



***Facultad  
de  
Ciencias***

**ANÁLISIS, DISEÑO, REALIZACIÓN Y  
GESTIÓN DE UNA BASE DE DATOS CON  
ELEMENTOS OPEN SOURCE Y CON  
DIVERSIDAD DE DISPOSITIVOS CLIENTES**  
(Analysis, design, implementation and  
management of a database with open source  
elements and with a variety of client devices)

Trabajo de Fin de Grado  
para acceder al

**GRADO EN INGENIERÍA INFORMÁTICA**

Autor: Víctor Lavid Gutiérrez

Director: Rafael Menéndez De Llano Rozas

Junio - 2016



## Tabla de contenido

Agradecimientos .....	VII
Resumen .....	IX
Abstract .....	X
1 INTRODUCCIÓN Y OBJETIVOS.....	- 1 -
1.1 INTRODUCCIÓN .....	- 1 -
1.2 OBJETIVOS .....	- 1 -
1.3 CASO PRÁCTICO DE LA IMPLMENTACIÓN .....	- 2 -
1.4 ESTRUCTURA DE LA MEMORIA .....	- 2 -
2 HERRAMIENTAS Y TECNOLOGÍAS.....	- 4 -
2.1 Estado del arte .....	- 4 -
2.2 Lenguajes de programación .....	- 4 -
2.2.1 HTML4.1/HTML5 5 .....	- 4 -
2.2.2 JAVASCRIPT .....	- 5 -
2.2.3 CSS3 .....	- 6 -
2.3 Tecnologías.....	- 6 -
2.3.1 GMAP 3.....	- 6 -
2.3.2 Selenium.....	- 7 -
2.4 Herramientas.....	- 8 -
2.4.1 Visual Studio Code.....	- 8 -
2.4.2 GIT .....	- 8 -
2.4.3 PUTTY .....	- 8 -
2.4.4 FileZilla.....	- 9 -
2.4.5 Dia .....	- 9 -
2.4.6 Office .....	- 9 -
2.4.7 Kanban tool .....	- 9 -
2.5 Desarrollo full stack MEAN.....	- 10 -
2.5.1 NODE.JS .....	- 11 -
2.5.1.1 Problemas que resuelve NODE.JS .....	- 12 -
2.5.3 Descripción Express.js .....	- 12 -
2.5.1.2 ¿Por qué EXPRESS?.....	- 13 -
2.5.2 AngularJS.....	- 14 -
2.5.2.1 Descripción AngularJS .....	- 14 -
2.5.2.2 Problemas que soluciona AngularJS.....	- 15 -

2.5.3 MongoDB.....	- 16 -
2.5.3.1 Bases de datos (BD).....	- 16 -
2.5.3.1.1 Base de datos relacionales.....	- 16 -
2.5.3.1.2 Bases de datos documentales.....	- 17 -
2.5.3.2 Description MongoDB .....	- 17 -
2.5.4 Paquetes NODE.JS .....	- 18 -
2.6 Metodologías .....	- 19 -
2.6.1 Metodologías tradicionales.....	- 19 -
2.6.2 Metodologías Agiles .....	- 20 -
2.6.2.1 KANBAN.....	- 20 -
3. DESARROLLO DEL PROYECTO .....	- 22 -
3.1 Análisis del sistema .....	- 22 -
3.1.1 Actores .....	- 22 -
3.2 Requisitos funcionales del sistema .....	- 23 -
3.3 Requisitos no funcionales .....	- 27 -
3.4 Especificación de casos de uso.....	- 28 -
3.4.1 Área de administración .....	- 29 -
4 ESTRATEGIA DE DISEÑO E IMPLEMENTACIÓN .....	- 33 -
4.1 Descripción de la arquitectura .....	- 33 -
4.2 ¿Que arquitecturas tiene nuestro proyecto?.....	- 34 -
4.2.1 Descripción MVC .....	- 34 -
4.2.2 Flujo de control MVC.....	- 34 -
4.3 Diagrama despliegue.....	- 35 -
4.4 Diagrama de componentes .....	- 36 -
4.5 Implementación .....	- 36 -
4.5.1 Vista.....	- 37 -
4.5.1.1 JADE.....	- 38 -
4.5.2 Controlador .....	- 40 -
4.5.3 Modelo .....	- 42 -
5 PRUEBAS Y DESPLIEGLE .....	- 44 -
5.1 Pruebas.....	- 44 -
5.1.1 MonchaJs.....	- 45 -
5.1.2 Webdriver.io.....	- 45 -
5.1.2 ApacheBench.....	- 46 -
5.2 Despliegue .....	- 47 -

5.2.1 Especificaciones del entorno.....	- 47 -
5.2.2 Configuración del entorno .....	- 48 -
5.2.2.1 Servidor FTP.....	- 48 -
5.2.2.1 Base de datos MongoDB .....	- 49 -
5.2.2.1 NODE.JS .....	- 49 -
5.2.2 Puesta en producción.....	- 50 -
6 CONCLUSIONES Y TRABAJOS FUTUROS .....	- 51 -
6.1 Conclusiones.....	- 51 -
6.2 Líneas futuras .....	- 52 -
7 REFERENCIAS.....	- 53 -
ANEXO I .....	- 55 -
Caso práctico .....	- 55 -
1. Entrar en la plataforma .....	- 55 -
2. Acceso al área de administración.....	- 55 -
3. Área privada. ....	- 56 -
4. Incidencias.....	- 57 -
4.1. Introducir nueva incidencia.....	- 57 -
4.2. Ver incidencias .....	- 58 -

## Lista de acrónimos

- (TFG) Trabajo final de grado
- (JS) JavaScript
- (CMS) Content Management System
- (HTML) Hyper Text Markup Language
- (WHATWG) Web Hypertext Application Technology Working Group
- (DB) Data Base
- (CSS) Cascading Style Sheets
- (W3C) World Wide Web Consortium
- (XML) eXtensible Markup Language
- (CSV) Comma-Separated Values
- (IDE) Integrated Development Environment
- (SSH) Secure Shell
- (FTP) File Transfer Protocol
- (UML) Unified Modeling Language
- (MEAN) MongoDB, ExpressJS, AngularJS, NODE.JS (web application framework)
- (ASP) Active Server Page (Microsoft script engine)
- (OS) Operating System
- (POO) Object Oriented Programming
- (RAM) Random-Access Memory
- (API) Application Programming Interface
- (HTTP) Hypertext Transfer Protocol
- (MVC) Model–View–Controller
- (SQL) Structured Query Language (database query language)
- (BSON) Binary Structured Object Notation
- (ORM) Object Relational Mapping
- (CRUD) Create Read Update Delete
- (RUP) Rational Unified Procces
- (MSF) Microsoft Solution Framework
- (JSON) JavaScript Object Notation
- (SSD) Solid-State Drive
- (VPS) Virtual Private Server
- (SSL) Secure Sockets Layer
- (URL) Uniform Resource Locator

## Lista de ilustraciones

Ilustración 1. Logotipos de los CMS más populares.....	- 4 -
Ilustración 2. Logotipo de HTML 5. ....	- 5 -
Ilustración 3. Logotipo de JavaScript.....	- 6 -
Ilustración 4. Logotipo de CSS3. ....	- 6 -
Ilustración 5. Logotipo de Google Maps. ....	- 7 -
Ilustración 6. Logotipo de Selenium.....	- 7 -
Ilustración 7. Logotipo de Visual Studio Code.....	- 8 -
Ilustración 8. Logotipo de GIT. ....	- 8 -
Ilustración 9. Logotipo de FileZilla.....	- 9 -
Ilustración 10. Logotipo de DIA. ....	- 9 -
Ilustración 11. Logotipo de Kanban Tool.....	- 10 -
Ilustración 12. Arquitectura de un desarrollo MEAN.....	- 11 -
Ilustración 13. Logotipo de NODE.JS. ....	- 11 -
Ilustración 14. Logotipo de Express.....	- 13 -
Ilustración 15. Ilustración de una comparativa entre MEAN y Meteor.Js. ....	- 14 -
Ilustración 16. Logotipo del framework AngularJS ....	- 15 -
Ilustración 17. Logotipo de MongoDB.....	- 18 -
Ilustración 18. Imagen del tablero de la aplicación kanbantool.com. ....	- 21 -
Ilustración 19. Reporte del estado de las tareas en el método Kanban. ....	- 21 -
Ilustración 20. Casos de uso de alto nivel. ....	- 23 -
Ilustración 21. Casos de uso del área de administración. ....	- 29 -
Ilustración 22. Casos de uso del sistema global. ....	- 30 -
Ilustración 23. Diagrama de despliegue de la aplicación. ....	- 35 -
Ilustración 24. Diagrama de componentes. ....	- 36 -
Ilustración 25. Imagen del código de renderizado de una plantilla Jade.....	- 37 -
Ilustración 26. Imagen de la plantilla utilizada en la aplicación.....	- 39 -
Ilustración 27. Imagen de la plantilla utilizada para el área de administración.....	- 39 -
Ilustración 28. Mapa de la aplicación web.....	- 40 -
Ilustración 29. Código para realizar la autenticación a la aplicación. ....	- 40 -
Ilustración 30. Código para la resolución de peticiones no autenticadas.....	- 41 -
Ilustración 31. Controladores de la aplicación.....	- 41 -
Ilustración 32. Código para renderización de una vista. ....	- 41 -
Ilustración 33. Código para la conexión con la BD. ....	- 43 -
Ilustración 34. Modelo de hostelería en Mongoose. ....	- 43 -
Ilustración 35. Comando para realizar el benchmark del servidor. ....	- 46 -
Ilustración 36. Resultado del test al servidor NODE.JS. ....	- 46 -
Ilustración 37. Gráfico del tiempo de respuesta por el número de peticiones. ....	- 47 -
Ilustración 38. Página principal de la aplicación. ....	- 55 -
Ilustración 39. Página de acceso al área privada. ....	- 56 -
Ilustración 40. Formulario del CMS de información. ....	- 56 -
Ilustración 41. Área de nueva incidencia. ....	- 57 -
Ilustración 42. Introducir información de la incidencia. ....	- 57 -
Ilustración 43. Lista de incidencias.....	- 58 -

## Agradecimientos

Quiero aprovechar estas líneas para agradecer a todas las personas que me han ayudado y me han apoyado a lo largo de estos años de dura andadura por la universidad de Cantabria.

En primer lugar quiero agradecer a mis familiares, desde mis padres, mis hermanos, novia... Que han estado ayudándome y apoyándome y siempre a mi lado, desde que empecé a estudiar esta carrera como ingeniero informático.

Quiero mostrar mis más sinceros agradecimientos a mis compañeros de clase en especial a Gelo, que gracias a él, la carrera se lleva más amena. Estos estudios no solo han sido la forma de formarme como ingeniero informático, sino que en ellos he encontrado muchas cosas más. Me ha formado, me ha hecho que madurara y he encontrado unos buenos amigos.

No quería pasar por alto la oportunidad de agradecer a mi jefe de proyecto Rafael Menéndez y a todas los profesores que he tenido durante mi vida académica, y en especial a los que he tenido durante en la carrera, que me han ayudado a que sea la persona que soy ahora.

Por todo esto quiera daros las gracias.





## Resumen

En este trabajo de fin de grado (TFG), se van a definir las diferentes fases que se llevan a cabo para un desarrollo web. Se definirá el análisis, diseño, implementación y las pruebas de una aplicación para la gestión de incidencias y gestión de contenidos para diversos sectores.

Con el aumento del uso de internet en la vida cotidiana, es importante para cualquier empresa estar presente en la red. A su vez, es importante que sus clientes puedan notificar las incidencias producidas en su entorno de forma rápida y en tiempo real para que la propia empresa lo pueda solucionar lo antes posible.

Esta aplicación es una forma eficaz de organizar y tener controladas todas estas notificaciones, además de poder gestionar el contenido que la compañía quiere tener visible en internet.

La aplicación web esta implementada en un desarrollo *full stack mean*, en el cual la mayoría del desarrollo está realizado mediante el lenguaje de programación JavaScript. El motor central de la aplicación se basa en NODE.JS, que se encarga de implementar el servidor de la aplicación.

La metodología de desarrollo ha sido realizada mediante el método Kanban, con el cual garantizamos la entrega de las partes más importantes del proyecto en el tiempo establecido.

Al ser un desarrollo de una aplicación web basada en el *framework Express* la arquitectura de la aplicación es Modelo Vista Controlador, es más usual a la hora de realizar una aplicación de este estilo además de que Express apoya este tipo de desarrollo.

**Palabras clave:** Aplicación web, NODE.JS, JavaScript, Express framework, Kanban, modelo vista controlador (MVC).

## Abstract

This Degree Final Project (DFP), it will define the different steps for implement a web development. In this document will be explained how we can make an analysis, design, implementation and testing for an application for incident and CMS for any sectors.

The use of internet is increasing in all processes in daily life, all companies want to be in Internet because it is very important for them. At the same time, it is essential that their clients can notify the incidents quickly and in real time.

This application is an effective way to organize and have controlled all notifications. The company can manage the content that wants to be present on the Internet.

The web application is carry out in a full *development stack mean*, all development is write with javascript programming language. The central engine of application server is based on NODE.JS.

In development methodology has been used the Kanban method, it is a good way to guarantee that the parts of project are delivered on time.

This development is an application web that it is based in Express framework. The architecture is MVC.

**Keywords:** Web application, NODE.JS, JavaScript, Express framework, Kanban, MVC.

## 1 INTRODUCCIÓN Y OBJETIVOS

En este primer apartado se introducirá al lector en la materia del proyecto, metodología de desarrollo, objetivos que se pretenden conseguir y medios técnicos a utilizar. Así mismo, se expondrá la estructura que sigue este documento.

### 1.1 INTRODUCCIÓN

En el pasado siglo, el ordenador e internet sólo estaban permitidos en el ámbito profesional o académico, en cambio en los últimos años hemos podido comprobar el incremento de dispositivos que están conectados a la red de internet. Esto es un factor importante en el cambio de la comunicación entre personas y entidades, y cada vez más personas están adoptando internet como un elemento cotidiano.

Uno de los cometidos principales para una empresa o entidad pública, es que los clientes estén informados sobre temas relativos a la organización y que a su vez puedan opinar e informar sobre el estado de la misma.

A menudo las entidades públicas o empresas registran numerosas incidencias en papel o incluso en correos electrónicos, todas ellas sin una correcta organización y coordinación.

Muchas quedan en el olvido por una mala administración o gestión, gracias a la tecnología tenemos muchos medios de centralizar toda esa información y mantenerla controlada.

Por otro lado es importante que cualquier organización este visible en internet, ya que a día de hoy si no estás en internet, no existes. Es importante que aparte de informar a los visitantes de la información más relevante, la plataforma tenga disponibles una serie de servicios añadidos.

Con la aplicación propuesta pretendemos que haya una presencia de cualquier tipo de empresa o entidad pública en internet, de la misma manera que la gestión de incidencias sea más eficiente. Es de vital importancia que este servicio esté unificado en una sola aplicación, ya que esto permitirá un trato más cercano con los clientes.

### 1.2 OBJETIVOS

En este proyecto, se proponen una serie de soluciones para solventar los problemas diagnosticados. Se darán los pasos para diseñar, implementar y desplegar una aplicación web, que permita a cualquier empresa o entidad pública gestionar la

información que quiere tener en internet a su vez que gestionar incidencias detectadas por sus clientes.

A continuación se definen de manera detallada los objetivos mínimos que deben cumplirse:

- Gestionar toda la información con la que la entidad quiere estar presente en internet.
- Construir una aplicación funcional y accesible para distintos dispositivos web.
- Conexionar con la información de la Wikipedia.
- Construir un área de gestión de incidencias para los vecinos.
- Herramienta de geolocalización de las incidencias en tiempo real.
- Servidor de correo electrónico: que permita informar al ayuntamiento de incidencias en tiempo real.

Además cumplimos otros objetivos, como poner en práctica el método de desarrollo iterativo e incremental, que se basa en dividir el Proyecto en sub-proyectos e ir aplicando interacciones en cada uno de ellos.

### 1.3 CASO PRÁCTICO DE LA IMPLEMENTACIÓN

A la hora de implementar un caso práctico de esta aplicación hemos escogido el de un ayuntamiento.

En un ayuntamiento en el cual la mayoría de sus vecinos están conectados a internet, es imprescindible que esté de los servicios adecuados además de ofrecer soluciones tecnológicas para su municipio. Es importante que la información esté actualizada, de esta manera, todo ciudadano estará informado de lo que acontezca en su municipio.

Otra cuestión importante, es que los vecinos puedan notificar libremente en un lugar centralizado todo aquello que el pueblo quiera hacer llegar a oídos de su ayuntamiento, sin la necesidad de tenerse que trasladar al mismo

En el pasado, los vecinos se enteraban de la información relacionada con el ayuntamiento mediante bandos pegados por las calles o incluso mediante el boca a boca. Con la implementación de esta herramienta, el pueblo podrá estar informado a la vez de que podrá notificar rápidamente una incidencia.

### 1.4 ESTRUCTURA DE LA MEMORIA

Vamos a enumerar los apartados en los que está dividido el trabajo de fin de grado:

- Apartado 1: Planteamos el problema al que vamos a dar solución y como lo vamos a hacer.
- Apartado 2: Recorremos el estado del arte actual. Tecnologías y herramientas utilizadas para dar solución al problema.
- Apartado 3: Describimos y analizamos los requisitos que se presentan en el problema.
- Apartado 4: Describimos la estructura de diseño de ingeniería de software para la solución, y la llevamos a cabo.
- Apartado 5: Describimos la evaluación, las pruebas de verificación y como desplegar la aplicación web.
- Apartado 6: Conclusión y líneas futuras.
- Apartado 7: Bibliografía.

## 2 HERRAMIENTAS Y TECNOLOGÍAS

En este segundo capítulo, describiremos el estado del arte, donde puntualizaremos las diferentes herramientas que hay en el mercado con las cuales podemos dar solución a nuestro desarrollo, en segundo lugar se detallan los lenguajes, las tecnologías y herramientas software utilizadas, por último, se describe la metodología escogida para dar solución al problema.

### 2.1 Estado del arte

Buscando por internet nos encontramos aplicaciones CMS (Content Management System) que implementan módulos para la realización de alguna de las características requeridas.

Alguna de estas herramientas que se adjuntan son Drupal, Wordpress o Joomla.



*Ilustración 1. Logotipos de los CMS más populares*

Cualquiera de estos CMS puede resolver algunos de los problemas que se nos presentan: editor de contenidos, gestión de usuarios, etc. Aunque la personalización y el posterior desarrollo de aplicaciones móviles con el mismo núcleo sería muy complejo.

### 2.2 Lenguajes de programación

#### 2.2.1 HTML4.1/HTML5

HTML [1] es un elemento de construcción de contenido web, un lenguaje de marcas que solamente define el contenido y estructura de la página, no su funcionalidad. Es un lenguaje interpretado por el navegador, encargado de mostrar el contenido web tal y como lo especifica el HTML.

Fue creado en 1990 y es un estándar internacional, otra características es que está regulado por *World Wide Web Consortium* y el *WHATWG*.

HTML5: es la versión utilizada en el proyecto, ya que facilita el desarrollo de componentes gracias a nuevas características que incorpora a la anterior versión, aunque versiones antiguas de los navegadores no den soporte a este lenguaje, todos los navegadores actuales poseen esta característica, además de que esta versión cumple con los requisitos no funcionales de la aplicación.

Algunas de las características de HTML5 son:

- Etiquetas sintácticas.
- Etiquetas multimedia.
- Nuevas capacidades en los formularios.
- Geolocalización



*Ilustración 2. Logotipo de HTML 5.*

### 2.2.2 JAVASCRIPT

JavaScript [2] es un lenguaje ligero, orientado a objetos e interpretado. En un primer momento se ha encargado de dar dinamismo a las web.

Hoy en día JavaScript se está utilizando para otros propósitos, como podemos ver en NODE.JS o Apache CouchDB. En nuestro caso JavaScript juega un papel muy importante, ya que es el lenguaje de programación con el que se realiza la aplicación del lado de servidor y la forma de acceder a la base de datos, además de ser la base del *framework* de la vista de la aplicación.

Algunas de las características de JavaScript son:

- Lenguaje dinámico.
- Soporta programación funcional, orientada a objetos e imperativa.
- Lenguaje no tipado.
- Lenguaje interpretado.





*Ilustración 3. Logotipo de JavaScript.*

### 2.2.3 CSS3

CSS [3] es el lenguaje encargado de describir la presentación de documentos HTML o XML, está regulado por la W3C, en este proyecto se ha utilizado la versión 3 ya que da más poder visual a la aplicación que su antecesora. Este lenguaje puede ir incrustado dentro del propio HTML al que le va a dar el estilo o en archivos aparte con extensión .css.

Algunas características de CSS3 son:

- Esquinas redondeadas o bordes redondeados
- Cajas con sombra
- Sombra para texto
- Múltiples columnas



*Ilustración 4. Logotipo de CSS3.*

## 2.3 Tecnologías

### 2.3.1 GMAP 3

GMAP 3 [4] es un API de desarrollo de componentes web basados en los mapas de google, dicho API proporciona los diferentes métodos para incorporar nuevos elementos dentro del mapa y a su vez poder manipularlo con facilidad.

Hace una década los mapas de google eran cerrados y solo podían realizar aplicaciones su propia empresa, a día de hoy es un servicio muy interesante a la hora de requerir un mapa en una aplicación.

Las principales características de GMAPS son:

- Posicionamiento de *markers* (marcadores) personalizados en los mapas de google.
- Eventos a los diferentes objetos del mapa.
- Herramientas de dibujo.
- Servicios de cálculo de rutas.



*Ilustración 5. Logotipo de Google Maps.*

### 2.3.2 Selenium

Selenium [5] es un entorno para la realización de pruebas, que permite automatizar procesos que un usuario cotidiano de la aplicación realiza, con esta herramienta podemos reproducir todas las tareas, pudiendo probar la funcionalidad de una aplicación web de forma automática.

Este entorno está compuesto por dos productos:

- Selenium WebDriver: gracias a este producto podemos conectar nuestra aplicación directamente con el navegador en el que se quieren realizar las pruebas, este driver esta implementado para diversos lenguajes como JAVA o JavaScript. Además permite la utilización de los navegadores más recientes para realizar las pruebas como Chrome o Mozilla.
- Selenium IDE: es un software aparte de la aplicación que se desarrolla, posee un entorno para generar secuencias en aplicaciones web, únicamente indicando que acciones tiene que realizar de forma intuitiva sin tener que desarrollar ningún código en nuestra aplicación.

En nuestro caso hemos utilizado Selenium WebDriver, ya que nos proporciona la facilidad de tener los test de las pruebas en el mismo código de la aplicación y de esta manera integrarlo de forma continua, además nos da la versatilidad de realizar pruebas más concluyentes de nuestra aplicación.



*Ilustración 6. Logotipo de Selenium.*

## 2.4 Herramientas

### 2.4.1 Visual Studio Code

Es un entorno de desarrollo de código libre desarrollado por Microsoft, este IDE es idóneo para trabajar con JavaScript, en nuestro caso el 70% del código fuente esta implementado con este lenguaje, posee soporte para depurar y además tiene embebido un control de versiones GIT muy importante cuando estamos trabajando con software.

Otra de las características importantes es que posee un buen resaltado en la sintaxis de los lenguajes de programación que se utilizan para resolver el problema planteado.



*Ilustración 7. Logotipo de Visual Studio Code.*

### 2.4.2 GIT

Es un software de control de versiones de código libre, con el cual tendremos un historial de cambios en los diferentes ficheros de nuestra aplicación, una característica importante es que es distribuido por lo que tenemos la facilidad de desarrollar el proyecto en diferentes máquinas.



*Ilustración 8. Logotipo de GIT.*

### 2.4.3 PUTTY

Putty [6] es un cliente SSH y telnet de código abierto que nos permite acceder a servidores remotos.



*Ilustración 9. Logotipo de PUTTY.*

Utilizamos este cliente para poder acceder al servidor en el que tenemos instalado el servidor NODE.JS y para poder configurar la base de datos documental, además es una forma sencilla de poder configurar las diferentes partes de las que depende la aplicación.

#### 2.4.4 FileZilla

FileZilla [7] es un cliente de FTP libre que nos permite acceder a los directorios de un servidor remoto, permite añadir, modificar o borrar archivos que se encuentran en el servidor.



*Ilustración 9. Logotipo de FileZilla.*

Con este programa pondremos en producción en el servidor remoto nuestra aplicación.

#### 2.4.5 Dia

Aplicación libre que nos sirve para modelar diferentes tipos de diagramas de forma fácil, gracias a este programa hemos modelado los diferentes diagramas software de la aplicación.

Algún ejemplo de los diagramas que se pueden modelar son: UML, Casos de uso...



*Ilustración 10. Logotipo de DIA.*

#### 2.4.6 Office

Es una suite de ofimática creada por Microsoft, con esta herramienta podemos crear todo tipo.

#### 2.4.7 Kanban tool

Se trata de una herramienta para poder controlar el seguimiento de un desarrollo basado en el método Kanban [20] de forma online, de manera que podemos tener

tableros de trabajo en los cuales mi jefe de proyecto puede seguir mis diferentes evoluciones.

Aparte de crear diferentes tableros para seguir el método, nos da la facilidad de crear gráficas indicando los diferentes estados de la aplicación.



*Ilustración 11. Logotipo de Kanban Tool.*

Es una herramienta perfecta, ya que es gratuita hasta dos usuarios.

## 2.5 Desarrollo full stack MEAN

En el apartado anterior hemos comentado, las diferentes tecnologías con las cuales se ha desarrollado la aplicación web, para la implementación de esta aplicación se ha optado por un desarrollo *Full Stack MEAN* [8]. El acrónimo MEAN viene de (M)ongo + (E)xpress + (A)ngular + (N)ode, en este caso todas las partes que componen la aplicación (frontEnd, backEnd) y la base de datos están desarrolladas con el lenguaje de programación JavaScript, por lo que un punto a favor es que solo debemos aprender un lenguaje.

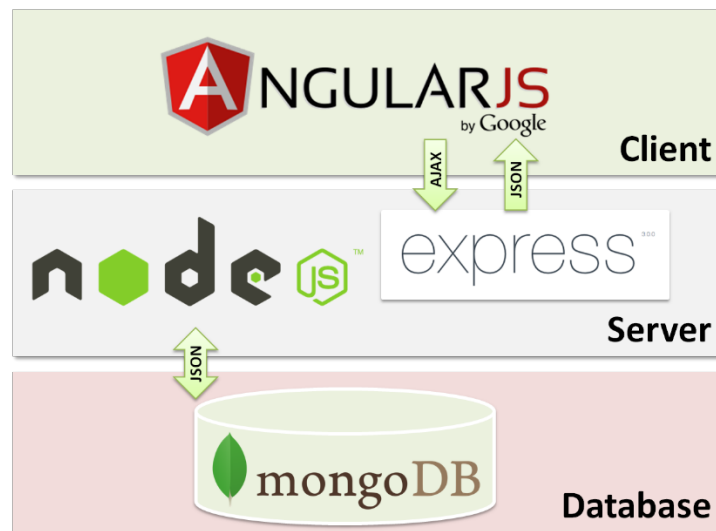
Este tipo de desarrollos apareció gracias a la implementación de NODE.JS en 2009, el lenguaje JavaScript se creó para realizar tareas menores en el navegador, aunque se intentaron crear alternativas a JavaScript para el lado del servidor pero no triunfaron, algunas de estas tecnologías fueron: lenguaje propio de NETSCAPE o ASP de Microsoft.

JavaScript es un lenguaje de programación con el mismas características que C# o Java, una de las cosas interesantes que tiene es su orientación a eventos, además de una comunidad amplia de desarrolladores que dan soporte.

Al estar construidas todas las partes en JavaScript, podemos portar fácilmente librerías de utilidades desde la capa de persistencia hasta el *frontEnd*, una de las cosas destacables es el uso del motor V8 del navegador Chrome, que es el motor principal que transforma el código JS a código máquina. Este motor ha sido perfeccionado al cabo del tiempo, ya que es la pieza fundamental del navegador Chrome (navegador más utilizado hoy en día).

Uno de los puntos más importantes es que la curva de aprendizaje es muy corta, ya que cualquier desarrollador que conozca JavaScript puede construir todas las partes de la aplicación web.

En la ilustración 12 podemos ver las diferentes partes que componen un desarrollo full stack MEAN.



*Ilustración 12. Arquitectura de un desarrollo MEAN.*

### 2.5.1 NODE.JS

Es un entorno de tiempo de ejecución y una librería para el lenguaje JavaScript, NODE.JS [9] utiliza la máquina virtual V8 de google que es el intérprete del lenguaje JavaScript y la encargada de transformar ese lenguaje a lenguaje máquina, este motor es el mismo que utiliza Google para sus navegadores, utiliza una programación orientada a eventos, la entrada y salida no es bloqueante, además de ser de código abierto, está disponible para las plataformas más populares: Windows, Linux y Mac OS.

Utiliza un gestor de paquetes NPM en el cual encontramos un gran ecosistema de librerías de *open source*.

Las características más importantes de NODE.JS son:

- Programación Asíncrona.
- POO.
- Open Source.
- Lazo de eventos.



*Ilustración 13. Logotipo de NODE.JS.*

### 2.5.1.1 Problemas que resuelve NODE.JS

Como se ha definido anteriormente JavaScript, es un lenguaje de programación que tiene un excelente modelo para la programación orientada a eventos, este paradigma de programación es muy apreciado en aplicaciones web.

Una de las cosas que queríamos conseguir es una aplicación escalable, ya que es una implementación en la que el número de usuarios puede ir creciendo con el tiempo. Además de que puede ir evolucionando, ofreciendo cada vez más servicios a sus usuarios.

Actualmente las soluciones tradicionales con lenguajes JAVA o PHP [10] crean un hilo con 2MB de memoria RAM por cada conexión, por lo que este tipo de implementaciones nos ponen limitaciones en cuanto a la escalabilidad en un futuro, ya que para poder mantener la concurrencia deberíamos ampliar las prestaciones del servidor en memoria y procesador.

NODE.JS soluciona este problema ya que posee un *event loop* (bucle de eventos) que está compuesto por una cola en la que se van añadiendo los eventos, este bucle de eventos resuelve el problema cambiando la forma en la que se hace una conexión al servidor. En vez de generar un hilo, lo que hace es añadir en la cola un evento que será ejecutado cuando le llegue su turno, otro punto a destacar es que el *event loop* no tendrá tiempos muertos porque ni siquiera se bloqueará, en el caso de una operación de entrada/salida.

NODE.JS es un entorno basado en eventos, utiliza el motor V8 que está perfectamente optimizado para la conversión de JavaScript a lenguaje máquina y además posee un *event loop*. Todas estas cualidades hacen que NODE.JS sea rápido y escalable.

### 2.5.3 Descripción Express.js

Framework de desarrollo de aplicaciones web para NODE.JS [11] proporciona una API con multitud de métodos para HTTP y middleware, para una creación de aplicaciones rápida y sencilla, el rendimiento es alto ya que no oculta las características de NODE.JS.

Es una forma de desarrollo de aplicaciones web de forma rápida, clara y a la vez avalada por una comunidad. Te proporciona facilidades como:

- Direccionamiento
- Manejo de errores
- Depuración
- Conexión con base de datos



*Ilustración 14. Logotipo de Express.*

#### 2.5.1.2 ¿Por qué EXPRESS?

Cuando se planteó el proyecto se buscó en el panorama actual que tecnología era la más apropiada para el lado del servidor.

A causa de la complejidad de proyecto, se tuvo que dividir en dos partes, una primera, que fue la creación de una aplicación web, la cual englobaba al servidor. Éste, aparte de dar respuesta al *frontEnd* de la aplicación web, también tenía que dar respuesta a la segunda parte del proyecto, que es una aplicación para móviles.

Después de indagar en las diferentes soluciones y debatirlo con mi jefe de proyecto, llegamos a la conclusión de que la mejor solución sería la utilización de un *framework* JavaScript para el lado del servidor que nos permitía dar la velocidad y concurrencia necesarias, además de la baja curva de aprendizaje.

Después de nuestra elección del lenguaje de programación, indagamos los diferentes *framework*:

- Express
- Meteor
- SailsJs
- SocketStream
- ActionHero
- Derby

Una vez recorridos los diferentes *framework* para NODE.JS, nos quedamos con los dos más populares y orientado a soluciones generales: Express y Meteor.

Vamos a realizar una comparativa entre ambos:



Express	Meteor
Es muy ligero y solo se encarga de la parte de backEnd	Es un framework full stack que se encarga de la parte de backEnd como la del frontEnd
Estable	Está madurando actualmente
Gran comunidad	Comunidad menos numerosa
Poca abstracción	Alta abstracción

Tras leer varios documentos sobre los diferentes *framework* se llegó a la conclusión de que Express es el mejor para nuestro propósito, es un *framework* liviano con mucha potencia y además muy buena forma de aprender la plataforma NODE.JS.

Gracias a este *framework* nos simplifica las tareas a la hora de realizar cualquier proceso habitual en una aplicación de web, además de hacernos comprender como funciona NODE.JS al no abstraer los procesos como lo hace Meteor.

Podríamos comparar estos *frameworks* como Meteor un coche montado y Express como las piezas de un coche sin montar.



Ilustración 15. Ilustración de una comparativa entre MEAN y Meteor.Js.

Es destacable mencionar que Meteor hace la parte de *frontEnd* como la de *backEnd* y en el caso de Express sólo se dedica al *backEnd*.

Nosotros queríamos independizar las diferentes partes del proyecto y utilizar el *framework* que más nos conviene para cada parte.

## 2.5.2 AngularJS

### 2.5.2.1 Descripción AngularJS

Framework de desarrollo del frontEnd de una aplicación web, AngularJS [12] implementa el patrón MVC, se utiliza para mantener aplicaciones web de una o pocas

páginas web, dotándolas de dinamismo al igual que una aplicación de escritorio, esto fue desarrollado Google.

Esta tecnología permite extender el vocabulario HTML de nuestra aplicación, con el propósito de ahorrarnos tiempo en el desarrollo al igual que darle dinamismo de una forma sencilla a nuestra aplicación web.

Algunas de las características de AngularJS son:

- Patrón MVC.
- Plantillas.
- Directivas.
- Dinamismo.



*Ilustración 16. Logotipo del framework AngularJS*

#### 2.5.2.2 Problemas que soluciona AngularJS

Antes de aparecer internet todo el dominio de aplicaciones era de escritorio, una vez que apareció internet y los navegadores, ese panorama empezó a cambiar el uso de aplicaciones web era cada vez más importante.

En el inicio de internet los navegadores solo se encargaban de mostrar texto, pero con el paso de los años el propósito empezó a cambiar. Se empezó a introducir maquetado del texto para dar estilo mediante el lenguaje CSS y a dotarlo con dinamismo gracias a JavaScript.

Cuando las aplicaciones web empezaron a coger protagonismo frente a las aplicaciones de escritorio, comenzaron a aparecer *frameworks* para la parte del *frontEnd* que se encargaron de dar a las aplicaciones web un comportamiento parecido a los del software de escritorio.

Hoy en día encontramos multitud de *frameworks* de la parte del cliente que nos facilitan la creación de interfaces amigables, algunos de ellos son:

- BackboneJS
- EmberJS
- AngularJS
- JQuery

Después de un exhaustivo análisis nos quedamos con los dos mejores para nuestro proyecto que son AngularJS y BackboneJS. Vamos a comentar las diferentes propuestas de cada uno:

AngularJS	BackboneJS
<b>Comunidad alta y gran cantidad de módulos</b>	Menos comunidad y módulos
<b>Peso de la librería más alto</b>	Peso ligero de la librería
<b>Curva de aprendizaje alta</b>	Curva de aprendizaje baja
<b>Poder muy alto</b>	Poder más bajo

Después de comparar los diferentes *frameworks* para la parte del *frontEnd* nos decantamos por AngularJS, ya que la comunidad es más superior y la búsqueda de soluciones o documentación es mucho más sencilla, además en nuestro caso teníamos una gran formación en este *framework*, por lo que la curva de aprendizaje no es importante en nuestro caso.

Gracias a este *framework* hemos podido fragmentar el *frontEnd* en un MVC, un echo impensable cuando no existían este tipo de *frameworks* y trabajábamos únicamente con JavaScript.

### 2.5.3 MongoDB

#### 2.5.3.1 Bases de datos (BD).

Actualmente el panorama de la persistencia de datos es muy variopinto, los tipos que vamos a analizar son: el tradicional denominado bases de datos relacionales y un concepto más actual que son las bases de datos documentales. Estos dos tipos tienen formas diferentes de agrupar y persistir los datos, es importante analizar el tipo de aplicación que vamos a desarrollar para elegir uno u otro.

##### 2.5.3.1.1 Base de datos relacionales.

Las bases de datos relacionales [14] fueron desarrolladas por IBM en 1970. La forma de persistir los datos es gracias a tablas que están relacionadas entre sí, cada tabla está compuesta por una serie de columnas que pueden contener relaciones con otras.

Uno de los lenguajes más utilizados en este tipo de base de datos es el SQL. Hay numerosas tecnologías que implementan este tipo de BD: Mysql, Oracle, Microsoft Sql Server etc.

### 2.5.3.1.2 Bases de datos documentales

Este tipo de base de datos se originan por el cambio producido por la aparición de aplicaciones web, el uso de aplicaciones con millones de usuarios necesitan una alta escalabilidad, lo cual no quiere decir que una BD relacional no pueda ofrecer una alta escalabilidad, sino que este nuevo modelo de BD ofrece una mayor gestión a la hora de escalar.

En este caso la información se guarda en documentos en vez de tablas, los documentos de una BD no tienen relación con otros, este tipo de base de datos es un subconjunto de las BD No-Sql. Algún ejemplo de este tipo de base de datos son: MongoDB, CouchDB, Apache Cassandra etc.

Las diferencias fundamentales basándonos en los requerimientos de nuestra aplicación son:

BD Relacional	BD Documental
<b>Necesidad de esquemas</b>	Ausencia de esquemas
<b>Escalabilidad posible pero costosa</b>	Escalabilidad sencilla
<b>Velocidad alta</b>	Velocidad extraordinaria

Después de analizar los diferentes tipos de bases de datos en el mercado nos decidimos por utilizar una base de datos documental, nuestra aplicación necesita una alta escalabilidad y no necesita un esquema relacional, ya que no tenemos relaciones entre los diferentes datos de nuestra aplicación.

Una vez decidido el tipo de base de datos nos encontramos con diferentes implementaciones, en nuestro caso como la aplicación es full stack JavaScript, nos decantamos por la implementación MongoDB.

### 2.5.3.2 Description MongoDB

Sistema de bases de datos de tipo NO-SQL [13], orientado a documentos y de código abierto, este tipo de bases de datos no guardan el contenido en tablas como las BD relacionales, la forma de guardar los datos es mediante documentos tipo BSON.

Las características más importantes son:

- Orientada a documentos.
- Alta disponibilidad.
- Escalabilidad.
- Auto balanceado de carga.



Ilustración 17. Logotipo de MongoDB.

En nuestro caso para la realización del proyecto hemos utilizado la versión 3.2.4, la última versión estable.

#### 2.5.4 Paquetes NODE.JS

Como hemos comentado anteriormente, hay una amplia comunidad que está detrás de NODE.JS, gracias a la cual podemos encontrar numerosos paquetes que nos hacen la vida más fácil a la hora de hacer un desarrollo web y además de no tener que desarrollar todas las partes desde cero.

Algunos de los paquetes que hemos utilizado a la hora de desarrollar nuestra aplicación son:

- Mongoose ORM

Es un ORM para MongoDB [15], se encuentra en el gestor de paquetes NPM de NODE.JS. Mongoose, nos permite tener un acceso MongoDB mediante el lenguaje de JavaScript además de proporcionarnos un CRUD y otras operaciones cotidianas de manera fácil y sencilla.

- PassportJs

Es un middleware [16] para NODE.JS para realizar la autenticación, este módulo es extremadamente flexible a la hora de la autenticación, ya que podemos autenticar a un usuario con el apoyo de diferentes estrategias como usuario y contraseña, Facebook, Twitter... con muy poca complejidad. Este *framework* de autenticación es modular por lo que le podemos acoplar con el *framework* Express.

- MochaJs

Es un *framework* de desarrollo [17] de pruebas unitarias para NODE.JS con el cual hemos hecho pruebas para la parte del controlador del servidor, este *framework* facilita la depuración a la hora de realizar los test unitario.

- HTTP

Módulo que se encarga de darnos todas las facilidades para la creación de un servidor http, está incluido en el *core* de NODE.JS.

- Path

Este módulo contiene todas las utilidades de manipulación y transformación de rutas de los archivos que componen la aplicación, también está incluido en el *core* de NODE.JS.

- Nodemailer

Este módulo [18] se encarga de darnos todas las facilidades para enviar email, en nuestro caso es muy importante esta implementación, ya que le necesitamos para enviar las diferentes incidencias a los clientes de la aplicación.

## 2.6 Metodologías

Lo primero en lo que hay que enfatizar [19] es que una metodología de desarrollo software es un conjunto de reglas para poder estructurar, planificar y controlar un proyecto software.

Nos disponemos a realizar un recorrido entre las diversas metodologías que hay y ha habido a lo largo de la historia del desarrollo de software, en el inicio del software no existía ningún tipo de metodología ya que el software estaba creado para resolver problemas a la misma persona que los desarrolla. Con el paso del tiempo se empezó a pensar en la ingeniería del software porque cada vez el software evolucionaba y ocupaba un lugar más importante dentro de la tecnología, en ese momento se planteó la necesidad de realizar un conjunto de reglas para ofrecer una buena calidad al desarrollo.

El número de metodologías en el desarrollo del software es muy elevado a lo largo de la historia, por lo que vamos a agruparlas en: metodologías tradicionales y metodologías ágiles.

### 2.6.1 Metodologías tradicionales

El desarrollo software en sus inicios era rudimentario y se empezó por intentar definir una serie de etapas al desarrollo. Este tipo de metodologías en lo que se centraban es en imponer una metodología de trabajo en la cual lo más importante es cumplir el plan de proyecto, el cual que está definido estrictamente de principio a fin.

Actualmente estas metodologías tienen problemas con el tipo de desarrollo que se lleva a cabo, uno de los mayores problemas es que no se adaptan adecuadamente a cambios.

Algunos de las metodologías tradicionales más importantes son:

- RUP (Rational Unified Procces)
- MSF (Microsoft Solution Framework)
- Win-Win Spiral Model
- Iconix

### 2.6.2 Metodologías Agiles

En la actualidad, la realidad es que los requisitos de un proyecto son cambiantes e incluso en algunos casos, en un primer momento, se carecen de ellos. El desarrollo de software con metodologías tradicionales es impensable si se dan este tipo de circunstancias.

Para solucionar este tipo de problemas se desarrollaron nuevas metodologías, las metodologías ágiles, que se adaptan perfectamente a cambios según va avanzando el proyecto.

Hoy en día nos encontramos infinidad de metodologías. Indagando y estudiando las diferentes alternativas, no se puede concluir la superioridad de ninguna. Es importante pararnos a pensar en las diferentes alternativas a la hora de comenzar un nuevo proyecto, ya que una u otra se puede adaptar mejor a los requerimientos de nuestro proyecto.

Algunas de las metodologías ágiles más populares son:

- SCRUM
- Kanban

Después de comparar entre estas dos metodologías nos decantamos por Kanban, dado que es un proyecto de pequeña magnitud, por lo que no tiene sentido la entrega gradual que ofrece SCRUM. Es importante tener una cola de trabajos priorizados y una vez finalizados, probarlos y pasarlos a producción de forma paulatina.

Esto es una buena práctica para que nuestro jefe de proyecto vea cómo se van solucionando los diferentes requisitos de la aplicación. Es importante destacar que en Kanban no se producen iteraciones sobre el trabajo realizado, como en SCRUM, por lo que se centra en las tareas pendientes.

#### 2.6.2.1 KANBAN

Es un tipo de metodología japonesa [20] que cada vez está ganando más popularidad, nos facilita la gestión de una forma fluida. Uno de sus principios básicos es obtener el mayor rendimiento de su flujo de trabajo.

Este método se basa en una serie de tarjetas que indican las diferentes tareas a realizar, el color indica la prioridad de cada una de ellas, y se van colocando en diferentes filas según el estado en el que se encuentren. Los estados mínimos son 3: tareas para hacer, tareas haciéndose y tareas finalizadas. Esta clasificación se le puede añadir más estados según nuestras necesidades en el proyecto.



Ilustración 18. Imagen del tablero de la aplicación kanbantool.com.

En nuestro caso hemos utilizado el siguiente software que nos ha permitido seguir la evolución del proyecto [20].

Este software nos permite tener un control de la metodología Kanban de forma fácil y sencilla, mostrarnos reportes de la situación del proyecto de forma automática y además tenemos la posibilidad de exportarlo a CSV.

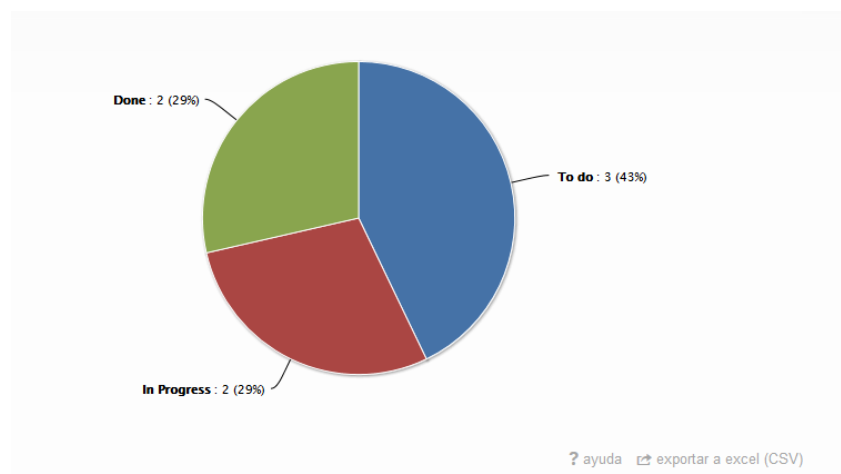


Ilustración 19. Reporte del estado de las tareas en el método Kanban.

En nuestro caso ha sido muy útil, ya que es una aplicación web y mi jefe de proyecto puede seguir la evolución del proyecto sin estar trabajando en el mismo sitio.



### 3. DESARROLLO DEL PROYECTO

Una vez definido los objetivos y las diferentes tecnologías con las que se desarrollará el proyecto, llega el momento de conocer los diferentes requisitos, el primer paso a realizar fue una reunión con los trabajadores del ayuntamiento y con mi jefe de proyecto para poder conocer todos los requisitos que formaran parte de la aplicación.

La persona del ayuntamiento que se encargaba de dirigir este proyecto, expuso la necesidad de un aplicativo para dar a conocer la región a sus visitantes a la vez de que los vecinos pudieran interactuar dando a conocer las diferentes incidencias que pasan a diario.

Una vez llegado a este punto y tras numerosas reuniones con las partes citadas, empezamos a capturar los diferentes requisitos del sistema además de empezar a dar forma a la estructura de la aplicación web.

En las siguientes secciones indicaremos los pasos a seguir para realizar un análisis al problema planteado, además de darle solución.

#### 3.1 Análisis del sistema

En este apartado vamos a analizar las diferentes funcionalidades que requiere el ayuntamiento para nuestro aplicativo.

En un primer momento vamos a analizar los actores que componen nuestro sistema, además detallaremos los casos de uso para poder obtener los requisitos del aplicativo.

Por último definiremos los diferentes requisitos funcionales y no funcionales del sistema que nos explicarán el funcionamiento del aplicativo.

##### 3.1.1 Actores

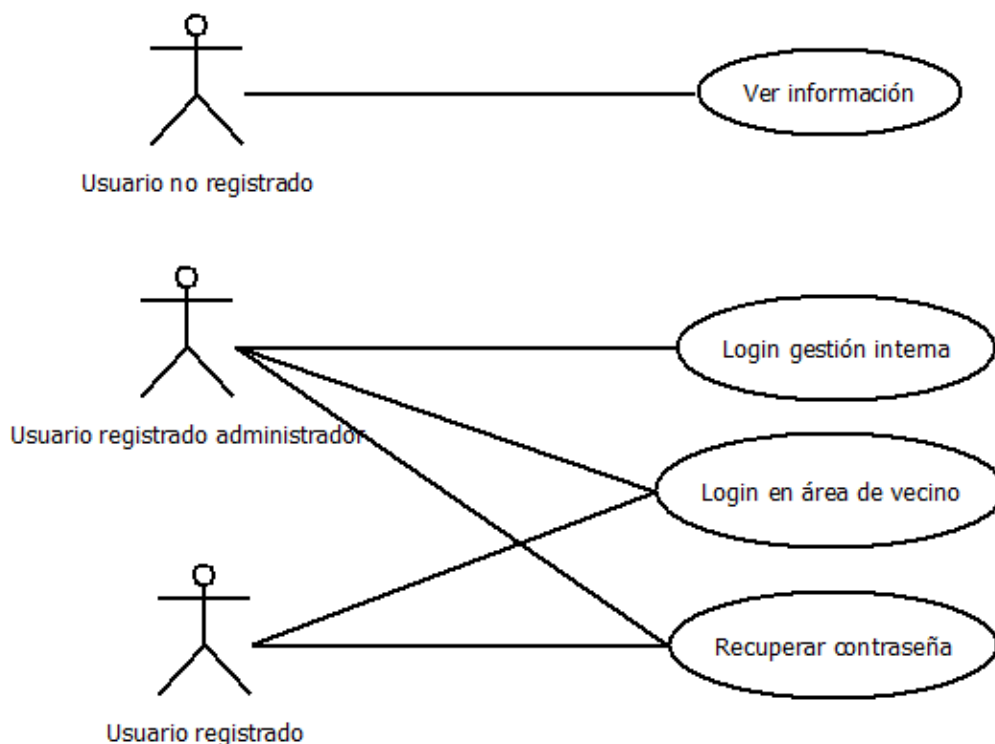
Después de analizar las conclusiones obtenidas en las diferentes reuniones con el jefe del ayuntamiento, llegamos a la conclusión de que el uso del sistema será para cualquier usuario que quiera demandar los servicios de la plataforma.

La aplicación está formada por dos partes diferenciadas, la primera parte será el muestrario de información y la segunda la administración de la misma. Según este criterio podemos obtener una clasificación de actores:

- Usuario no registrado: este actor puede englobar a cualquier persona que este navegando por nuestra aplicación web, únicamente podrá ver la información procesada por el aplicativo.

- Usuario registrado: persona que se haya registrado previamente y que no tiene el rol de administrador, esta persona podrá ver la información procesada por el aplicativo y además podrá añadir incidencias al sistema.
- Usuario registrado administrador: personal del ayuntamiento que puede cambiar cualquier información del sistema además de procesar incidencias y ver la información procesada por el aplicativo.

A continuación mostraremos un diagrama general del aplicativo para poder detallar después los diferentes escenarios que componen el aplicativo:



*Ilustración 20. Casos de uso de alto nivel.*

### 3.2 Requisitos funcionales del sistema

Los requisitos funcionales son aquellos que nos indican la labor del sistema.

Vamos a dividir los diferentes requisitos funcionales según el actor al que corresponden, para una mayor organización distribuiremos los diferentes números de los requisitos funcionales de la siguiente forma:

- Rol usuario no registrado: desde el 100 al 199
- Rol usuario registrado administrador: desde el 200 al 299
- Rol usuario registrado: desde el 300 al 399

**Rol usuario no registrado**

RF101	<b>R Descripción general</b>
	El sistema deberá permitir ver la descripción general del ayuntamiento
	Alta/Requerido
RF102	<b>R Hostelería</b>
	El sistema deberá permitir ver la hostelería presente en la región
	Alta/Requerido
RF103	<b>R Núcleos de población</b>
	El sistema deberá permitir información relativa a la población: localización, demografía y población.
	Alta/Requerido
RF104	<b>R Patrimonio</b>
	El sistema deberá permitir ver la información relativa a el patrimonio de la región.
	Alta/Requerido
RF105	<b>R Otros comercios</b>
	El sistema deberá permitir ver otros comercios presentes en la comarca
	Alta/Requerido
RF106	<b>R Servicios ayuntamiento</b>
	El sistema deberá permitir ver los servicios que ofrece el ayuntamiento.
	Alta/Requerido
RF107	<b>R Personajes ilustres</b>
	El sistema deberá permitir ver los personajes ilustres que viven o han vivido en la comarca.
	Alta/Requerido
RF108	<b>R Información turismo</b>
	El sistema deberá permitir ver información turística de la comarca.
	Alta/Requerido

RF109	<b>Recuperar contraseña</b>
	El sistema deberá permitir restablecer la contraseña.
	Alta/Requerido

RF110	<b>Registrarse</b>
	El sistema deberá permitir registrarse.
	Alta/Requerido

### Rol usuario registrado administrador

RF201	<b>CRU Descripción general</b>
	El sistema deberá permitir ver/añadir/modificar la descripción general del ayuntamiento
	Alta/Requerido

RF202	<b>CRUD Hostelería</b>
	El sistema deberá permitir ver/añadir/modificar/borrar la hostelería presente en la región
	Alta/Requerido

RF203	<b>CRUD Núcleos de población</b>
	El sistema deberá permitir ver/añadir/modificar/borrar información relativa a la población: localización, demografía y población.
	Alta/Requerido

RF204	<b>CRUD Patrimonio</b>
	El sistema deberá permitir ver/añadir/modificar/borrar la información relativa al patrimonio de la región.
	Alta/Requerido

RF205	<b>CRUD Otros comercios</b>
	El sistema deberá permitir ver/añadir/modificar/borrar otros comercios presentes en la comarca
	Alta/Requerido

<b>RF206</b>	<b>CRUD Servicios ayuntamiento</b>
	El sistema deberá permitir ver/añadir/modificar/borrar los servicios que ofrece el ayuntamiento.
	Alta/Requerido

<b>RF207</b>	<b>CRUD Personajes ilustres</b>
	El sistema deberá permitir ver/añadir/modificar/borrar los personajes ilustres que viven o han vivido en la comarca.
	Alta/Requerido

<b>RF208</b>	<b>CRUD Información turismo</b>
	El sistema deberá permitir ver/añadir/modificar/borrar información turística de la comarca.
	Alta/Requerido

<b>RF209</b>	<b>RD Incidencias</b>
	El sistema deberá permitir ver/borrar incidencias previamente registradas en el sistema.
	Alta/Requerido

<b>RF208</b>	<b>CRU Ver información personal</b>
	El sistema deberá permitir ver/añadir/modificar información personal.
	Alta/Requerido

### **Rol usuario registrado**

<b>RF301</b>	<b>CRU Ver información personal</b>
	El sistema deberá permitir ver/añadir/modificar la información personal
	Alta/Requerido

<b>RF202</b>	<b>CRU Incidencia</b>
	El sistema deberá permitir ver/añadir/modificar la incidencias
	Alta/Requerido

<b>RF203</b>	<b>Enviar correos al ayuntamiento</b>
	El sistema deberá permitir enviar correos al ayuntamiento
	Alta/Requerido

### 3.3 Requisitos no funcionales

Son aquellos que nos indican características generales o restricciones del sistema.

En este caso vamos a dividir los requisitos no funcionales en cuatro: eficiencia, seguridad, usabilidad y dependencia, cada uno de ellos son imprescindibles a la hora de realizar un buen desarrollo de una aplicación, ya que verifican como debería ser un sistema.

Para una mayor organización distribuiremos los diferentes números de los requisitos no funcionales de la siguiente forma:

- Eficiencia: desde el 100 al 199
- Seguridad: desde el 200 al 299
- Usabilidad: desde el 300 al 399
- Dependencia: desde el 400 al 499

#### Eficiencia

RF101	<b>Velocidad del sistema</b>
	El sistema deberá ser capaz de responder a cualquier petición en menos de 6 segundos.
	Alta/Requerido

RF101	<b>Concurrencia</b>
	El sistema deberá permitir al menos 100 usuarios concurrentes en la aplicación
	Alta/Requerido

#### Seguridad

RF101	<b>Acceso por roles</b>
	El sistema deberá prohibir el acceso a los diferentes recursos si no se tiene el rol adecuado.
	Alta/Requerido

RF101	<b>Respaldo</b>
	Todos los datos tienen que estar respaldados al menos cada 48 horas.
	Alta/Requerido

<b>RF101</b>	<b>Encriptado de contraseñas</b>
	El sistema deberá encriptar las contraseñas a la hora de almacenarla en la base de datos.
	Alta/Requerido

### Usabilidad

<b>RF101</b>	<b>Interface grafica</b>
	El sistema deberá tener una interface fácil e intuitiva.
	Alta/Requerido

<b>RF101</b>	<b>Compatibilidad de navegadores</b>
	El sistema deberá permitir su visualización en los últimos navegadores compatibles con HTML 5.
	Alta/Requerido

### Dependencia

<b>RF101</b>	<b>Disponibilidad</b>
	El sistema tiene que estar disponible al menos el 97% del tiempo.
	Alta/Requerido

<b>RF101</b>	<b>Tiempo de mantenimiento</b>
	El sistema no puede estar más de 3h de mantenimiento.
	Alta/Requerido

## 3.4 Especificación de casos de uso

Una vez definidos los diferentes requisitos del sistema pasamos a obtener los casos de uso del sistema, los casos de uso de un sistema son una forma de expresar como alguien o algo (otros sistemas) interactúan con nuestro sistema.

En este caso vamos a dividir el sistema en dos partes diferenciadas; el primero engloba el área del administración, que es la parte que van a ver los usuarios registrados administración, que permite configurar la información de la plataforma y la gestión de las incidencias. La segunda parte engloba el sistema global que se caracteriza por gestionar todas las tareas que puede realizar un usuario.

### 3.4.1 Área de administración

En la ilustración podemos ver todos los procesos que puede realizar un administrador del aplicativo, desde modificar cualquier información que será procesada y visualizada por el usuario, hasta la gestión de las diferentes incidencias anotadas por los usuarios.

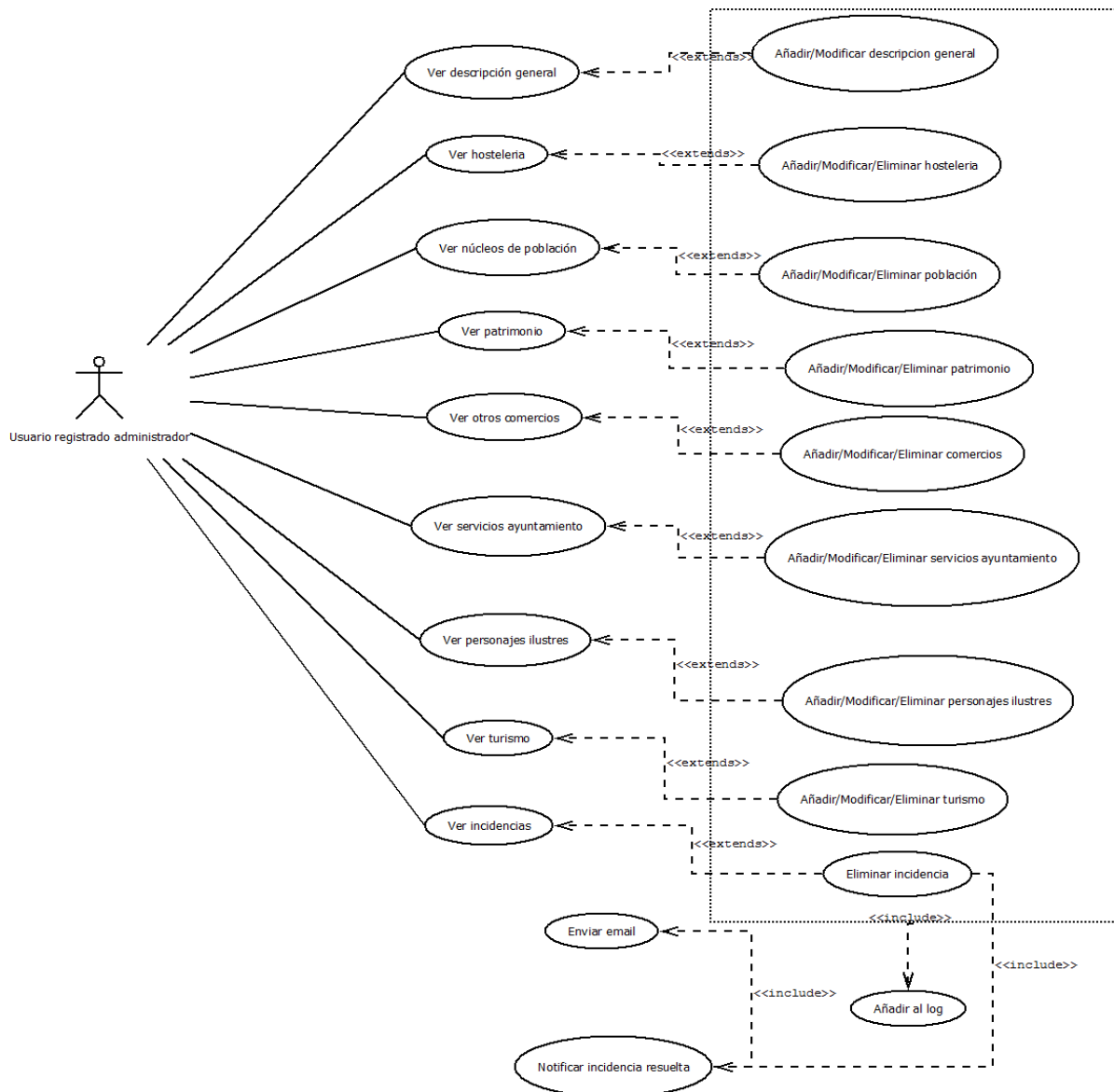


Ilustración 21. Casos de uso del área de administración.

A continuación vamos a detallar los casos de usos más relevantes mediante una plantilla:



ID	P001
<b>Nombre</b>	Eliminar incidencia
<b>Descripción</b>	El administrador ha leído una incidencia y una vez solucionado la elimina del sistema.
<b>Actores primarios y secundarios</b>	Usuario registrado administrador
<b>Precondiciones</b>	Usuario registrado con rol administrador, un usuario haya registrado previamente una incidencia.
<b>Flujo principal</b>	<ol style="list-style-type: none"> <li>1. El administrador selecciona en la lista de incidencias la que quiere borrar.</li> <li>2. El sistema muestra los datos de la incidencia.</li> <li>3. El administrador selecciona la opción borrar incidencia</li> <li>4. El sistema comunica al usuario que interpuso la incidencia de que se a solucionado.</li> <li>5. El sistema guarda el log la incidencia.</li> </ol>
<b>Flujo alternativo</b>	
<b>Post condiciones</b>	Se elimina la incidencia del sistema

### 3.4.2 Sistema global

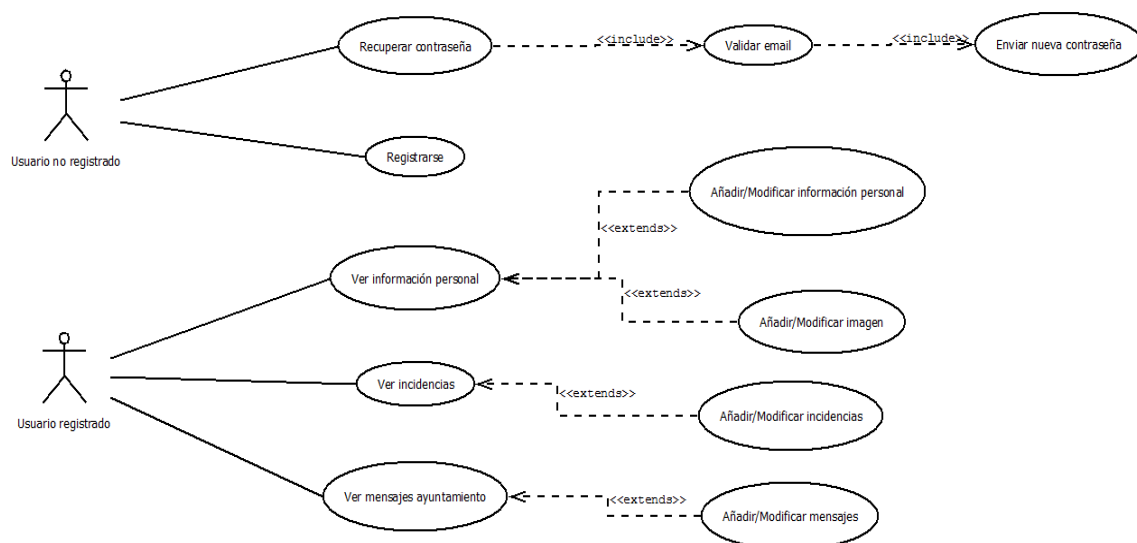


Ilustración 22. Casos de uso del sistema global.

En la ilustración anterior podemos ver los diferentes roles que puede adoptar cualquier persona que acceda al sistema, como podemos observar un usuario que accede al sistema y no tiene ningún privilegio se puede registrar y adoptar los privilegios de unos usuarios registrados.

A continuación vamos a detallar los casos de usos más relevantes mediante una plantilla:

<b>ID</b>	<b>P001</b>
<b>Nombre</b>	Recuperar contraseña
<b>Descripción</b>	El usuario a olvidado la contraseña y solicita recupérala.
<b>Actores primarios y secundarios</b>	Usuario
<b>Precondiciones</b>	Que el usuario este registrado.
<b>Flujo principal</b>	<ol style="list-style-type: none"> <li>1. El usuario selecciona contraseña olvidada</li> <li>2. El usuario introduce su email</li> <li>3. El sistema valida si el email está en la base de datos.</li> <li>4. El sistema envía un email con un enlace para cambiar la contraseña</li> </ol>
<b>Flujo alternativo</b>	<ol style="list-style-type: none"> <li>1. Email no está en la base de datos <ol style="list-style-type: none"> <li>a. El sistema muestra un error.</li> <li>b. El sistema nos envía a la página principal</li> </ol> </li> </ol>
<b>Post condiciones</b>	Se cambia la contraseña del usuario

<b>ID</b>	<b>P002</b>
<b>Nombre</b>	Añadir incidencia
<b>Descripción</b>	El usuario introduce una incidencia al sistema.
<b>Actores primarios y secundarios</b>	Usuario registrado
<b>Precondiciones</b>	Que el usuario este registrado.
<b>Flujo principal</b>	<ol style="list-style-type: none"> <li>1. El usuario selecciona la opción añadir incidencia</li> <li>2. El usuario selecciona el tipo de incidencia a insertar</li> <li>3. El usuario selecciona la posición de la incidencia</li> <li>4. El usuario selecciona guardar incidencia</li> <li>5. El sistema comprueba que los datos introducidos son correctos.</li> <li>6. Se almacena la incidencia.</li> </ol>
<b>Flujo alternativo</b>	<ol style="list-style-type: none"> <li>1. La posición de la incidencia no es correcta. <ol style="list-style-type: none"> <li>a. El sistema indica un mensaje de error</li> <li>b. El sistema da la posibilidad de volver a insertar la posición</li> </ol> </li> </ol>
<b>Post condiciones</b>	Se añade la incidencia del sistema

## 4 ESTRATEGIA DE DISEÑO E IMPLEMENTACIÓN

En este apartado se realiza la elección de la arquitectura con la que se desarrolla el proyecto, indicaremos los diagramas de despliegue y sus componentes y para terminar, concluiremos con el capítulo de la implementación de diferentes capas de la aplicación.

### 4.1 Descripción de la arquitectura

Llega el momento de hacer una descripción de la arquitectura que vamos a utilizar y los componentes que se van a utilizar.

Esta es la capa más alta del desarrollo software, la arquitectura define la estructura que compone el proyecto y sus propiedades, además de las relaciones entre ellos.

Existen numerosos patrones arquitectónicos que con el tiempo se han ido consolidando en la arquitectura software, estos permiten dar una solución entendible a determinados problemas, dando unos principios organizativos al desarrollo software.

Hay que recalcar que la aplicación que estamos desarrollando, es una aplicación distribuida por lo que nos vamos a centrar en arquitecturas de este tipo.

Vamos a nombrar alguna de las arquitecturas [21] de sistemas distribuidos más importantes:

- Cliente-servidor: es la arquitectura más simple que encontramos en los sistemas distribuidos, la aplicación está formada por dos partes, el servidor que se encarga de ofrecer los servicios y los clientes de consumir los servicios generados por el servidor.
- Peer-to-peer: son sistemas que no están centralizados, el cálculo computacional se realiza en cualquiera de los nodos presentes.
- Arquitectura orientada a servicios: es un servicio web para compartir algún recurso computacional.
- Modelo Vista Controlador: arquitectura de tres capas que está compuesta por persistencia, negocio e interface.

Después de analizar las diferentes arquitecturas nos decantamos por la de Modelo Vista Controlador (MVC), ya que nuestro *framework* Express es la que tiene implementada.

Es la solución más utilizada en el desarrollo web actualmente y hay numerosos *framework* que lo implementan.

Gracias a esta solución el programador tiene mucha más flexibilidad a la hora de desarrollar o mantener un proyecto.

Hay que recalcar que la arquitectura global de nuestro proyecto es MVC, pero al utilizar AngularJS la parte de la vista implementa otro patrón MVC, esto quiere decir que poseemos un patrón dentro de otro patrón.

## 4.2 ¿Que arquitecturas tiene nuestro proyecto?

Como hemos comentado anteriormente nuestro patrón se basa en MVC, por lo que vamos a definir en que consiste esta arquitectura.

### 4.2.1 Descripción MVC

Es un patrón de desarrollo software que está compuesto por tres capas: modelo, vista y controlador; esto ofrece un control específico de desarrollo de una aplicación. Este tipo de patrón es un estándar en aplicaciones web y es muy utilizado en el desarrollo de aplicaciones escalables.

Procedemos a describir cada una de las partes del MVC:

- **Modelo:** corresponde a todo lo relacionado con la lógica de datos, es el encargado de obtener los datos para pasarlo a la lógica de negocio o recibirlos de ella.
- **Vista:** es la parte que se encarga de dar la lógica del interface de usuario de la aplicación, en nuestro caso en la vista contenemos el *framework* AngularJS, que por sí mismo implementa el patrón MVC, por lo que dentro de la vista contenemos otro patrón MVC.
- **Controlador:** se encargan de ser una interfaz entre la vista y el modelo, procesa toda la lógica de negocio y todas las peticiones entrantes a la aplicación.

### 4.2.2 Flujo de control MVC

Estos son los pasos que se llevan a cabo a la hora de realizar una petición en un sistema MVC [22].

- i. El usuario realiza una petición al sistema mediante la vista.
- ii. El controlador de la aplicación recibe la petición, la controla y la trata.
- iii. El controlador se comunica con el modelo, puede cambiar el estado en caso de que no sea solo una consulta.
- iv. El controlador genera una nueva vista con los datos que recibe del modelo.
- v. La vista espera una nueva interacción del usuario para repetir los pasos anteriores.

### 4.3 Diagrama despliegue

Como podemos ver en la figura siguiente, nos encontramos con un servidor central, en el cual tenemos instalado NODE.JS, gracias al *framework* Express tenemos una arquitectura MVC.

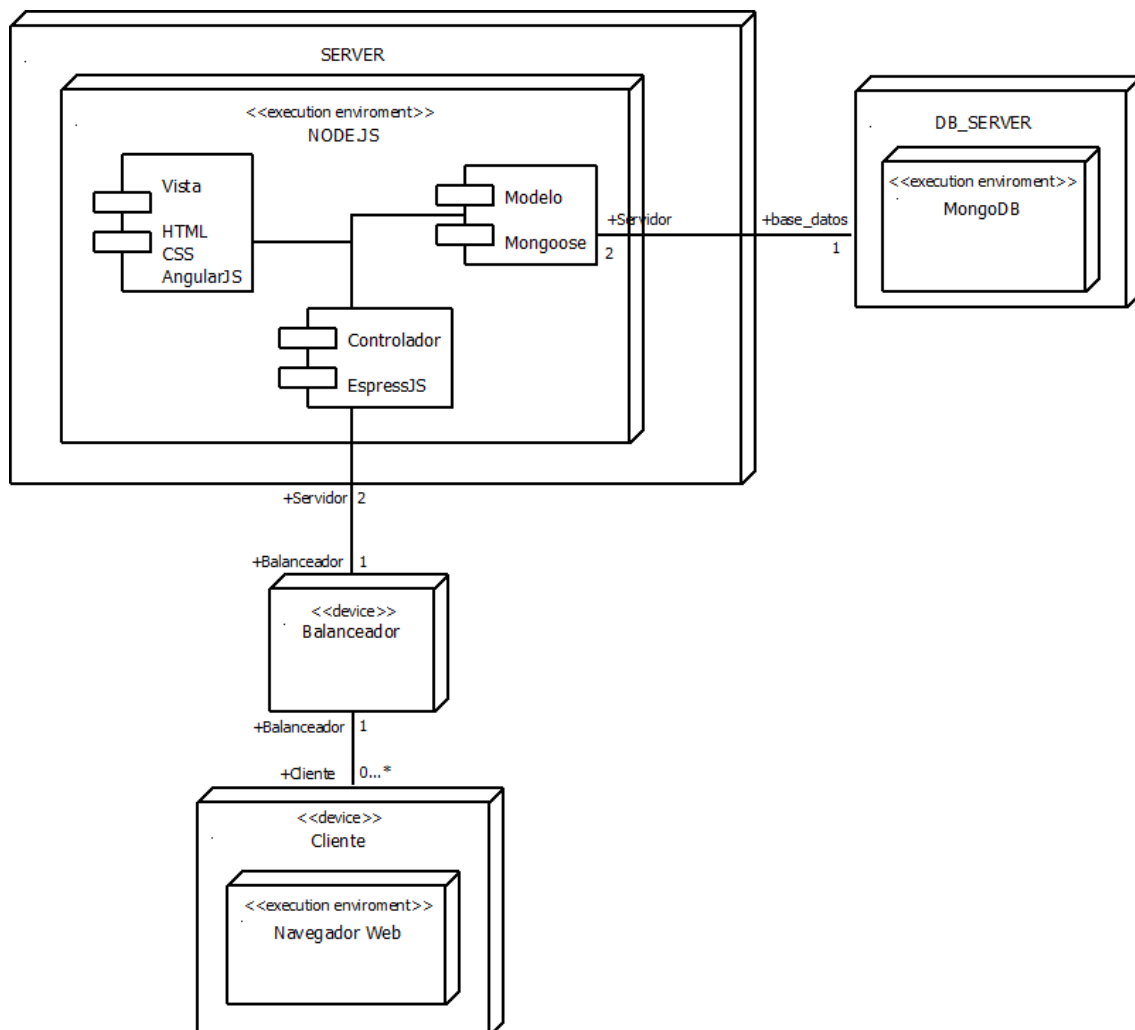


Ilustración 23. Diagrama de despliegue de la aplicación.

También nos encontramos con un servidor de base de datos, que será el encargado de ejecutar la base de datos MongoDB y un balanceador de carga que nos permitirá cumplir los requisitos no funcionales de disponibilidad.

#### 4.4 Diagrama de componentes

En este diagrama hemos simplificado la parte del modelo ya que utilizamos un ORM, encargado de abstraer el acceso a la base de datos.

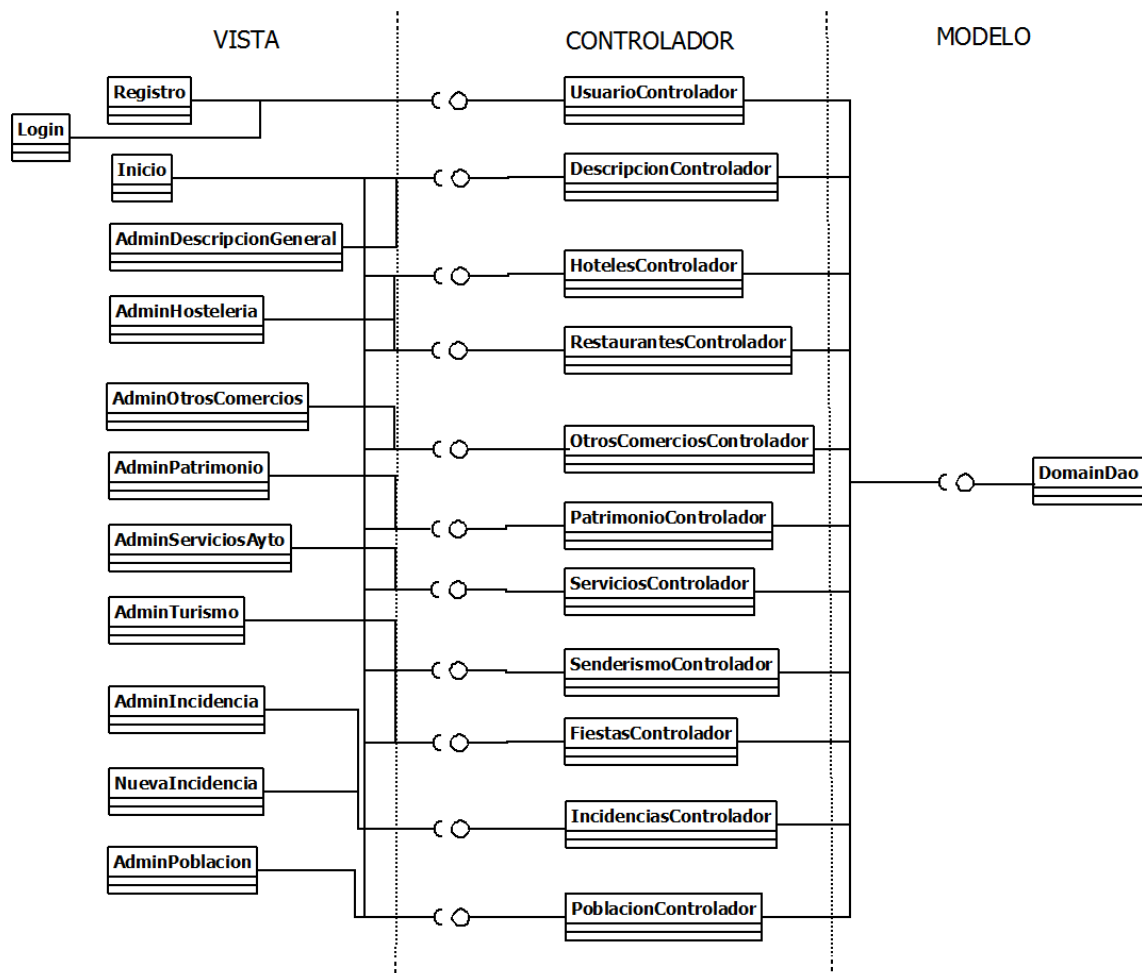


Ilustración 24. Diagrama de componentes.

#### 4.5 Implementación

Después de definir el análisis y diseño de la aplicación, continuamos con la implementación de la misma, en esta sección se explicara cómo se ha implementado cada capa de la aplicación.

A consecuencia de utilizar la metodología Kanban vamos implementando las tareas con más prioridad.

Además de las capas que están definidas en el patrón MVC, añadimos otra parte al proyecto que es el *testing* de la aplicación.

Procedemos a indicar el orden en el que vamos a explicar cada una de las capas existentes en la aplicación:

- Vista: definiremos como están construidas las diferentes vistas de la aplicación.
- Controlador: engloba la definición de cómo implementar la gestión de las peticiones y procesamiento de las mismas.
- Modelo: explicaremos como implementamos cada documento y como lo relacionamos con la base de datos documental.
- *Testing*: implementaremos las pruebas para todas las capas de la aplicación.

#### 4.5.1 Vista

En esta parte de la implementación nos centraremos en como el usuario puede interactuar con la aplicación, para la gestión de las vistas utilizaremos un motor de plantillas, en este caso utilizaremos un motor que viene implementado en el *framework* Express, Jade.

Al configurar Express el mismo se encargara de importar el motor sin que el programador tenga que importar los módulos necesarios para la utilización del mismo.

Una vez importado el motor, este se encargara de *renderizar* el código .jade a HTML sin ninguna complicación.

```
// Configuración de node para poder usar el motor Jade
app.set('view engine', 'jade');

//Forma de renderizar una pagina jade a HTML
app.get('/', function(req, res) {
  res.render('index.jade');
});
```

*Ilustración 25. Imagen del código de renderizado de una plantilla Jade.*

Antes de continuar con la explicación de la implementación de la vista describiremos y nombraremos las características de este fantástico motor de plantillas:



#### 4.5.1.1 JADE

Jade es un poderoso motor de plantillas escrito en JavaScript que nos permite escribir de una forma fácil y clara HTML con una sintaxis parecida a Python.

Es una forma de desarrollar HTML de forma rápida y limpia, Jade nos permite crear un *layout* genérico para las partes web en las cuales siempre tiene que tener ese contenido y así poder mantener el código de forma rápida y eficaz, sin tener que modificar cada uno de los archivos HTML, únicamente modificando el genérico.

Algunas de las características son:

- Claridad en el código fuente.
- Creación de *layout*.
- Introduce una forma de dinamizar el código HTML mediante variables.
- Uso de condicionales.

Para la creación de las vistas se ha utilizado plantillas ya hechas las cuales nos han facilitado la maquetación a la vez que nos han dado un diseño nuevo e innovador.

Procedemos a identificar las diferentes plantillas que se han utilizado para el desarrollo de la aplicación.

Nuestra tiene dos partes diferenciadas a nivel visual.

- Para el área informativa de la aplicación se ha utilizado una plantilla de estilo *landing page* (página de presentación), esta plantilla es de *open source* y a partir de ella se han hecho las modificaciones oportunas para adaptarla a nuestras necesidades.

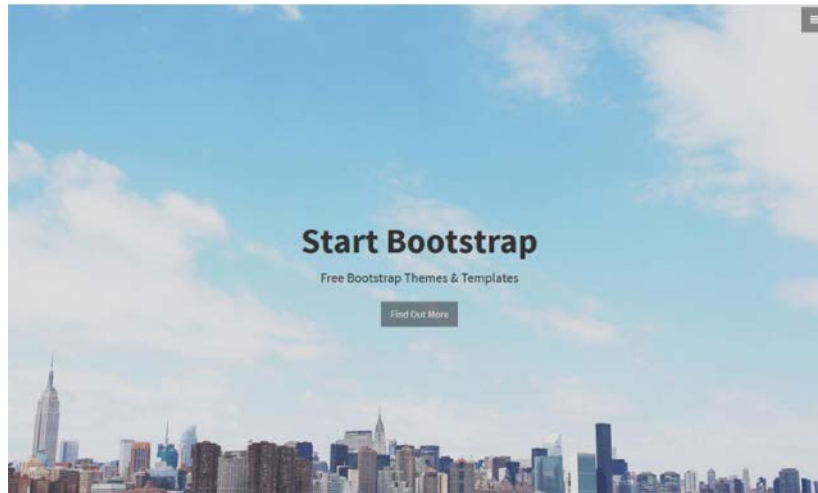
El creador de esta *landing page* es David Miller y se puede obtener de forma fácil a través de su página web [27].

Las tecnologías utilizadas por esta plantilla son:

- Bootstrap 3.3.6
- JQuery 1.11.1
- CSS 3
- Para el área administrativa y área de usuario se ha utilizado una fantástica plantilla llamada AdminLTE, también open source y creada por Abdullah Almsaeed está también se puede obtener de una forma fácil atreves de la siguiente web [28].

Las tecnologías utilizadas para esta área de usuario son:

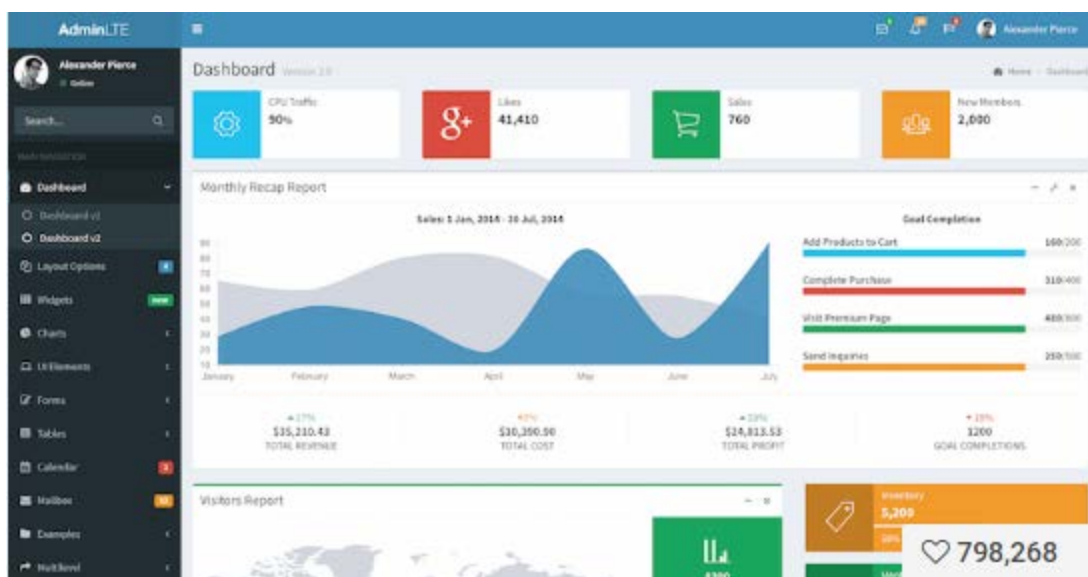
- Bootstrap 3
- JQuery 1.11.1
- CSS 3



*Ilustración 26. Imagen de la plantilla utilizada en la aplicación.*

Algunas de las librerías que utiliza para dar dinamismo y usabilidad son:

- Ckeditor (Editor de texto)
- Datepicker
- Chartjs(Graficas)



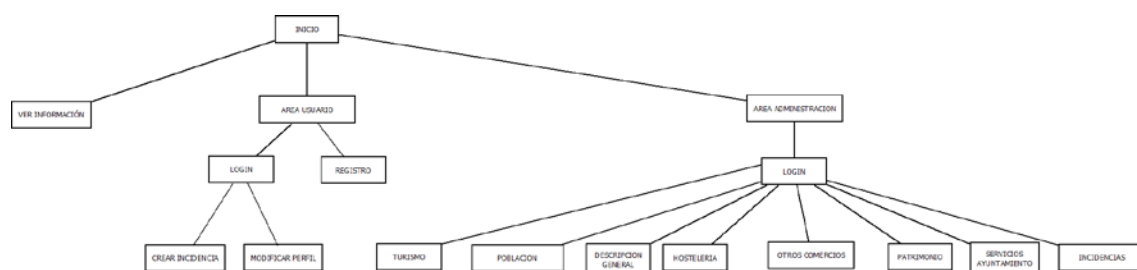
*Ilustración 27. Imagen de la plantilla utilizada para el área de administración.*

El único problema de estas plantillas, es que están desarrolladas con HTML y se ha tenido que transformar ese código a Jade para poder añadir las características propias que se han comentado anteriormente.

Este tipo de plantillas traen todo lo necesario para crear tu aplicación, en este caso, hemos elegido los componentes que nos han interesado para nuestra aplicación y hemos adaptado la vista a los requisitos del ayuntamiento.

Además en esta aplicación utilizamos AngularJS, que nos sirve para conectar las peticiones al controlador además de dar dinamismo de forma fácil y sencilla.

Por último vamos a mostrar el mapa web de la aplicación web:



*Ilustración 28. Mapa de la aplicación web.*

#### 4.5.2 Controlador

El controlador es el encargado de realizar las operaciones de negocio de la aplicación, para facilitar el mantenimiento de la aplicación creamos un controlador exclusivo para procesar las peticiones del usuario, con este controlador definimos todas las rutas de nuestra aplicación y posteriormente encaminarlas al controlador oportuno.

En este controlador además se aplica el filtro de autenticación de acceso a los recursos, gracias a la librería PassportJs, de esta manera, los usuarios no autenticados no pueden acceder a las áreas restringidas.

```

//Método al que pueden acceder cualquier usuario de la aplicacion
app.get('/', function (req, res) {
  res.render('login', { user : req.user });
});

//Método al que solo pueden acceder los usuarios registrados
app.get('/principal', ensureAuthenticated, function (req, res) {
  res.render('principal', { user : req.user });
});
  
```

*Ilustración 29. Código para realizar la autenticación a la aplicación.*

En el caso de que no esté autenticado el usuario que quiere acceder al recurso, el sistema le mandará directamente a la página de inicio, esto se realizara mediante la implementación del siguiente *callback*:

```
//comprobar que esta autenticado
function ensureAuthenticated(req, res, next) {
  if (req.isAuthenticated()) { return next(); }
  res.redirect('/')
}
```

Ilustración 30. Código para la resolución de peticiones no autenticadas.

Llegado a este punto vamos a describir en la siguiente figura los diferentes controladores que tiene nuestra aplicación:

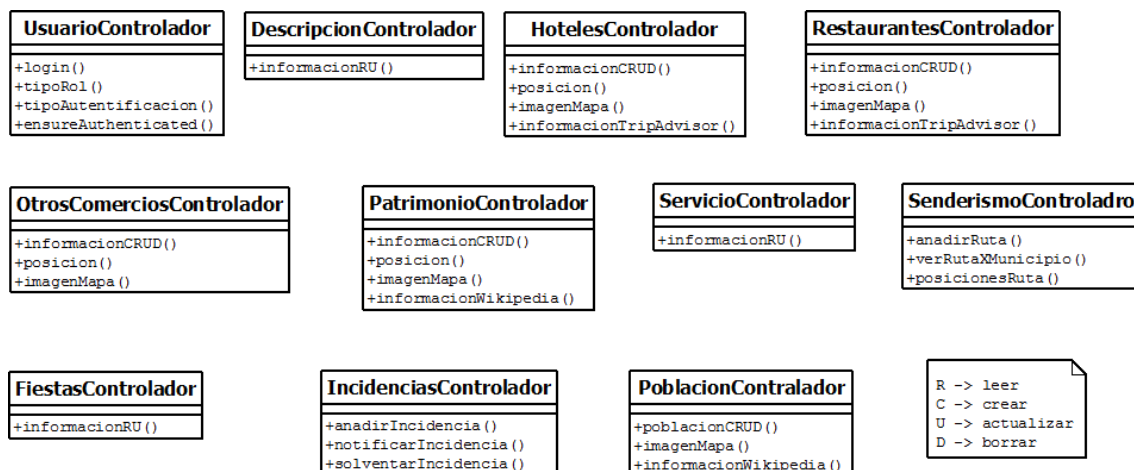


Ilustración 31. Controladores de la aplicación.

Omitimos el RutaControlador, que es el encargado de encaminar las peticiones a cada uno de los controladores correspondientes por la gran cantidad de métodos que tiene.

En nuestra aplicación el RutaControlador puede realizar tres tipos de operaciones según el recurso que pida el usuario:

- Procesa la petición y *renderiza* una vista.

```
app.get('/', function (req, res) {
  res.render('login', { user : req.user });
});
```

Ilustración 32. Código para renderización de una vista.

- Procesa la petición, accede al controlador correspondiente del cual obtiene los datos, ya sea por composición del mismo o por petición al modelo y por último compone la vista.

```
app.post('/register/restaurante', function (req, res) {
    hosteleria.createRestaurante(req, res);
});
```

*Ilustración 32. Código para generar una respuesta.*

- Procesa la petición, accede al controlador correspondiente y únicamente devuelve los datos en formato JSON.

```
app.get('/menu/data', ensureAuthenticated, function (req, res) {
    res.json({user:req.user});
});
```

*Ilustración 33. Código para generar un JSON de respuesta.*

La forma en la cual un controlador usa un recurso de otra capa es importando mediante la expresión *requiere("urlRecurso")* y gracias a esto podremos acceder a los métodos implementados en las diferentes capas de la aplicación.

#### 4.5.3 Modelo

Esta es la capa de datos que engloba el acceso a la base de datos para persistir los datos de la aplicación.

Gracias al framework Mongoose (ORM), esta tarea se convierte en algo sencillo, ya que abstrae la estructura y el modo de acceder a la base de datos documental.

Mongoose proporciona un acceso a los datos mediante funciones predefinidas en el lenguaje JavaScript, a causa de ORM podemos mapear la estructura de los documentos de la base de datos con objetos JavaScript para poder manipularlos de forma fácil y eficiente.

Simplemente con una básica configuración de la conexión a la base de datos y de unos modelos definidos en el modelo de la aplicación podemos realizar cualquier operación con la base de datos.

En el siguiente fragmento indicaremos un ejemplo de conexión con la base de datos MongoDB:

```
//Importamos el modulo del ORM de Moongose
var mongoose = require('mongoose');
//Realizamos la conexión a la base de datos MongoDB
mongoose.connect('localhost', 'TFG');
var db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function callback() {
  console.log('Connected to DB');
});
```

*Ilustración 33. Código para la conexión con la BD.*

Con el código anterior ya nos encontramos conectados a la base de datos.

En nuestro caso hemos implementado una base de datos local para realizar el desarrollo en nuestro equipo sin tener problemas en la conectividad, una vez puesto en producción solo deberíamos cambiar la dirección en la que se encuentra la base de datos y todo funcionaría correctamente.

Para mapear los diferentes modelos de la base de datos documental con el ORM solo sería necesario definir el modelo como a continuación.

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var userSchema = new Schema({
  nombre: String,
  email: String,
  aforo: Number,
  precio: Number,
  telefono: Number,
  facebook: String,
  website: String,
  latitud: Number,
  longitud: Number,
  icono:String,
});

module.exports = mongoose.model('hosteleria', userSchema);
```

*Ilustración 34. Modelo de hostelería en Mongoose.*

Como podemos ver en la figura anterior se ha mapeado el documento de la base de datos hostelería con el objeto JavaScript proporcionado por Mongoose.

Otro de los puntos fuerte que nos ofrece ORM son los métodos suficientes para realizar todas las operaciones de la base de datos, como CRUD y filtrado de búsquedas por un parámetro establecido.

## 5 PRUEBAS Y DESPLIEGUE

En este apartado vamos a realizar las diferentes pruebas de las diferentes partes del MVC para comprobar su correcto funcionamiento y asegurar el cumplimiento de los requisitos, concluiremos con los pasos que se siguen para desplegar la aplicación web.

### 5.1 Pruebas

Una vez explicados los pasos a seguir para realizar la implementación, llega el momento de explicar cómo se ha realizado el testeo de la aplicación.

El *testing* es una parte muy importante en el desarrollo software ya que nos va a ayudar a detectar los fallos en el desarrollo y el correcto cumplimiento de los requisitos del sistema, otro punto a destacar es que nos va a ayudar a encontrar errores a la vez que vamos desarrollando nuevas funcionalidades del mismo.

Hay que decir que no sólo buscamos errores en el software, también buscamos que no se cumpla un requisito del cliente.

Recordamos que la metodología que utilizamos para nuestro proyecto es una metodología ágil, por lo que en este tipo de metodologías es muy importante probar cada implementación cada vez que esta se finaliza, gracias a las pruebas podremos estar seguros que los entregables van a estar cumpliendo los requisitos.

Vamos a definir los diferentes tipos de pruebas de software según el objetivo en el que se centran:

- Pruebas funcionales: nos centramos en el comportamiento del sistema o de un componente dado, en nuestro caso hemos realizado pruebas funcionales con Selenium.
- Pruebas no funcionales: serían las pruebas que indican cómo se comporta el sistema, gracias a ApacheBench hemos medido la eficiencia de la aplicación.
- Pruebas estructurales: son pruebas que se aplican a las funciones dentro de un componente, con las pruebas unitarias en el controlador hemos comprobado que los métodos funcionan correctamente.
- Pruebas de regresión: sería volver a probar un componente después que ha sido modificado, cada vez que hemos integrado una nueva funcionalidad se ha vuelto a comprobar todas las pruebas relativas al componente.

En nuestro caso al ser una aplicación de tipo web tenemos que testear además de las pruebas habituales de cualquier desarrollo, temas relativos al servidor web, sistema de gestión de base de datos o conjunto del sistema.

Vamos a realizar un recorrido de las pruebas realizadas en cada capa de la aplicación:

- Vista: la representación en la vista está compuesta por HTML/CSS, es importante comprobar que estos datos transmitidos por el servidor son correctos.

Para ello validamos todo el contenido transmitido por el servidor, uno de los validadores utilizados es el de W3C [29].

- Controlador: es importante comprobar que cada método del mismo se comporta como hemos esperado, para ello utilizamos test de pruebas unitarias para realizar una cobertura de los datos críticos de los diferentes métodos que componen los diferentes módulos del controlador de la aplicación.

Para realizar las pruebas unitarias utilizaremos el siguiente *framework*: MochaJs (ver 5.1.1).

- Modelo: vamos a comprobar que la conexión esta correcta y que la base de datos esta subida.

Vamos a explicar las tecnologías [23] y las formas de realizar los testeo en las diferentes capas de la aplicación.

#### 5.1.1 MochaJs

Para la parte del controlador utilizaremos el siguiente *framework* para la realización de test unitarios de métodos.

En nuestra aplicación vamos a realizar los test unitarios para comprobar que la lógica de negocio de nuestra aplicación funciona correctamente.

Debido a la gran cantidad de métodos que tienen las diferentes clases que componen la capa de negocio, nos vamos a centrar en una de ellas para explicar el funcionamiento del *framework*.

Una vez realizados los test para cada una de las capas, lo siguiente que se ha sugerido es realizar unas pruebas de integración que es la forma de comprobar que todas las partes en su conjunto funcionen correctamente.

Para ello utilizaremos webdriver.io:

#### 5.1.2 Webdriver.io

Es un Framework [24] para NODE.JS que implementará el driver de Selenium para los diferentes navegadores del mercado.



Con este *framework* podemos predefinir los diferentes pasos que un usuario puede seguir para obtener un recurso, gracias a esto podemos ver que todos los requisitos funcionales de la aplicación están funcionando correctamente.

Webdriver.io, permite que con el mismo código generado, únicamente cambiando el web driver podemos probar nuestra aplicación en diferentes navegadores.

### 5.1.2 ApacheBench

Mediremos el rendimiento de la aplicación para así cumplir alguno de los requisitos no funcionales mentados anteriormente.

Para ello lo que hacemos es ejecutar el siguiente comando:

RF101 Comprobamos el requisito funcional de concurrencia, en este caso tiene que responder a 100 usuarios concurrentes durante 1000 peticiones.

```
C:\wamp\bin\apache\apache2.4.9\bin>ab -r -n 1000 -c 100 http://5.196.20.235:8080/
This is ApacheBench, Version 2.3 <$Revision: 1554214 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

*Ilustración 35. Comando para realizar el benchmark del servidor.*

Esperamos a los resultados que son los siguientes.

```
Server Software:
Server Hostname:      5.196.20.235
Server Port:          8080

Document Path:        /
Document Length:      13035 bytes

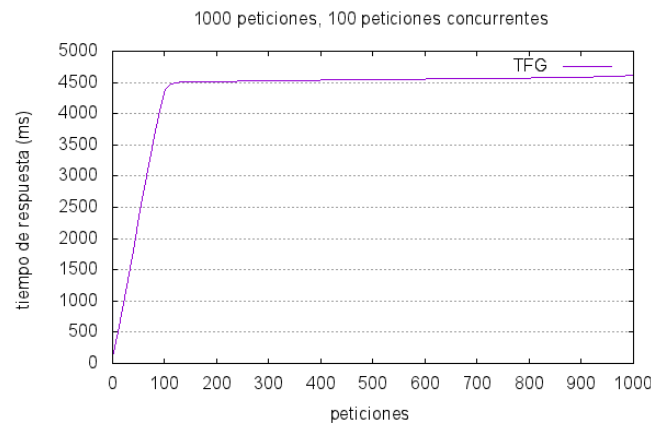
Concurrency Level:     100
Time taken for tests:  45.725 seconds
Complete requests:     1000
Failed requests:        0
Total transferred:     13326000 bytes
HTML transferred:     13035000 bytes
Requests per second:   21.87 [#/sec] (mean)
Time per request:      4572.538 [ms] (mean)
Time per request:      45.725 [ms] (mean, across all concurrent requests)
Transfer rate:         284.61 [Kbytes/sec] received

Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:        42    45   2.8      45     62
Processing:     52  4296  797.0    4506   4629
Waiting:        47  2251 1307.7    2244   4514
Total:          97  4341  797.0    4551   4673

Percentage of the requests served within a certain time (ms)
 50%    4551
 66%    4565
 75%    4581
 80%    4603
 90%    4622
 95%    4632
 98%    4649
 99%    4660
100%    4673 (longest request)
```

*Ilustración 36. Resultado del test al servidor NODE.JS.*

La prueba se ha realizado contra la página de inicio de la aplicación una de las más pesadas de la aplicación. Como podemos ver en la ilustración 37 el tiempo medio por petición se encuentra en 4572 ms por lo que cumple el requisito funcional RF101 de no superar los 6 minutos.



*Ilustración 37. Gráfico del tiempo de respuesta por el número de peticiones.*

Como podemos ver en la ilustración 38 el servidor se mantiene estable a partir de la petición número 100, por lo que podemos ver que cumple el requisito funcional RF101

## 5.2 Despliegue

En este apartado se explicarán los pasos que se han llevado a cabo para configurar el entorno de producción, configuración de la base de datos y el posterior despliegue de la aplicación.

### 5.2.1 Especificaciones del entorno

Para la configuración del entorno se alquiló un servicio de VPS (Servidor Privado Virtual), en OVH una compañía dedicada al alquiler de servidores en la nube.

Este servidor fue el más básico de la gama de VPS SSD, una gama de servicios con gran rendimiento y que se ajustan a los requisitos de nuestra aplicación, elegimos ese servidor ya que posee una garantía de servicio de 99,95, uno de los requisitos no funcionales de nuestro sistema.

Las características más destacadas del servidor son la siguiente:

- Un core virtualizado Intel Xeon E5v3 @2,4 GHz.
- 2Gb de RAM.
- 10GB de almacenamiento SSD.
- 100Mb de ancho de banda.

Este tipo de servicios son muy ventajosos a la hora de la escalabilidad, ya que se puede aumentar las prestaciones de número de *cores*, RAM y almacenamiento únicamente contactando con el equipo de atención al cliente y aumentando la cuota mensual. El equipo técnico migrará el servidor si tener que hacerlo nosotros mismos.

### 5.2.2 Configuración del entorno

Al contratar este tipo de servicios VPS, la máquina que nos alquilan es una máquina desnuda, esta sin ningún tipo de sistema operativo en un principio, pero la instalación del SO es algo sencillo, ya que OHV posee de un panel en el cual podemos indicarle el tipo de sistema operativo que queremos que nos instale.

En nuestro caso elegimos un Debian 7(Wheezy) de 64bits, una vez instalado el sistema operativo accedemos al servidor con un cliente SSH, en nuestro caso utilizamos Putty, una vez en el sistema como superusuario empezamos a configurar el diferente software necesario para desplegar la aplicación.

#### 5.2.2.1 Servidor FTP

Lo primero que hacemos es configurar un servidor FTP [25] para poder acceder con un cliente y subir los archivos que componen la aplicación. El servidor que montaremos será el ProFTPD que le encontramos en los repositorios de Debian.

La forma de instalarlo será con el siguiente comando:

```
aptitude install proftpd-basic proftpd-doc
```

Una vez instalado el servidor ftp únicamente configuramos el directorio raíz donde accederá cualquier conexión a la base de datos, para realizar la configuración accedemos a la siguiente ruta donde encontramos el archivo de configuración.

```
/etc/proftpd/proftpd.conf
```

Por seguridad es importante crear usuarios para el acceso al ftp y no usar los usuarios del sistema Debian.

Por último levantamos el servicio de ftp:

```
/etc/init.d/proftpd restart
```

### 5.2.2.1 Base de datos MongoDB

Es el momento de instalar la base de datos que va a albergar los datos de nuestra aplicación, para instalar la base de datos utilizaremos los repositorios que no están incluidos en Debían, por lo que tendremos que introducir en los repositorios del sistema la ubicación de los de MongoDB [26]:

Creamos el siguiente archivo con los repositorios:

```
/etc/apt/sources.list.d/mongodb-org-3.0.list
```

Añadimos el siguiente contenido:

```
echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/3.0
main" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

Una vez actualizados los repositorios instalamos MongoDB:

```
apt-get install -y mongodb-org
```

Por último iniciamos el servicio de la base de datos:

```
service mongod start
```

Una vez instalado el sistema de base de datos probamos que la base de datos está funcionando correctamente, mediante Robomongo, que es un administrador de bases de datos MongoDB.

### 5.2.2.1 NODE.JS

Sólo nos queda la instalación de la parte esencial para el despliegue de la aplicación, el servidor de NODE.JS, para instalar NODE.JS y el gestor de paquetes NPM:

Instalamos primero todo el paquete de Python:

```
apt-get install python-software-properties
```

Añadimos los repositorios de NODE.JS:

```
add-apt-repository ppa:chris-lea/NODE.JS
```

Instalamos NODE.JS y gestor de paquetes npm:

```
apt-get install nodejs npm
```

### 5.2.2 Puesta en producción

Para poner en producción la aplicación, lo primero de todo es acceder mediante el cliente fileZilla al servidor, crearemos una carpeta donde albergaremos toda la aplicación.

Accederemos mediante Robomongo a la base de datos y subiremos todos los datos que necesitamos para que la aplicación pueda arrancar, es importante puntualizar que no se necesita cargar ningún tipo de esquema, ya que estamos trabajando con una base de datos documental.

Se cambia la dirección de la base de datos de desarrollo a producción y por último nos descargamos todos los módulos necesarios para la ejecución de la aplicación, para ello ejecutaremos el siguiente comando en la carpeta de la aplicación:

```
npm install
```

## 6 CONCLUSIONES Y TRABAJOS FUTUROS

En el presente epígrafe vamos a realizar un recorrido de todos los pasos que se han seguido para realizar el proyector fin de grado y los trabajos en un futuro.

### 6.1 Conclusiones

En el presente documento hemos podido ver las diferentes etapas de desarrollo de un producto software, haciendo hincapié en su análisis, diseño, implementación y despliegue de la aplicación de gestión de contenidos e incidencias para un ayuntamiento. Hay que destacar que la aplicación siempre se ha diseñado de manera abierta para que se pueda utilizar en cualquier tipo de organismo, tanto público como privado.

En la primera fase del proyecto descubrimos como se captan las necesidades de una entidad pública para desarrollar un análisis del problema, una vez hecho, aprendimos a diferenciar los requisitos del sistema. Para mí fue la parte más importante del proyecto.

A continuación indico los hitos logrados:

- La gestión de contenidos de la aplicación:
  - Se ha creado una web pública, en la cual se publican todos los contenidos que el ayuntamiento quiere tener presentes en internet, todo ello con un diseño responsivo para dispositivos de escritorio, móvil o Tablet.
  - Se ha realizado un área privada administrativa, donde los administradores (en este caso empleados del ayuntamiento) puedan gestionar los contenidos que se muestran en la web pública.
  - Se ha implementado una herramienta visual basada en los mapas de Google, donde encontramos los diferentes servicios que existen en el municipio de manera georreferenciada. Entre estos servicios podemos destacar la hostelería, las incidencias presentadas o elementos patrimoniales.
  - Se ha creado una herramienta visual para poder visualizar en Google Maps, las distintas rutas turísticas introducidas por el ayuntamiento.
  - Se ha hecho una conexión con Wikipedia para ampliar el contenido de determinados servicios como la información actualizada del municipio.

- La gestión de incidencias
  - Se ha creado un área de usuario en el cual cualquier persona de internet se puede registrar e introducir incidencias relativas al ayuntamiento.
  - Se ha implementado una gestión de incidencias, que mediante un servidor de correo electrónico, avise a la persona responsable de su resolución.
  - Se ha diseñado una interfaz gráfica para indicar la solvencia de las incidencias.

Una vez implementado la aplicación se realizó la configuración de un servidor VPS para poder poner en producción la aplicación y un despliegue de la misma.

Desde el punto de vista personal creo que he aplicado muchos de los conocimientos que he aprendido en el grado y que me han sido muy útiles para conseguir realizar con éxito este proyecto.

## 6.2 Líneas futuras

Como todos los proyectos, se plantean mejoras a lo que está implementado o incluso pueden surgir nuevos requerimientos. En nuestro caso al ser un proyecto de relativa envergadura se dividió en dos partes en un primer momento, la parte del cliente web (y el servidor), que es la que ocupa el presente documento y en segundo lugar la aplicación móvil cliente para consumir los datos que procesa el servidor o enviar peticiones o incidencias.

Vamos a enumerar las líneas futuras a desarrollar. Algunas son relativas a la aplicación móvil y otras a la aplicación web que se han ido planteando mientras esta se desarrollaba:

- Creación de WebServices para que otras aplicaciones puedan consumir los datos procesador por el servidor de NODE.JS.
- Creación de una aplicación móvil en la cual se pueda consumir los datos facilitados por los WebServices. Esta aplicación debería contener notificaciones *push* para informar a los usuarios de que hay una nueva incidencia.
- Añadir servicios a la aplicación sobre realidad virtual.
- Mejorar en el trazado de las rutas en Google Maps.
- Aumentar la seguridad para que sólo los usuarios empadronados en el municipio puedan registrarse para introducir incidencias.

## 7 REFERENCIAS

- [1] Web corporativa de Mozilla HTML <https://developer.mozilla.org/es/docs/Web/HTML>. (Visitado: Febrero 2016).
- [2] Web corporativa de Mozilla JS <https://developer.mozilla.org/es/docs/Web/JavaScript> . (Visitado: Febrero 2016).
- [3] Web corporativa de Mozilla CSS <https://developer.mozilla.org/es/docs/Web/CSS>. (Visitado: Marzo 2016).
- [4] Web para desarrolladores de Google Maps <https://developers.google.com/maps/?hl=es>. (Visitado: Marzo 2016).
- [5] Web Selenium <http://www.seleniumhq.org/> . (Visitado: Marzo 2016).
- [6] Web PUTTY <http://www.putty.org/> (Visitado: Marzo 2016).
- [7] Web FileZilla <https://filezilla-project.org/>. (Visitado: Marzo 2016).
- [8] Adam Bretz(2014) "Full Stack JavaScript Development With MEAN" <https://www.sitepoint.com/full-stack-javascript-development-mean/>. (Visitado: Febrero 2016)
- [9] Web NODE.JS <https://nodejs.org/en/> (Visitado: Febrero 2016)
- [10] Michael Abernethy(2011) "¿Simplemente qué es NODE.JS?" (Visitado: Febrero 2016) <https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/>
- [11] Web del framework Express <http://expressjs.com/es/> (Visitado: Febrero 2016)
- [12] Web del framework AngularJs <https://angularjs.org/> (Visitado: Marzo 2016)
- [13] Juan Roy (2014) "MongoDB: Características y futuro" (Visitado: Enero 2016) <http://www.mongodbspain.com/es/2014/08/17/mongodb-characteristics-future/>
- [14] Margaret Rouse (2015) "Base de datos relacional" (Visitado: Febrero 2016) <http://searchdatacenter.techtarget.com/es/definicion/Base-de-datos-relacional>
- [15] Web del paquete para NODE.JS Mongoose <http://mongoosejs.com/> (Visitado: Marzo 2016)
- [16] Web del paquete para NODE.JS PassportJs <http://passportjs.org/> (Visitado: Febrero 2016)
- [17] Web del paquete para NODE.JS MochaJS <http://mochajs.org/>(Visitado: Febrero 2016)
- [18] Web del paquete para NODE.JS Nodemailer <https://nodemailer.com/> (Visitado: Febrero 2016)
- [19] Brito Acuña, K. (2009) Selección de Metodologías de Desarrollo para Aplicaciones Web en la Facultad de Informática de la Universidad de Cienfuegos, Edición electrónica gratuita. (Visitado: Enero 2016)
- [20] Web herramienta KanbanTool <http://kanbantool.com/es/metodologia-kanban> (Visitado: Enero 2016)



- [21] Web de FING  
<https://www.fing.edu.uy/tecnoinf/mvd/cursos/ingsoft/material/teorico/is05-ArquitecturaDeSoftware.pdf> (Visitado: Enero 2016)
- [22] Juan Pavón. UCM. "El patrón Modelo-Vista-Controlador (MVC)" (Visitado: Febrero 2016)  
<https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf>
- [23] Stefan (2014) "How to test your MongoDB models under Node & Express"  
<https://www.terlici.com/2014/09/15/node-testing.html>. (Visitado: Enero 2016)
- [24] Web del framework para pruebas WebDriver <http://webdriver.io/> (Visitado: Enero 2016).
- [25] Web servidor Debian "ProFTPD" (Visitado: Enero 2016)  
<https://servidordebian.org/es/squeeze/internet/ftp/proftpd>
- [26] Web corporativa MongoDB <https://docs.mongodb.com/v3.0/tutorial/install-mongodb-on-debian/>) (Visitado: Enero 2016)
- [27] David Miller plantilla stylish (Visitado: Enero 2016)  
<http://startbootstrap.com/template-overviews/stylish-portfolio/>
- [28] Almsaedd Studio (Visitado: Enero 2016) plantilla adminLTE .  
<https://almsaeedstudio.com/>
- [29] Web validador de sintaxis HTML, <http://validator.w3.org/>. (Visitado: Enero 2016).

## ANEXO I

### Caso práctico

Para mejorar la comprensión de la aplicación web que se ha desarrollado, mostraremos un caso práctico para introducir una incidencia en el sistema.

#### 1. Entrar en la plataforma

Para entrar en la plataforma nos dirigimos a la siguiente url (por ahora provisional): <http://5.196.20.235:3000>, en donde nos encontraremos el área pública. En esta área encontraremos la diferente información que se ha gestionado en el gestor de contenidos y un enlace al área de administración.

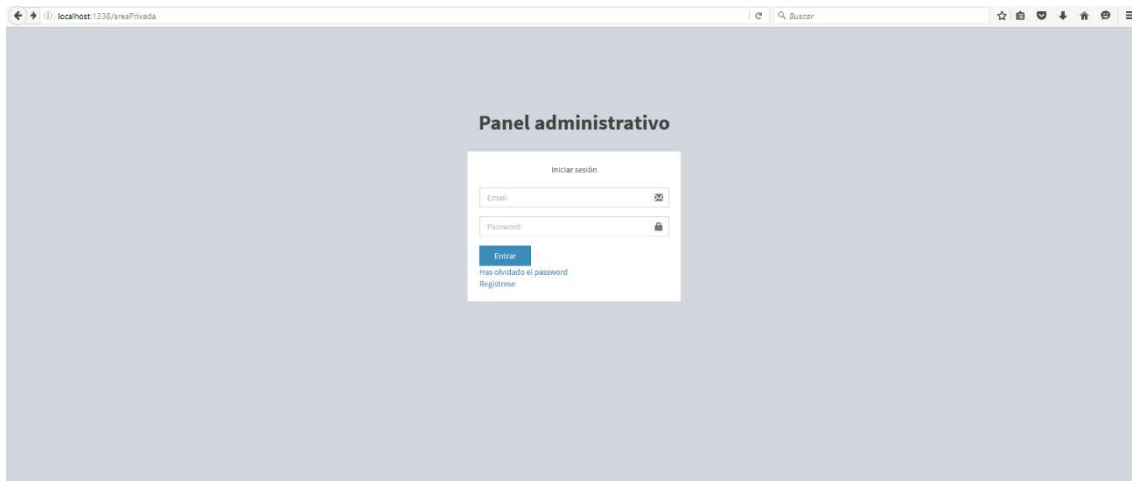


*Ilustración 38. Página principal de la aplicación.*

En la ilustración 38 podemos ver la página de inicio de la aplicación. En nuestro caso nos vamos a centrar en la creación y gestión de una incidencia, por lo que pincharemos en el área privada.

#### 2. Acceso al área de administración

Una vez pinchado con el botón izquierdo del ratón en el área privada nos saldrá un área para la autenticación en el interior del área de gestión de incidencias.



*Ilustración 39. Página de acceso al área privada.*

En la ilustración 39 podemos ver un formulario para autenticarse en la aplicación y así poder introducir una nueva incidencia.

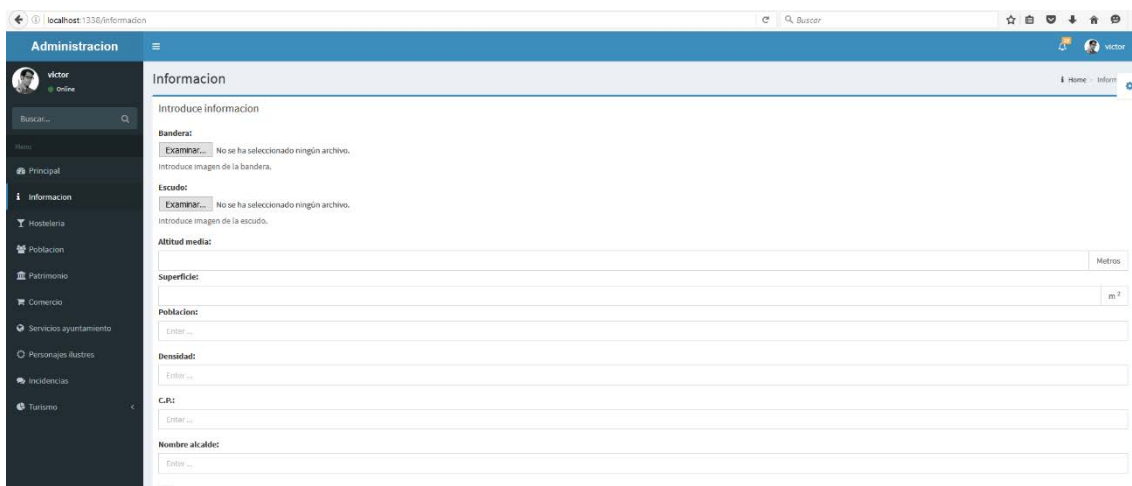
Introducimos los siguientes credenciales para introducirnos en la aplicación.

Email: xxx

Password: xxx

### 3. Área privada.

En esta área veremos todas las funcionalidades del gestor de contenidos y del gestor de incidencias. En este caso al ser una demostración nos hemos *logueado* con una cuenta de administración para ver todas las posibilidades de la aplicación.



*Ilustración 40. Formulario del CMS de información.*

En la ilustración 40 vemos una captura del formulario para introducir la información sobre el municipio, esta información será visualizada en la página principal.

Para introducirnos en el área de incidencias pinchamos en incidencias en el menú izquierdo.

#### 4. Incidencias.

En el siguiente panel nos saldrá la posibilidad de introducir una nueva incidencia y visualizar las incidencias que hay en el sistema.

##### 4.1. Introducir nueva incidencia

Para introducir una nueva incidencia:

1. Colocamos el *marker* de color rojo de la ilustración 39 en la posición deseada.

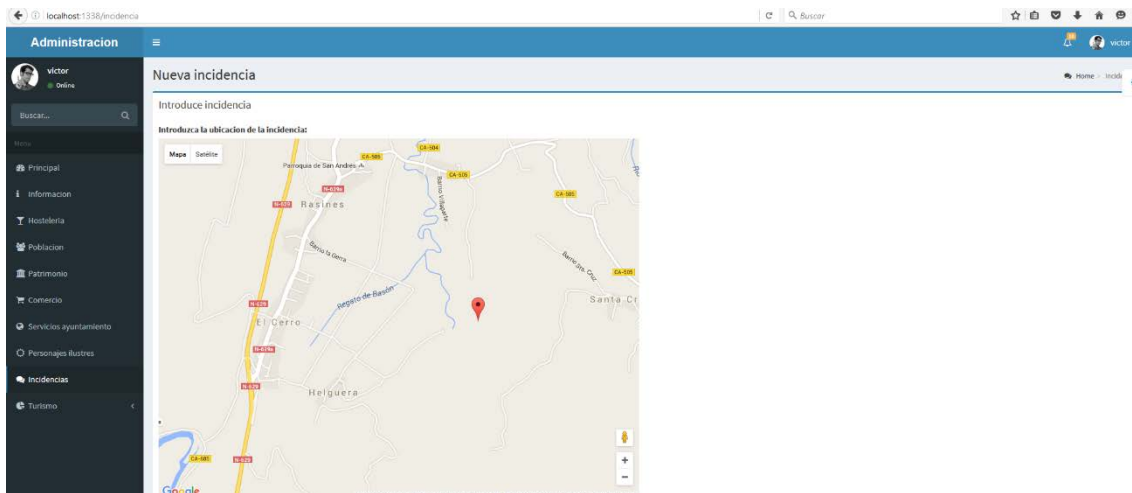


Ilustración 41. Área de nueva incidencia.

2. De inmediato se obtendrán las coordenadas y nos las indicará en los campos de texto que podemos ver en la ilustración 41.
3. Introducimos en el área de texto el motivo de la incidencia.
4. Pulsamos en guardar. La incidencia se ha guardado correctamente en el sistema.

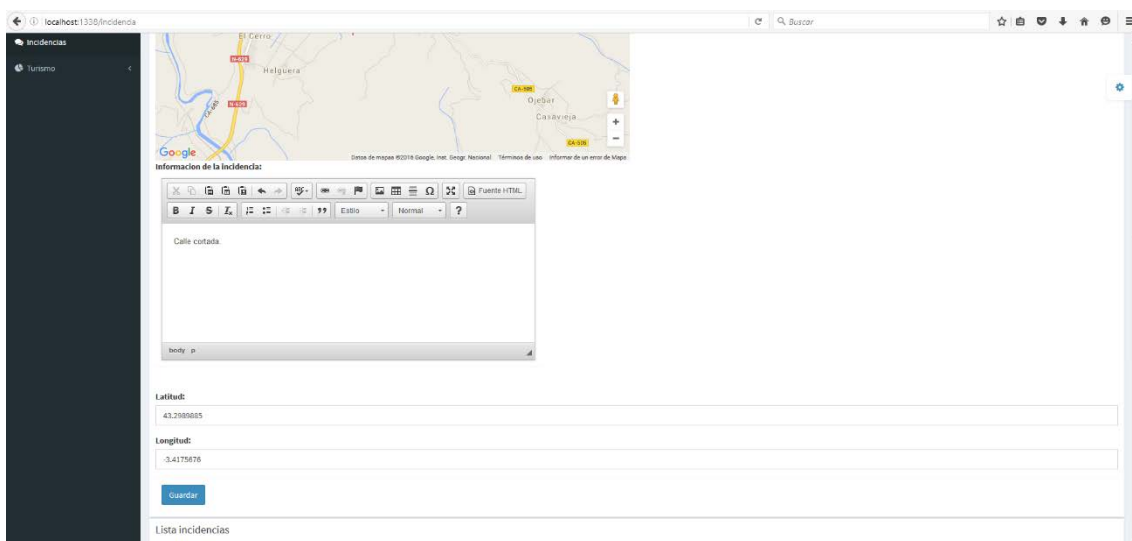
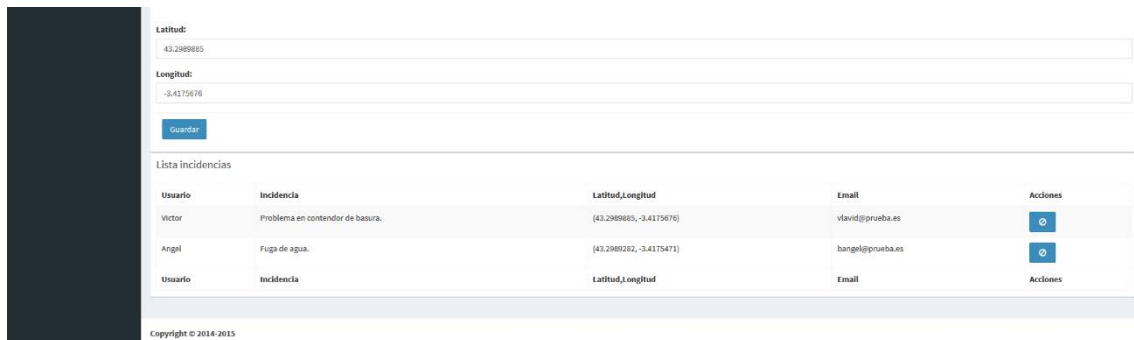


Ilustración 42. Introducir información de la incidencia.

## 4.2. Ver incidencias

Para ver las incidencias que están en el sistema, bajamos hasta el final de la ventana de la aplicación donde encontramos las incidencias registradas, además como somos administradores, tenemos la posibilidad de solucionar la incidencia pulsando el botón que está al lado de la incidencia que queremos resolver.





Latitude: 43.298985

Longitude: -3.4175678

Guardar

Lista incidencias

Usuario	Incidencia	Latitud,Longitud	Email	Acciones
Victor	Problema en contenedor de basura.	(43.298985, -3.4175678)	victor@prueba.es	
Angel	Fuga de agua.	(43.298985, -3.4175471)	angel@prueba.es	

Copyright © 2014-2015

Ilustración 43. Lista de incidencias.