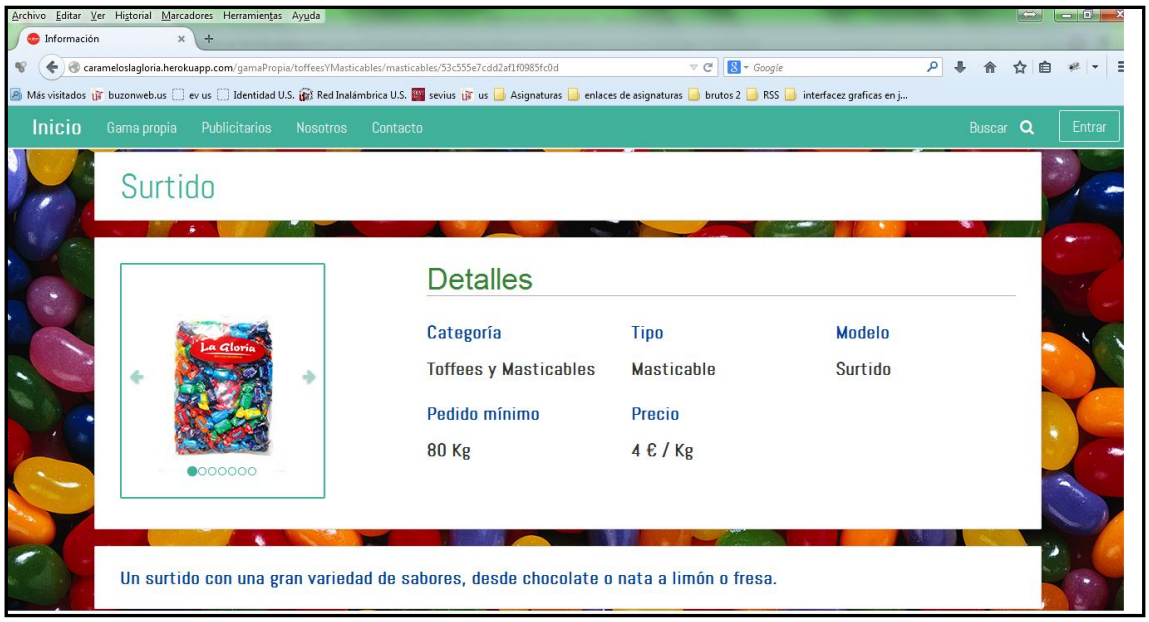



***Ilustración 66:*** *Pruebas de renderizado - Mozilla Firefox 31.0 – Windows 7*

Pasamos al iPad:

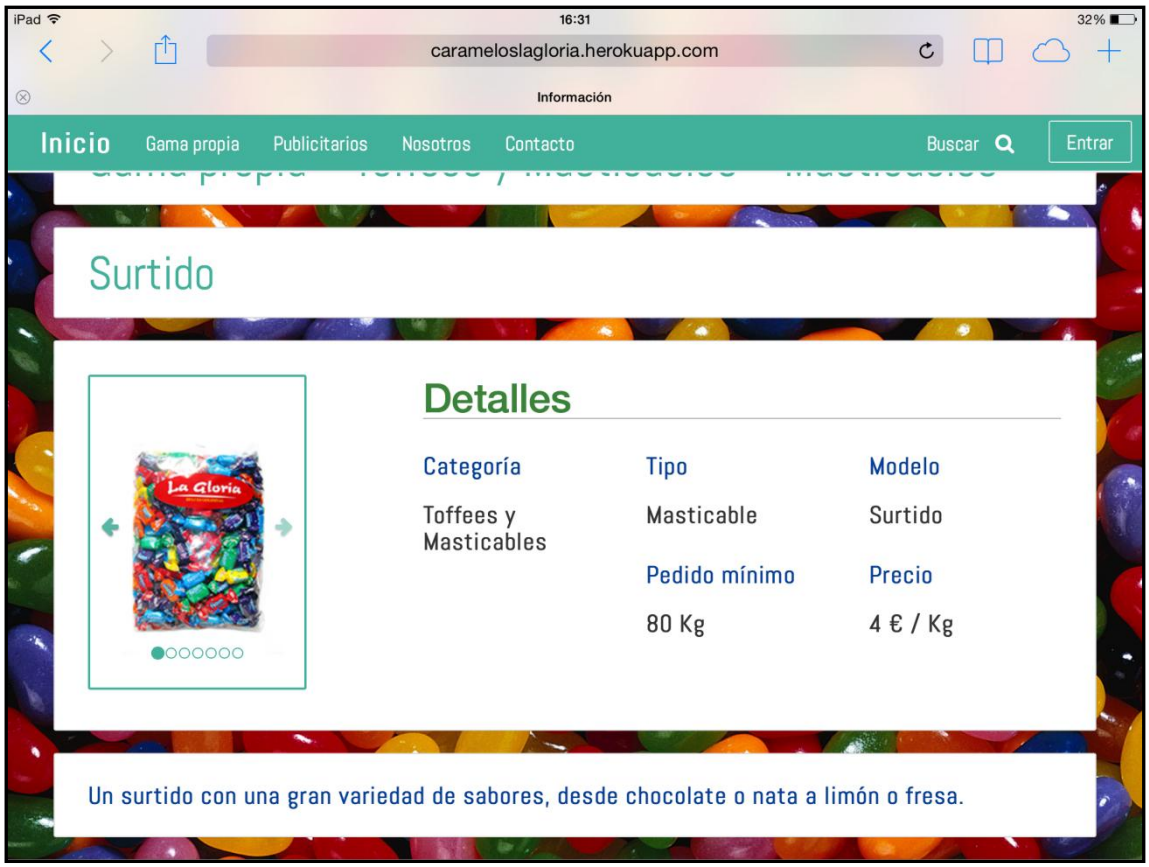



***Ilustración 67:*** *Pruebas de renderizado - Safari 7.0 – iOS 7.1.2 (orientación horizontal)*

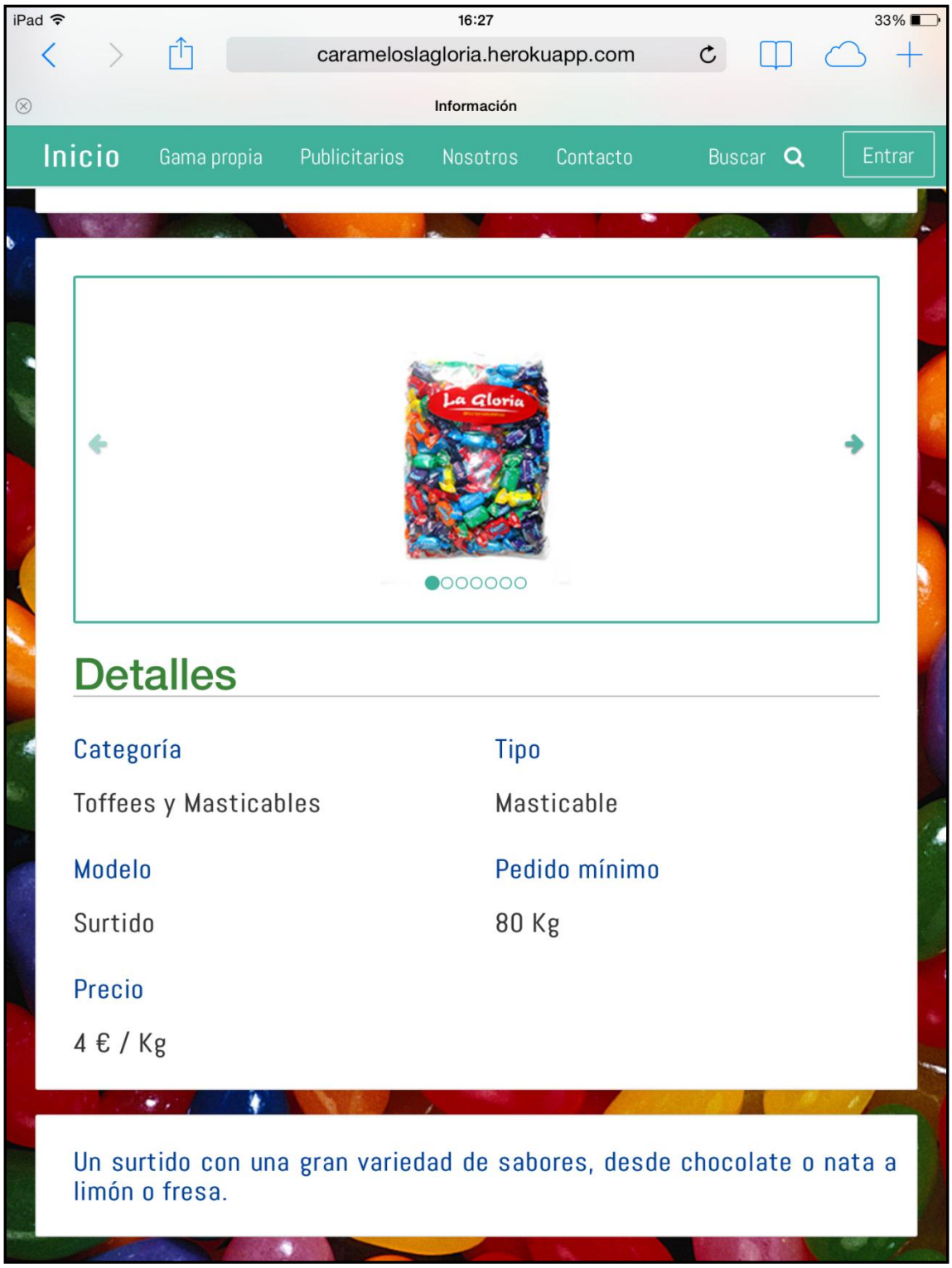



***Ilustración 68:*** *Pruebas de renderizado - Safari 7.0 – iOS 7.1.2 (orientación vertical)*

Para finalizar se muestra la captura en un smartphone:

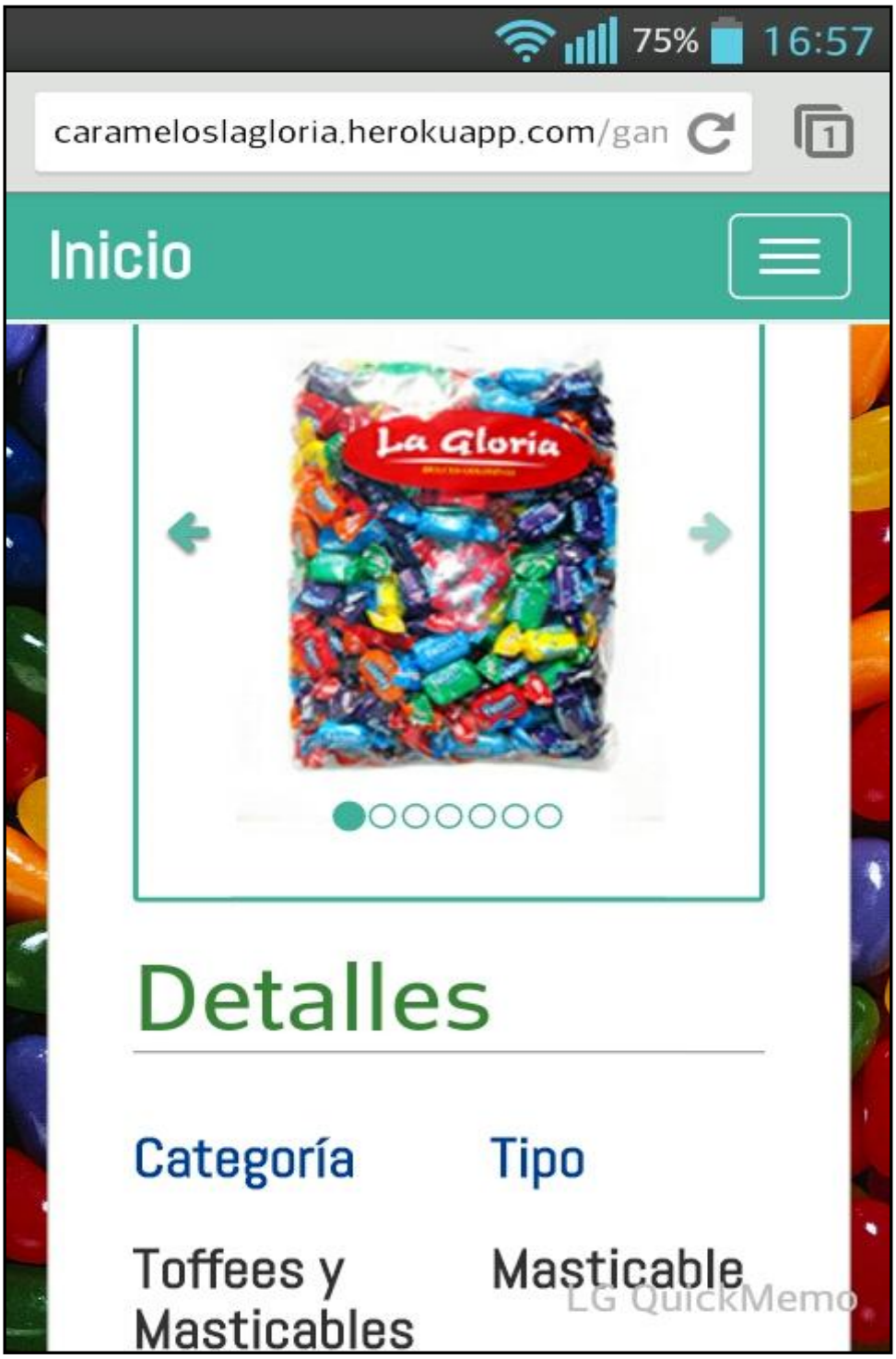



***Ilustración 69:*** *Pruebas de renderizado - Chrome 37.0.2062.94 m – Android 4.1.2*

# 11.2. Pruebas unitarias

Las pruebas unitarias sirven para verificar el correcto funcionamiento de ciertas funcionalidades de un sistema.

En esta sección se muestran las pruebas unitarias realizadas sobre la aplicación. Estas pruebas han sido realizadas con la librería **Protractor**[13], una librería especializada en pruebas para **AngularJS**, contando con un programa **Node.js** creado con otra librería: **WebDriverJS**.

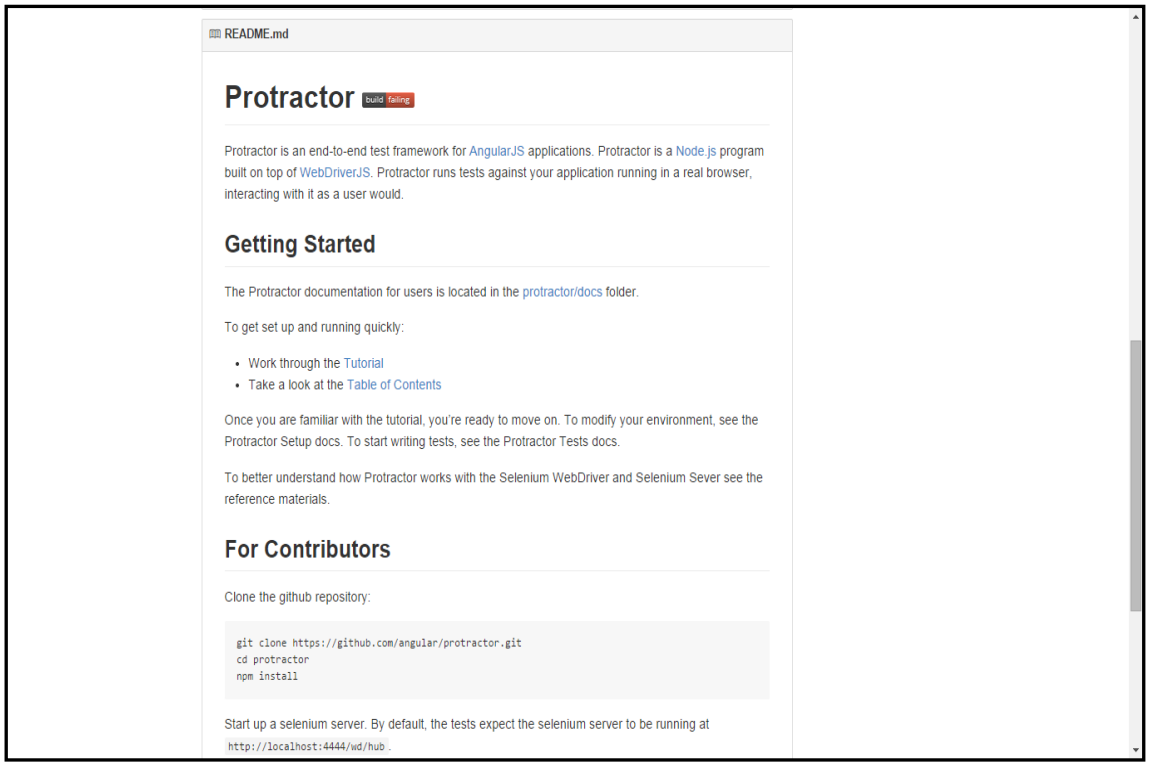



***Ilustración 70:*** *Pruebas unitarias - Repositorio de* ***Protractor*** *en* ***GitHub***

Esta librería engloba a otras librerías **JavaScript** para evaluar el funcionamiento de las implementaciones realizadas. Por defecto, Protractor usa **Jasmine**, un framework para la realización de pruebas de código **JavaScript**.

Por último tenemos el **Webdriver-manager**, una herramienta que proporciona una instancia de un servidor **Selenium**. Este tipo de servidor tiene amplios y diversos propósitos, pero nosotros lo usaremos para interpretar los resultados de las pruebas, los cuales se muestran en el terminal de comandos.

---

[13] https://github.com/angular/protractor

El fin de Protractor no es más que verificar que las funcionalidades implementadas operan correctamente y que la información provista por el servidor es enviada al cliente y renderizada correctamente en las vistas.

# Instalación

Para instalar todas las herramientas tenemos que seguir los siguientes pasos:

*npm install –g protractor*

Con este comando instalamos de forma global Protractor en nuestro pc. Instalar una librería en **Node.js** de forma global quiere decir que se instala en el pc, y no en el proyecto como dependencia de **Node.js**. Esto se hace para evitar que dicha librería quede desplegada en el servidor donde se aloje la aplicación, ya que las pruebas se van a realizar en local.

webdriver-manager update

A continuación se actualiza el **Webdriver-manager**, descargando todos los binarios necesarios para lanzar el servidor **Selenium**.

webdriver-manager start

Haciendo esto lanzamos el servidor **Selenium**, cuyo funcionamiento se puede comprobar en la consola y en el navegador en la dirección: http://localhost:4444/wd/hub:

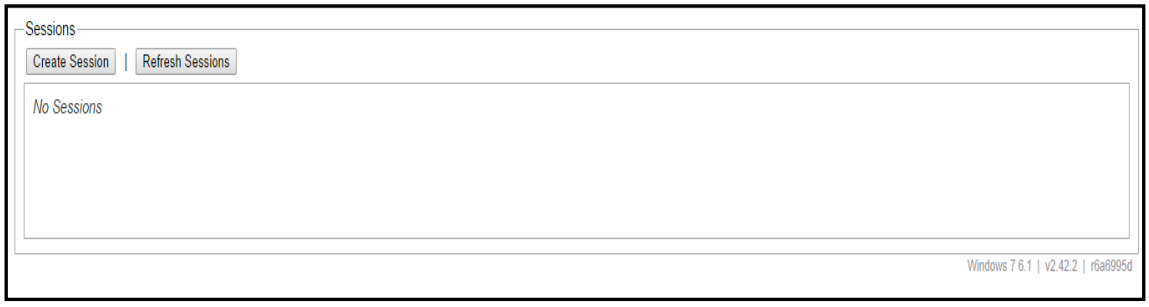



***Ilustración 71:*** *Pruebas unitarias - Selenium server iniciado*

Sin embargo, no vamos a usar el navegador, ya que nos es suficiente con el terminal de comandos.

Una vez que se han instalado todas las herramientas podemos diseñar las primeras pruebas:

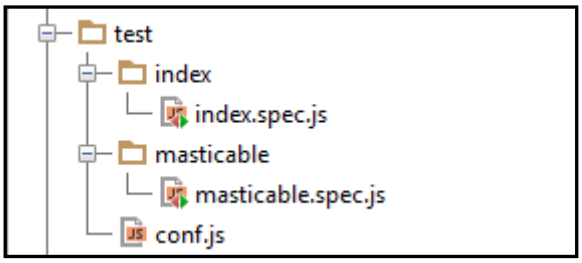



***Ilustración 72:*** *Pruebas unitarias - Estructura de las pruebas*

# Plan de pruebas

Usualmente se suelen añadir estas pruebas dentro de una carpeta llamada ‘test’, la cual se encuentra en la raíz del proyecto al mismo nivel que la carpeta ‘app’.

El archivo principal es el ‘conf.js’:

```
1
2  exports.config = {
3      rootElement: '.lagloria', //nombre del app de angular
4      seleniumAddress: 'http://localhost:4444/wd/hub',
5      specs: ['*/*.spec.js']
6  }
7
```

***Ilustración 73:*** *Pruebas unitarias - Fichero de configuración de Protractor: Conf.js*

En este fichero se indica el nombre del módulo app de **AngularJS** utilizado en las plantillas, la dirección del servidor **Selenium** y la ubicación de los archivos que contendrán las pruebas unitarias.

```
describe('Pruebas unitarias - Index', function() {
    it('Debe tener el titulo establecido', function() {
        browser.get('http://localhost:8888');

        expect(browser.getTitle()).toEqual('Caramelos La Gloria');
    });

    it('No debe calcular la fecha de ultima actualizacion', function() {
        browser.get('http://localhost:8888');

        expect(element(by.binding("fecha")).getText())
            .toEqual('Última actualización: Estamos en local');
    });

});
```

**Ilustración 74:** Pruebas unitarias - Pruebas de la página de inicio

Este archivo es el ‘index.spec.js’, donde se realizan algunas pruebas en las que se verifica que el contenido de la vista mostrada es el esperado.

# 11.3. Pruebas de rendimiento.

En esta ocasión se abordan las pruebas de rendimiento. Las pruebas de rendimiento sirven para comprobar el comportamiento de la aplicación web, especialmente en el servidor, ante posibles escenarios. Estos escenarios pueden contener varios usuarios, usuarios simultáneos o numerosas peticiones simultáneas.

El comportamiento de una aplicación web se mide analizando las métricas que proporciona la herramienta de pruebas utilizada.

En un primer instante pretendí usar **jMeter**, una herramienta que he usado con anterioridad para realizar pruebas sobre una aplicación web. Inicialmente se diseñó para aplicaciones desarrolladas en **Java**. Sin embargo, en cada nueva versión soporta más lenguajes de programación y plataformas.

Esta herramienta debe soportar **Node.js**, ya que en un gran número de casos se utilizan peticiones **HTTP** en el servidor implementado. Sin embargo, no funcionaba de la forma que yo esperaba. Debido al comportamiento inusual que mostraba decidí usar una herramienta más específica de **Node.js**.

En Internet podemos encontrar un elevado número de herramientas y librerías para analizar el rendimiento de una aplicación web **Node.js**. En este caso, la herramienta elegida ha sido **Loadtest**[14].

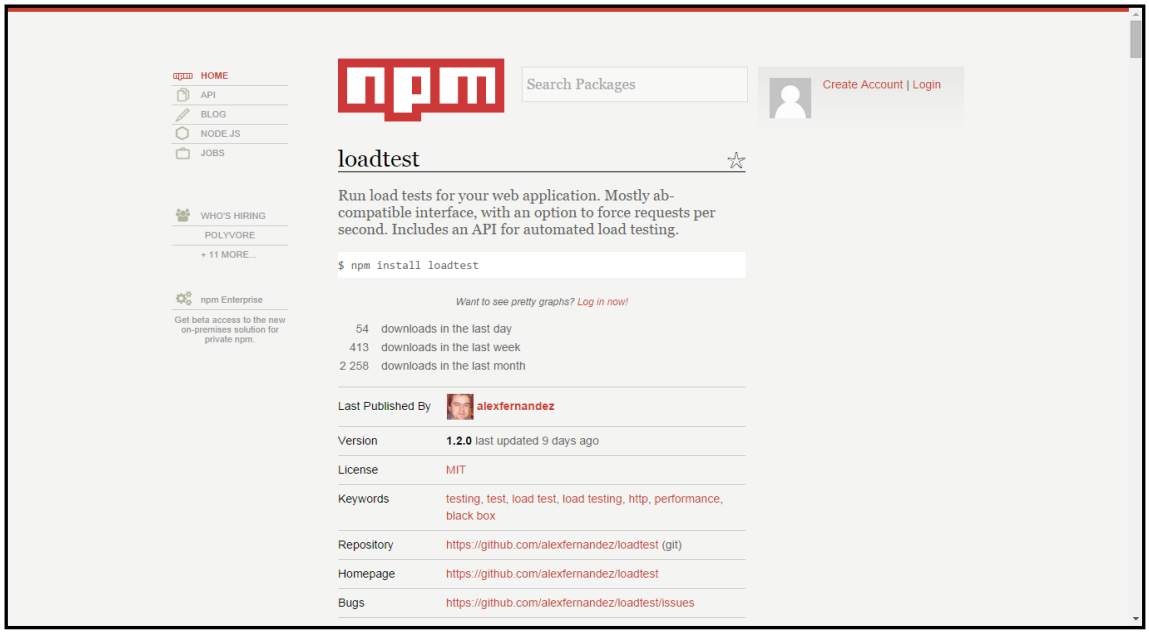



***Ilustración 75:*** *Pruebas de rendimiento - Pantalla de* ***Loadtest*** *en el registro NPM*

[14] https://www.npmjs.org/package/loadtest

El motivo por el que se ha elegido este herramienta es porque se instala muy fácilmente, se usa con apenas 1 comando en el terminal de comandos y proporciona suficiente información del comportamiento del servidor.

Además, se pueden configurar a la perfección los escenarios de prueba indicando el número de usuarios concurrentes o el número de peticiones.

# Instalación

Para instalar el benchmark **Loadtest** se procede de la misma forma que se procedió en el apartado anterior para instalar **Protractor**:

*npm install –g loadtest*

Igual que en el caso anterior, es preferible instalar la herramienta de forma global antes que incluirla como dependencia en el proyecto ya que sólo la usaremos en local.

# Plan de pruebas

Para evaluar el rendimiento de la aplicación web se plantea una serie de casos, en los que se configuran parámetros distintos para simular distintos escenarios.

Además, se harán unas pruebas relacionadas con vistas de la aplicación web, mientras que otras estarán relacionadas con la **API REST** implementada.

(Nota: como las métricas se imprimen en el terminal de comandos, para mayor comodidad se agruparán en tablas)

### Vistas de la aplicación web

En este subapartado se muestran 2 peticiones de vistas estáticas.

| Comandos |
| --- |
| **GET /** |

| |
|---|
| loadtest –n 1 –c 1 http://localhost:8888 |
| loadtest –n 10 –c 5 http://localhost:8888 |
| **GET /gamaPropia/duros/especiales** |
| loadtest –n 1 –c 1 http://localhost:8888/gamaPropia/duros/especiales |
| loadtest –n 10 –c 5 http://localhost:8888/gamaPropia/duros/especiales |

***Tabla 61:*** *Pruebas de rendimiento - Comandos de las pruebas de vistas estáticas*

| URL | Número de peticiones | Número de usuarios concurrentes | Tiempo total |
|---|---|---|---|
| GET / | 1 | 1 | 0.2457 s |
| | 10 | 5 | 1.7427 s |
| GET /gamaPropia/duros/especiales | 1 | 1 | 0.2954 s |
| | 10 | 5 | 2.3908 s |

***Tabla 62:*** *Pruebas de rendimiento - Resultados de las pruebas de vistas estáticas*

Como podemos ver el rendimiento de la aplicación con pocos usuarios y peticiones es bastante bueno. Sin embargo, a medida que se aumenta el valor de algún parámetro el rendimiento baja un poco, aunque sin llegar a ser malo.

**API REST**

Ahora es el turno de las pruebas relacionadas con la **API REST**:

| Comandos |
|---|
| **GET /api/especiales** |
| loadtest –n 1 –c 1 http://localhost:8888/api/especiales |
| loadtest –n 10 –c 5 http://localhost:8888/api/especiales |
| **GET /api/especiales/53c7eeb02d9b00e2354fe1ad** |
| loadtest –n 1 –c 1 http://localhost:8888/api/especiales/53c7eeb02d9b00e2354fe1ad |
| loadtest –n 10 –c 5 http://localhost:8888/api/especiales/53c7eeb02d9b00e2354fe1ad |

***Tabla 63:*** *Pruebas de rendimiento - Comandos de las pruebas de la* ***API REST***

| URL | Número de peticiones | Número de usuarios concurrentes | Tiempo total |
|---|---|---|---|
| GET /api/especiales | 1 | 1 | 0.0366 s |
| | 10 | 5 | 0.0711 s |
| GET /api/especiales/ /53c7eeb02d9b00e2354fe1ad | 1 | 1 | 0.0351 s |
| | 10 | 5 | 0.0692 s |

***Tabla 64:*** *Pruebas de rendimiento - Resultados de las pruebas de la* ***API REST***

En esta ocasión, los resultados son muchísimo mejores. Los tiempos de respuesta son muy bajos, ya que cualquier consulta a la **API REST** devuelve un simple fichero en formato **JSON**, mientras que en el apartado anterior la respuesta es una vista en formato **HTML**.

# 11.4. Pruebas de usuarios

En este apartado se muestran las pruebas que han realizado diferentes usuarios de la aplicación web desarrollada.

A continuación se muestra una tabla en la que se detallan las valoraciones realizadas por usuarios sin conocimientos técnicos:

| Usuario | Diseño | Usabilidad | Rendimiento | Comodidad |
|---|---|---|---|---|
| José Olmo López | 6 | 4 | 8 | 7 |
| María Gamero Alonso | 7 | 7 | 9 | 8 |
| Francisco José Sánchez Cárdenas | 7 | 7 | 7 | 8 |
| Jesús Martínez Fernández | 7 | 8 | 8 | 7 |

***Tabla 65:*** *Pruebas de usuarios - Valoración de usuarios sin conocimientos técnicos*

En general, la valoración de los usuarios es bastante buena. La valoración media es entorno al 7.

Podemos ver que hay un usuario algo más exigente, pero tanto en estética como en rendimiento la web está muy aceptada. Ambos objetivos parecen haberse conseguido.

# BLOQUE VI: MANUALES

# 12. MANUAL DE USUARIO

En este capítulo se encuentran los manuales de usuario, los cuales son de gran utilidad para saber qué cosas puede hacer cada usuario y cómo se puede usar la web para hacer uso de las funcionalidades que ofrece.

Así pues, existen 3 manuales específicos según el rol del usuario que esté usando la aplicación web.

# 12.1. Manual de usuario para anónimo

A continuación se detalla un pequeño índice con las acciones que puede realizar un usuario anónimo en la aplicación web:

- Gestión de contenidos.
  - Ver los productos de una categoría.
  - Ver los detalles de un producto en una categoría.
  - Buscar producto en el sistema.
- Gestión de usuarios.
  - Registrarse como usuario.
- Gestión de emails.
  - Enviar email.

# Gestión de contenidos

En este apartado se explican las acciones que un usuario anónimo puede realizar relacionadas con los contenidos mostrados en la aplicación web.

### Ver los productos de una categoría

La página de inicio de la aplicación web es la que se muestra a continuación:

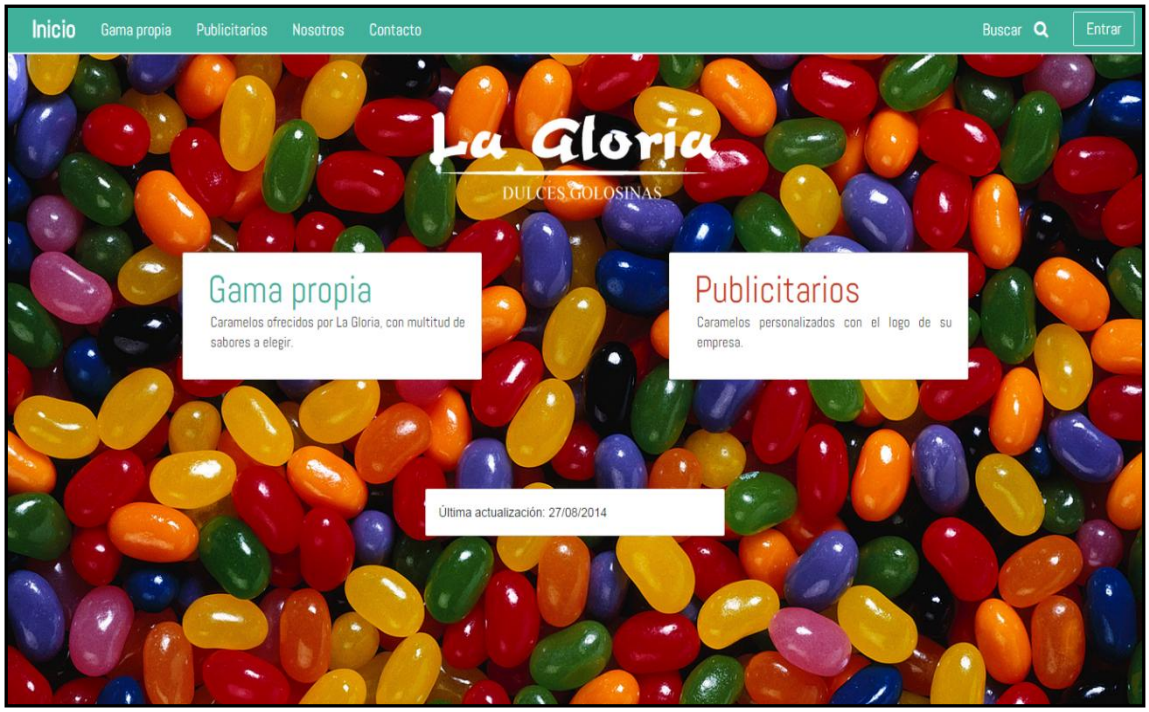



***Ilustración 76:*** *Manual de usuario anónimo - Página de inicio*

Para ver los productos de una categoría primero tenemos que seleccionar una gama ofrecida por **La Gloria S.L.**, por ejemplo, la gama propia:

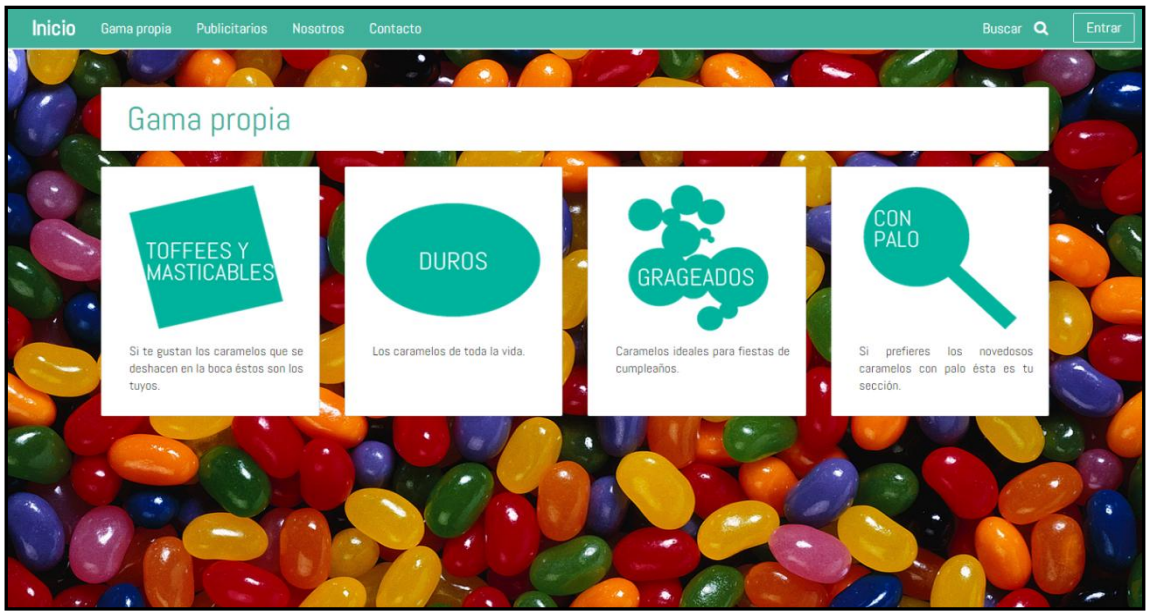



***Ilustración 77:*** *Manual de usuario anónimo - Gama propia*

A continuación seleccionamos una categoría, por ejemplo la categoría ‘Duros’. De este modo podremos ver los productos de dicha categoría clasificados por tipo, tal y como se puede ver a continuación:

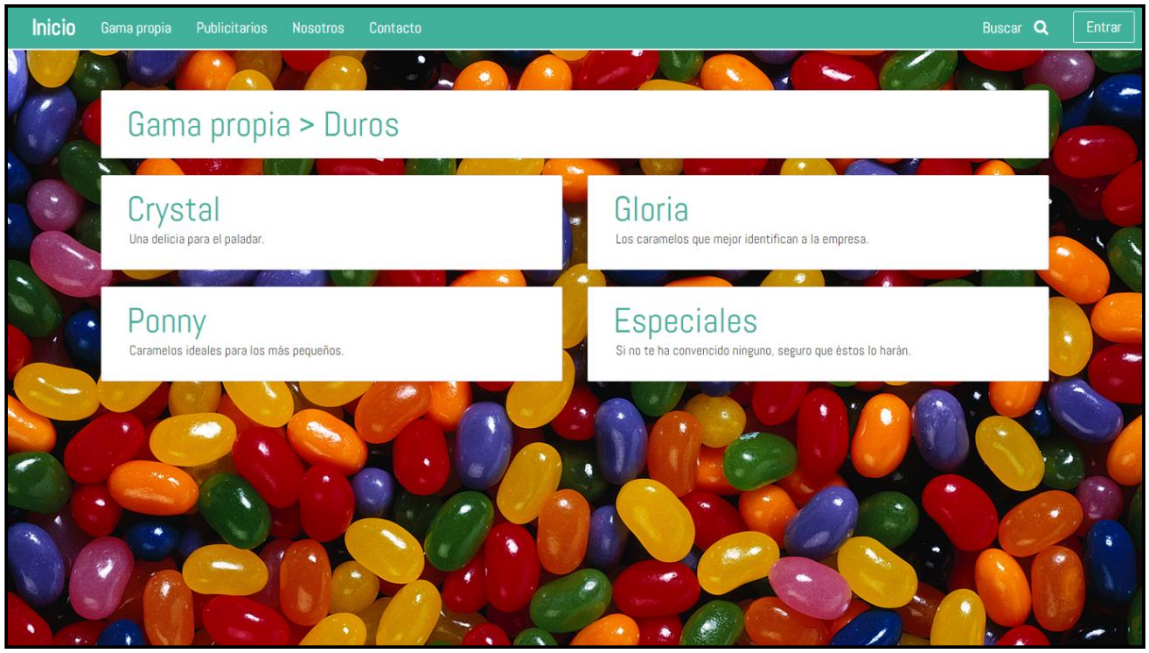



***Ilustración 78:*** *Manual de usuario anónimo - Tipos de caramelo duros de la gama propia*

A continuación elegimos un tipo de producto, por ejemplo, el tipo Gloria, lo que nos lleva a la última pantalla:

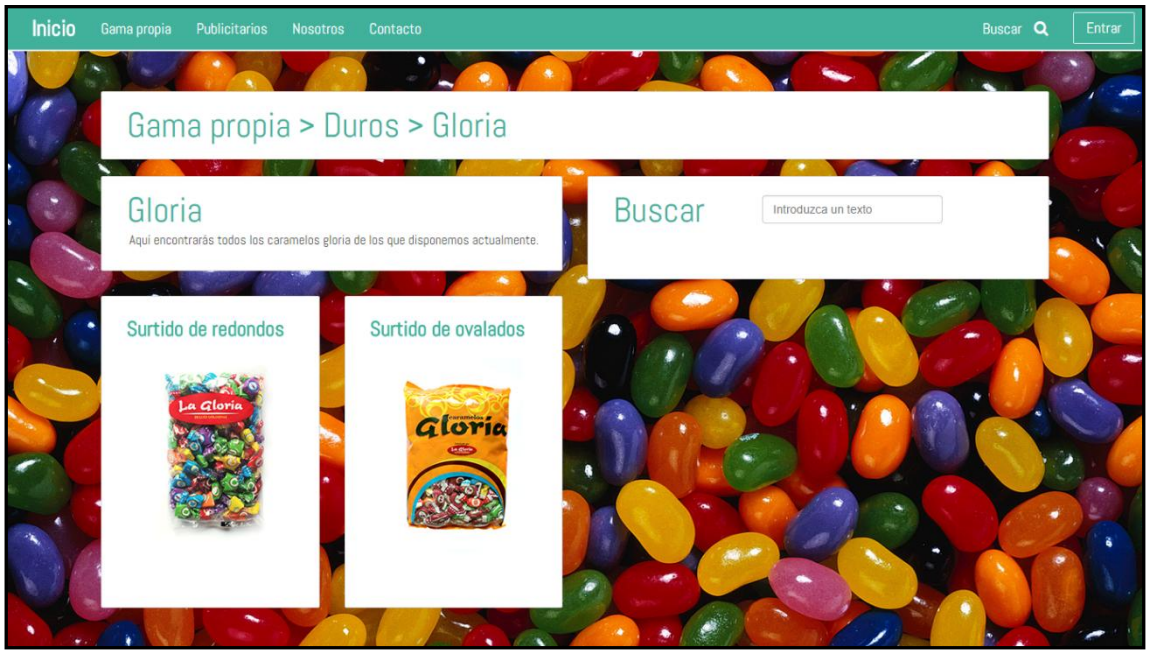



***Ilustración 79:*** *Manual de usuario anónimo - Modelos de caramelos Gloria*

### Ver los detalles de un producto en una categoría

Para ver los detalles de un producto primero tenemos que realizar la acción anterior. Una vez hecho sólo tenemos que seleccionar el producto que más nos guste. Siguiendo la navegación anterior, vamos a ver los detalles del surtido de ovalados:

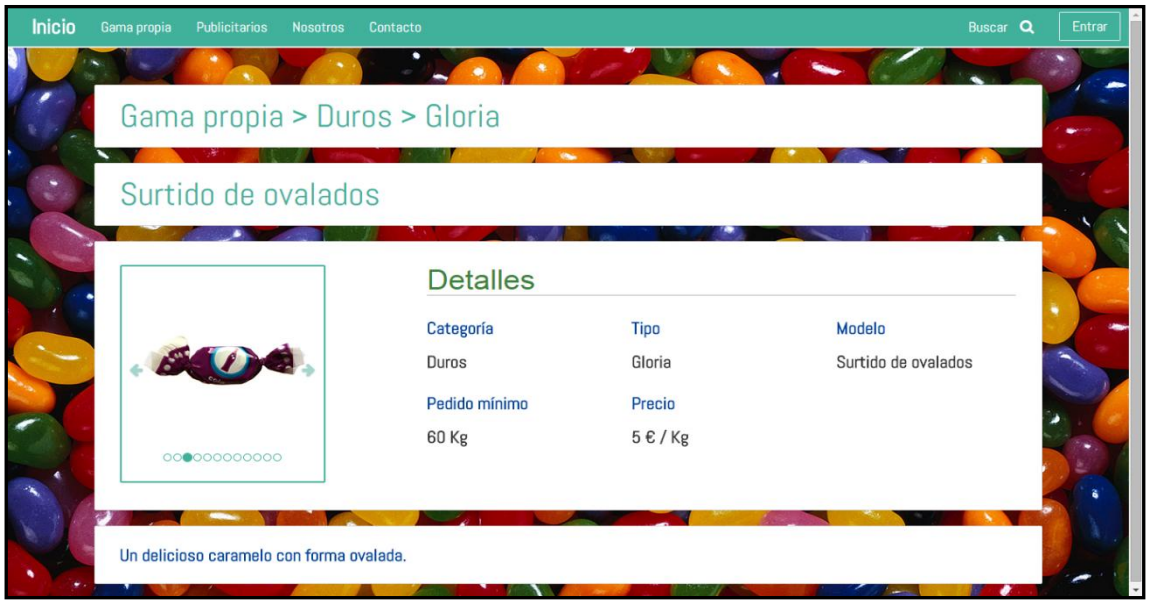



***Ilustración 80:*** *Manual de usuario anónimo - Detalles del surtido de ovalados*

En esta pantalla se pueden ver todos los detalles del producto seleccionado, así como una serie de imágenes en un carousel, las cuales se corresponden con los caramelos que contiene el surtido de ovalados seleccionado.

### Buscar producto en el sistema

Esta acción se puede realizar desde cualquier página de la web. Para ello sólo tenemos que seleccionar el botón 'Buscar' situado en la barra de navegación principal de la aplicación web. Tras hacer esto veremos una pantalla como ésta:

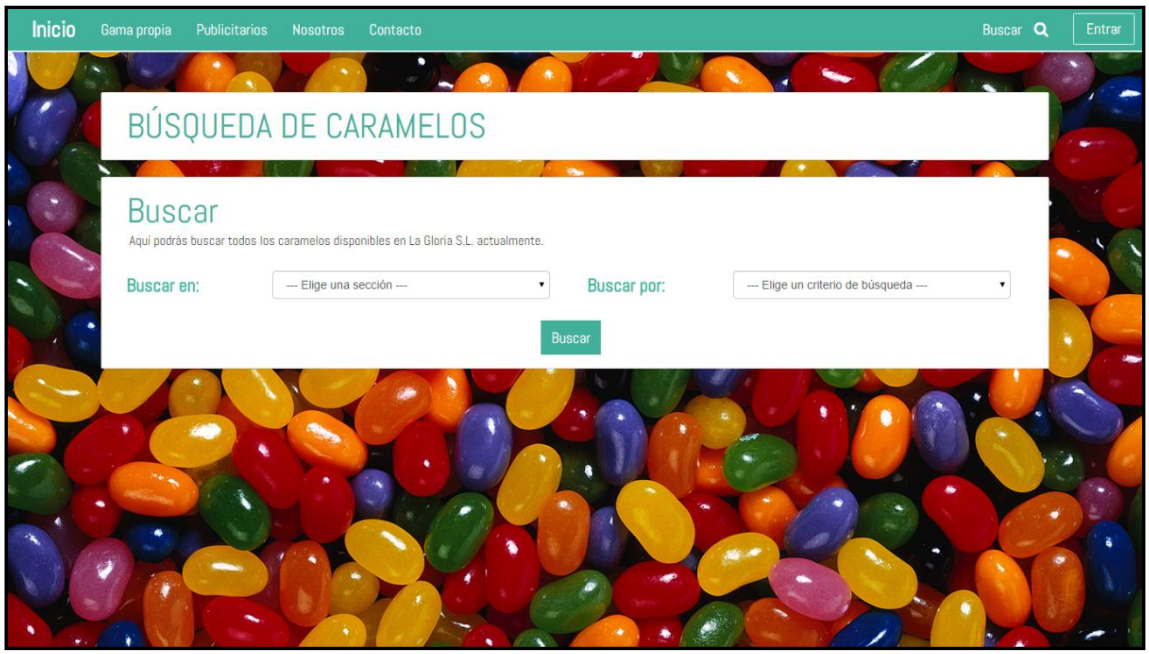



***Ilustración 81:*** *Manual de usuario anónimo - Búsqueda de caramelos*

Vamos a realizar una búsqueda en la gama propia, por categoría y, en concreto, Toffees y masticables, y seleccionamos el botón 'Buscar'. En unos instantes aparecerá el resultado de la búsqueda:

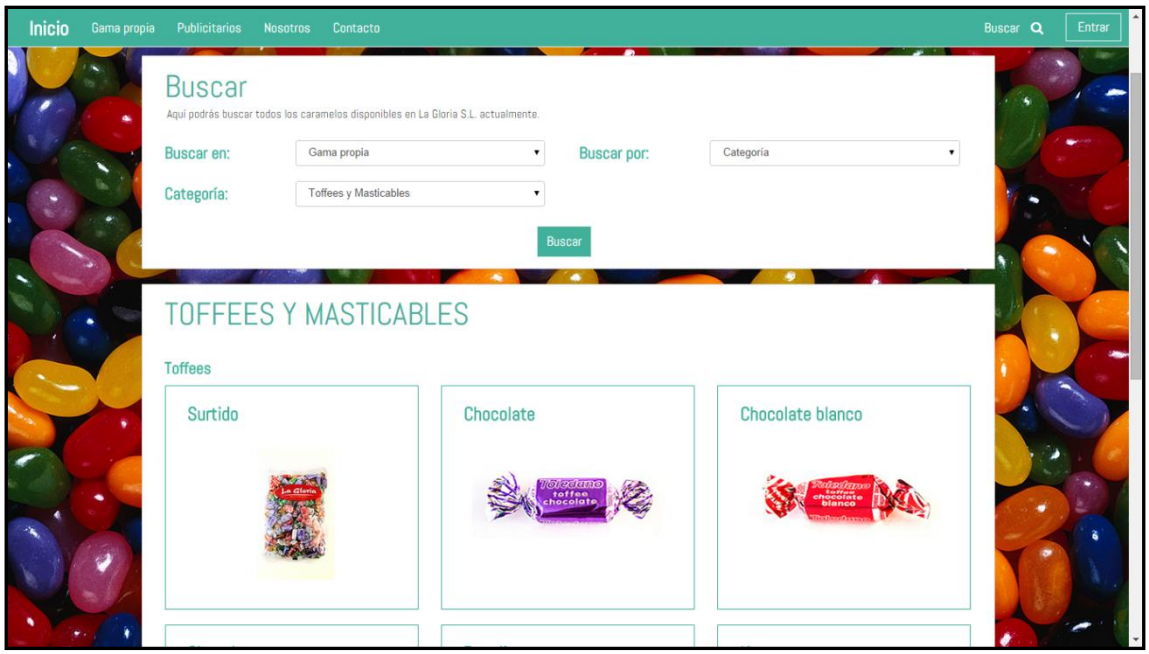



***Ilustración 82:*** *Manual de usuario anónimo - Búsqueda de toffees y masticables*

Si seleccionamos cualquier producto, accederemos a la vista de la información detallada del producto, similar a la mostrada en el apartado anterior.

# Gestión de usuarios

Pasemos ahora con las acciones relacionadas con la gestión de usuarios.

### Registrarse como usuario

Para dejar de ser un usuario anónimo y registrarnos en el sistema tenemos que pulsar el botón ‘Entrar’, situado en la zona derecha de la barra de navegación principal de la aplicación. La pantalla que veremos será ésta:

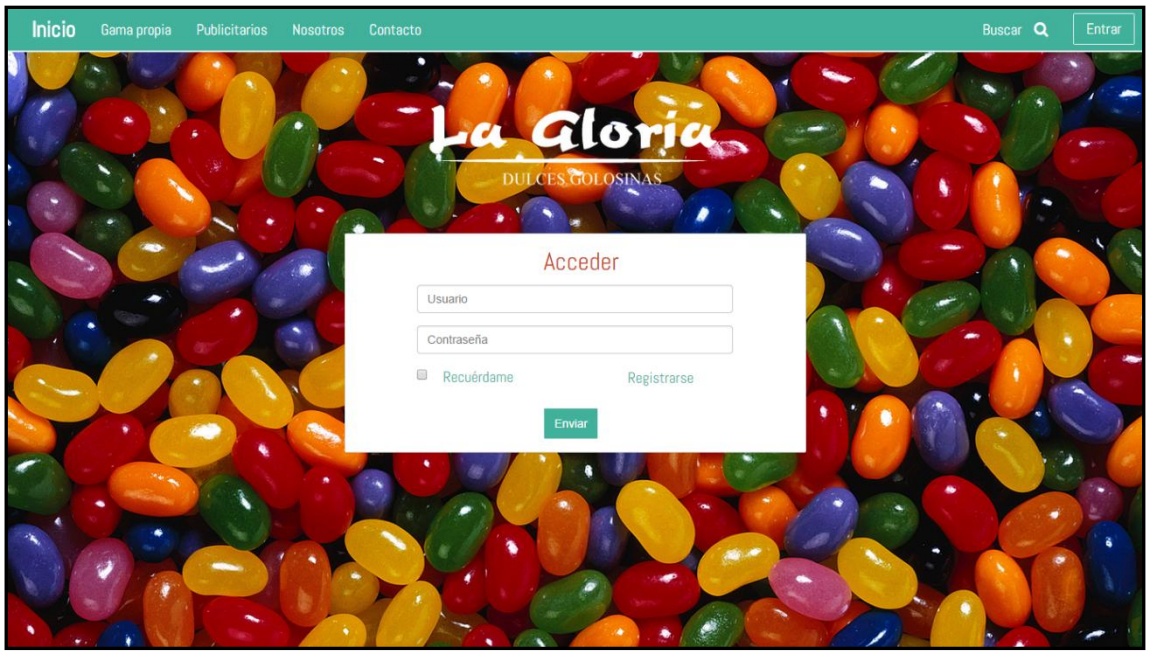



***Ilustración 83:*** *Manual de usuario anónimo - Pantalla de login*

Como aún no tenemos una cuenta vamos a crearla. Para registrarnos tenemos que pulsar en la parte inferior derecha del formulario, donde se encuentra la palabra 'Registrarse', lo cual nos lleva al formulario de registro:

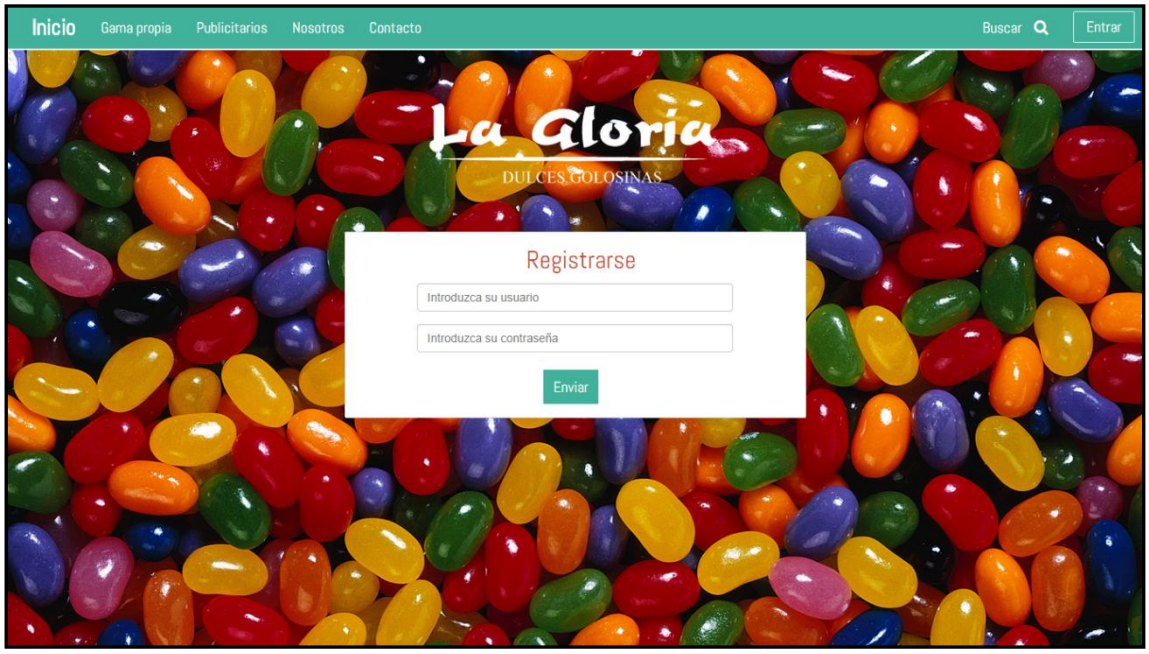



***Ilustración 84:*** *Manual de usuario anónimo - Pantalla de registro*

Rellenamos el formulario, por ejemplo, con los datos ‘usuario’ y ‘usuario’, como usuario y contraseña respectivamente y enviamos la solicitud de registro. Como ese usuario no está en uso, el resultado será satisfactorio:

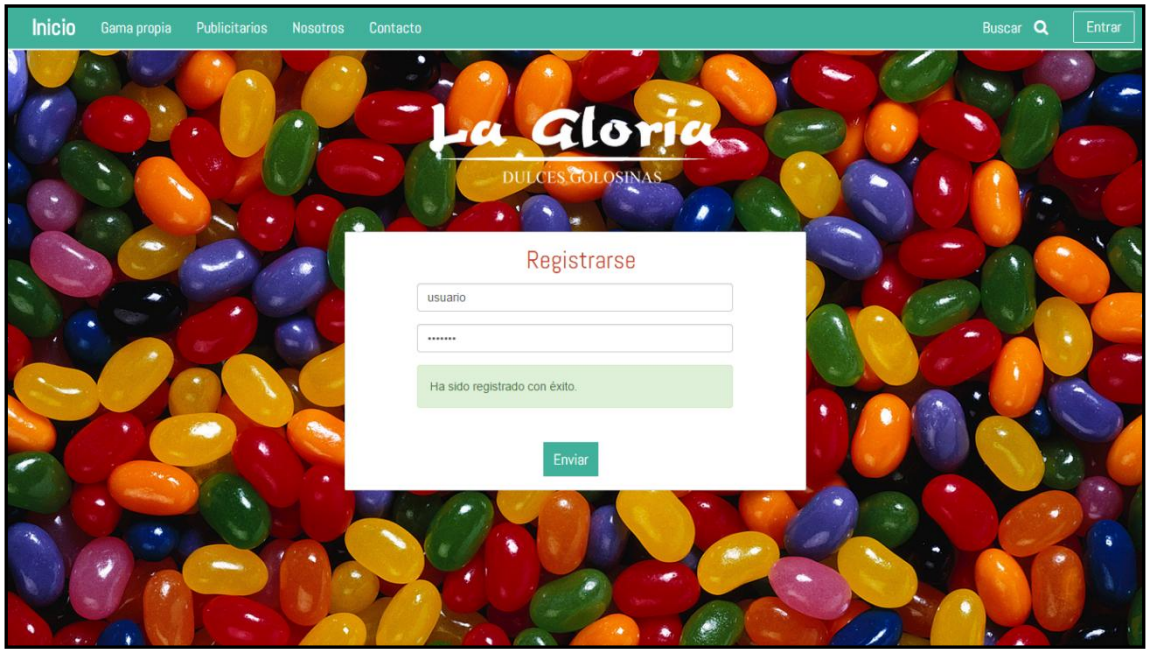



***Ilustración 85:*** *Manual de usuario anónimo - Registro con éxito*

Tras ver esta pantalla durante 3 segundos, la aplicación nos redirige automáticamente a la página de login, la cual se explica en el manual de usuario para proveedor.

# Gestión de emails

Otra forma con la que puede interactuar el usuario anónimo con el sistema es mediante los emails.

### Enviar email

Para enviar un email a **La Gloria S.L.** nos dirigimos a la sección ‘Contacto’, ubicada en la barra de navegación principal de la aplicación. Nuevamente veremos un formulario mediante el que podemos ponernos en contacto con la empresa para cualquier duda o sugerencia:

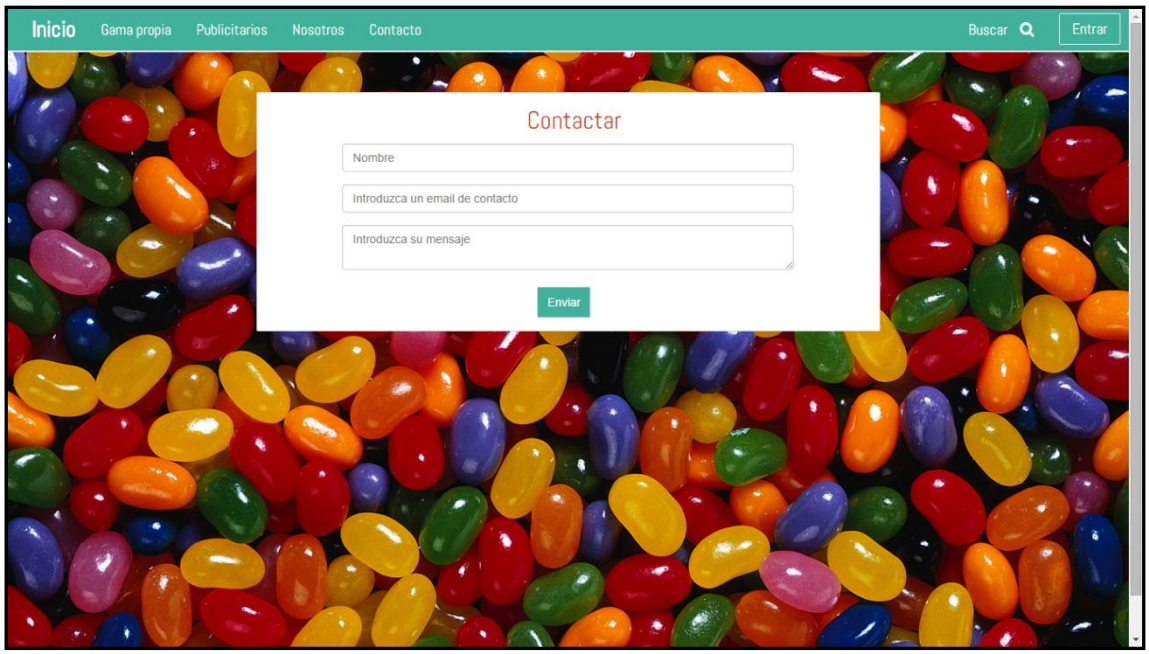



***Ilustración 86:*** *Manual de usuario anónimo - Pantalla de envío de email*

El funcionamiento de esto formulario es idéntico al de registro: primero se validan los datos y, en caso de ser correctos, se envía el email y se notifica al usuario de que ha sido enviado correctamente.

# 12.2. Manual de usuario para proveedor

Ahora es el turno de explicar las acciones que puede realizar un proveedor. Recordemos que un proveedor puede realizar sus acciones propias y las de un usuario anónimo. Así pues, en esta apartado vamos a explicar sólo las acciones características de un proveedor, que son:

- Gestión de contenidos.
  - Comprar producto.
- Gestión de usuarios.
  - Hacer login en el sistema.
  - Hacer logout en el sistema.
- Gestión de aspectos sociales.
  - Comentar producto.
  - Editar comentario realizado.
  - Eliminar comentario realizado de producto.
  - Valorar producto.
- Gestión de pedidos.
  - Realizar pedido.

# Gestión de contenidos

Aquí se detallan las acciones que puede realizar un proveedor respecto a los contenidos de la aplicación.

### Comprar producto

Para comprar un producto primero tenemos que realizar la acción **Ver los detalles de un producto en una categoría**, explicada en el manual de usuario para anónimo. Sin embargo, al visualizar esta vista como proveedor veremos un botón para 'Comprar' el producto:

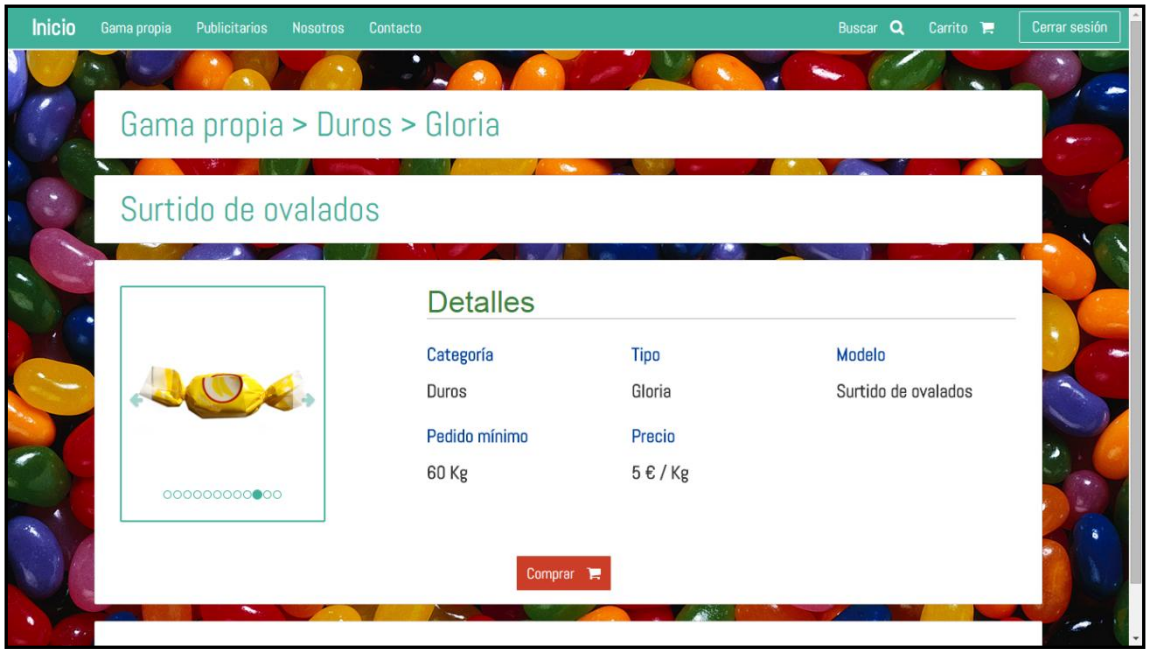



***Ilustración 87:*** *Manual de usuario proveedor - Comprar producto*

Si pulsamos sobre el botón comprar y el producto no estaba ya en el carrito, se añadirá correctamente y se nos notificará:

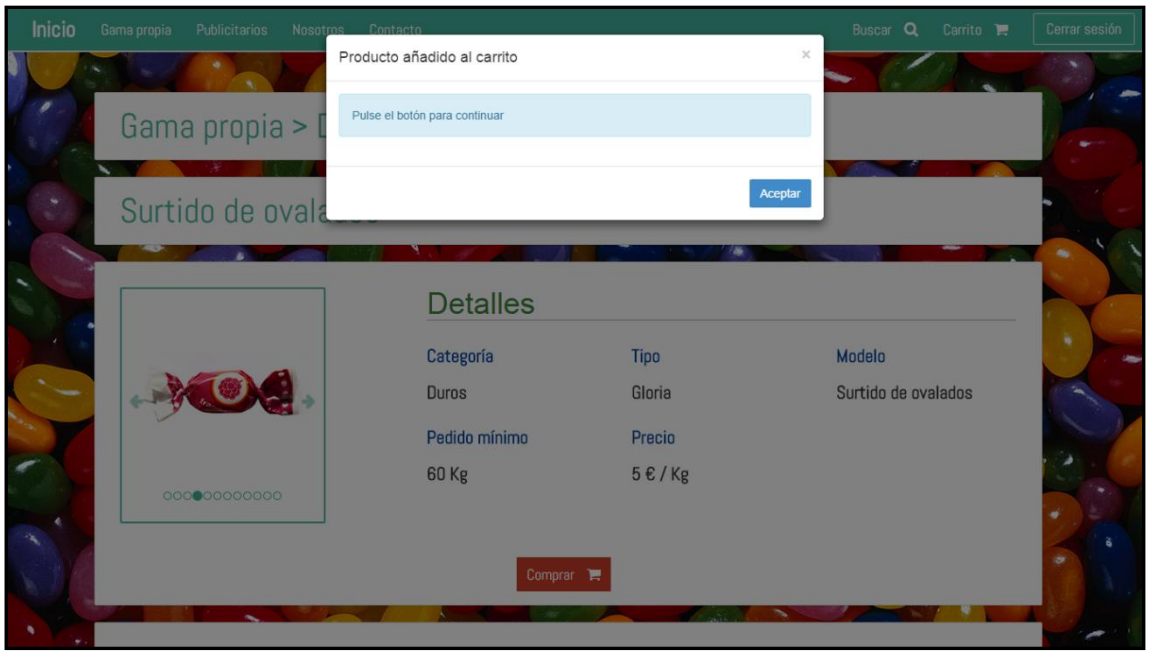



***Ilustración 88:*** *Manual de usuario proveedor - Compra de producto con éxito*

Si queremos ver el producto en el carrito de la compra no tenemos más que seleccionarlo en la barra de navegación principal:

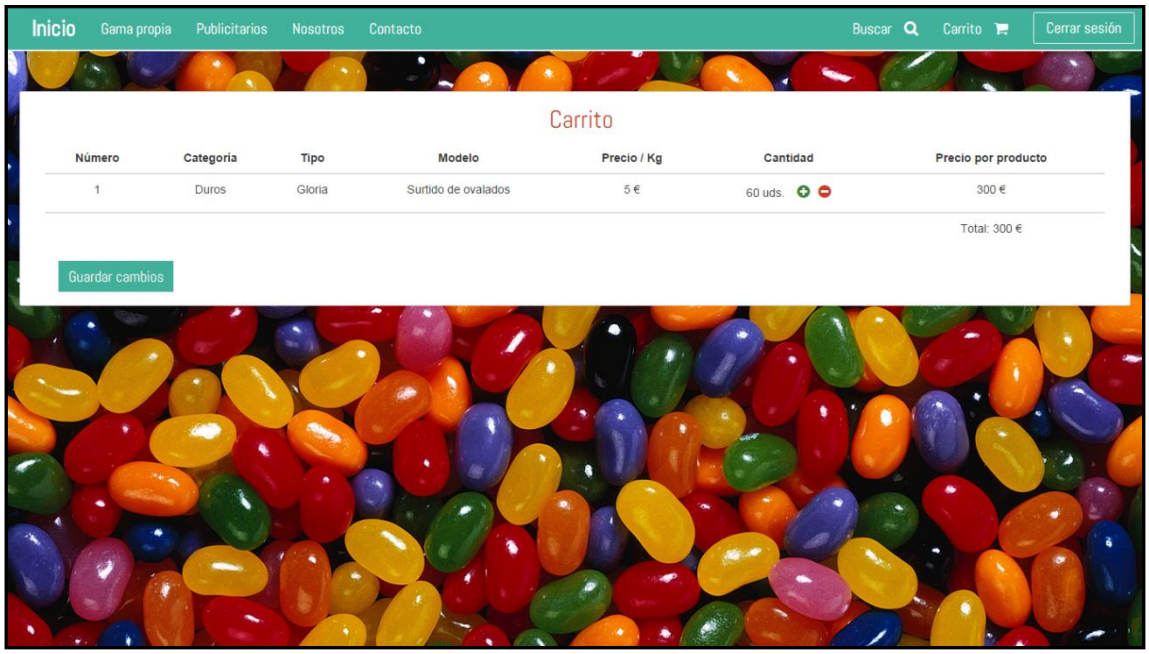



***Ilustración 89:*** *Manual de usuario proveedor - Carrito de la compra*

# Gestión de usuarios

En este apartado vamos a detallar acciones como el login o el logout de la aplicación.

### Hacer login en el sistema

Desde cualquier página de la aplicación web seleccionamos el botón ‘Entrar’, situado a la derecha de la barra principal. Veremos una página bastante similar a la de registro que se detalla en el manual de usuario para anónimo:

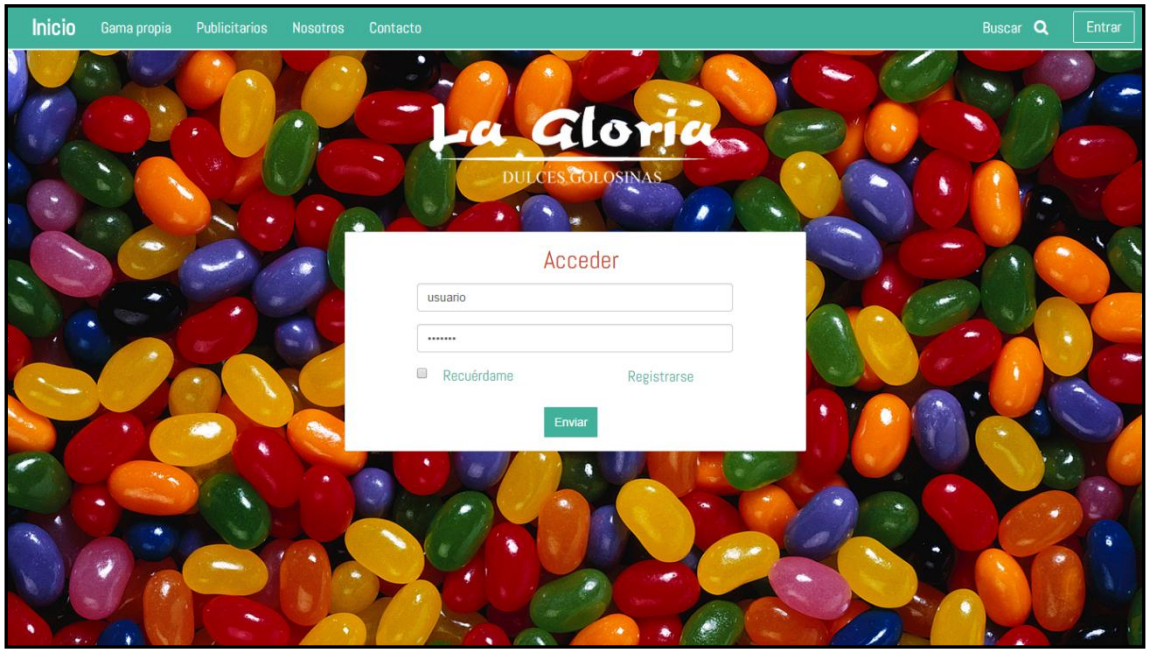



***Ilustración 90:*** *Manual de usuario proveedor - Login en la aplicación*

Rellenamos el formulario con los datos de registro en el manual de usuario para anónimo (“usuario” para los dos campos del formulario) y, si todo está correcto, se nos notificará el inicio de la sesión como proveedor:

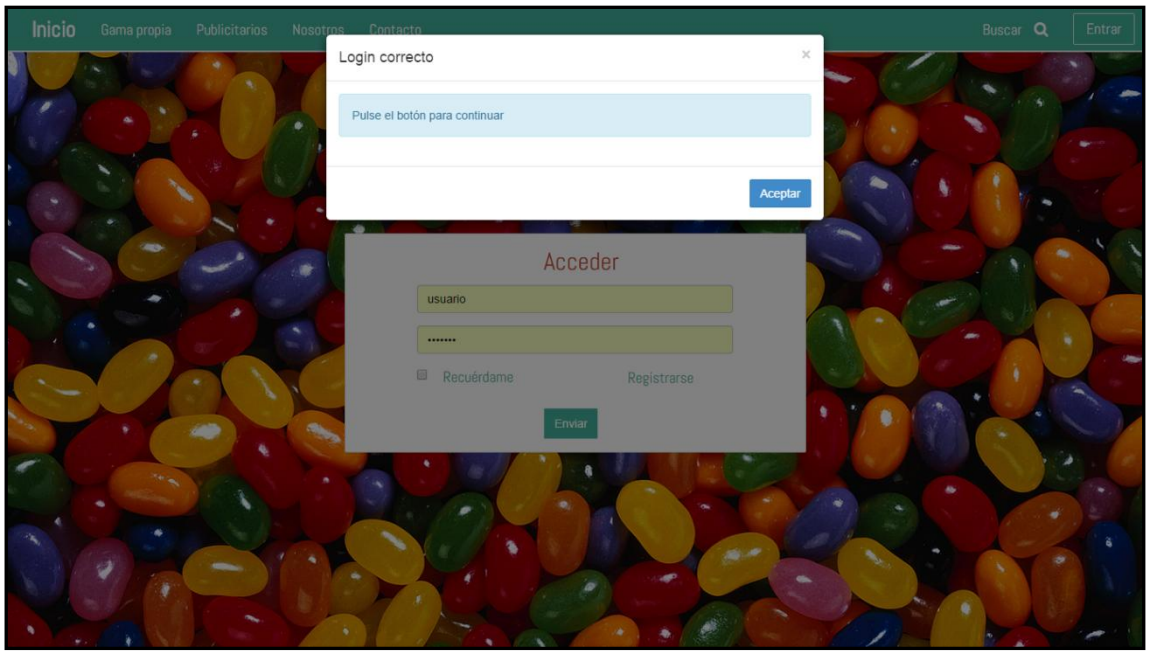



***Ilustración 91:*** *Manual de usuario proveedor - Login con éxito*

### Hacer logout en el sistema

Una vez iniciada la sesión como usuario podremos salir de la aplicación cuando queramos. En esta ocasión, en lugar de aparecer en la barra principal de navegación un botón ‘Entrar’, aparecerá un botón ‘Cerrar sesión’. Haciendo click en el mismo cerraremos la sesión sin problemas.

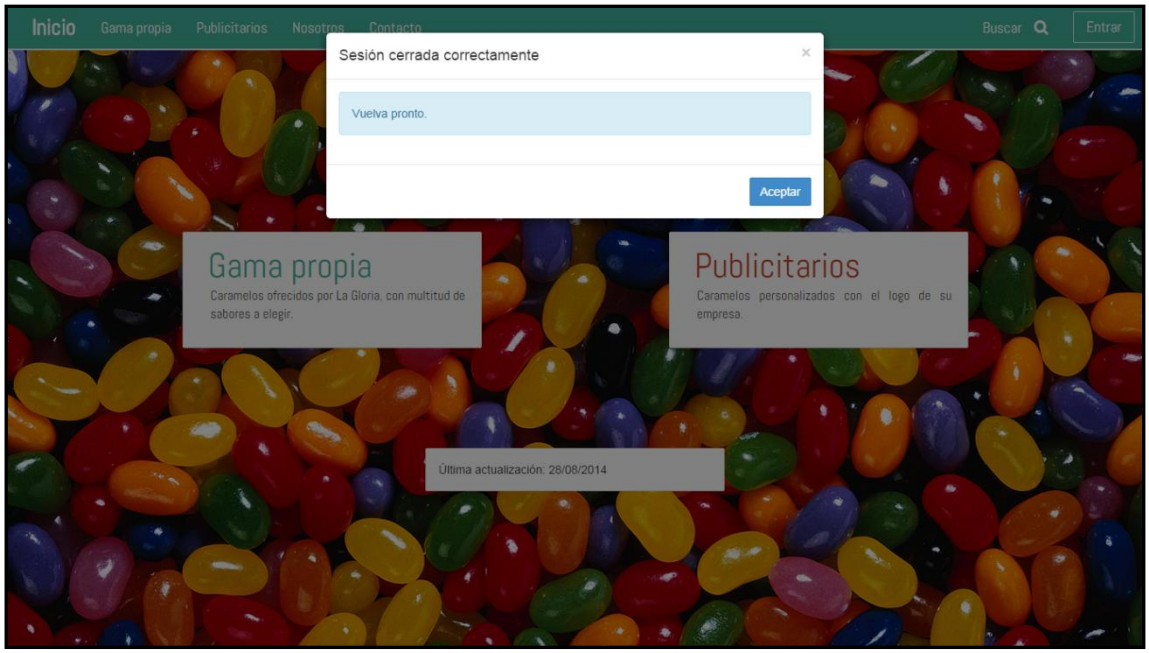



***Ilustración 92:*** *Manual de usuario proveedor - Logout en la aplicación*

# Gestión de aspectos sociales

En este apartado se comentarán todas las acciones relacionadas con aspectos sociales, así como los comentarios o las valoraciones.

### Comentar producto

Para comentar un producto primero hay que realizar la acción **Ver los detalles de un producto en una categoría**, explicada en el Manual de usuario para anónimo.

Al final de la página veremos el panel con los comentarios realizados, además de un text área para realizar un nuevo comentario.

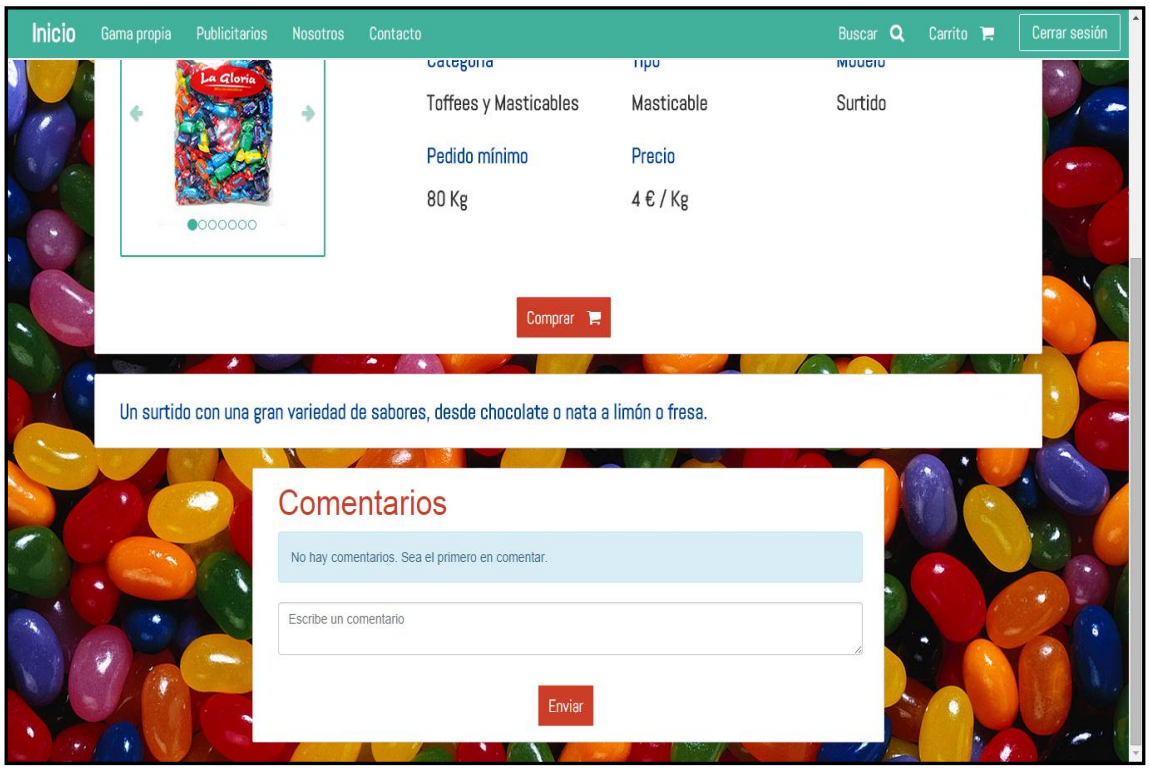



***Ilustración 93:*** *Manual de usuario proveedor - Comentarios de un producto*

Aún no se han realizado comentarios, pero podemos comentar el producto para dejar constancia de alguna opinión o información en general:

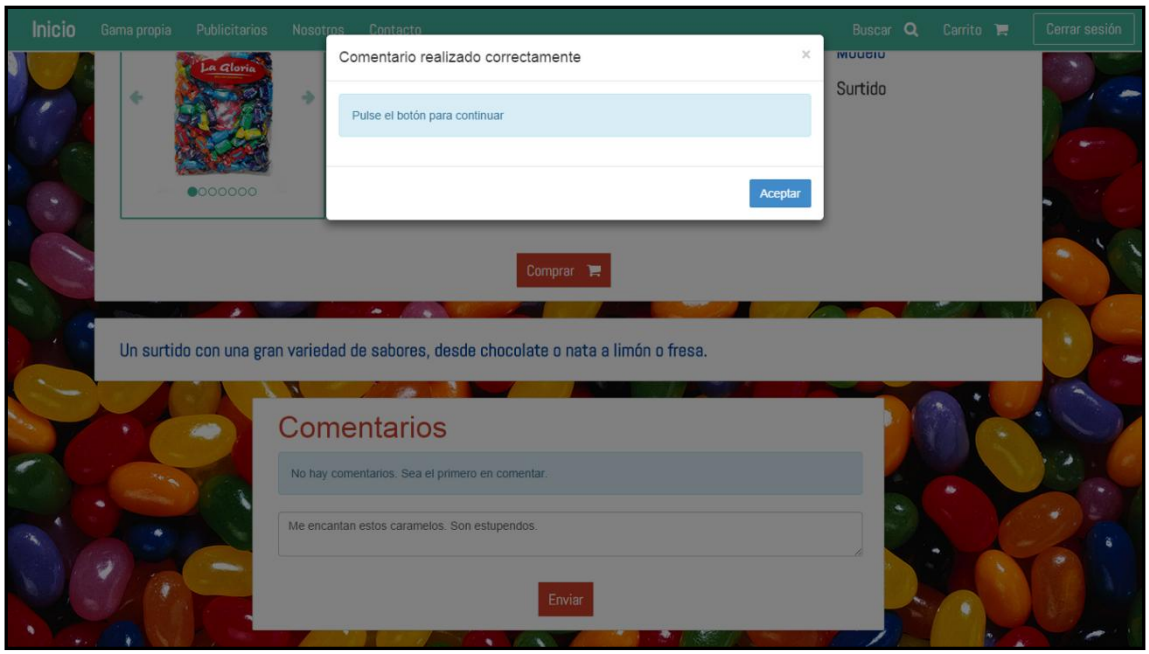



***Ilustración 94:*** *Manual de usuario proveedor - Comentario realizado*

Si pulsamos el botón aceptar, veremos que el comentario aparece de inmediato en el panel:

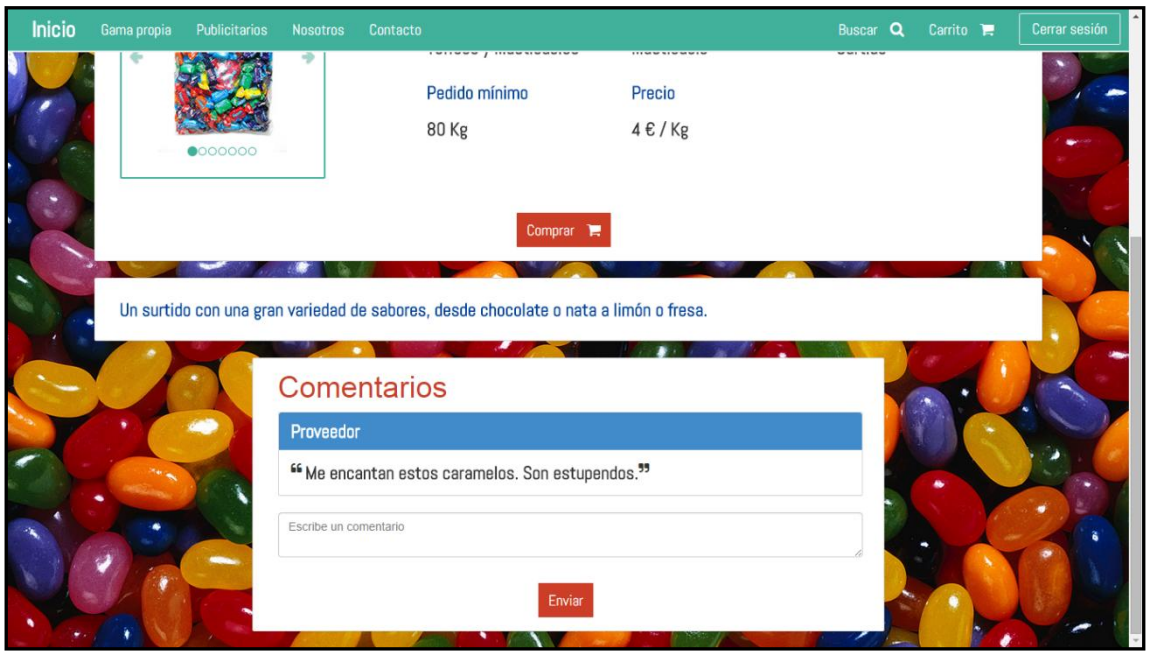



***Ilustración 95:*** *Manual de usuario proveedor - Visualización del comentario realizado*

### Editar comentario realizado

Una vez que hemos realizado un comentario, podemos editarlo posteriormente:

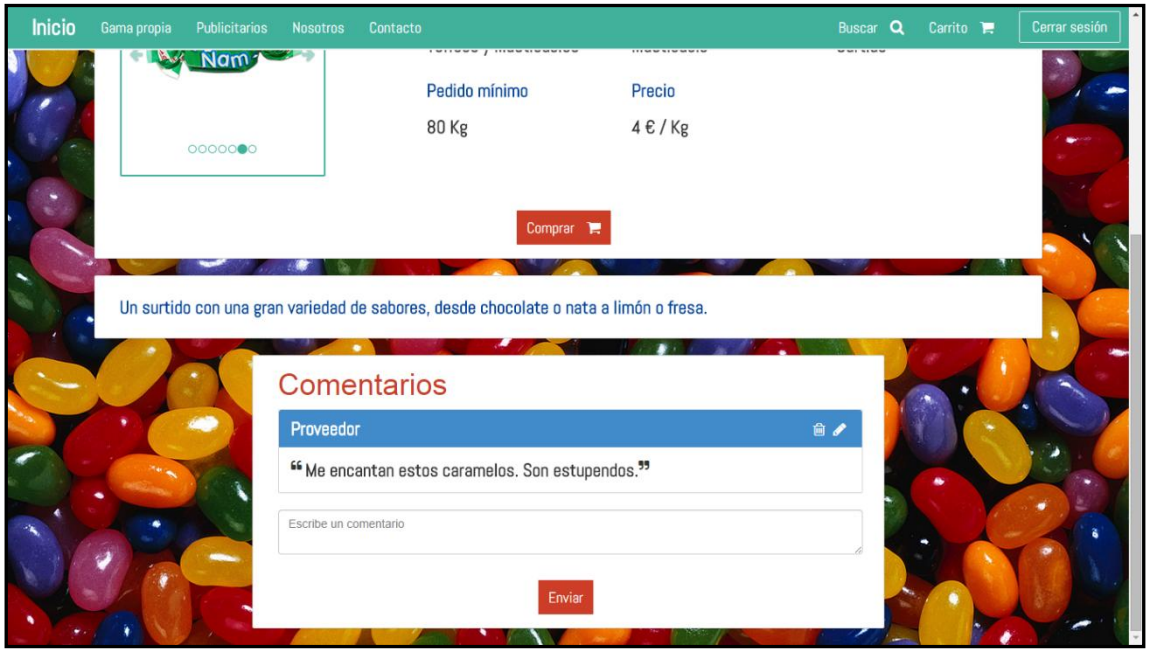



***Ilustración 96:*** *Manual de usuario proveedor - Editar comentario*

Si hacemos click sobre el icono del lápiz podremos editar el comentario directamente. Cuando confirmemos la edición volverá a aparecer una notificación avisando del cambio:

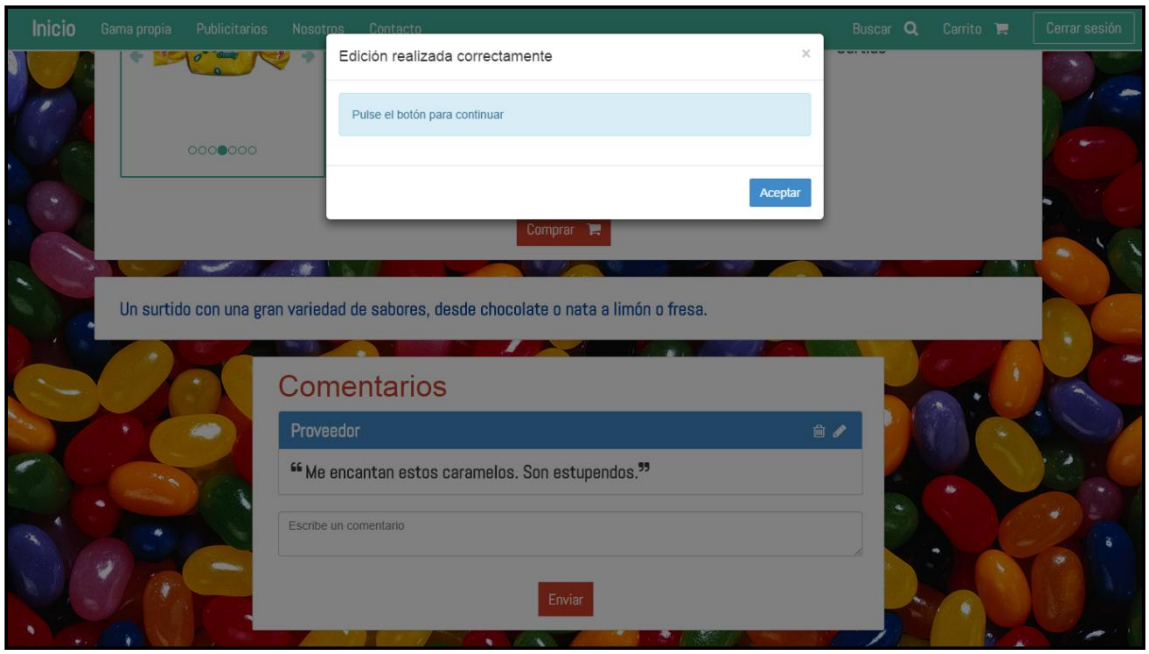



**Ilustración 97:** Manual de usuario proveedor - Comentario editado correctamente

El resultado es el que se puede ver a continuación:

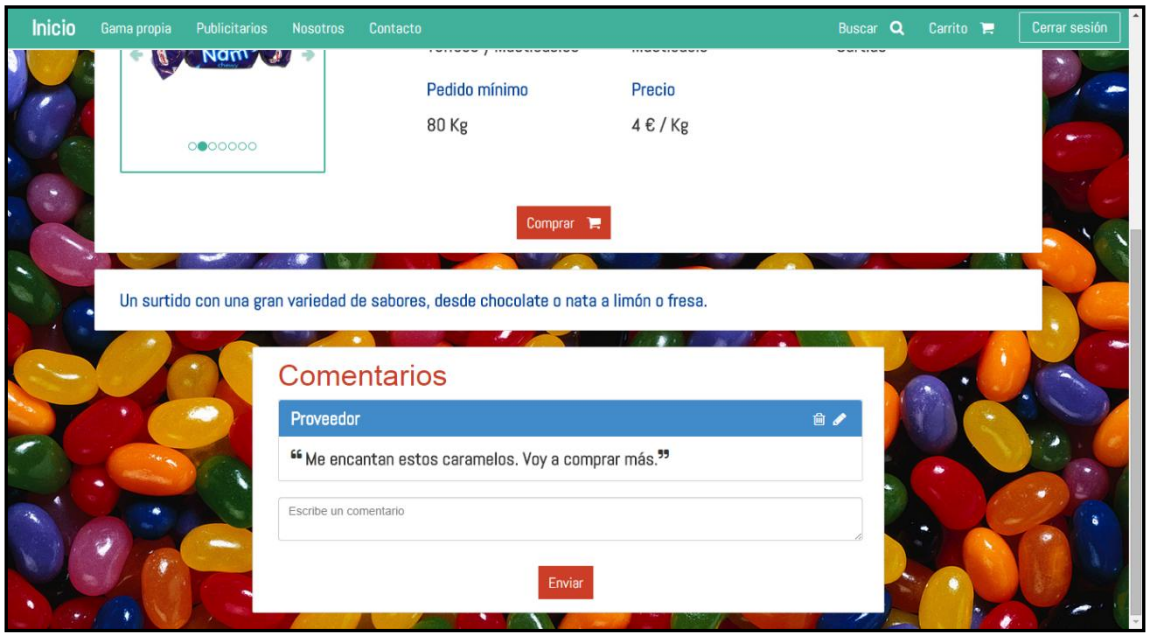



***Ilustración 98:*** *Manual de usuario proveedor - Comentario editado*

### Eliminar comentario realizado de producto

Para eliminar un comentario hay que realizar la misma secuencia de pasos que en la edición. La única diferencia radica en el icono sobre el que pulsamos. En esta ocasión tenemos que pulsar sobre el icono de la papelera, quedando el siguiente resultado:

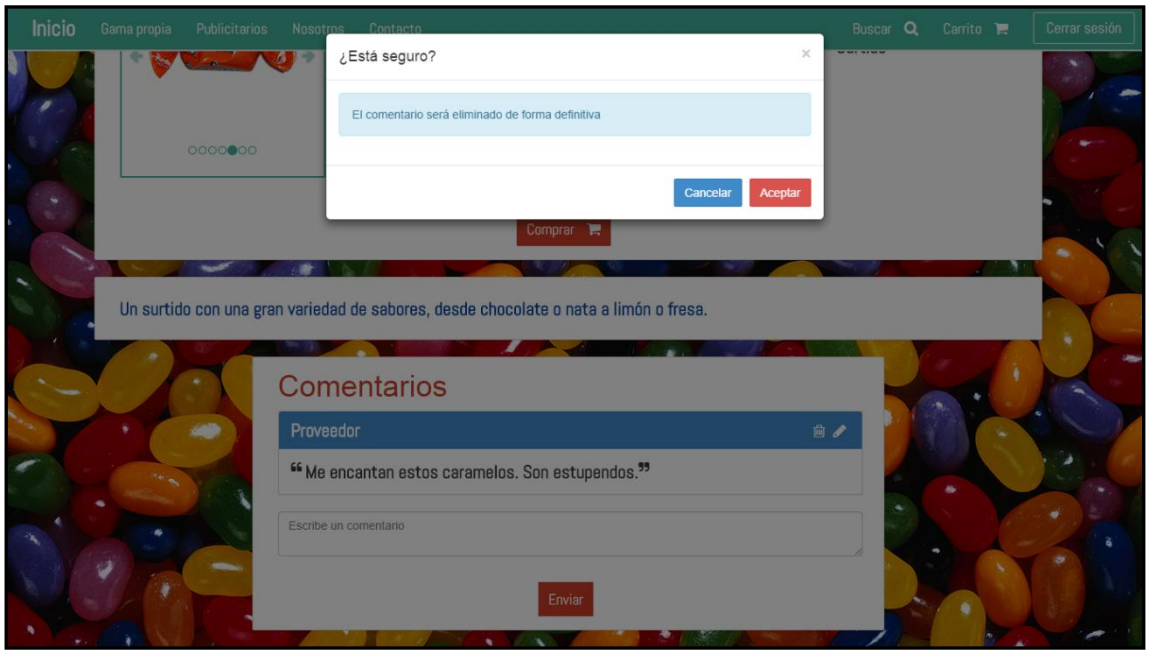



**Ilustración 99:** Manual de usuario proveedor - Eliminar comentario

### Valorar producto

Para valorar un producto no tenemos más que pulsar sobre uno de los dos iconos de las manos. Si nos gusta el producto, no tenemos más que pulsar sobre la mano hacia arriba situada bajo el carousel de imágenes:

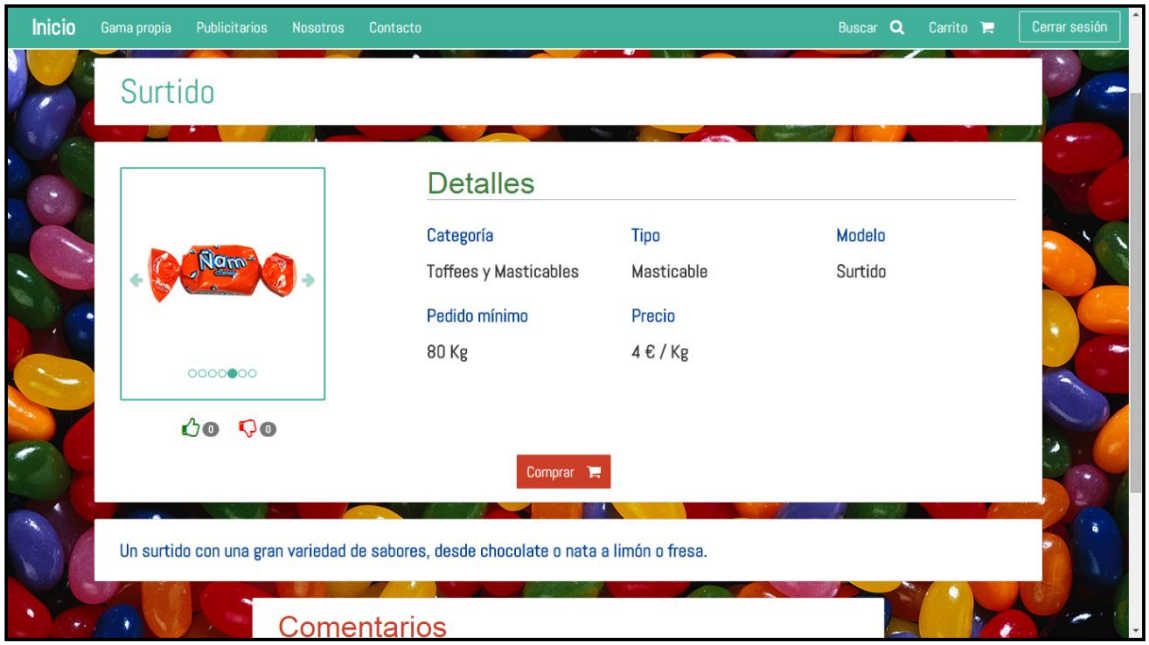



***Ilustración 100:*** *Manual de usuario proveedor - Valorar producto*

El resultado se puede comprar de inmediato:

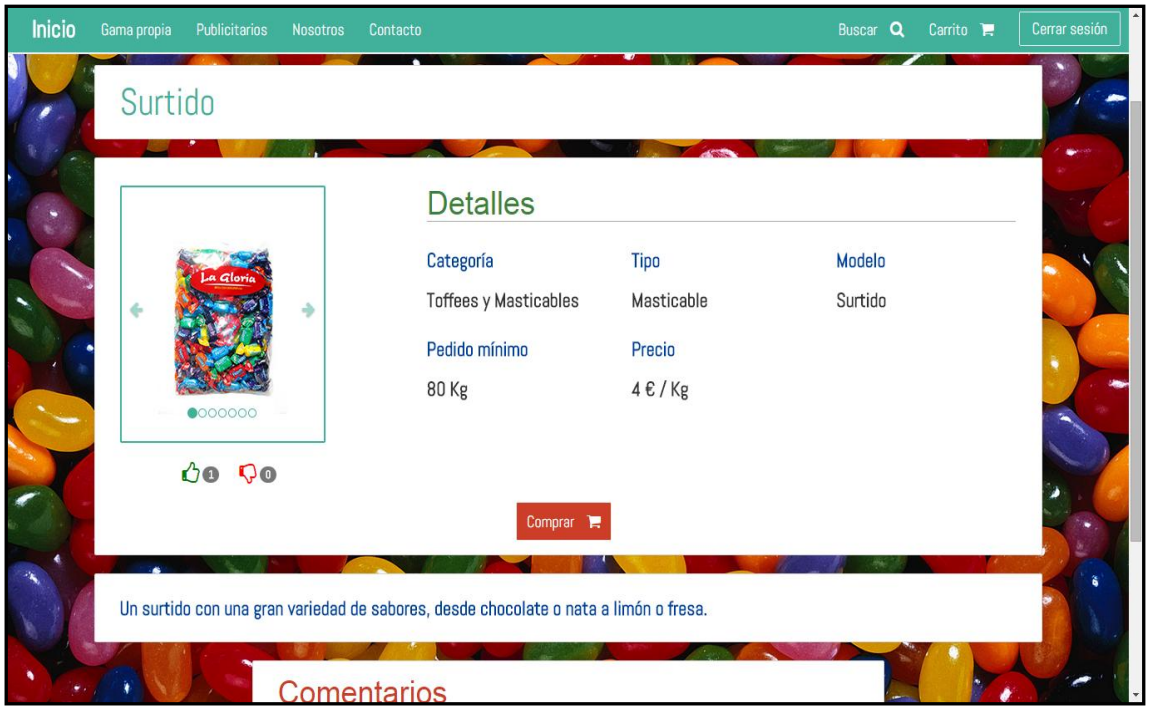



**Ilustración 101:** Manual de usuario proveedor - Producto valorado

# Gestión de pedidos

Para finalizar con el manual para un usuario proveedor se explica cómo realizar un pedido.

### Realizar un pedido

Para realizar un pedido primero hay que realizar la acción **Comprar un producto** con cuantos productos deseemos encargar. Una vez realizada vamos al carrito de la compra y seleccionamos 'realizar pedido':

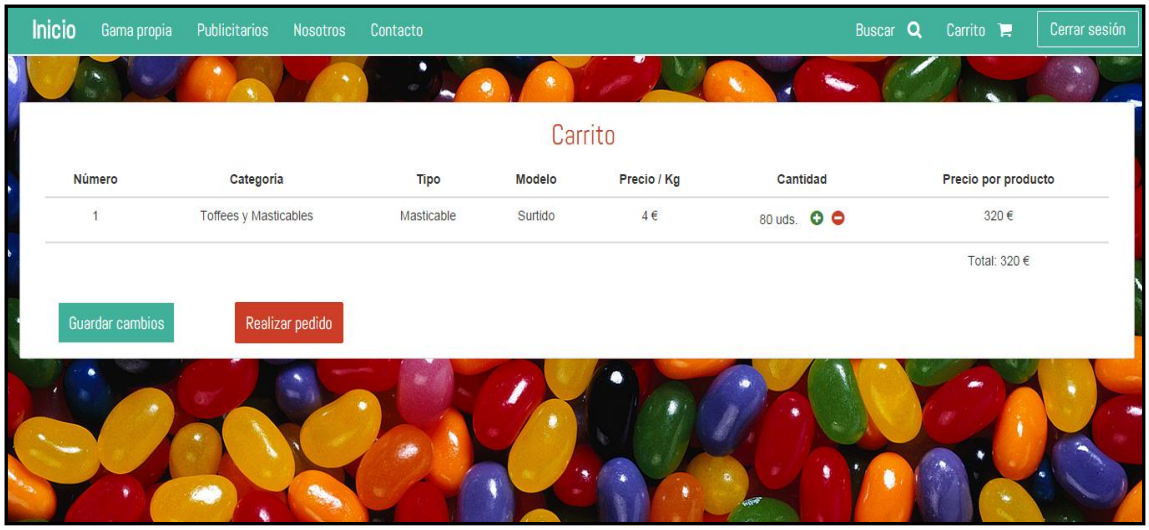



***Ilustración 102:*** *Manual de usuario proveedor - Realizar pedido*

Si lo pulsamos veremos lo siguiente:

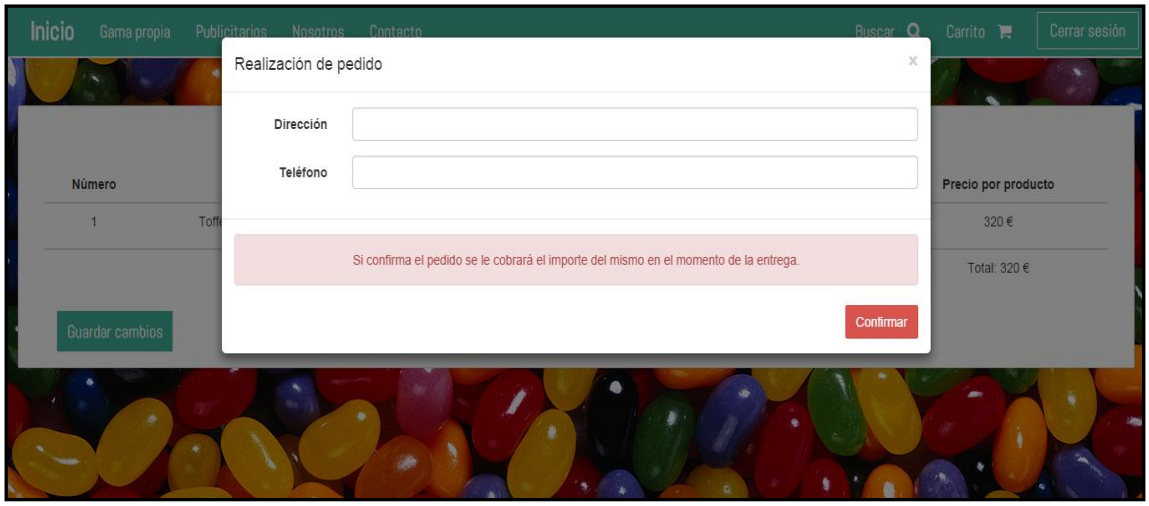



***Ilustración 103:*** *Manual de usuario proveedor - Confirmar pedido*

Para confirmar el pedido no tenemos más que cumplimentar el formulario y pulsar sobre ‘Confirmar’, creándose así nuestro pedido.

# 12.3. Manual de usuario para administrador

Por último tenemos el manual de usuario para un administrador. Las vistas del administrador son únicas y a las que sólo él tiene acceso. Un administrador puede realizar todas las acciones desde un panel de administración. Dichas acciones se detallan a continuación:

- Gestión de contenidos.
  - Editar producto.
  - Eliminar producto.
- Gestión de usuarios.
  - Listar usuarios registrados en el sistema.
  - Editar usuario registrado en el sistema.
  - Eliminar usuario registrado en el sistema.
- Gestión de aspectos sociales.
  - Listar comentarios de un producto.
  - Eliminar comentario de un producto.
- Gestión de pedidos.
  - Ver los pedidos realizados.
  - Editar pedido.
  - Eliminar un pedido realizado.
- Gestión de emails.
  - Ver los emails recibidos.
  - Eliminar email.

# Gestión de contenidos

En primer lugar se explican las acciones que se pueden realizar en relación con los contenidos de la aplicación web.

### Editar producto

Para editar un producto tenemos que seleccionar la categoría a la que pertenece el producto a editar, la cual está ubicada en el panel vertical del panel de administración. De esta forma, veremos todos los productos de esa categoría. Si seleccionamos la categoría 'Toffees y Masticables' veremos la siguiente pantalla:

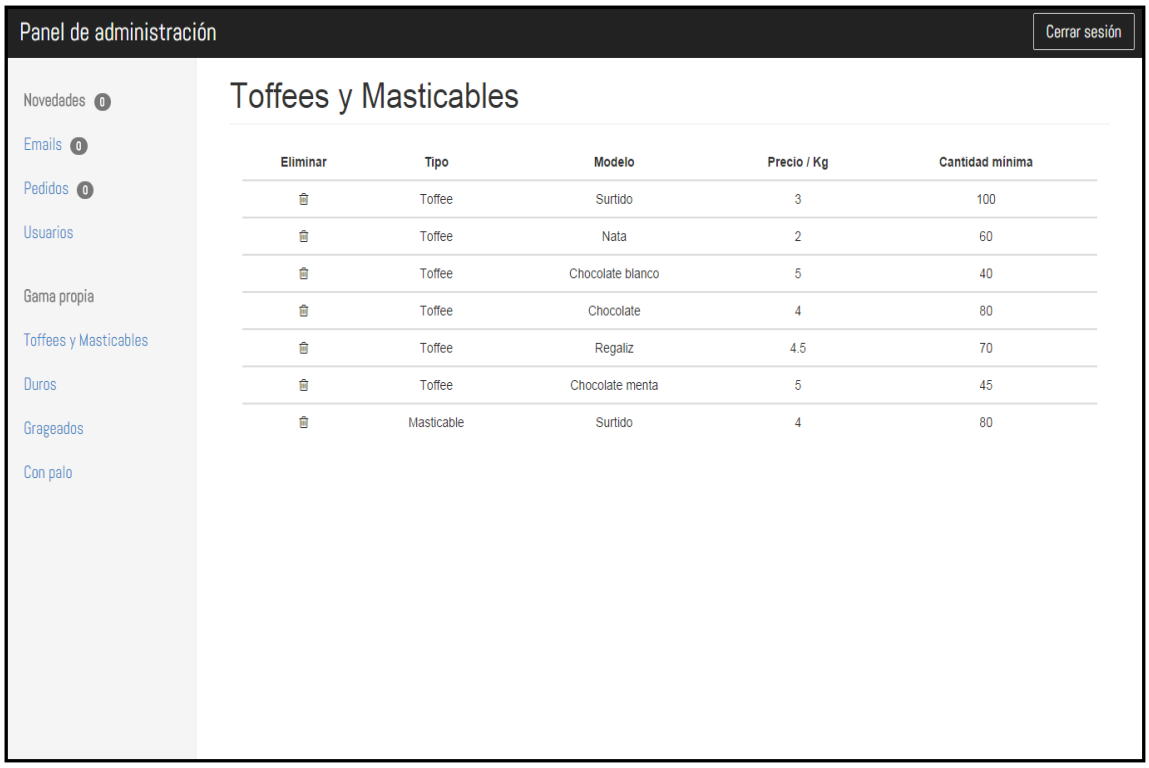



***Ilustración 104:*** *Manual de usuario administrador - Toffees y Masticables*

Al igual que en muchas acciones del administrador, debemos seleccionar un producto para editar la información del mismo:

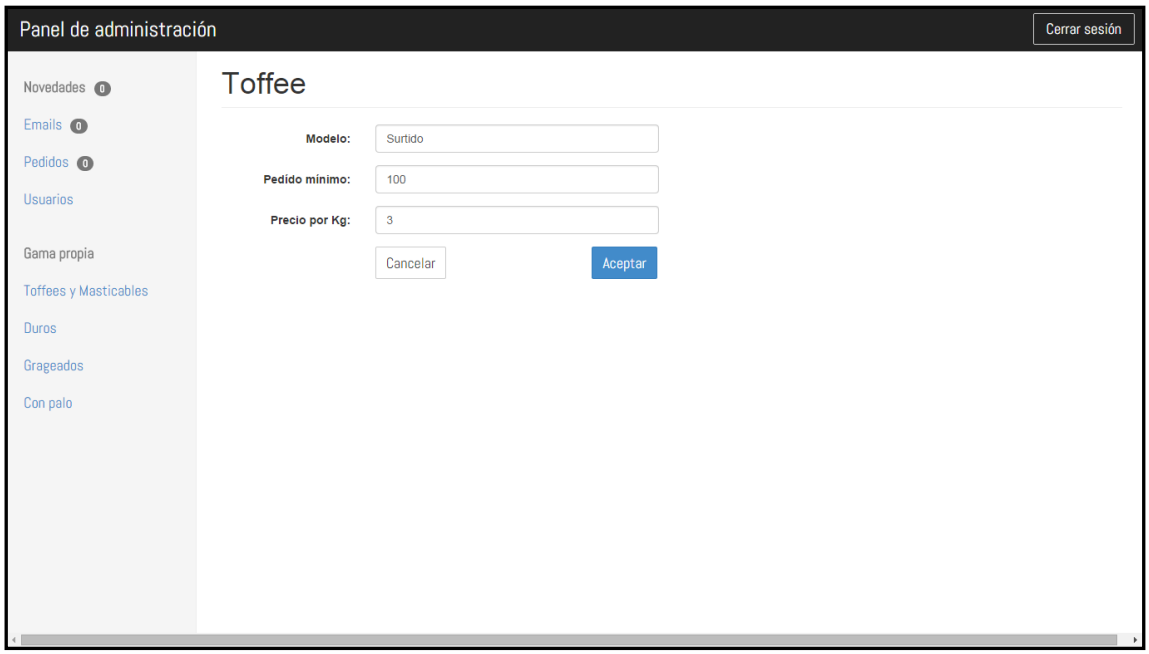



***Ilustración 105:*** *Manual de usuario administrador - Editar producto*

## Eliminar producto

Para eliminar un producto debemos realizar el primer paso de la acción **Editar producto**. Una vez que visualicemos todos los productos no tendremos más que hacer click sobre el icono de la papelera. Borremos, por ejemplo, el surtido de masticables. Al igual que en otras ocasiones, se nos preguntara por el borrado del producto:

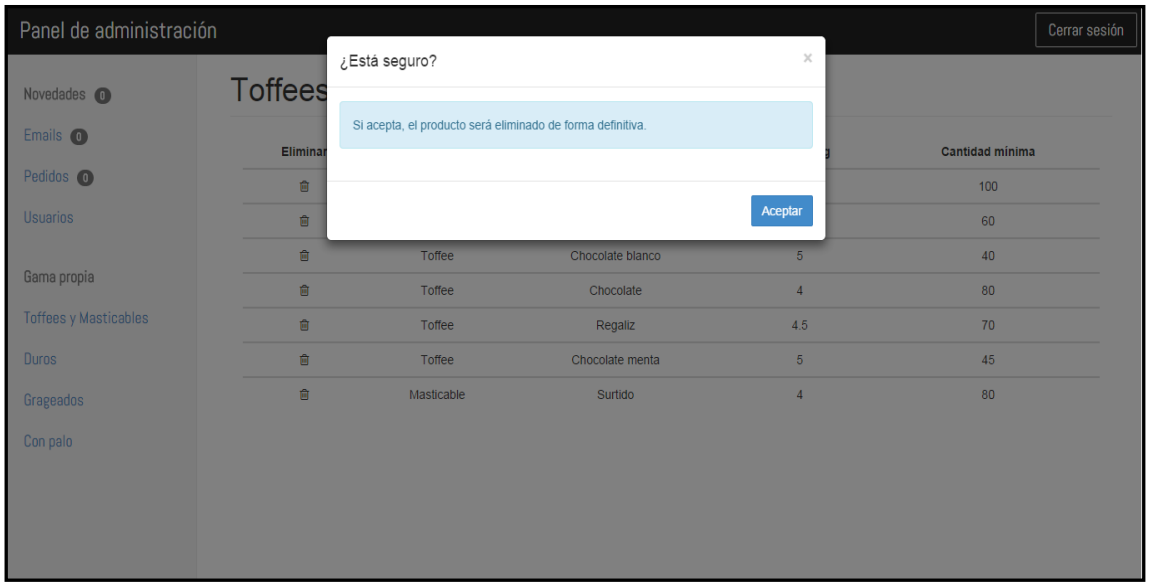



***Ilustración 106:*** *Manual de usuario administrador - Eliminar producto*

Si aceptamos, veremos que el surtido de masticables ya no se encuentra en el sistema:

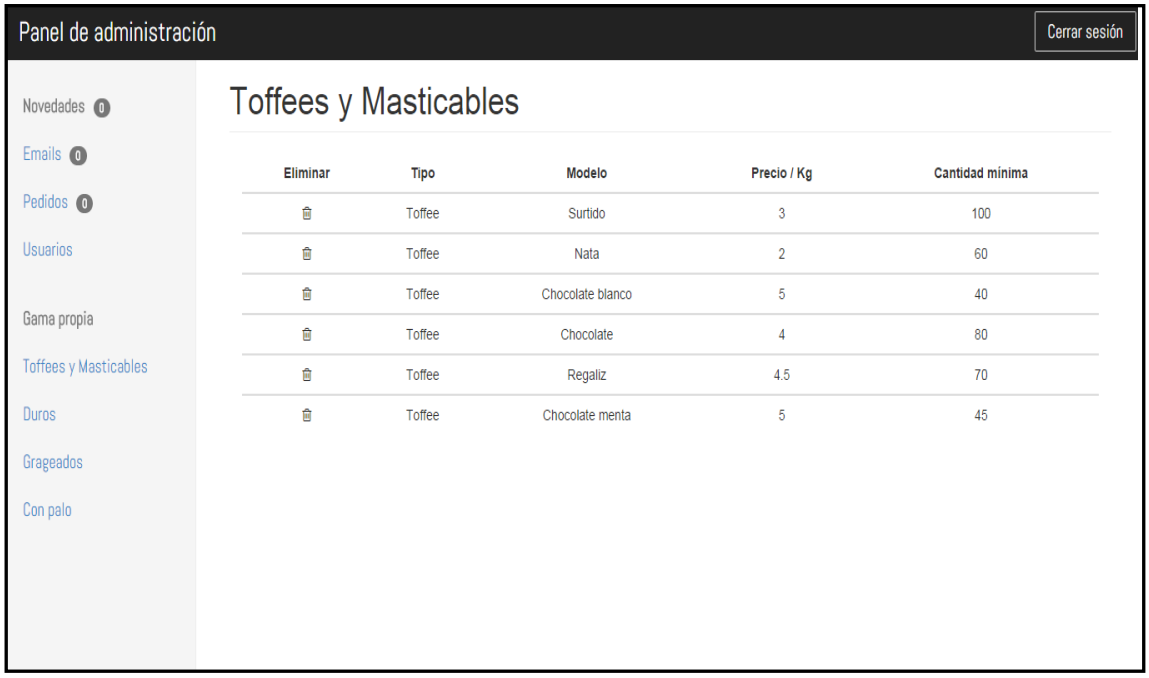



***Ilustración 107:*** *Manual de usuario administrador - Surtido eliminado*

# Gestión de usuarios

También se pueden realizar ciertas acciones en lo que a gestión de usuarios se refieres, las cuales se detallan a continuación.

### Listar usuarios registrados en el sistema

Para listar los usuarios del sistema sólo tenemos que hacer click en Usuarios' en el panel vertical del panel de administración, lo cual nos lleva a la siguiente vista:

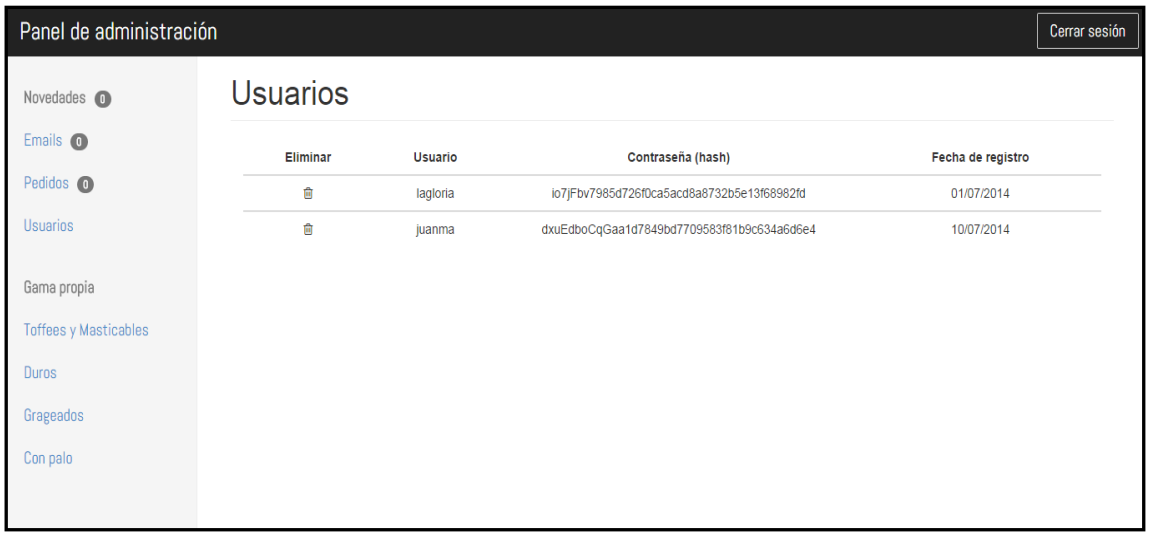



***Ilustración 108:*** *Manual de usuario administrador - Usuarios en el sistema*

En esta pantalla se pueden ver todos los usuarios registrados en el sistema, así como toda la información asociada a los mismos.

### Editar usuario registrado en el sistema

Para editar un usuario debemos realizar la acción **Listar usuarios registrados en el sistema** y seleccionar un usuario:

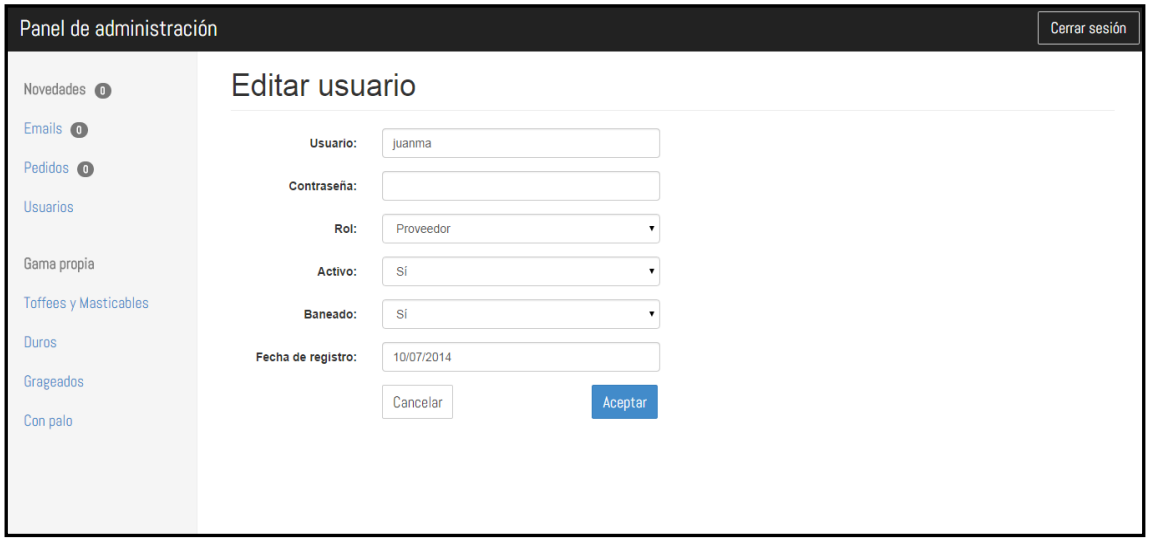



***Ilustración 109:*** *Manual de usuario administrador - Editar usuario*

Una vez realizadas las modificaciones necesarias sólo tenemos que pulsar el botón ‘Aceptar’.

### Eliminar usuario registrado en el sistema

Para eliminar un usuario debemos realizar la acción **Listar usuarios registrados en el sistema**. En dicha pantalla no tenemos más que hacer click en el icono de la papelera que esté ubicado en la fila del usuario a eliminar. Haciendo esto, aparecerá un aviso advirtiéndonos:

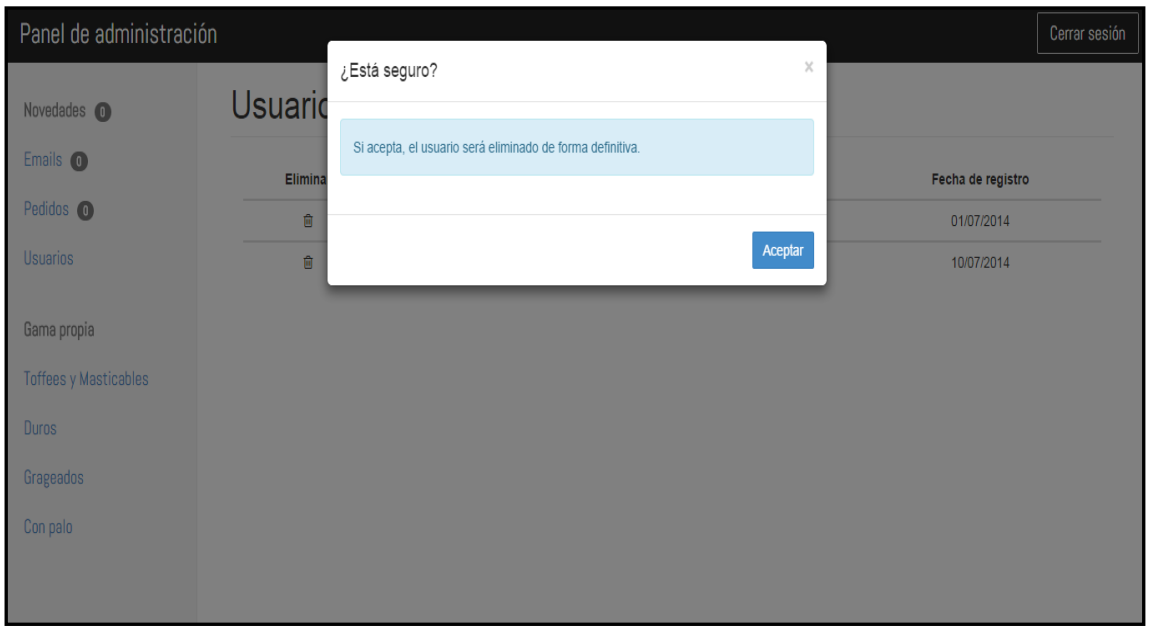



***Ilustración 110:*** *Manual de usuario administrador - Eliminar usuario*

Si aceptamos, veremos de inmediato que el usuario en cuestión ha desaparecido de la vista general de usuarios:

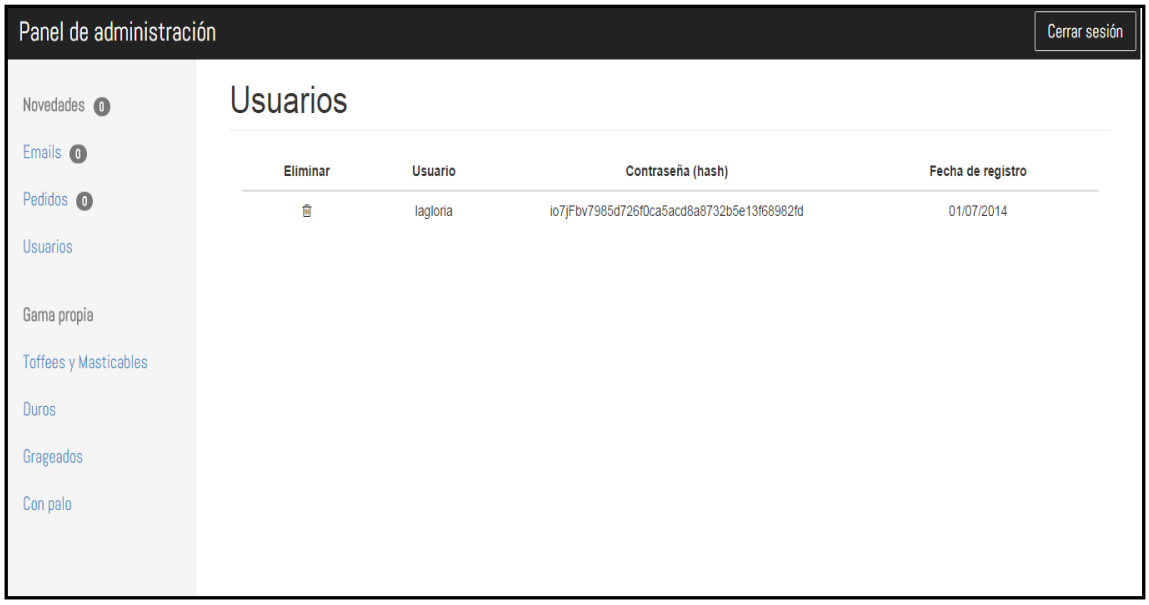



***Ilustración 111:*** *Manual de usuario administrador - Usuario borrado*

# Gestión de aspectos sociales

Ahora es turno de los aspectos sociales.

### Listar comentarios de un producto

Para ver los comentarios debemos realizar los primeros pasos de la acción **Editar producto**. Una vez que visualizamos todos los productos, hacemos click sobre el icono de los comentarios. Así pues, veremos los comentarios realizados sobre el producto:

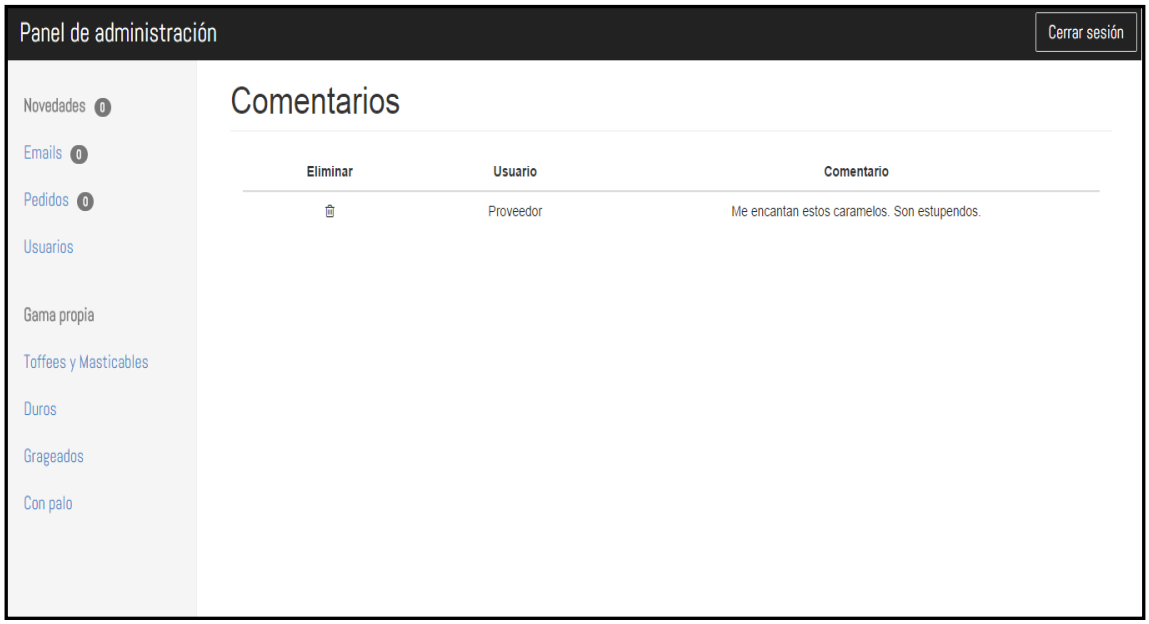



***Ilustración 112:*** *Manual de usuario administrador - Comentarios de un producto*

### Eliminar comentario de un producto

Para eliminar un comentario de un producto tenemos que realizar la acción anterior: **Listar comentarios de un producto**.

Cuando estemos en la pantalla en la que se ven todos los comentarios no tenemos más que hacer click sobre el icono del comentario a borrar. Una ocasión más, se nos advertirá de que borrar un contenido del sistema, en este caso un comentario, será de forma definitiva:

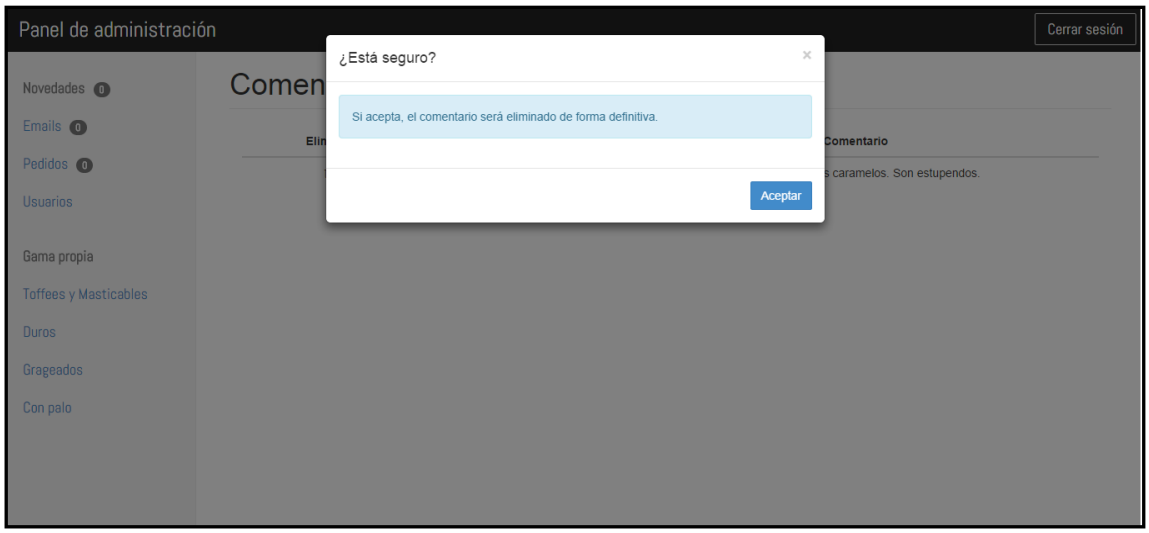



***Ilustración 113:*** *Manual de usuario administrador - Eliminar comentario*

Al igual que en otras acciones, el elemento seleccionado queda borrado del sistema.

# Gestión de pedidos

Pasemos ahora con lo que a gestión de pedidos se refiere.

### Ver los pedidos realizados

Para ver los pedidos realizados tenemos que seleccionar ‘Pedidos’ en el panel de administración:

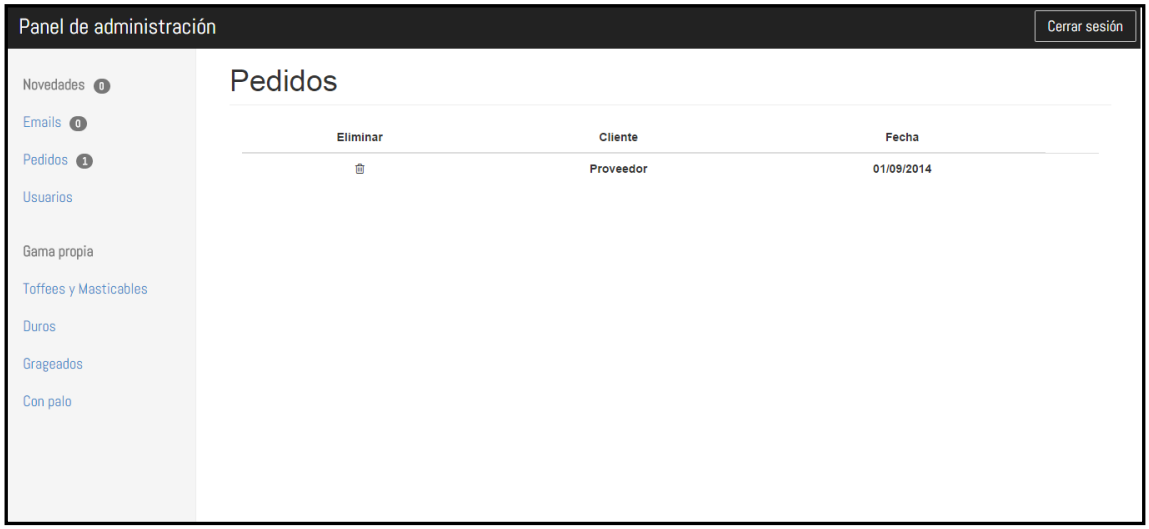



***Ilustración 114:*** *Manual de usuario administrador - Pedidos realizados*

### Editar pedido

Para editar un pedido tenemos que realizar la acción **Ver los pedidos realizados**. Una vez hecho seleccionamos el pedido que queramos editar y aparecerá una pantalla similar a esta:

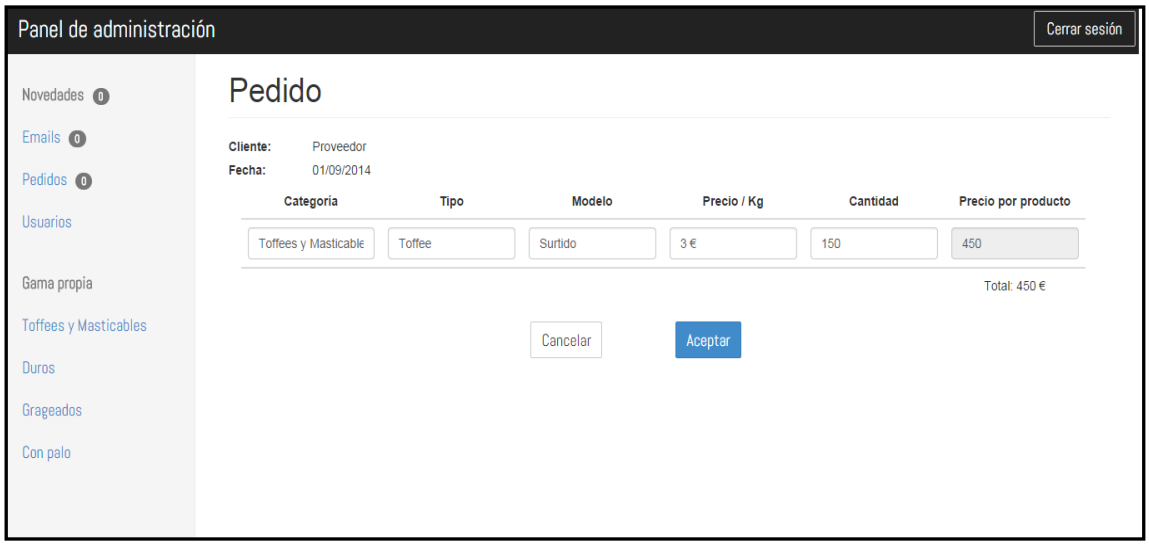



***Ilustración 115:*** *Manual de usuario administrador - Editar pedido*

Ésta es sin duda la vista de edición dentro del panel de administración más personal, ya que se pueden editar los campos directamente, manteniendo la estructura que se definió para el carrito de la compra.

### Eliminar un pedido realizado

Para eliminar un pedido primero tenemos que realizar la acción **Ver los pedidos realizados** hasta llegar a la vista general de pedidos. A continuación hacemos click en el icono de la papelera en la fila del pedido que queremos eliminar.

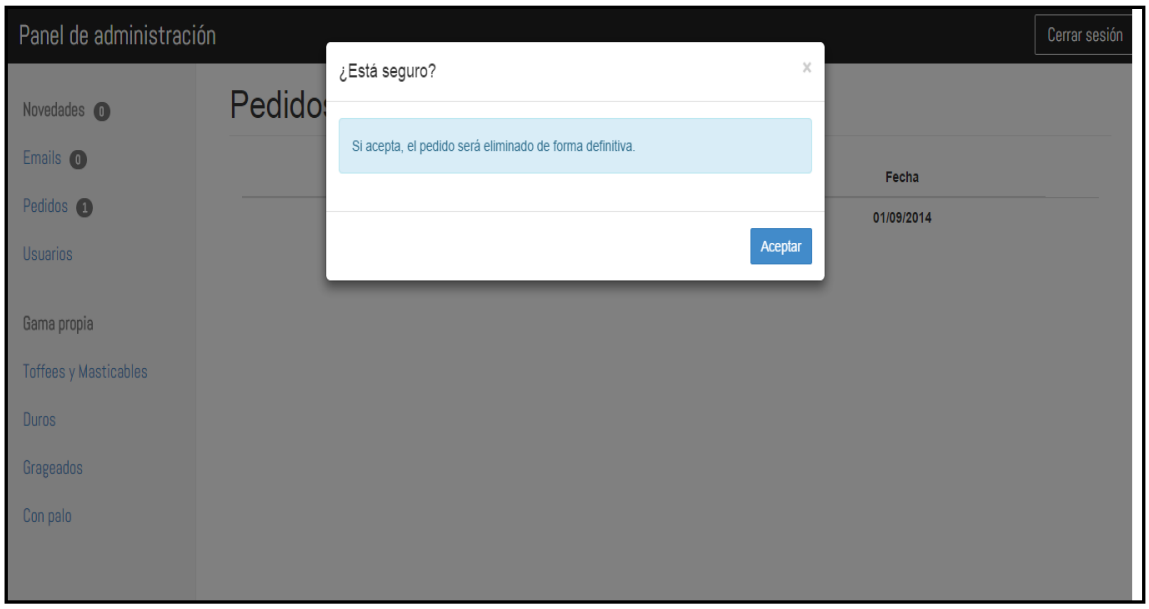



***Ilustración 116:*** *Manual de usuario administrador - Eliminar pedido*

Nuevamente, se nos pregunta si queremos eliminar definitivamente el elemento, en este caso un pedido.

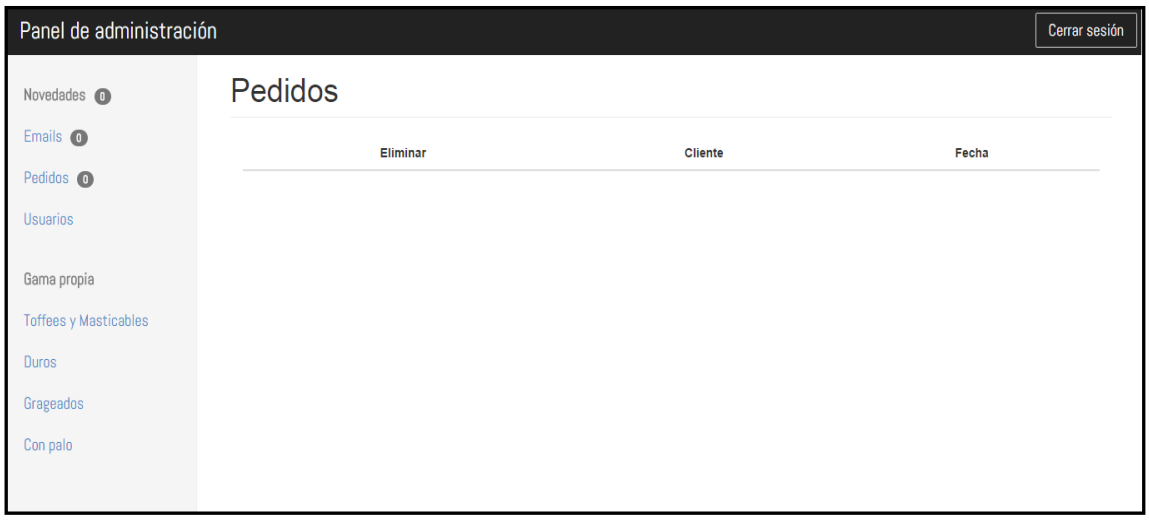



***Ilustración 117:*** *Manual de usuario administrador - Pedido borrado*

Podemos comprobar que ha sido borrado del sistema.

# Gestión de emails

En este apartado explicaremos qué puede realizar un administrador con los emails que son enviados por los usuarios.

### Ver los emails recibidos

Desde el panel de administración seleccionamos el enlace situado a la izquierda denominado ‘Emails’, con lo cual llegaremos a la pantalla en la que se visualizan los emails recibidos.

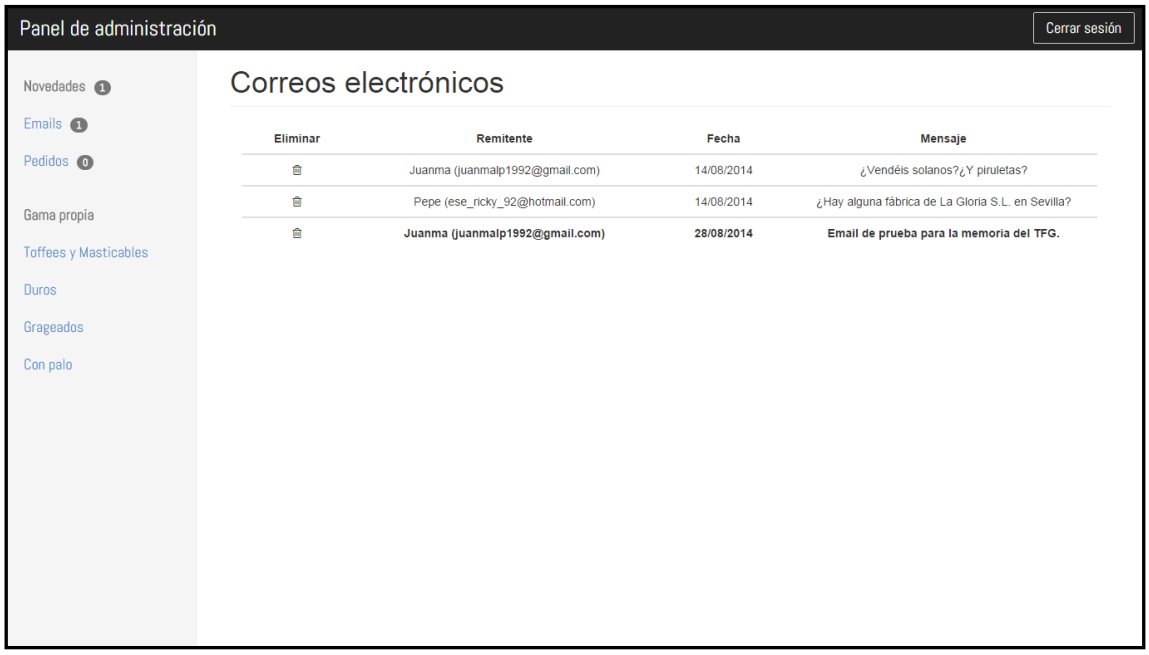



***Ilustración 118:*** *Manual de usuario administrador - Pantalla de emails recibidos*

Si hacemos click sobre algún campo del email podremos abrirlo y verlo con más detalles.

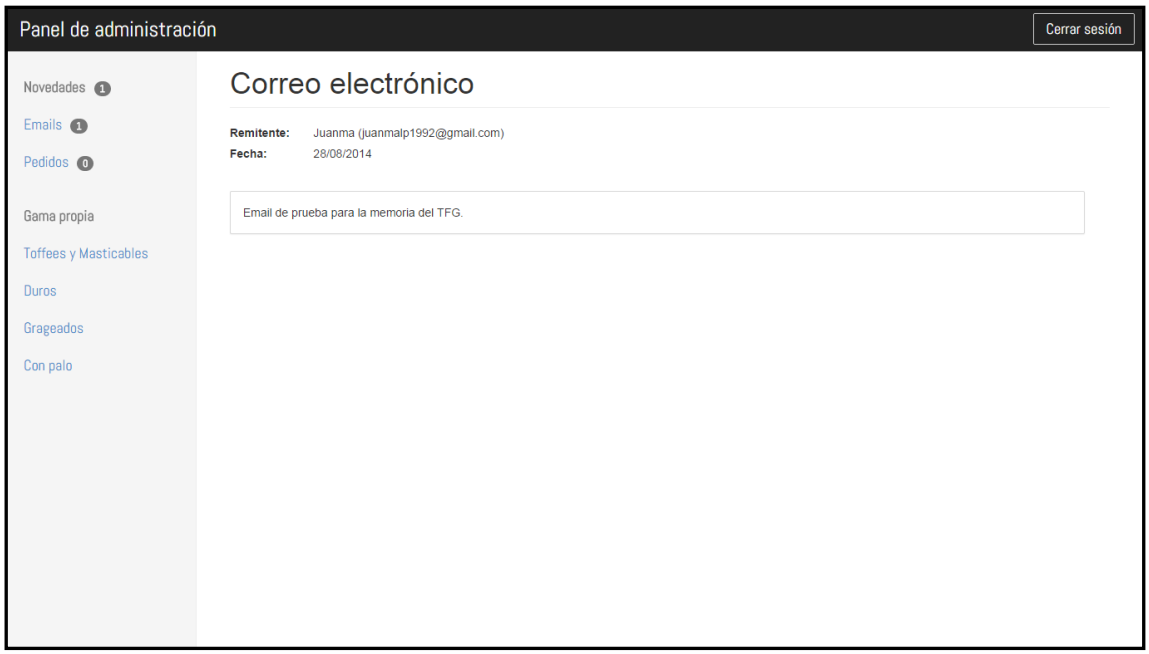



***Ilustración 119:*** *Manual de usuario administrador - Detalles de email recibido*

Si volvemos a la vista general de los emails recibidos, veremos que este email estará marcado como 'leído':


***Ilustración 120:*** *Manual de usuario administrador - Pantalla de emails recibidos sin mensajes por leer*

### Eliminar email

Para realizar esta acción hay que realizar previamente la anterior. Una vez que estamos en la pantalla de los emails recibidos no tenemos más que pulsar sobre el icono de la papelera y se nos preguntará lo siguiente:

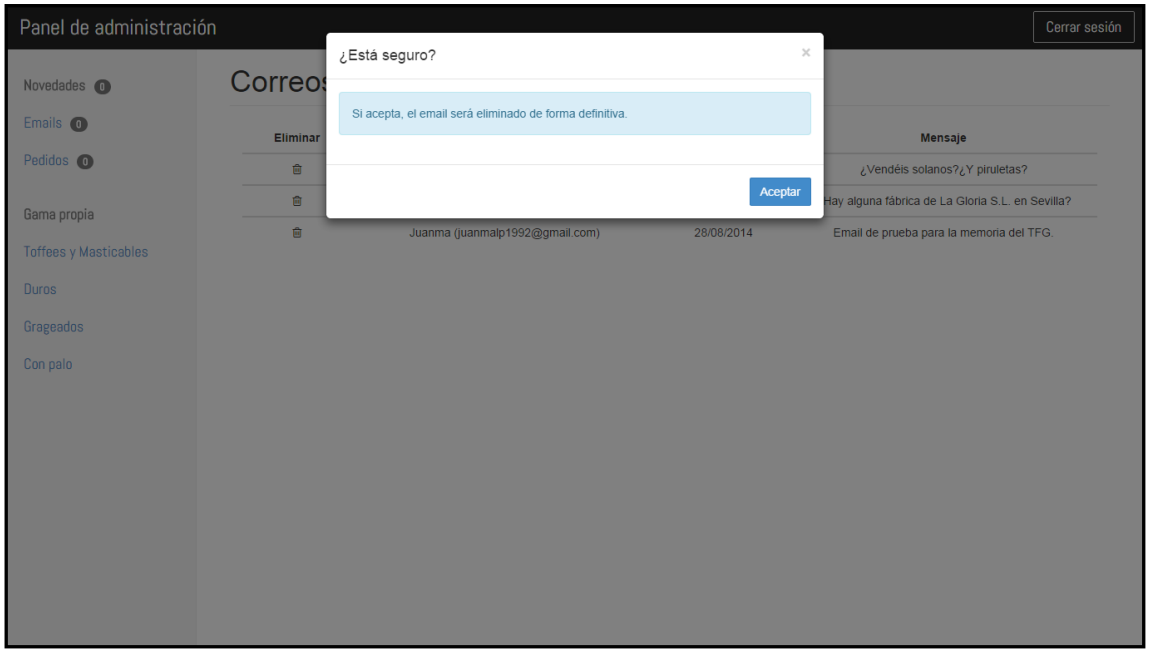



***Ilustración 121:*** *Manual de usuario administrador - Eliminación de un email*

Si pulsamos aceptar, el email quedará borrado definitivamente y al instante, tal y como se puede comprobar en la siguiente imagen:


***Ilustración 122:*** *Manual de usuario administrador - Pantalla de emails sin el email borrado*

# 13. MANUAL DE IMPLANTACIÓN

En este capítulo se explican una serie aspectos relacionados con la implantación del sistema, así como la configuración del **Node.js** para que funcione sobre **Heroku** o la conexión con la base de datos, ya sea local o remota.

Por tanto, este capítulo es de alta importancia a la hora de conocer cómo funciona la aplicación web en una plataforma como **Heroku**.

# 13.1. Configuración del servidor Node.js

En este apartado se explica cómo está configurado el servidor **Node.js**. Para que la aplicación web funcione en un servidor hay que realizar una serie de configuraciones. En el proyecto ha sido necesario realizar 3 tipos de configuraciones: la del servidor en sí, la de la conexión con la base de datos, tanto local como remota, y la configuración para que la aplicación funcione en **Heroku**.

## Configuración específica de Node.js

La configuración específica de **Node.js** se encuentra mayormente en el archivo 'app.js', el principal de la aplicación.

```
9  var express     = require('express');
10 var app         = express();
11 var path        = require("path");
12
13 var port = process.env.PORT || 8888;
14
15 app.configure(function() {
16     app.set('port', port);
17     app.set('views', path.join(__dirname + '/app/server/views'));
18     app.set('view engine', 'jade');
19     app.use(express.static(__dirname + '/public'));      // Localización de los ficheros estátic
20     app.use(express.logger('dev'));                      // Muestra un log de todos los request
21     app.use(express.bodyParser());                       // Permite cambiar el HTML con el métod
22     app.use(express.cookieParser());
23     app.use(express.session({ secret: 'super-duper-secret-secret' }));
24     app.use(express.methodOverride());                   // Simula DELETE y PUT
25     app.use(express.favicon(path.join(__dirname, '/app/public/img/logo.png')));
26     app.use(express.static(__dirname + '/app/public'));
27     app.use('/img', express.static(__dirname + '/app/public/img'));
28     app.use('/js', express.static(__dirname + '/app/public/js'));
29 });
30
31 require('./app/server/router')(app);
32
33
34 app.listen(port, function() {
35     console.log('App listening on port ' + port);
36 });
37
```

***Ilustración 123:*** *Manual de implantación - Configuración específica de* ***Node.js***

La primera configuración que observamos es el puerto. Se le asigna una variable disyuntiva de **JavaScript**. Esto quiere decir que se le asigna el primer valor, excepto si es 'undefined'. En ese caso, se le asigna el segundo. De ese modo, cuando la aplicación se ejecute en **Heroku** (cuyo puerto de ejecución es

aleatorio) se escuchará en el puerto que haya definido **Heroku**. Sin embargo, cuando ejecutemos la aplicación en local siempre estará escuchando peticiones en el puerto 8888.

Luego podemos ver un método en el que se configuran aspectos muy específicos, así como: el puerto, la ubicación de las vistas en el proyecto, el motor de plantillas (en este caso **JADE** en lugar de **HTML**), la habilitación de la variable de sesión, para funcionalidades como el carrito de la compra; y las rutas de los archivos estáticos como los archivos **JavaScript** de **AngularJS** o las imágenes.

A continuación se importa el archivo 'router.js', donde están definidas las rutas de la aplicación, y para finalizar se activa la escucha del servidor en el puerto que se haya establecido.

# Configuración de la conexión con la Base de Datos

La configuración para establecer la conexión con la base de datos se realiza en el módulo del servidor que llamamos 'data-base-manager.js'.

```
1  var crypto      = require('crypto');
2  var MongoDB     = require('mongodb').Db;
3  var Server      = require('mongodb').Server;
4  var moment      = require('moment');
5  var mongoose    = require('mongoose');
6
7
8  var dbPort, dbHost, dbName, dbUser, dbPass;
9
10 if (process.env.MONGOHQ_URL){
11     var elems   = String(process.env.MONGOHQ_URL).split(":");
12     dbPort      = Number(elems[elems.length-1].split("/")[0]);
13     dbHost      = elems[2].split("@")[1];
14     dbName      = elems[elems.length-1].split("/")[1];
15     dbUser      = elems[1].split("//")[1];
16     dbPass      = elems[2].split("@")[0];
17 }else{
18
19     dbPort      = 27017;
20     dbHost      = 'localhost';
21     dbName      = 'lagloria';
22 }
23
24 console.log("Puerto: " + dbPort);
25 console.log("DbHost: " + dbHost);
26 console.log("dbName: " + dbName);
27 console.log("dbUser: " + dbUser);
28 console.log("dbPass: " + dbPass);
29
```

***Ilustración 124:*** *Manual de implantación: Configuración del driver de* ***MongoDB*** *y* ***Node.js***

Éste es el primer fragmento del fichero 'data-base-manager.js'. Aquí se importan las librerías necesarias para la gestión de la conexión con la base de datos y inicializan una serie de variables de una forma u otra dependiendo si la aplicación se está ejecutando en **Heroku** o en local.

```
var db = new MongoDB(dbName, new Server(dbHost, dbPort, {auto_reconnect: true}), {w: 1});
    db.open(function(e, d){
    if (e) {
        console.log(e);
    }   else{
        console.log('connected to database :: ' + dbName);

        if(process.env.MONGOHQ_URL){
            db.authenticate(dbUser, dbPass, function(err, result){
                if(err){
                    console.log("Error en la autenticacion");
                }else{
                    console.log("Autenticado");
                }
            });
        }
    }
});

// Collections definition

var accounts                = db.collection('accounts');
var mails                   = db.collection('mails');
var orders                  = db.collection('orders');
var toffeesYMasticables     = db.collection('toffeesYMasticables');
var duros                   = db.collection('duros');
var grageados               = db.collection('grageados');
var conPalo                 = db.collection('conPalo');
```

***Ilustración 125:*** *Manual de implantación - Conexión con la base de datos*

Esta imagen se corresponde con la continuación del fichero. Aquí se inicia la conexión con la base de datos según los parámetros que han sido definidos en la imagen anterior.

Además, en caso de que estemos en **Heroku**, tendremos que acceder a la base de datos remota en **Compose** (antiguamente **MongoHQ**), la cual está protegida con un usuario y una contraseña, los cuales se obtienen de unas variables definidas en **Heroku**, las cuales se explicarán en breve.

# Configuración de Node.js para desplegar en Heroku

Para finalizar este capítulo vamos a explicar las configuraciones que hay que realizar para que el servidor, una vez desplegado en **Heroku**, funciona a la perfección y del mismo modo que si estuviera en local.

Primero tenemos que añadir las dependencias de **Node.js** en el archivo Package.json.

```
{
  "name": "web-angular",
  "version": "0.0.1",
  "description": "Sweets are you looking for are here.",
  "main": "server.js",
  "author": "Juan Manuel Lopez Pazos <juanmalp1992@gmail.com>",
  "dependencies": {
    "express": "~3.0.6",
    "mongoose": "~3.6.2",
    "mongodb": "^1.3.23",
    "url": "^0.7.9",
    "jade": "^1.3.1",
    "moment": "^2.7.0",
    "nodegit": "^0.1.4",
    "octonode": "^0.6.4",
    "request-json": "^0.4.11"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/jualoppaz/caramelosla gloria.git"
  },
  "license": "MIT",
  "engines": {
    "node": "0.10.x"
  }
}
```

***Ilustración 126:*** *Manual de implantación - Configuración del 'Package.json'*

Realmente este archivo contiene información específica de **Node.js**. Sin embargo, es utilizado por **Heroku** para establecer las relaciones entre las librerías. Por lo tanto, es recomendable tener todas las dependencias declaradas en este fichero para, de ese modo, garantizar que la aplicación no entrará en conflicto al ser desplegada en **Heroku**.

Una última configuración que tenemos que realizar específica de **Heroku** es mediante el fichero 'Procfile'. El fichero tiene que llamarse así, sin ninguna extensión y sin cambiar ni un solo carácter. En caso contrario, **Heroku** no lo detectará.

Procfile

```
web: node app.js
```

***Ilustración 127:*** *Manual de implantación - Configuración del 'Procfile'*

El contenido es bastante simple, ya que sólo hay que indicar el tipo de proyecto que es (node) y el nombre del principal archivo del proyecto (app.js). Si usáramos otro lenguaje o cambiáramos el nombre del archivo principal del proyecto, entonces habría que modificar la configuración de este fichero.

# 13.2. Despliegue en Heroku

En la sección anterior se ha comentado con todo detalle cuál es la configuración que hay que realizar en **Node.js** para que la aplicación funcione perfectamente al ser desplegada en **Heroku**. Sin embargo, también hay que realizar una serie de configuraciones directamente en **Heroku**, por ejemplo, las variables de ejecución.

Estas variables, a las que se accede desde el servidor mediante 'process.env.[nombre de la variable]', sirven para acceder a una información que nos interesa conocer en el momento en que la aplicación está desplegada en **Heroku**.

En el caso de esta aplicación hay una serie de variables de ejecución definidas en **Heroku**, como pueden ser el usuario y la contraseña a la base de datos remota en **Compose**.

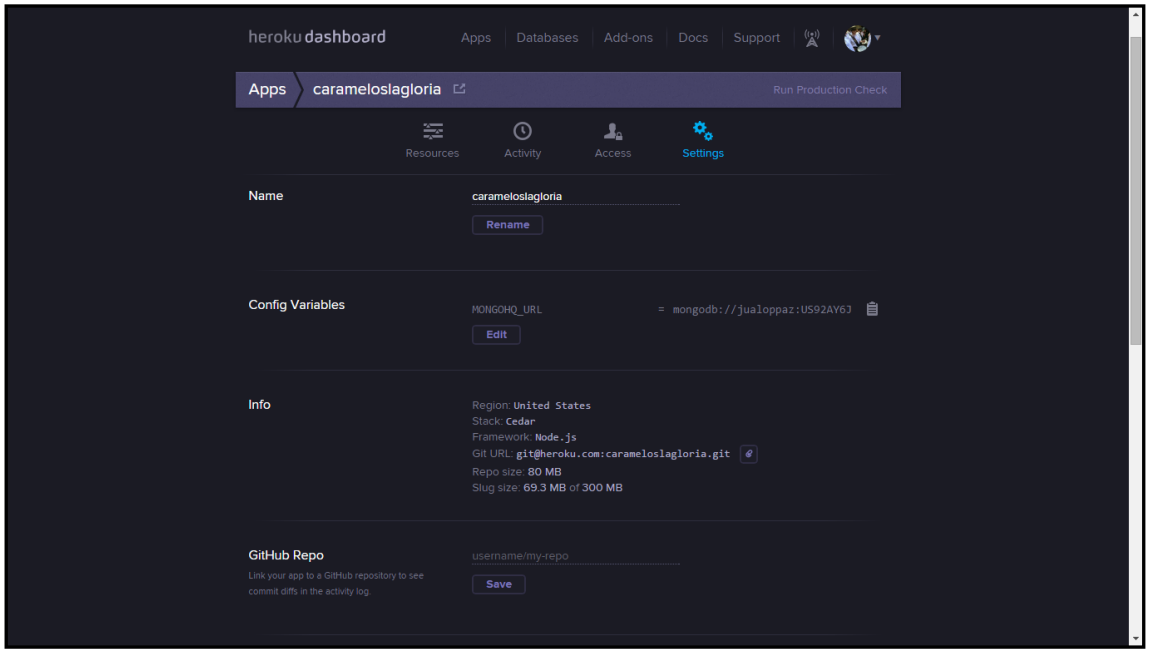



***Ilustración 128:*** *Manual de implantación - Dashboard de* ***Heroku***

Como se puede observar, en el panel de control de **Heroku** está creada un aplicación llamada ‘caramelosla gloria’ y que tiene una variable de ejecución configurada: ‘MONGOHQ_URL’. En esta variable se encuentran el usuario y la contraseña de la base de datos alojada en **Compose**.

Dichos datos son consultados en el ‘data-base-manager’, tal y como se pudo comprobar en un apartado anterior de este capítulo.

# BLOQUE VII: CONCLUSIONES

# 14. ANÁLISIS DEL PROYECTO

En este capítulo se realiza un análisis del proyecto, analizando lo que ha supuesto realizar este trabajo desde el punto de vista de la planificación y otros aspectos técnicos.

Empezaremos realizando un análisis de la planificación, donde se analizará el tiempo y los costes empleados.

Finalmente se indicarán una serie de posibles mejoras o extensiones que podría tener este proyecto.

# 14.1. Análisis de la planificación

Una vez finalizado el proyecto, las horas invertidas han sido de 372 h, lo que supone 72 horas más de las previstas. Esta circunstancia podía ser previsible, sin embargo no esperaba que el margen de error fuera tan amplio, ya que no es el primer proyecto que realizo.

No obstante, hay que considerar una serie de factores que han podido implicar este retraso, por ejemplo: tecnologías desconocidas, problemas técnicos y los problemas típicos que suelen suceder en este tipo de proyectos, así como fallos en el diseño, en el inventario de contenidos o en alguna funcionalidad de la aplicación web.

En término de costes, este retraso supondría un presupuesto final de 4464 €, lo que supone 864 € más de lo que se había presupuestado inicialmente.

## Costes reales de personal

| Tarea | Duración (horas) | Rol | Coste (€ / rol) | Coste (€) |
|---|---|---|---|---|
| **Estructura de la aplicación web** | | | | |
| Elicitación de requisitos | 12 | Jefe de proyecto | 23,11 | 277,32 |
| Diseño | 13 | Analista | 19,86 | 258,18 |
| Codificación | 20 | Programador | 16,46 | 329,2 |
| Pruebas unitarias | 5 | Programador | 16,46 | 82,3 |
| Pruebas de validación | 4 | Jefe de proyecto | 23,11 | 92,44 |
| Documentación | 23 | Jefe de proyecto, Analista y Programador | 19,81 | 455,63 |
| **Inventario de productos** | | | | |
| Elicitación de requisitos | 15 | Jefe de proyecto | 23,11 | 346,65 |

| | | | | |
|---|---|---|---|---|
| Diseño | 12 | Analista | 19,86 | 238,32 |
| Codificación | 65 | Programador | 16,46 | 1069,9 |
| Pruebas unitarias | 7 | Programador | 16,46 | 115,22 |
| Pruebas de validación | 6 | Jefe de proyecto | 23,11 | 138,66 |
| Documentación | 6 | Jefe de proyecto, Analista y Programador | 19,81 | 118,86 |
| **Gestión de usuarios** | | | | |
| Elicitación de requisitos | 11 | Jefe de proyecto | 23,11 | 254,21 |
| Diseño | 9 | Analista | 19,86 | 178,74 |
| Codificación | 35 | Programador | 16,46 | 576,1 |
| Pruebas unitarias | 7 | Programador | 16,46 | 115,22 |
| Pruebas de validación | 6 | Jefe de proyecto | 23,11 | 138,66 |
| Documentación | 10 | Jefe de proyecto, Analista y Programador | 19,81 | 198,1 |
| **Aspectos sociales** | | | | |
| Elicitación de requisitos | 20 | Jefe de proyecto | 23,11 | 462,2 |
| Diseño | 15 | Analista | 19,86 | 297,9 |
| Codificación | 50 | Programador | 16,46 | 823 |
| Pruebas unitarias | 14 | Programador | 16,46 | 230,44 |
| Pruebas de validación | 9 | Jefe de proyecto | 23,11 | 207,99 |
| Documentación | 25 | Jefe de proyecto, Analista y Programador | 19,81 | 336,77 |
| **Implantación** | | | | |
| Elicitación de requisitos | 7 | Jefe de proyecto | 23,11 | 161,77 |
| Diseño | 9 | Analista | 19,86 | 178,74 |

| | | | | |
|---|---|---|---|---|
| Codificación | 12 | Programador | 16,46 | 197,52 |
| Pruebas unitarias | 14 | Programador | 16,46 | 230,44 |
| Pruebas de validación | 14 | Jefe de proyecto | 23,11 | 323,54 |
| Documentación | 14 | Jefe de proyecto, Analista y Programador | 19,81 | 277,34 |
| **Documentación del proyecto** | | | | |
| Memoria del TFG | 98 | Jefe de proyecto, Analista y Programador | 19,81 | 1941,38 |
| **Total** | | | | 10652,74 |

Los costes reales de personal se elevan a 10652,74 €, lo que supone un incremento de 926,77 € respecto a los costes de personal estimados.

# Costes reales de software

Los costes reales de software han sido los mismos que se estimaron:

| | |
|---|---|
| Licencia de JetBrains WebStorm 7 | 121,60 € |
| **Total** | 121,60 € |

# Costes totales reales

Una vez que han sido calcularos todos los costes reales, se tiene que el coste del proyecto ha sido de:

| | |
|---|---|
| Costes reales de personal | 10652,74 € |
| Costes reales de software | 121,60 € |
| **Total** | **10774,34 €** |

En el desarrollo del proyecto pensé que las horas de trabajo tendrían que ser muchas más y que los costes se iban a disparar. Sin embargo, a pesar de haber sido requeridas más horas de trabajo, no ha sido tan alto.

Sin embargo, sí ha habido un error muy común en todos los Sprints. Ha habido tareas que requerían más horas de dedicación que las que se habían estimado inicialmente, pero también se ha dado el caso contrario. También encontramos tareas que han requerido mucho menos tiempo del que se esperaba.

Esa última circunstancia ha sido, sin duda, la que ha evitado que se dispararan de forma muy elevada los costes.

## 14.2. Posibles mejoras del proyecto

Antes de añadir cualquier funcionalidad nueva al proyecto, lo primero que se podría hacer es refactorizar el código de la aplicación web al completo. Por ejemplo, cada categoría en la web tiene sus propias vistas: una vista general de los productos y una vista específica. Se podrían implementar sólo dos plantillas que sirvieran para todos los productos. Quizás quedarían más grandes de lo recomendable, pero facilitaría mucho los cambios en la web, ya que las vistas estarían más centralizas.

En cuanto a funcionalidades, se podría implementar algún mecanismo para subir fotos online del logo de una empresa, de modo que el proveedor que quiera comprar caramelos pueda ver cómo quedarían los caramelos con el logo enviado.

Respecto al diseño en sí, se podría incluir alguna novedad distinta en las vistas para que no parecieran todas iguales. Por ejemplo, en lugar de paneles rectangulares se podrían usar algunos paneles circulares, por ejemplo para las imágenes de los caramelos o alguna característica similar.

# 15. CONCLUSIONES PERSONALES

Realizar este proyecto me ha aportado muchos conocimientos. Por ejemplo, yo no conocía las tecnologías que he utilizado para desarrollar esta aplicación web. Así que me siento muy satisfecho de haber obtenido tantos conocimientos y de ver el resultado final tras un trabajo que ha resultado muy largo y difícil.

Me ha parecido una iniciativa muy interesante haber trabajado en uno de los proyectos propuestos por **SINERGIA**. La idea de reunir a gente desconocida y de diferentes disciplinas es una buena forma de practicar y ganar experiencia de cara a nuestro futuro profesional inmediato, ya que es algo muy parecido a lo que nos podemos encontrar en una empresa.

Sin embargo, a pesar de los aspectos positivos que reflejo en las líneas anteriores también concluyo que, en ciertos aspectos técnicos del proyecto, me hubiera hecho falta algo más de tutoría, cosa que, por diversos motivos, no ha podido llevarse a cabo con la correspondiente pérdida de conocimientos, teniendo que ser aún más autodidacta de lo que esperaba. En momentos puntuales me hubiera resultado útil mayor orientación.

Dicho todo esto espero aprender de este proyecto de las cosas positivas pero también de las negativas, lo cual me ayudará a evitarlas en el futuro.

# 16. BIBLIOGRAFÍA Y WEBGRAFÍA

A continuación se detallan tanto la bibliografía como la webgrafía consultada para la realización de este trabajo:

## Bibliografía


- Ebook de **Carlos Azaustre** titulado "*Desarrollo Web ágil con Angular.js*".
- Libro de Pedro Teixeira titulado "*Professional Node.js, building JavaScript-Based scalable software*".


## Webgrafía


**Comparación de metodologías**

- http://metodologiasagiles.wikispaces.com/metodos+agiles+vs+metodos+tradicionales
- http://www.beeva.com/agilismo-vs-metodologia-tradicional
- http://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software
- http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software
- http://www.ecured.cu/index.php/Metodolog%C3%ADas_Tradicionales

**Scrum**

- http://es.wikipedia.org/wiki/Scrum

**Estimación de costes**

- http://www.aeiciberseguridad.es/descargas/categoria6/8774932.pdf
- www.ine.es

**MongoDB**

- http://docs.mongodb.org
- http://es.wikipedia.org/wiki/MongoDB
- http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no
- http://planetubuntu.es/post/mongodb-lider-de-las-alternativas-a-las-tradicionales-bases-de-datos-sqls
- http://www.genbetadev.com/bases-de-datos/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap

**Node.js**

- https://github.com/braitsch/node-login (Proyecto base de la aplicación web)
- http://es.wikipedia.org/wiki/Node.js
- http://www.nodehispano.com/2011/11/que-es-node-js-nodejs
- http://nodejs.org/api
- http://www.rmunoz.net/introduccion-a-node-js.html
- http://blog.beeva.com/las-ventajas-de-apostar-por-node-js
- http://blog.cballesterosvelasco.es/2012/03/presento-nodenet-alternativa-real.html

**AngularJS**

- https://docs.angularjs.org/guide
- http://tutorials.jenkov.com/angularjs/ajax.html

**jQuery**

- http://es.wikipedia.org/wiki/JQuery
- http://blog.capacityacademy.com/2013/03/16/jquery-que-es-origenes-ventajas-desventajas
- http://www.etnassoft.com/2011/03/28/alternativas-a-jquery
- http://api.jquery.com/jquery.ajax
- http://tutorialzine.com/2013/04/50-amazing-jquery-plugins/

**JADE**

- http://jade-lang.com
- http://codehero.co/node-y-express-jade-js
- http://www.kabytes.com/programacion/sistema-de-plantillas-para-node-js

**Bootstrap**

- http://getbootstrap.com

**Fontawesome**

- http://fortawesome.github.io/Font-Awesome

**Pruebas**

- https://www.npmjs.org/package/loadtest

# APÉNDICE A:
# LA GLORIA

# A.1. INVENTARIO DE CONTENIDOS

→INICIO
→CONTACTE
- Formulario con datos del usuario
- Cuadro: comentarios

→MENÚS

1. Empresa: misión
2. Modelos:

| MODELO | INFORMACIÓN DEL PRODUCTO | ENVOLTORIO | PEDIDO MÍNIMO |
|---|---|---|---|
| 10. Corazones | Fotos | Flowpack | 50 kg |
| 9. Piruletas | Fotos | Específico | 6000 unidades |
| 4. Crystal | Fotos + "Caramelo duro de frutas" | 2 lazos, Flowpack | 50 kg |
| 3. Ovalado | Fotos + "Caramelo duro de frutas" | 2 lazos, Flowpack | 50 kg |
| 7. Redondo | Fotos + "Caramelo duro de frutas" | 1 lazo | 50 kg |
| 6. Bloke | Fotos + "Caramelo duro de frutas" | Flowpack | 50 kg |
| 5. Ponny | Fotos + "Minicaramelo de frutas" | 2 lazos, Flowpack | 50 kg |
| 8. Gajitos | Fotos + "Naranja y limón" | 2 lazos | 50 kg |
| 2. Masticable | Fotos + "Caramelo masticable de frutas" | 2 lazos | 50 kg |
| 1. Toffee | Fotos + "Caramelo de Toffee" | 2 lazos | 50 kg |

***Tabla 66:*** *Apéndice A - Inventario de contenidos*

3. Catálogos (-)

4. Propuesta de pedidos:
- Formulario con datos del usuario
- Producto deseado (Modelo + cantidad)

Atención al cliente
- Formulario con datos del usuario
- Cuadro: motivo de contacto

Buzón de sugerencias (mejoras en la web, opiniones de los productos,…)
- Formulario con datos del usuario
- Cuadro: comentarios

# Caramelos LA GLORIA

En la gama propia LA GLORIA podríamos agregar los artículos según similitudes, por ejemplo:

A. Caramelos Toffees y Masticables

009. Toffee Surtido
010. Toffee Toledano
011. Toffee Chocolate
012. Toffee Chocolate Menta
013. Toffee Regaliz
014. Toffee Chocolate Blanco
015. Caramelos Masticables de Frutas

B. Caramelos duros de Frutas

020. Crystal Rainbow
025. Crystal Mint
030. Surtido Gloria
050. Ponny Frutas
120. Ponny Mint
130. Ponny Anís
140. Ponny Colours
150. Ponny Express
060. Colorines
080. Ice Mint
082. Verde Mint
090. Oval Gloria
110. Gajos Naranja y Limón

C. Caramelos Con palo

| | | |
|---|---|---|
| 420. Glorypop 7g | 420C | Colorful 7g |
| 440. Glorypop 9g | 440C | Colorful 9g |
| 450. Glorypop 16g | 450C | Colorful 16g |

D. Grageados

310. Peladillas Blancas
320. Peladillas Colores
350. Anises
370. Bolitas Anís Blancas
371. Bolitas Anís Colores
380. Maní

# A.2. DOSSIER DE DISEÑO


***Ilustración 129:*** *Apéndice A - Dossier de diseño*

# APÉNDICE B: CATÁLOGO DE WIDGETS

# 1. Modal

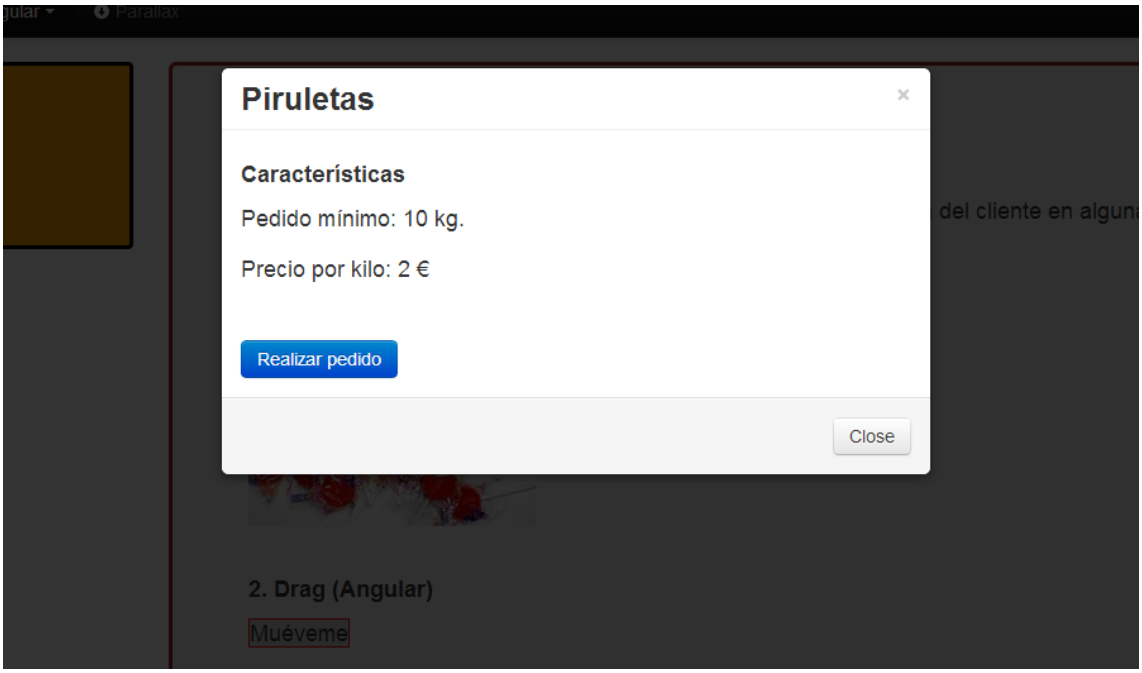



***Ilustración 130:*** *Widgets - Modal*

Como vemos, el modal aísla el resto de la página, dando importancia a la notificación pertinente. Esto crea una sensación de profundidad que hace inevitable centrar toda la atención en el cuadro blanco. Podría ser muy útil para cerrar presupuestos, encargar artículos o confirmar el cierre de la página, entre otros.

La forma más fácil de implementar este widget es mediante la librería **Bootstrap**. Por defecto, el modal no bloquea las partes sombreadas de la página; es decir, si hacemos click en una zona sombreada el modal desaparecerá. No obstante, podemos hacer que el modal bloquee el resto de la página. Para ello sólo tenemos que seguir la documentación de **Bootstrap**.

# 2. Dropdown

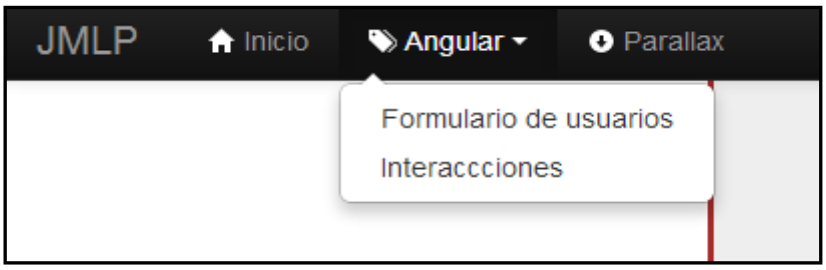



***Ilustración 131:*** *Widgets - Dropdown*

Los dropdowns son muy usados hoy en día y muy fáciles de implementar. Por su apariencia, son muy habituales en el despliegue de submenús, ya que facilitan mucho que el usuario conozca la jerarquía de la navegación, pues se denota gráficamente que el dropdown procede de la barra superior y esta puede volver a su estado inicial.

**Bootstrap** nos proporciona una amplia gama con diferentes estilos, además de encargarse de manejar todo tipo de eventos asociados a los mismos.

# 3. Tabs (pestañas)

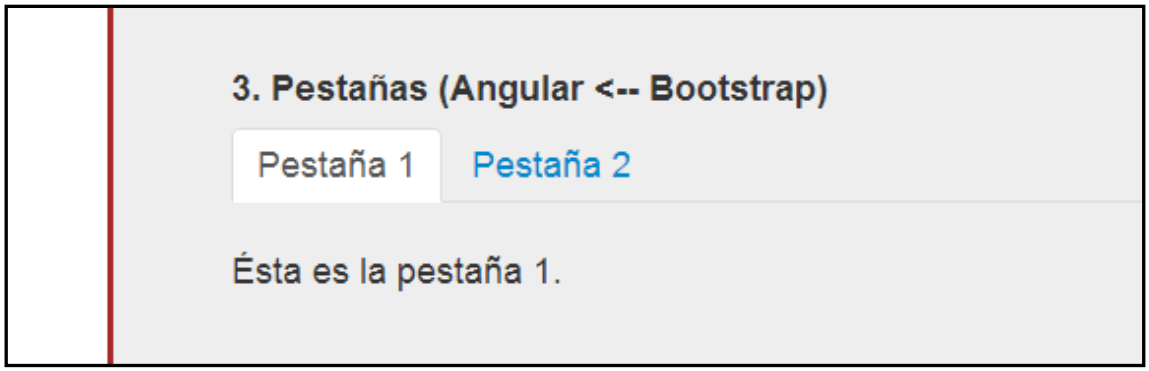



***Ilustración 132:*** *Widgets - Tabs*

Las pestañas son un widget básico para la navegación global. Hacen posible que el usuario pueda saber inmediatamente en qué menú se encuentra, ya que está resaltado, y cambiar a otro en cualquier momento. Un punto en su contra es que no son muy estéticas.

En cuanto a su implementación, se puede realizar tanto con **Bootstrap** como con el framework **AngularJS**, el cual hace uso de **Bootstrap**. De entre

estas dos opciones, la forma más fácil de implementar las pestañas es sólo con **Bootstrap**, ya que con **AngularJS** se modulariza demasiado el código y en **Bootstrap** se puede realizar añadiendo un par de atributos en el campo “class” del div correspondiente.

# 4. Zoom de textos:

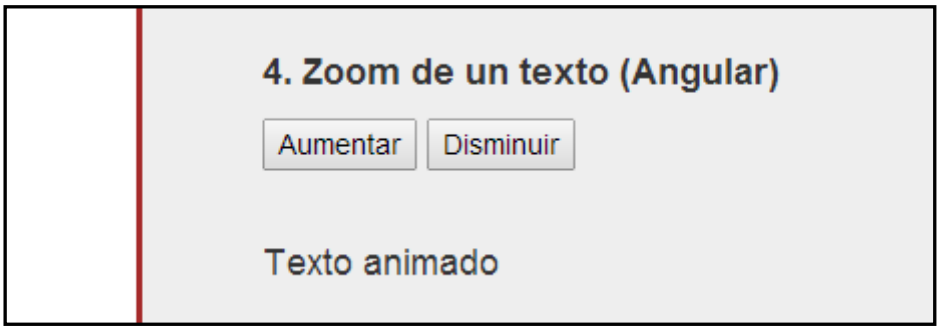



***Ilustración 133:*** *Widgets - Texto sin zoom*


***Ilustración 134:*** *Widgets - Texto con zoom*

Esta interacción es una de las muchas que ofrece **AngularJS**. Con sólo un fragmento de código **JavaScript** tendremos implementada esta funcionalidad, la cual resulta muy llamativa y puede ser útil para captar la atención de los usuarios.

# 5. Mostrar/Ocultar divs

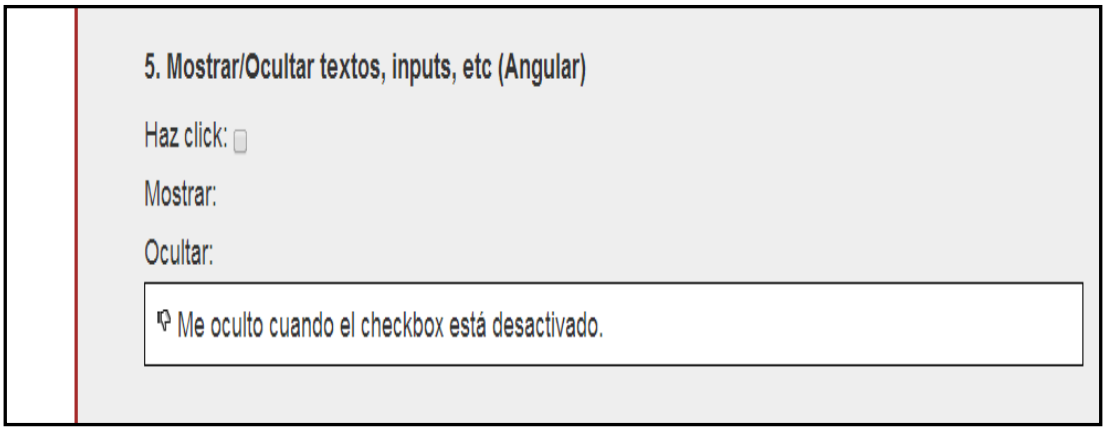



***Ilustración 135:*** *Widgets - Div oculto*

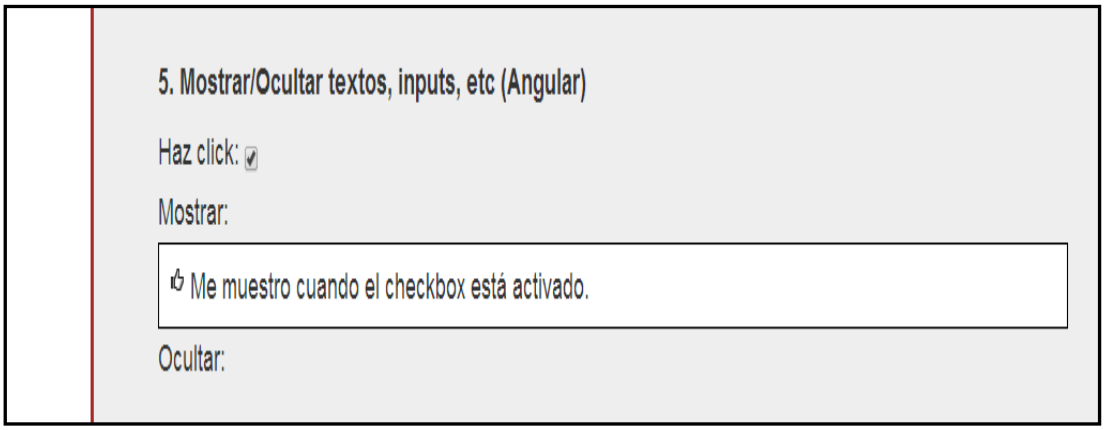



***Ilustración 136:*** *Widgets - Div visible*

Se trata de otra interacción interesante proporcionada por **AngularJS**. Dependiendo del estado del checkbox - si está seleccionado o no - se mostrará un div u otro. Dicha transición se realiza mediante una animación a pesar de que no se observe en las imágenes anteriores.

Creemos que es un widget muy útil para, por ejemplo, completar formularios e ir añadiendo información adicional importante para el usuario.

# 6. Data-binding de AngularJS

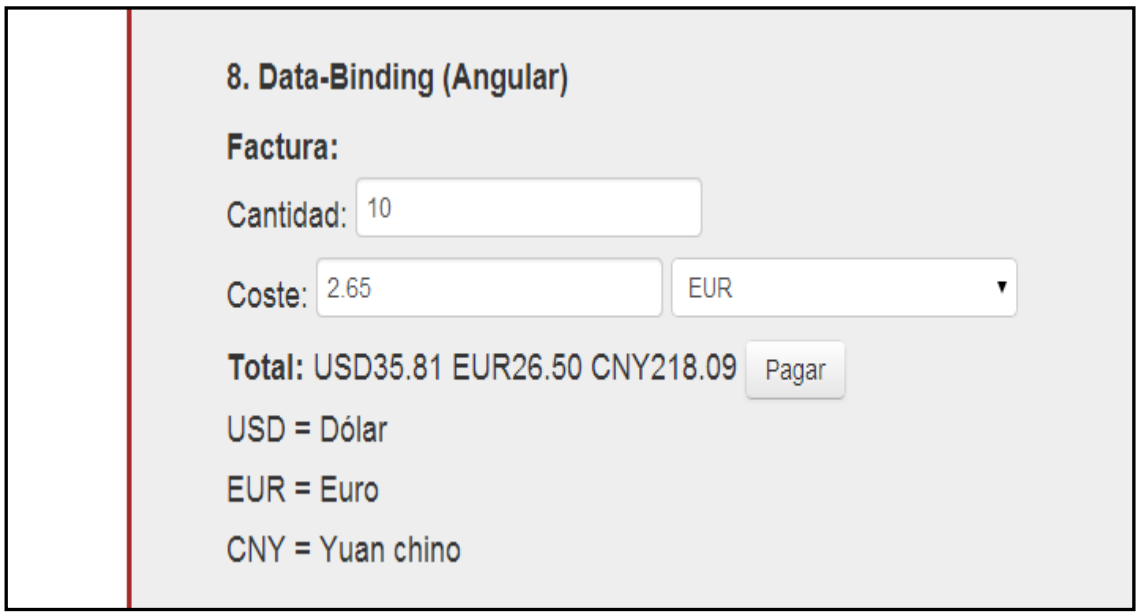



***Ilustración 137:*** *Widgets - Data-binding de* ***AngularJS***

Otra interacción exclusiva de **AngularJS**. Esta interacción está basada en los data-binding, la piedra angular de **AngularJS**, valga la redundancia. Con ella podemos realizar un cálculo instantáneo, algo ideal para realizar presupuestos online. Muy útil, por supuesto, en webs de e-commerce y además, como vemos, con la posibilidad de ofrecer el resultado en diferentes divisas, muy interesante también para páginas con público internacional.

# 7. Búsqueda instantánea full-text

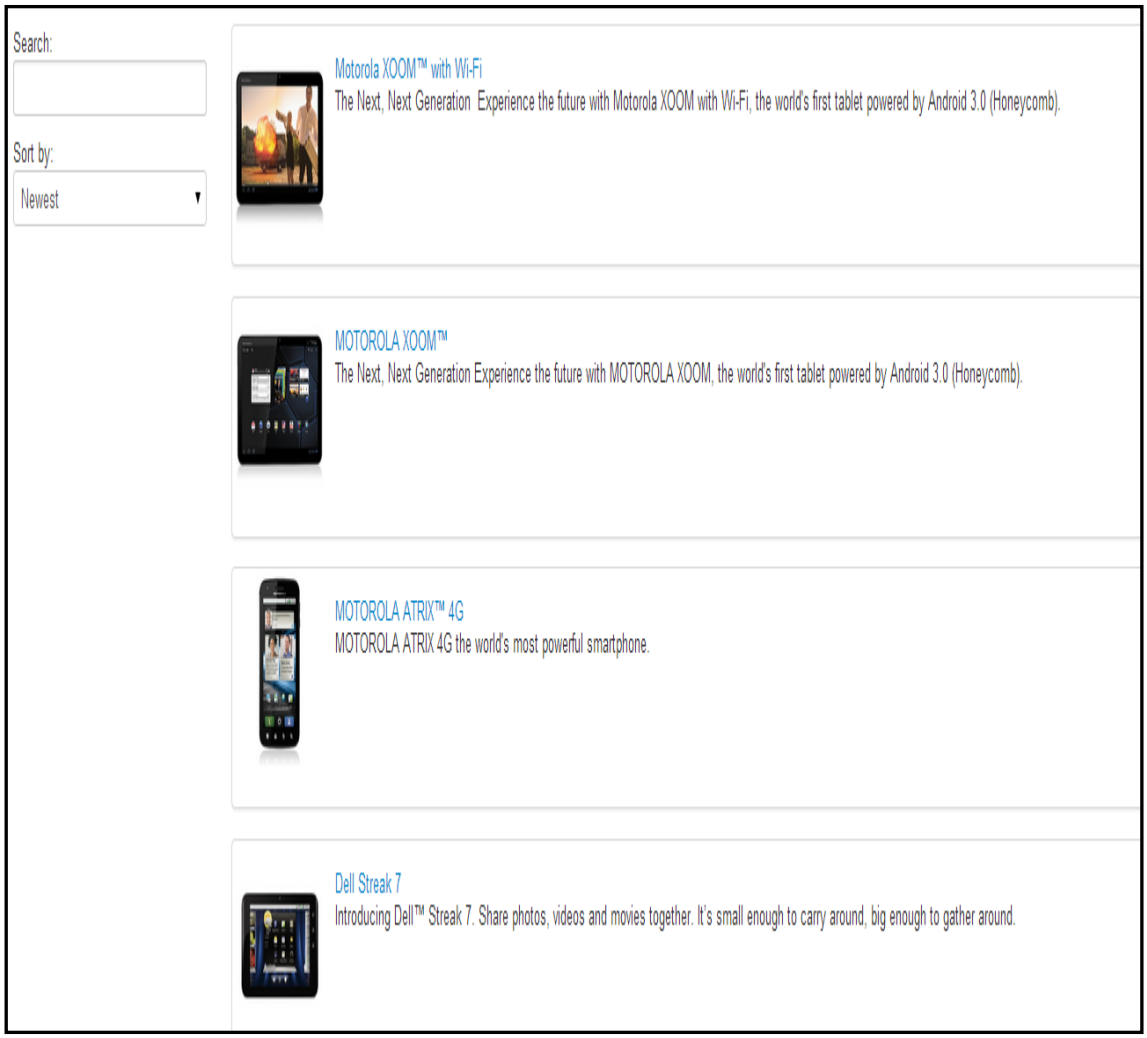



***Ilustración 138:*** *Widgets - Búsqueda instantánea full-text*

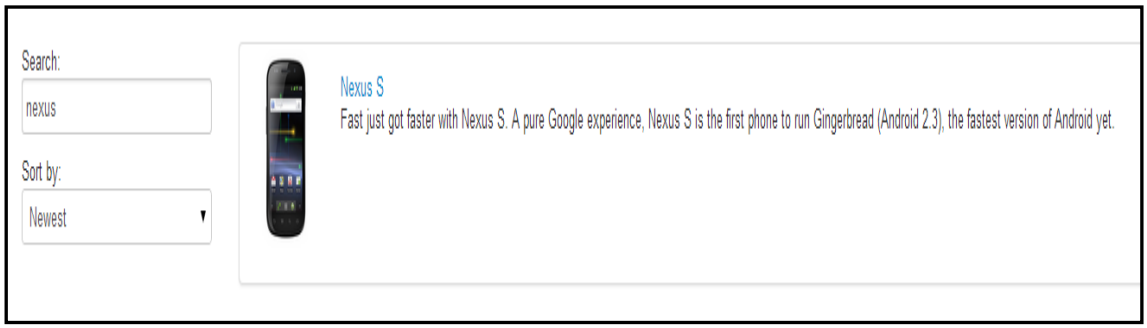



***Ilustración 139:*** *Widgets - Resultado de la búsqueda instantánea full-text*

**AngularJS** nos ofrece la posibilidad de realizar búsquedas instantáneas y sin necesidad de recargar la página. Este ejemplo está extraído del tutorial phonecat de **AngularJS**, en el que se trabaja con marcas y modelos de

smartphones. Inicialmente se tiene una lista ordenada según el criterio que especifiquemos, y más abajo vemos una imagen en la que buscamos por la palabra “nexus”. Así vemos cómo sólo aparece un smartphone con esa palabra en el título o en la descripción.

Esta interacción puede resultar imprescindible en webs de comercio electrónico en las que se ofrezca gran cantidad de productos y estos sigan una clasificación. Esto supondría en cierto modo un filtro rápido que agilizaría la navegación del usuario y, por tanto, su afinidad a la página aumentaría.

# 8. Collapse

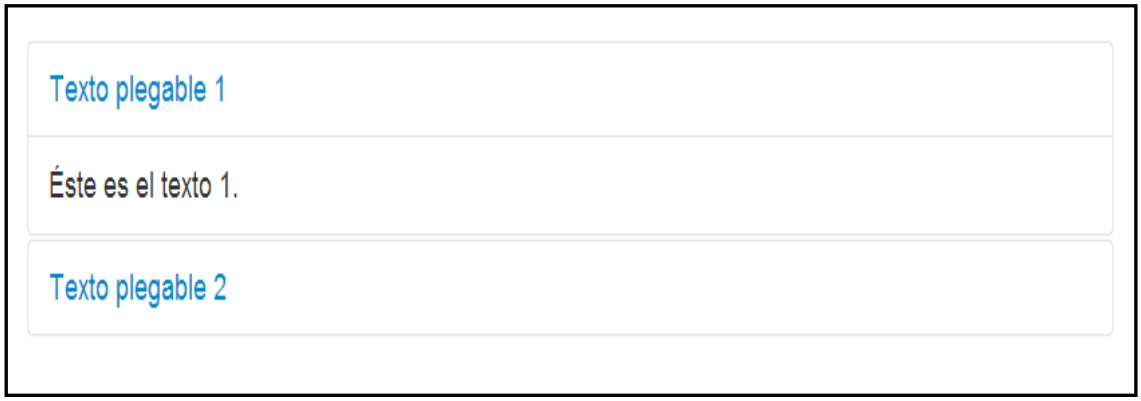



***Ilustración 140:*** *Widgets - Collapse desplegado*

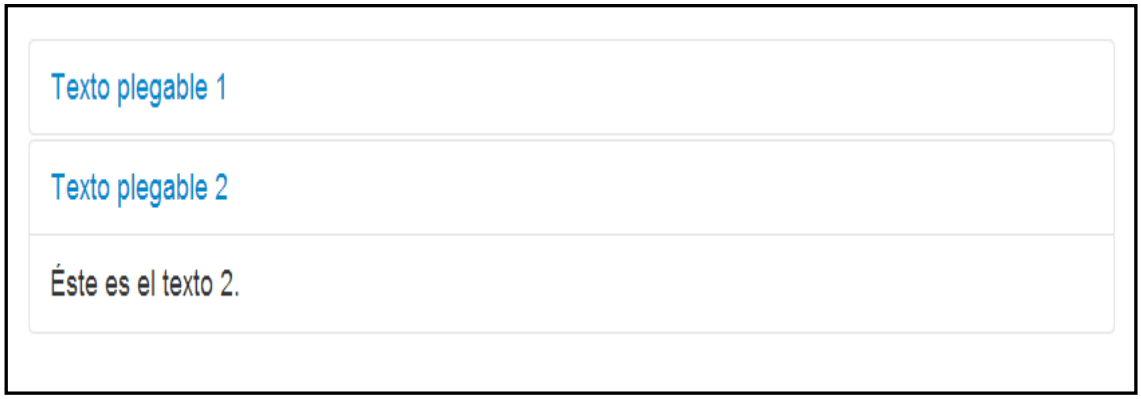



***Ilustración 141:*** *Widgets - Collapse plegado*

Una vez más nos encontramos ante una funcionalidad ofrecida por **Bootstrap**. El collapse es una forma de distribuir información por párrafos o textos completos, de modo que sólo se muestra una de las secciones plegables en cada momento.

Puede ser una interacción muy interesante para no extender mucho la longitud de la página y tener que realizar scroll muchas veces. Es muy importante que se muestre siempre la menor cantidad de información posible, para que exista orden y limpieza.

# 9. Datepicker

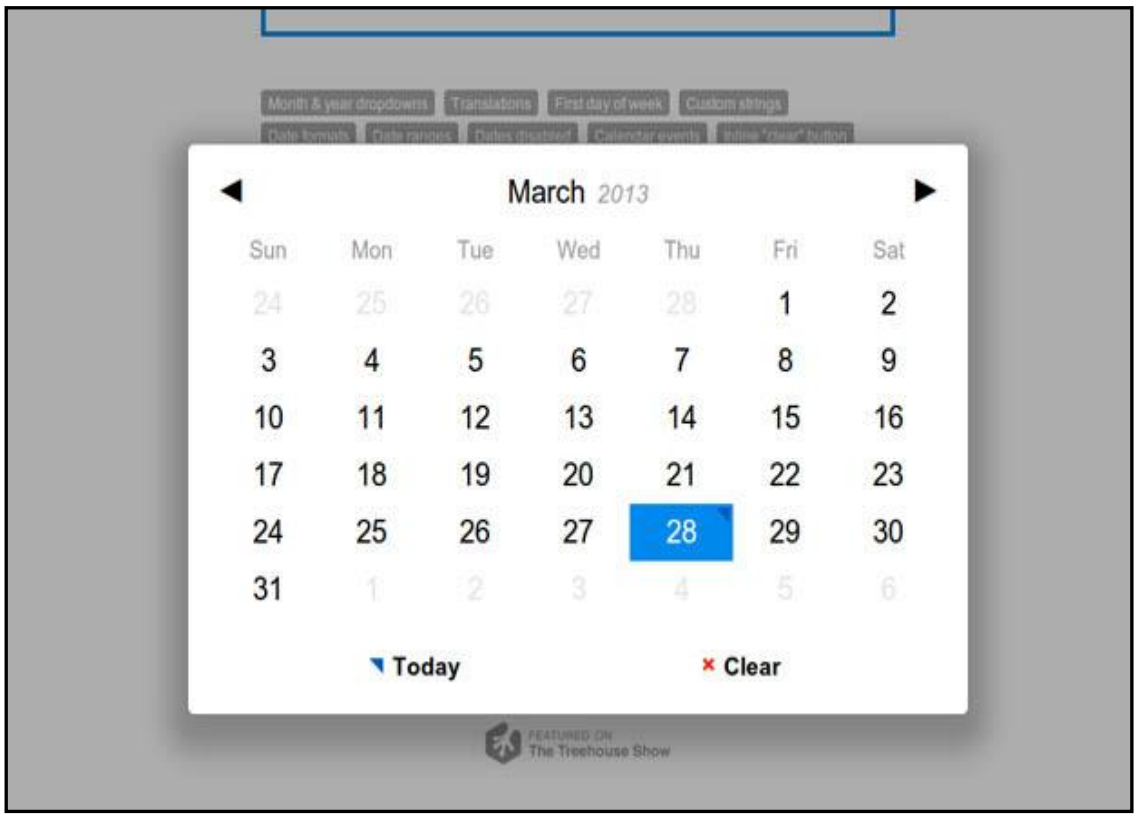



***Ilustración 142:*** *Widgets - Datepicker*

Este widget es proporcionado por **jQuery**, una librería **JavaScript** con una gran variedad de interacciones. Este datepicker es el estándar, en el que se muestran los días, los meses y los años. Sin embargo, **jQuery** proporciona otros datepickers con distintos formatos, así como algunos con la hora concreta, el día de la semana, etc.

Esta interacción no es útil sólo por estética, sino por simplicidad, ya que es posible controlar el formato de las fechas que introduce un usuario en un formulario. Sería muy razonable que se empleara en webs comerciales, en las que la fecha sea un apartado clave (ventas de billetes de tren, avión o autobús, citas médicas, reuniones,...).

# 10. Chosen

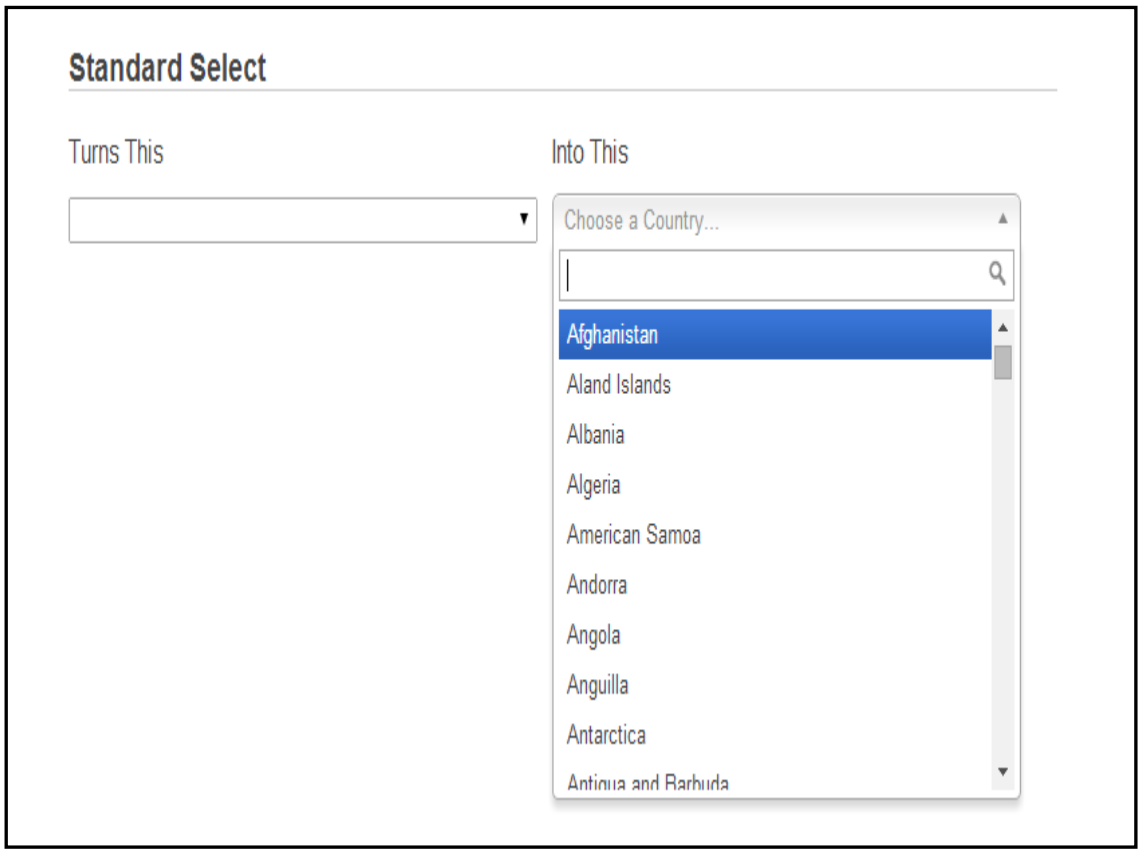



***Ilustración 143:*** *Widgets - Chosen*

Nos encontramos ante otra interacción ofrecida por **jQuery**. Es muy escaso el código necesario para hacer uso de esta interacción, conocida como "select" o "listbox".

La particularidad que añade **jQuery** es la búsqueda, la cual es instantánea y por palabras que empiecen por la subcadena introducida en la caja de texto.

Muy útil para incluir direcciones, como vemos en la imagen, o elegir productos, aunque los productos suelen identificarse más con imágenes que con su propio nombre.

# 11. Scrollpath

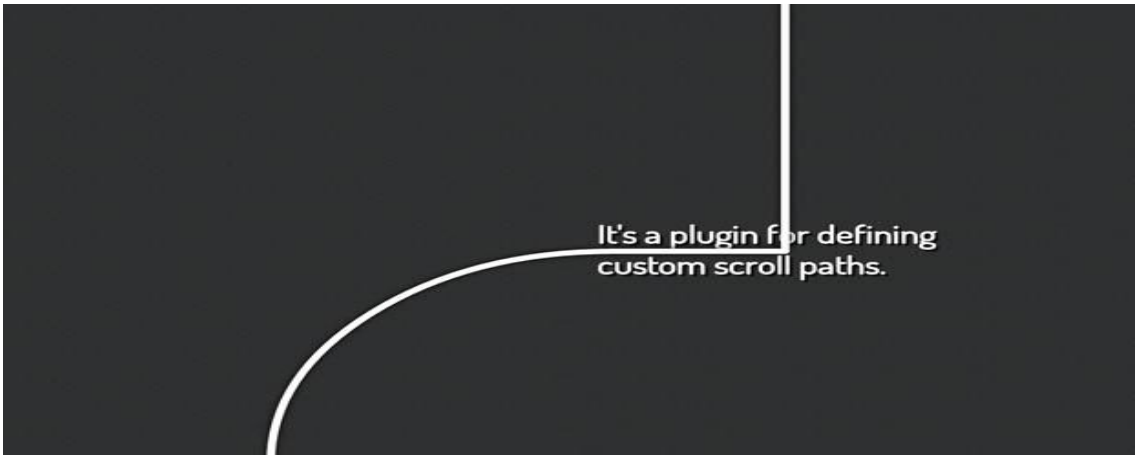



***Ilustración 144:*** *Widgets - Scrollpath*

Esta interacción también pertenece a **jQuery**. Con ella podemos simular un recorrido en nuestra web, de modo que el podemos rotar la imagen cuanto deseemos. Además, tenemos la posibilidad tanto de mostrar el camino recorrido como de ocultarlo.

Es una interacción que puede resultar interesante para una página de introducción de una página web. Es muy dinámica e interactiva, ya que sumerge al usuario en un recorrido por la página, abandonando el concepto plano de las interfaces típicas.

# 12. Arctext


***Ilustración 145:*** *Widgets - Arctext*

Una nueva interacción ofrecida por **jQuery**. Con esta interacción podemos curvar nuestros textos. Esta interacción hace uso de la algunas propiedades del estándar **CSS3** como el ángulo de rotación.

Es bastante útil para programadores sin conocimientos de diseño gráfico, ya que el hecho de que se aplique “apariencia” mediante programación facilita su flexibilidad para los diferentes formatos en los que aparecerá la web. Por otro lado, creemos que esta técnica podría ser suplida por diferentes diseños para cada tamaño de web, reduciendo su complejidad.

# 13. Gridster

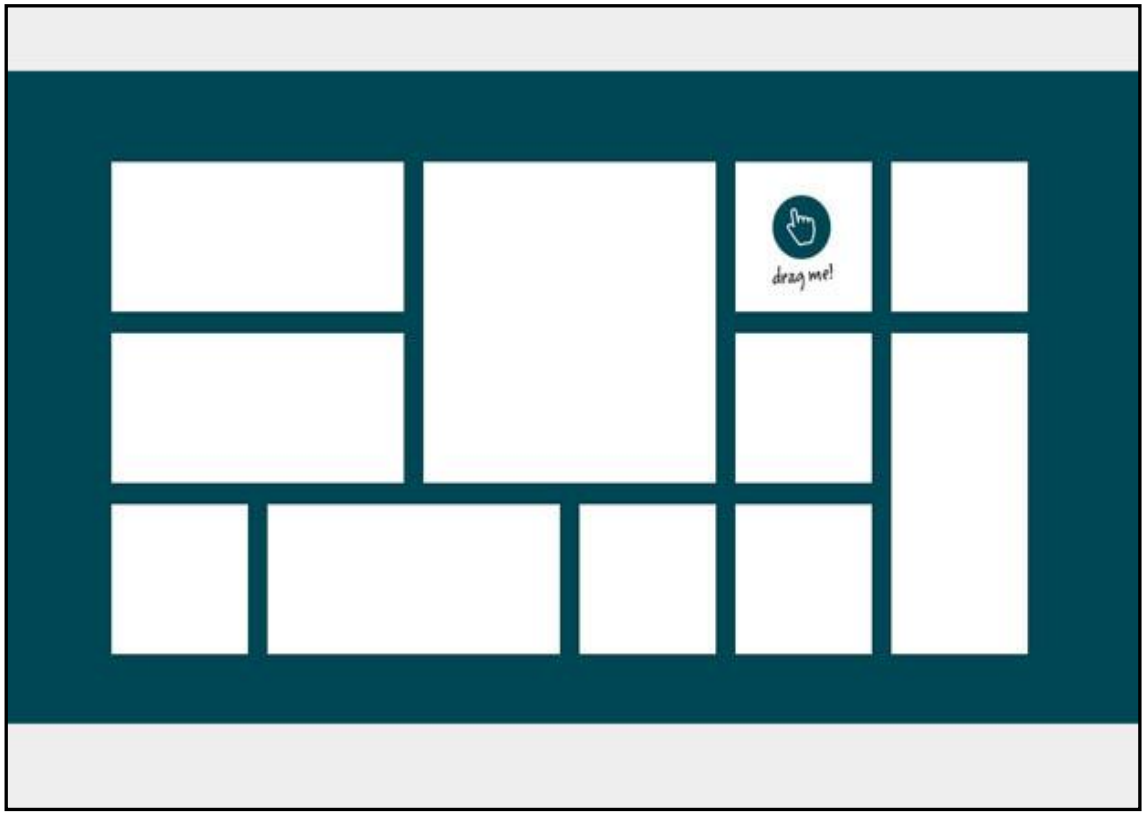



***Ilustración 146:*** *Widgets - Gridster*

Con esta interacción de **jQuery** podemos mover las cajas blancas de la imagen. De este modo podemos hacer partícipe al usuario de nuestra web. Cada caja que desplacemos provocará un reajuste automático de las demás, lo cual se puede configurar modificando los parámetros de la librería.

Como hemos dicho, podría ser interesante para aumentar la interactividad del usuario en nuestra web, aunque, por otro lado, una vez recargada la página, las cajas vuelven a su estado inicial.

# 14. Tubular

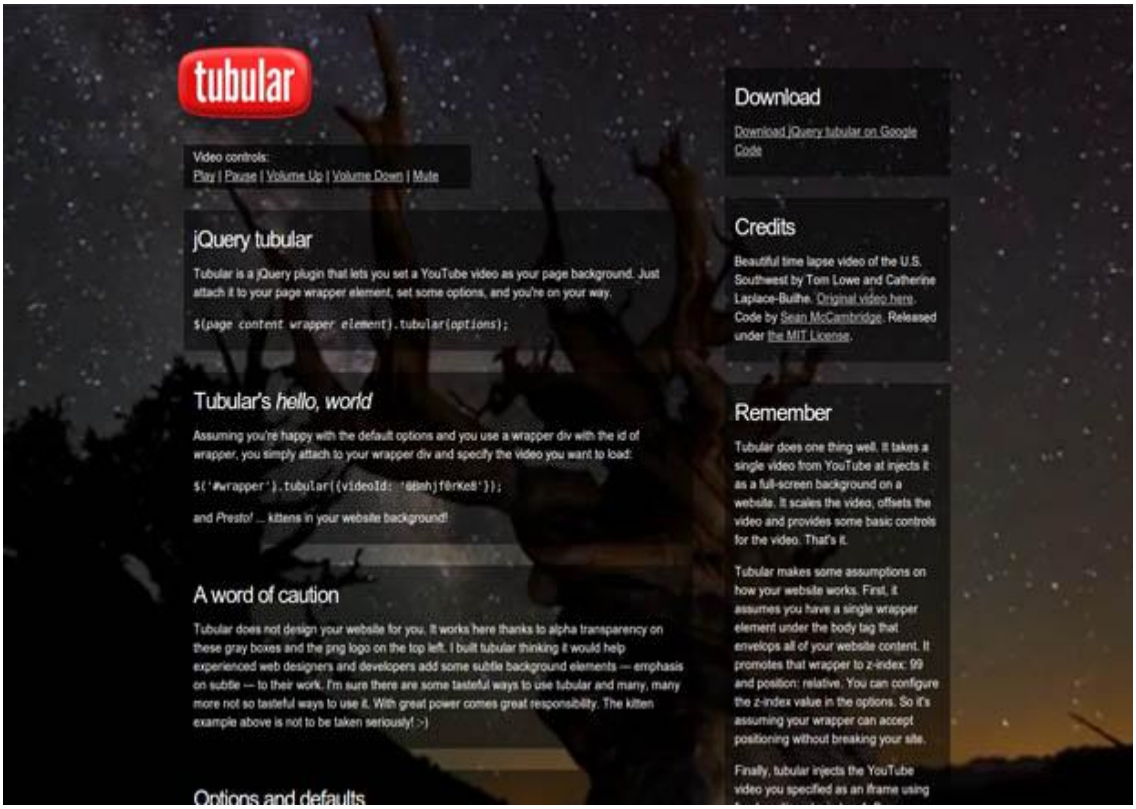



***Ilustración 147:*** *Widgets - Tubular*

Ésta es una de las interacciones más interesantes que podemos encontrar en la librería **jQuery**. Consisten en poner un vídeo de Youtube de fondo en nuestra página. Para ello sólo tenemos que invocar la función correspondiente con el id del vídeo que queramos establecer.

La principal desventaja es que ralentiza mucho el uso de la web y se requiere de un ordenador potente para ver el vídeo sin cortes. Sin embargo, el resultado queda muy atractivo, aunque también puede distraer la lectura o el visionado del web.

# 15. TillShift


***Ilustración 148:*** *Widgets - TillShift con filtro*


***Ilustración 149:*** *Widgets - TillShift sin filtro*

Esta interacción es otra de las que ofrece **jQuery** y resulta bastante llamativa. Consiste en la aplicación de los filtros que ofrece **CSS3** a las imágenes. Esta librería ofrece una amplia gama de posibilidades para la aplicación de filtros. Podemos desde aplicar filtros por defecto (imagen de la izquierda) y eliminarlos cuando ubiquemos el puntero sobre ella (imagen de la derecha) hasta modificar el tiempo que transcurre en dicha interacción.

TillShift consigue un efecto ‘enfoque’ que puede simular movimiento y parecer un vídeo o un gif, elementos muy de moda en las webs actuales y que, con poco peso, pueden añadir valor a la página.

# 16. JQueryPointPoint

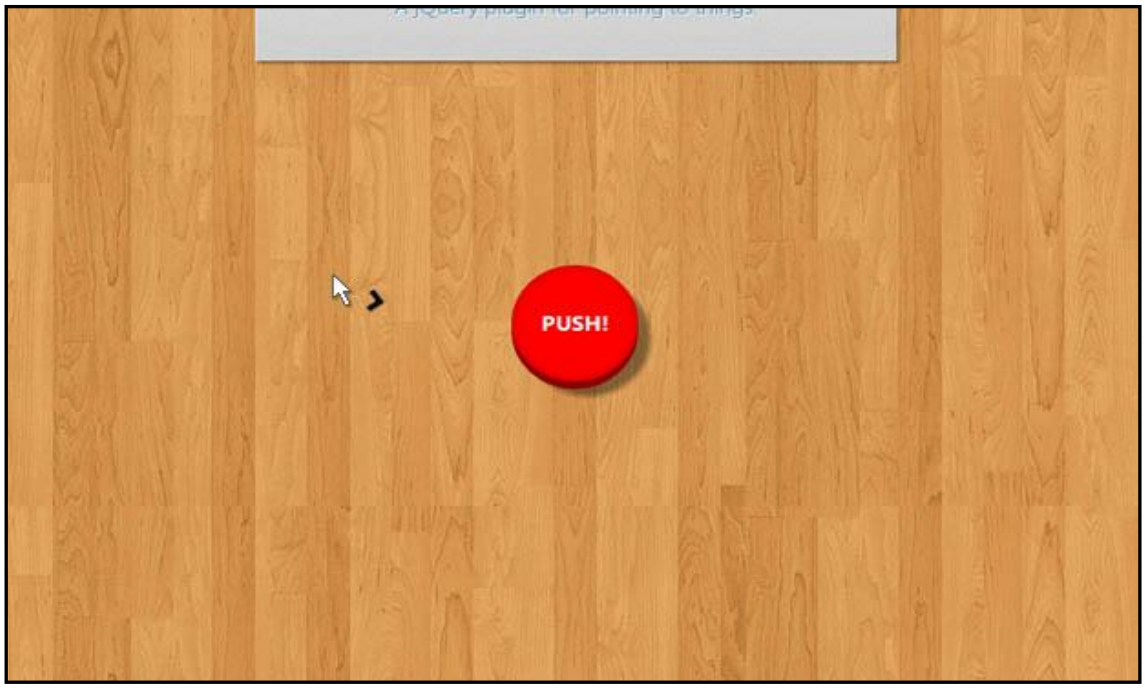



***Ilustración 150:*** *Widgets - JQueryPointPoint*

Una nueva interacción de **jQuery** que nos sirve claramente para llamar la atención del usuario. Si queremos que un usuario preste especial atención a algún elemento de nuestra web éste es el widget que necesitamos. Básicamente, se genera un símbolo con forma de flecha que se sitúa entre el puntero y el elemento que queramos resaltar. Lo particular de esta 'flecha' es que siempre apuntará hacia el elemento, independientemente del lugar de la web en el que nos encontremos, lo que hará imposible que el elemento pase desapercibido para el usuario.

La parte negativa de esta interacción es que a veces desaparece dicho símbolo y lo encontramos en una zona de la web totalmente alejado del puntero. Esto se debe a que hay muchos factores que influyen en esta interacción, así como la resolución o el css que se use en la página.

# 17. GmapsJS

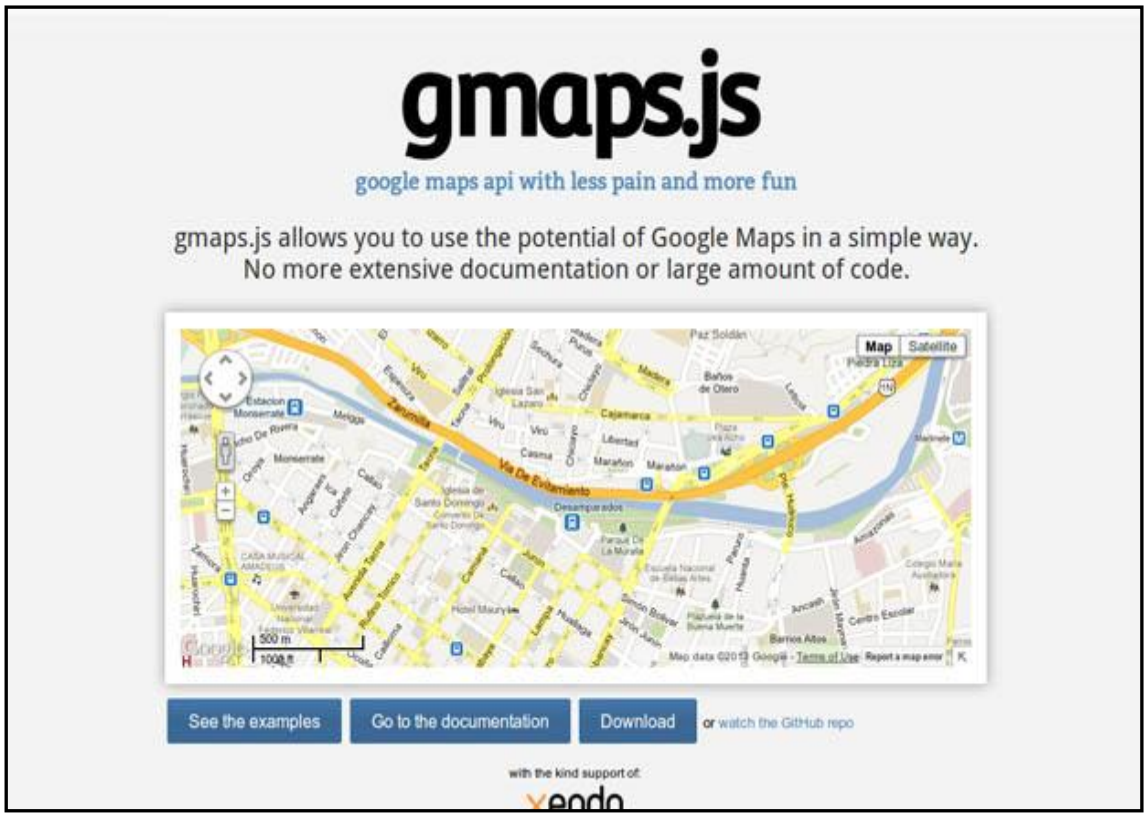



***Ilustración 151:*** *Widgets - Gmaps.js*

Esta interacción de **jQuery** es muy simple pero puede resultar útil para los usuarios. Como indica el nombre, es una librería **JavaScript** de **jQuery** que usa Google Maps, lo cual puede sernos de gran utilidad para insertar un mapa en nuestra página web con la ubicación de nuestra empresa. De este modo, podemos facilitar a nuestros clientes el contacto con la empresa.

Además, no se trata de una imagen plana; el mapa puede aumentarse o reducirse. Lo único que sería deseable es que también proporcionara la opción de ‘Cómo llegar’, para establecer rutas desde los lugares de procedencia de los clientes, y así facilitarles, por ejemplo, a los distribuidores su camino.