



UNIVERSIDAD DE OVIEDO



MÁSTER UNIVERSITARIO EN INGENIERÍA WEB

TRABAJO FIN DE MÁSTER

“APLICACIÓN WEB ESCALABLE PARA GESTIÓN DE INFORMACIÓN GEOLOCALIZADA”

JAVIER MÉNDEZ MENÉNDEZ

El/la tutor/a autoriza la defensa del Trabajo Fin de Máster

(FIRMA)

Fdo. D. /Dña. Darío Álvarez Gutiérrez.....

Resumen

Durante los últimos años Internet y el mundo tecnológico en general han sufrido profundos cambios que han posibilitado la aparición de nuevos conceptos, tecnologías y herramientas que están transformando el modelo de desarrollo y explotación de aplicaciones existente hasta el momento. Ejemplos de esta nueva situación son conceptos como *NoSQL*, *Responsive Design*, escalabilidad, metodologías ágiles, *Test Driven Development*; tecnologías como *Cloud Computing*, Geolocalización; herramientas de desarrollo como *Ruby On Rails*, *Django*, *Node.js*, etc.

En este proyecto se ha abordado el desarrollo de una aplicación web y su posterior despliegue en un servidor real sobre una arquitectura empresarial, usando las nuevas técnicas mencionadas anteriormente. En concreto se ha desarrollado una aplicación que gestiona información(incluyendo su localización geográfica) sobre eventos de todo tipo(cine, música, teatro, etc). A dichos eventos se les pueden añadir comentarios realizados por los usuarios registrados, incluyendo de este modo características propias de las nuevas redes sociales. También es posible realizar búsquedas basadas en la localización geográfica de los eventos.

Para ello se ha usado *Django*(framework de desarrollo web basado en *Python*), con *MongoDB* como base de datos que aporta sus posibilidades de escalabilidad basadas en su sistema de distribución, además de la potencia de su API de geolocalización para realizar las búsquedas basadas en la posición geográfica..

Posteriormente se ha desplegado en la plataforma *AWS*(Amazon Web Services) donde se ha diseñado una arquitectura escalable, es decir que responde de manera flexible ante aumentos y caídas bruscas de tráfico, adaptando el número de servidores a la carga de trabajo en cada momento.

Además se ha diseñado un plan de crecimiento, de modo que en un futuro la arquitectura pueda adaptarse de modo rápido y eficaz a posibles incrementos de número de usuarios o de tráfico.

En todo caso, el fin último de este proyecto ha sido el aprendizaje de las tecnologías y herramientas usadas por parte del proyectante, con vistas a mejorar sus capacidades en el futuro mercado laboral.

Palabras Clave

NoSQL, Responsive Design, Django, MongoDB, MongoEngine, AWS(Amazon Web Services), *TDD*(Test Driven Design), *Scrum*, escalabilidad.

Índice General

CAPÍTULO 1. INTRODUCCIÓN.....	11
1.1 JUSTIFICACIÓN DEL PROYECTO.....	11
1.2 OBJETIVOS DEL PROYECTO.....	13
1.3 DESCRIPCIÓN DE LA APLICACIÓN WEB.....	13
1.4 ALCANCE DEL PROYECTO.....	15
CAPÍTULO 2. ASPECTOS TEÓRICOS Y TÉCNICOS	17
2.1 CONCEPTOS.....	17
2.1.1 <i>NoSQL</i>	17
2.1.2 <i>Metodologías Ágiles</i>	18
2.1.3 <i>Test Driven Development (TDD)</i>	18
2.1.4 <i>Responsive Design</i>	19
2.1.5 <i>Escalabilidad</i>	21
2.2 TECNOLOGÍAS.....	24
2.2.1 <i>Geolocalización</i>	24
2.2.2 <i>Cloud Computing</i>	25
2.2.3 <i>Ajax</i>	26
2.3 HERRAMIENTAS.....	26
2.3.1 <i>Django</i>	26
2.3.2 <i>MongoDB</i>	28
2.3.3 <i>Amazon Web Services</i>	29
2.3.4 <i>MongoEngine</i>	31
2.3.5 <i>Boto</i>	31
2.3.6 <i>Git</i>	32
2.3.7 <i>VirtualEnv</i>	32
2.3.8 <i>Gmap3</i>	32
CAPÍTULO 3. PLANIFICACIÓN DEL PROYECTO.....	35
3.1 PLANIFICACIÓN INICIAL.....	35
3.2 PLANIFICACIÓN INTERMEDIA	36
3.3 PLANIFICACIÓN FINAL	38
3.4 CONCLUSIONES FINALES	39
CAPÍTULO 4. DESARROLLO DE LA APLICACIÓN WEB	41
4.1 DESCRIPCIÓN DE LA METODOLOGÍA EMPLEADA	41
4.2 DESCRIPCIÓN DEL PLAN DE PRUEBAS	43
4.2.1 <i>Pruebas unitarias y funcionales</i>	43
4.2.2 <i>Pruebas de integración y rendimiento</i>	44
4.2.3 <i>Pruebas de usabilidad y accesibilidad</i>	44
4.3 ANÁLISIS PREVIO.....	45
4.3.1 <i>Product Backlog</i>	45
4.3.2 <i>Tabla de requisitos</i>	46
4.4 DISEÑO PREVIO	47
4.4.1 <i>Diseño de la jerarquía de desarrollo Django</i>	47
4.4.2 <i>Diseño previo de clases</i>	47

4.4.3	<i>Diseño previo de la base de datos</i>	48
4.4.4	<i>Diseño previo de la interfaz de usuario</i>	48
4.5	SPRINT 1	51
4.5.1	<i>Sprint Backlog</i>	51
4.5.2	<i>Consideraciones previas</i>	51
4.5.3	<i>Análisis de historias de usuario</i>	53
4.5.4	<i>Diseño</i>	56
4.5.5	<i>Tests unitarios</i>	58
4.6	SPRINT 2	62
4.6.1	<i>Sprint Backlog</i>	62
4.6.2	<i>Consideraciones previas</i>	63
4.6.3	<i>Análisis de historias de usuario</i>	64
4.6.4	<i>Diseño</i>	67
4.6.5	<i>Tests unitarios</i>	69
4.7	SPRINT 3	71
4.7.1	<i>Sprint Backlog</i>	71
4.7.2	<i>Consideraciones previas</i>	72
4.7.3	<i>Análisis de historias de usuario</i>	73
4.7.4	<i>Diseño</i>	74
4.7.5	<i>Tests unitarios</i>	76
4.8	DOCUMENTACIÓN FINAL.....	79
4.8.1	<i>Descripción detallada de clases y métodos</i>	79
4.8.2	<i>Mapa de rutas del sistema</i>	85
4.8.3	<i>Descripción detallada del esquema de base de datos</i>	86
4.8.4	<i>Descripción de la navegabilidad</i>	87
4.8.5	<i>Responsive design</i>	91
CAPÍTULO 5. DESPLIEGUE DE LA APLICACIÓN		95
5.1	ELECCIÓN DE LA PLATAFORMA.....	95
5.2	DISEÑO DE LA ARQUITECTURA.....	96
5.2.1	<i>Capa de aplicación</i>	96
5.2.2	<i>Capa de datos</i>	97
5.2.3	<i>Arquitectura final</i>	99
5.3	ADAPTACIÓN DE LA APLICACIÓN A LA PLATAFORMA ELEGIDA	101
5.4	ESTRUCTURA DEL SERVIDOR DE APLICACIONES.....	101
5.5	CONSIDERACIONES SOBRE EL HARDWARE	102
5.5.1	<i>Capa de aplicación</i>	102
5.5.2	<i>Capa de datos</i>	103
5.6	CONSIDERACIONES SOBRE LA LOCALIZACIÓN GEOGRÁFICA DE LOS SERVIDORES	105
5.7	ELECCIÓN DE LA SHARD KEY	107
5.8	TAREAS DE DESPLIEGUE.	108
5.8.1	<i>Despliegue de la capa de aplicación</i>	108
5.8.2	<i>Despliegue de la capa de datos</i>	109
5.9	PRUEBAS DE RENDIMIENTO.	110
5.9.1	<i>Volumen de datos</i>	110
5.9.2	<i>Balancedador</i>	111
5.9.3	<i>Conclusiones</i>	113
5.10	PLAN DE CRECIMIENTO.	113
5.10.1	<i>Capa de aplicación</i>	113
5.10.2	<i>Capa de datos</i>	114

CAPÍTULO 6. MANUALES DEL SISTEMA	117
6.1 MANUAL DE DESPLIEGUE.....	117
6.1.1 Crear instancia EC2 para base de datos en AWS.....	117
6.1.2 Instalar y configurar MongoDB en la instancia.....	118
6.1.3 Crear una imagen base para añadir nuevos servidores.....	119
6.1.4 Crear dos nuevas instancias a partir de la imagen.....	120
6.1.5 Configurar el "Replica Set".....	121
6.1.6 Crear la estructura de ficheros estáticos en AWS S3.....	121
6.1.7 Crear instancia EC2 para servidor de aplicaciones en AWS.....	122
6.1.8 Instalar el software necesario en la instancia.....	123
6.1.9 Clonar en el servidor el repositorio.....	123
6.1.10 Instalar los paquetes python necesarios.....	123
6.1.11 Modificamos el fichero de configuración de python.....	124
6.1.12 Configurar Nginx.....	124
6.1.13 Iniciar Nginx y Gunicorn.....	124
6.1.14 Configurar la máquina para que nginx y gunicorn se ejecuten al iniciarla.....	125
6.1.15 Creamos una imagen para usarla como base.....	125
6.1.16 Creamos un Elastic Load Balancer en AWS.....	126
6.1.17 Creamos un Auto Scaling Group AWS.....	128
6.2 MANUAL DE SHARDING	134
6.2.1 Crear "Config Servers".....	134
6.2.2 Arrancar y configurar procesos "mongos".....	135
6.2.3 Habilitar Sharding en el sistema.....	136
6.3 MANUAL DE USO DE LA APLICACIÓN WEB.....	137
CAPÍTULO 7. CONCLUSIONES Y AMPLIACIONES	145
7.1 CONCLUSIONES	145
7.2 AMPLIACIONES.....	145
CAPÍTULO 8. PRESUPUESTO.....	147
8.1 PRESUPUESTO INICIAL.....	148
8.2 PRESUPUESTO FINAL.....	149
CAPÍTULO 9. REFERENCIAS BIBLIOGRÁFICAS	150
9.1 LIBROS Y ARTÍCULOS.....	150
9.2 REFERENCIAS EN INTERNET	150
CAPÍTULO 10. APÉNDICES.....	151
10.1 CONTENIDO ENTREGADO EN EL ANEXO.....	151
10.2 ÍNDICE ALFABÉTICO.....	152
10.3 CÓDIGO FUENTE.....	153
10.3.1 Fichero "views.py".....	153
10.3.2 Fichero "models.py".....	156
10.3.3 Fichero "forms.py".....	157
10.3.4 Fichero "urls.py".....	159
10.3.5 Fichero "storages.py".....	159
10.3.6 Fichero "settings.py".....	159
10.3.7 Fichero "tests.py".....	161
10.3.8 Fichero "addevent.html".....	172
10.3.9 Fichero "base.html".....	173

10.3.10	Fichero "comments.html".....	173
10.3.11	Fichero "deleteevent.html".....	174
10.3.12	Fichero "editevent.html".....	174
10.3.13	Fichero "eventslist.html".....	175
10.3.14	Fichero "home.html".....	176
10.3.15	Fichero "login.html".....	177
10.3.16	Fichero "register.html".....	178
10.3.17	Fichero "searchevents.html".....	178
10.3.18	Fichero "home.js".....	179

Índice de Figuras

Figura 1.1. Tabla número de usuarios de internet por año	11
Figura 2.1 Responsive design tamaño ordenador	20
Figura 2.2 Responsive design tamaño tableta	20
Figura 2.3 Responsive design tamaño móvil.....	21
Figura 2.4 Esquema escalabilidad horizontal.....	22
Figura 2.5. Arquitectura MVT de Django	27
Figura 3.1 Planificación inicial	36
Figura 3.2 Planificación intermedia	37
Figura 3.3 Planificación final.....	39
Figura 4.1 Product Backlog	45
Figura 4.2 Diseño previo clases	47
Figura 4.3 Diseño previo BD	48
Figura 4.4 Diseño previo interfaz ordenador	49
Figura 4.5 Diseño previo interfaz tablet 1.....	49
Figura 4.6 Diseño previo interfaz tablet 2.....	50
Figura 4.7 Diseño previo interfaz móvil	50
Figura 4.8 Sprint1 Backlog	51
Figura 4.9 Sprint1 Diseño clases	56
Figura 4.10 Sprint1 Diseño datos.....	57
Figura 4.11 Sprint1 Diseño interfaz	57
Figura 4.12 Sprint2 Backlog	63
Figura 4.13 Sprint2 Diseño clases	67
Figura 4.14 Sprint2 Diseño datos.....	68
Figura 4.15 Sprint2 Diseño interfaz	68
Figura 4.16 Sprint3 Backlog	72
Figura 4.17 Sprint3 Diseño clases	75
Figura 4.18 Sprint3 Diseño datos.....	75
Figura 4.19 Sprint3 Diseño interfaz	76
Figura 4.20 Pagina inicial sin loguear	87
Figura 4.21 Diagrama navegabilidad sin loguear	88
Figura 4.22 Página inicial logueado	88
Figura 4.23 Diagrama navegabilidad logueado	89
Figura 4.24 Detalle navegación listas eventos	90
Figura 4.25 Diagrama navegabilidad listas eventos	91
Figura 4.26 Responsive design tamaño ordenador	91
Figura 4.27 Responsive design tamaño tablet	92
Figura 4.28 Responsive design tamaño móvil.....	93
Figura 5.1 Diseño arquitectura	96
Figura 5.2 Despliegue MongoDB producción	97
Figura 5.3 Replica Set	99
Figura 5.4 Despliegue final inicial	100
Figura 5.5 Despliegue Empresarial	101
Figura 5.6 Esquema servidor aplicaciones	102
Figura 5.7 Detalle precios instancias AWS C3	103
Figura 5.8 Detalle precios instancias AWS I2	104

Figura 5.9 Detalle precios instancias AWS Micro	104
Figura 5.10 Regiones AWS	105
Figura 5.11 AWS Regiones y zonas.....	106
Figura 5.12 Despliegue Replica Set	106
Figura 5.13 Datos rendimiento por volumen de datos	111
Figura 5.14 Datos rendimiento balanceador	112
Figura 5.15 Datos rendimiento por número servidores.....	112
Figura 5.16 Esquema arquitectura evolucionada	115
Figura 6.1 Creación instancia AWS - Add Storage.....	118
Figura 6.2 Crear imagen instancia AWS.....	119
Figura 6.3 Submenú AMIs AWS.....	120
Figura 6.4 Opción My AMIs AWS	120
Figura 6.5 Selección Subnet AWS.....	121
Figura 6.6 Detalle S3 AWS.....	122
Figura 6.7 Detalle instancias AWS.....	123
Figura 6.8 Opción Create Image AWS	125
Figura 6.9 Lista imágenes AWS	126
Figura 6.10 Submenú "Load Balancers" AWS	126
Figura 6.11 Crear Load Balancer AWS Paso 1	127
Figura 6.12 Crear Load Balancer AWS Paso 1	127
Figura 6.13 Lista balanceadores AWS	128
Figura 6.14 Submenú "Launch Configuration"	129
Figura 6.15 Lista imágenes AWS	129
Figura 6.16 Create Launch Configuration AWS paso 5.....	130
Figura 6.17 Launch Configuration AWS creada	130
Figura 6.18 Create Auto Scaling Group AWS paso 1	131
Figura 6.18 Create Auto Scaling Group AWS paso 1	131
Figura 6.19 Create Alarm AWS.....	132
Figura 6.20 Create Alarm AWS Increase condition	132
Figura 6.21 Create Alarm AWS decrease condition	133
Figura 6.22 Lista instancias Auto Scaling Group AWS.....	133
Figura 6.23 Página inicial WorldEvents	137
Figura 6.24 Mapa página inicial WorldEvents	138
Figura 6.24 Formulario Create Account	138
Figura 6.25 Formulario Login	139
Figura 6.26 Formulario Search	139
Figura 6.27 Detalle links Edit, Delete y Add comment.....	140
Figura 6.28 Detalle formulario Add Comment	141
Figura 6.29 Detalle comentario.....	141
Figura 6.30 Opción Map/Detail.....	142
Figura 6.31 Lista eventos pantalla móvil	143
Figura 6.32 Link Back to list	144
Figura 8.1 Precios Google Maps	147
Figura 8.2 Presupuesto inicial	148
Figura 8.3 Presupuesto inicial modificado	149
Figura 8.4 Presupuesto Final.....	149

Capítulo 1. Introducción

1.1 Justificación del Proyecto

Durante los últimos años el número de usuarios de Internet y en consecuencia el tráfico en la red, ha sufrido un progresivo incremento, principalmente impulsado por la aparición de nuevos dispositivos móviles (smartphones y tablets) que han multiplicado las posibilidades de acceso para los usuarios. Según CNN.com en el mes de febrero de 2014 el número de usuarios que accedieron a Internet en Estados Unidos usando un dispositivo móvil superó a los que lo hicieron a través del PC. En la siguiente tabla se puede apreciar que aproximadamente durante el periodo 2006-2014 se ha pasado de 1000 a 3000 millones de usuarios.

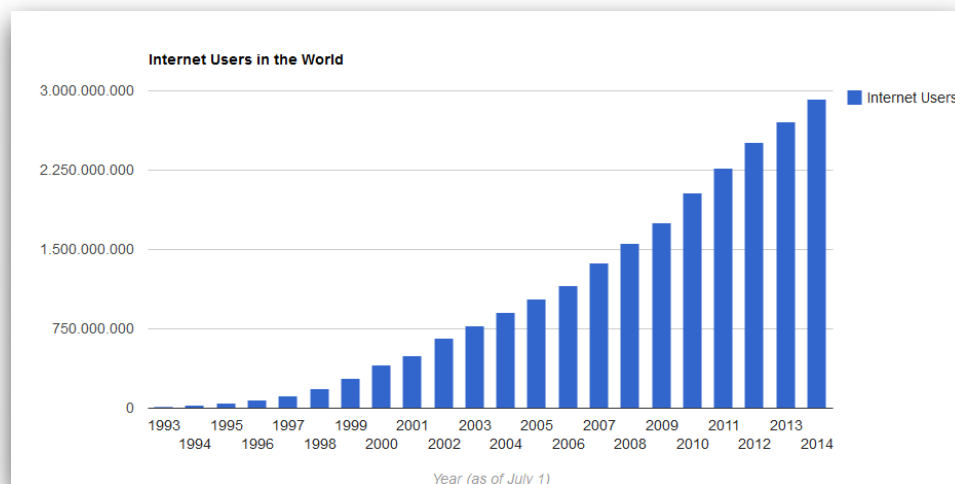


Figura 1.1. Tabla número de usuarios de internet por año

A estos datos se une el auge de las soluciones basadas en la nube, lo que ha incrementado de manera exponencial la cantidad de datos subidos a Internet. De hecho, se estima que el 90% de los datos que existen actualmente en Internet, han sido generados durante los dos últimos años (Estadística de 2012).

Por otra parte, actualmente se está desarrollando un activo ecosistema emprendedor en torno a las llamadas *startups*, tanto a nivel nacional como internacional. Esto está provocando la rápida proliferación de aplicaciones web que en muchos casos acaban superando las expectativas iniciales de sus creadores en cuanto a su éxito y uso, sin estar preparadas ni diseñadas para ello. Esto lleva a situaciones en las que dichas aplicaciones no pueden adaptarse al rápido incremento de usuarios y tráfico, y acaban fracasando, haciendo cada vez más habitual la expresión "morir de éxito".

Este nuevo escenario está provocando cambios profundos en todos los aspectos que afectan a la red de redes. Empezando por la metodología, tecnologías y herramientas usadas para crear sitios web, pasando por las arquitecturas y herramientas usadas para su despliegue y paso a producción y finalizando por los dispositivos, sistemas operativos y navegadores usados para su acceso.

Se pueden mencionar los siguientes aspectos:

- La aparición del concepto *Big Data* para referirse a las técnicas para el tratamiento y análisis de las enormes cantidades de datos que actualmente contiene Internet.
- *NoSQL* vs modelo relacional. Las bases de datos relacionales comenzaron a sufrir degradaciones importantes de rendimiento a partir de un determinado tamaño de datos. Para resolver esta carencia aparecieron las llamadas bases de datos *NoSQL*.
- La aparición de frameworks de desarrollo web como *Ruby On Rails* o *Django*, más modernos y adaptados a este nuevo escenario frente a opciones tradicionales como *JEE* o *PHP*.
- El progresivo éxito de servidores web ligeros como *NGINX* o *Node.js* frente a *Apache* o *IIS*, más lentos y poco adecuados para las nuevas arquitecturas.
- El auge de las metodologías ágiles como *SCRUM* o *XP*, frente a las tradicionales más lentas y menos flexibles frente a los cambios.
- *Responsive Design* como respuesta a la gran diversificación de los dispositivos de acceso a Internet.
- La inclusión de geolocalización en las aplicaciones apoyada por las capacidades de los dispositivos móviles y por herramientas como *Google Maps*.
- El éxito del Cloud Computing como plataforma de despliegue de aplicaciones, refrendado por el crecimiento de servicios como *Amazon Web Services*, *Google App Engine*, etc.
- La creciente importancia del concepto de "escalabilidad" definido como la capacidad de un sistema de adaptarse a variaciones de tráfico o de carga de trabajo sin perder calidad en sus servicios.

En este nuevo escenario, el dominio de estas nuevas "habilidades" está siendo cada vez más demandado por el mercado laboral. Perfiles de experto en *Big Data* o arquitecto Cloud son cada vez más habituales en las ofertas de empleo. Sin embargo el mundo de la educación, más lento y burocrático, no está dando una respuesta ágil a estas nuevas necesidades de las empresas.

Como consecuencia de esta situación, se ha planteado este proyecto como un ejercicio de investigación y aprendizaje de estas nuevas habilidades por parte del proyectante, buscando explotar, ampliar y adaptar los conocimientos adquiridos en el Máster en Ingeniería Web a este nuevo entorno.

1.2 Objetivos del Proyecto

Los objetivos principales del proyecto son dos:

- Investigar y aprender el uso de nuevos conocimientos que están siendo cada vez más demandados en el mercado laboral.
- Desarrollar usando esos conocimientos una aplicación web que sea escalable y desplegarla en un servicio *Cloud Computing* de modo que pueda adaptarse a variaciones en la carga de trabajo sin perder prestaciones.

Estos objetivos se pueden concretar en los siguientes puntos:

- Desarrollar un proyecto software usando metodologías ágiles.
- Usar un framework de desarrollo como *Django* basado en *Python*.
- Usar una base de datos *NoSQL* para garantizar el rendimiento de la aplicación incluso en escenarios con cantidades muy grandes de datos y un alto número de consultas
- Incluir en la aplicación características y capacidades de geolocalización, usando servicios como *Google Maps*.
- Desarrollar la aplicación usando *Responsive Design* para que pueda adaptarse a cualquier dispositivo o tamaño de pantalla.
- Diseñar la aplicación de manera que se pueda adaptar a cambios muy bruscos en la densidad del tráfico.
- Adaptar la aplicación para su despliegue en un servicio *Cloud Computing* empresarial.
- Diseñar una arquitectura usando los servicios de la plataforma *Cloud Computing*, de manera que la aplicación se adapte automáticamente al tráfico existente en cada momento y ofrezca en todo momento unas prestaciones mínimas, independientemente del tamaño de los datos, del número de usuarios, de los posibles fallos hardware, etc.

1.3 Descripción de la aplicación web

Para la realización de este proyecto se va a desarrollar un aplicación web. Esta servirá por una parte para practicar las técnicas y herramientas cuyo dominio es objetivo de este proyecto. Por otra parte será la base para luego abordar la segunda parte que es el diseño y posterior realización de una arquitectura escalable. Por ello se ha optado por una aplicación con unas características que dentro de lo posible se adapten a los objetivos del proyecto.

En primer lugar, la aplicación ha de tener un número elevado de potenciales usuarios, ya que uno de los objetivos del proyecto es poder analizar y diseñar su escalabilidad. Es decir, no serviría algo como "Guía de bares de Madrid" ya que el número de posibles usuarios estaría limitado. Por ello se ha optado por realizar una aplicación que maneje una información lo más general posible. La finalidad será entonces almacenar información o anuncios de eventos de todo tipo. Pueden ser conciertos musicales, obras de teatro, cine, conferencias, apertura de

restaurantes, fiestas, etc. La idea es que los usuarios de la aplicación tengan disponible todo tipo de información para organizar su tiempo libre.

Por otra parte, enlazando también con el punto anterior sería deseable que ese potencial alto número de usuarios se tradujera en un volumen también elevado de datos, de manera que se justificara el uso de soluciones *NoSQL*. Hay que recordar que la principal característica de este tipo de sistemas, así como el motivo de su creación, es su capacidad para tratar con grandes volúmenes de datos sin perder prestaciones. Por ello y buscando un alto número de usuarios y de datos no se va a limitar el ámbito geográfico de la aplicación, y se va a buscar que la geolocalización sea una de sus principales capacidades. Para ello será obligatorio que entre la información de los eventos se incluya su localización geográfica, y que el sistema ofrezca la posibilidad de realizar búsquedas seleccionando áreas geográficas. De este modo se aprovecha también una de las principales características de *MongoDB*, que es su soporte para este tipo de datos geográficos. Con este planteamiento el sistema ofrecerá la posibilidad de realizar búsquedas del tipo: "Dime los eventos culturales que haya en mi ciudad", "dime que partidos de fútbol hay anunciados en un mi región" o "busca los eventos relacionados con el cine que contengan la palabras 'De Niro' entre su información en un radio de 50 km desde mi posición actual."

También es necesario que la aplicación no tenga un esquema de datos muy complejo. Esto es debido a que una de las desventajas de los sistemas *NoSQL*, es que al no contar con relaciones, no son muy adecuados para modelar sistemas que tengan muchas entidades y relaciones complejas entre ellas. Por ello en la aplicación web se va a contar un mapa de datos bastante simple y que ejemplifique las técnicas de modelado de datos de los sistemas *NoSQL*. La base del sistema serán los eventos y se incluirán también usuarios que podrán darse de alta en el sistema y gestionar sus propios eventos. Además se incluirá la posibilidad de que los usuarios puedan realizar comentarios sobre los eventos. Esta característica es especialmente adecuada para resaltar las diferencias entre el modo de trabajo de los sistemas relacionales frente al mundo *NoSQL*. En una base de datos relacional habría una entidad Evento que estaría relacionada con otra entidad Comentario. En cambio la filosofía *NoSQL* aboga por incluir en la misma entidad Evento toda su información relacionada incluyendo sus comentarios. Además esta característica añade funcionalidades sociales típicas de las aplicaciones actuales, de esta manera se hace la aplicación más atractiva a los potenciales usuarios.

Teniendo en cuenta que la idea es contar con un número elevado de usuarios, es necesario también añadir capacidades de internacionalización, ya que se espera contar con usuarios de distintas nacionalidades. Por el mismo motivo parece también adecuado que la aplicación siga la filosofía *Responsive Design* para facilitar el acceso a la mayor cantidad de personas independientemente de su dispositivo de acceso.

A la hora de abordar el desarrollo de una aplicación de estas características parece natural no limitarse solo a implementar su funcionalidad, sino también plantearse si la aplicación seguirá estando operativa cuando el volumen de datos y tráfico se incremente. Y en base a ello tomar decisiones de diseño que solucionen o prevengan esos potenciales problemas.

1.4 Alcance del Proyecto

En un proyecto de estas características donde se va a afrontar la investigación y aprendizaje de un buen número de nuevos conceptos, tecnologías y herramientas es difícil definir un alcance realista. En efecto, es complicado calcular el tiempo necesario para adquirir un dominio aceptable de los nuevos conocimientos. También está la dificultad de estimar la velocidad de desarrollo usando técnicas con las que nunca se han trabajado. En este contexto y teniendo en cuenta que se va a utilizar una metodología iterativa, se va a seguir el plan de ir realizando sucesivas iteraciones en el desarrollo de la aplicación. En cada una de ellas se irán incorporando nuevas funcionalidades al proyecto, pero siempre obteniendo al final de cada una, un producto final terminado y operativo. De este modo, en todo momento se dispondrá de una aplicación prototipo sobre la que desarrollar la otra parte del proyecto que es su despliegue en un servidor real con una arquitectura escalable. De hecho, la aplicación web no se puede ver en este proyecto como un fin, sino más bien como un medio para alcanzar los objetivos. Por ello, solo se exigirá dentro del alcance de este proyecto una funcionalidad mínima, que en todo caso y como en todo proyecto real vendrá condicionada por los plazos de entrega.

La aplicación a desarrollar consistirá en un sitio web donde se podrá subir y consultar información asociada a todo tipo de eventos (cine, música, teatro, etc). Dicha información incluirá de manera obligatoria su localización geográfica, que servirá como parámetro para realizar búsquedas geolocalizadas dentro del sistema. La aplicación permitirá gestionar cuentas de usuario que podrán añadir, editar y eliminar sus propios eventos. Una vez alcanzada esta funcionalidad mínima y dependiendo del tiempo disponible ya se podrá pasar a la segunda fase del proyecto que será el diseño y despliegue de una arquitectura escalable sobre un servicio de *Cloud Computing* empresarial.

Capítulo 2. Aspectos Teóricos y técnicos

2.1 Conceptos

2.1.1 NoSQL

El primer campo que sufrió los problemas derivados del aumento de tráfico y datos fue el de las bases de datos relacionales. En un momento dado, las aplicaciones que contaban con mayor número de usuarios y tráfico dentro de Internet (Google, Facebook, Twitter, etc) constataron que a partir de un determinado tamaño de carga de trabajo las bases de datos tradicionales sufrían una degradación de rendimiento considerable. Debido a este problema cada una de estas grandes empresas tuvo que desarrollar su propia solución para almacenamiento y gestión de sus datos. Esto dio lugar a las bases de datos *NoSQL*, que como su nombre indica no siguen en absoluto el modelo relacional. Al eliminar toda la base matemática sobre la que se asientan las bases de datos relacionales, obtienen un mejor rendimiento pero a costa de perder ventajas como las propiedades ACID que caracterizan a dichas herramientas. En todo caso su uso actualmente está en constante incremento y es un campo abierto en investigación. Herramientas como *Cassandra*, *MongoDB*, *CouchDB*, etc, cada vez son más populares en el mundo informático.

Las principales características de estas bases de datos son:

- Ausencia de esquema. Es decir los datos no tienen que seguir una estructura rígida.
- Escalabilidad horizontal sencilla y eficiente.
- Gran velocidad en sus operaciones debido a la eliminación de relaciones y transacciones.
- No garantizan la consistencia de los datos en todo momento.
- No suelen incluir transacciones.
- Al no usar relaciones, no son adecuadas para modelar esquemas de datos muy complicados.

2.1.1.1 Aplicación en el proyecto

En este proyecto se usa *MongoDB*, una de las bases de datos *NoSQL* más usadas en la actualidad y que gracias a su sistema de replicas y distribución garantiza unas prestaciones adecuadas trabajando con grandes cantidades de datos y con altas tasas de tráfico.

2.1.2 Metodologías Ágiles

Las metodologías ágiles son una alternativa a los métodos tradicionales de desarrollo de software basados principalmente en el modelo en cascada, y que muchas corrientes de opinión actuales consideran demasiado rígidos por su excesiva dependencia de planificaciones y documentación detallada, y su poca flexibilidad ante los cambios durante la fase de desarrollo.

Su principal característica es que suelen apostar por un ciclo de vida iterativo frente al tradicional modelo en cascada. De ese modo en lugar de realizar un único producto final, se opta por desarrollar pequeños proyectos intermedios llamados iteraciones, cada uno incorporando nuevas funcionalidades y todos ellos plenamente operativos y listos para entregar al cliente, que de este modo puede valorar la marcha del proyecto e introducir modificaciones en cualquier momento.

2.1.2.1 Aplicación en el proyecto

Este proyecto está concebido como un ejercicio de auto aprendizaje y dado que nunca se puede predecir el tiempo necesario para adquirir un dominio adecuado de una nueva tecnología o herramienta, es difícil definir el alcance del proyecto o hacer una planificación real en este caso. Por ello en lugar de definir desde un principio y de modo estricto las funcionalidades de la aplicación y realizar todo el proceso de desarrollo siguiendo la metodología tradicional en cascada, se ha optado por seguir la filosofía ágil. De este modo se ha comenzado desarrollando una aplicación con una funcionalidad mínima, y a continuación en sucesivas iteraciones se han ido añadiendo nuevas funcionalidades. De este modo independientemente del tiempo disponible o de las dificultades encontradas durante el desarrollo, siempre se ha podido contar un producto terminado para presentar al hipotético cliente.

2.1.3 Test Driven Development (TDD)

Es una práctica de programación que consiste en escribir primero los test antes que el código a probar, y a continuación apoyándose en estos desarrollar el código. En la programación tradicional se suele seguir el proceso inverso. En la práctica se trata de a partir de los requisitos iniciales desarrollar los tests necesarios para validar el código que cumpla dichos requisitos. A continuación se escribe el código mínimo necesario para que los tests finalicen correctamente. Finalmente es habitual un proceso de refactorización, para añadir casos de prueba más detallados, control de errores y en general optimizar el código generado. Las principales ventajas de esta técnica son:

- Evita escribir código innecesario.

- Al estar en todo momento guiado por los tests, el código generado siempre tiene menos fallos que si se partiera de cero. Esto hace que la necesidad de depuración sea muy poco habitual.
- Cualquier modificación o añadido posterior siempre se va a hacer de modo seguro, ya que en todo momento las funcionalidades ya creadas están protegidas por los tests. Esto hace que las ampliaciones y evoluciones del código sean menos costosas. Está es una de las razones por las que metodologías ágiles y *TDD* suelen ir de la mano.

Como desventaja principal se puede citar que elaborar tests de calidad no es una tarea trivial, requiere experiencia y conocimientos profundos ya que en muchos casos es necesario el aprendizaje de librerías y técnicas específicas. Además la calidad de las pruebas elaboradas es un parámetro decisivo para medir el éxito final en el uso de esta técnica.

Por otra parte, esta técnica suele abarcar solo las llamadas pruebas unitarias. En proyectos de cierto tamaño es habitual que trabaje en combinación con otras técnicas como *Behaviour Driver Design(BDD)* y *Acceptance Test Driven Development(ATDD)*, que ayudan a validar no solo el código generado sino también los requisitos iniciales del cliente.

2.1.3.1 Aplicación en el proyecto

Para el desarrollo de la aplicación web se ha usado *TDD*. Inicialmente ha requerido una ardua tarea de aprendizaje dada la falta de conocimiento del proyectante no solo de la técnica en sí, sino incluso de las librerías usadas para la codificación de las pruebas. Pero al final, una vez adquirida una mínima soltura se han podido apreciar los beneficios de este modo de trabajar. Principalmente ha sido evidente su utilidad trabajando con metodologías ágiles. A la hora de desarrollar una nueva iteración los posibles fallos o conflictos que afectarán a funcionalidades ya desarrolladas fueron rápidamente resueltos gracias a la buena cobertura de pruebas con las que se contaba.

Además *Django*, el framework de desarrollo usado, incorpora un magnifico soporte para trabajar con *TDD*. Además de incluir *unittest*, la potente librería de *Python* para desarrollo de tests, proporciona de modo automático un entorno de pruebas aislado de la aplicación principal, incluso con una base de datos para pruebas independiente de la base de datos principal.

2.1.4 Responsive Design

Este es un concepto surgido recientemente debido a la proliferación y uso de diferentes tipos de dispositivos móviles para acceder a Internet. *Responsive Design* consiste en diseñar sitios web que se adapten a cualquier tamaño, densidad o calidad de pantalla y ofrezcan una buena experiencia de uso independientemente del dispositivo con el que se acceda al sitio. Las herramientas que se usan para ello son *CSS* y *JavaScript*. Aunque todavía en algunos

desarrollos se opta por realizar varias versiones del sitio adaptadas a distintos tamaños, la cada vez mayor variedad de dispositivos y por tanto de tipos de pantalla está haciendo que este tipo de diseño sea casi obligado en los nuevos proyectos.

2.1.4.1 Aplicación en el proyecto

En este proyecto se ha diseñado la aplicación web siguiendo esta filosofía. Se han definido tres tipos de pantalla a partir de su anchura, para adaptarse a los grupos de dispositivos más comunes actualmente: tabletas, móviles y ordenadores.



Figura 2.1 Responsive design tamaño ordenador



Figura 2.2 Responsive design tamaño tableta



Figura 2.3 Responsive design tamaño móvil

2.1.5 Escalabilidad

La escalabilidad se puede definir como la capacidad de un sistema de adaptarse a cambios en la carga de trabajo sin perder calidad ni prestaciones. Es un concepto que existe desde siempre en el mundo informático, ya que siempre existió la necesidad de adaptar y preparar los sistemas ante cambios en el número de usuarios o el aumento de peticiones de proceso. Sin embargo en los últimos tiempos, los cambios acaecidos en el mundo tecnológico han vuelto a poner esta idea de plena actualidad. En efecto, el rápido aumento de usuarios de Internet en los últimos años ha provocado en numerosas ocasiones el fallo de sistemas que no estaban preparados para soportar incrementos de actividad tan voluminosos. A raíz de estas situaciones cada vez es más popular la frase "morir de éxito", para referirse a estos casos.

Ante problemas de este tipo existen dos enfoques:

- Escalabilidad vertical

Consiste en aumentar las prestaciones de un sistema mejorando el hardware existente (aumentando la memoria, poniendo un procesador mejor, etc). Este sistema no es muy práctico y siempre está limitado por la potencia de los componentes actuales. Su ventaja es que no son necesarios cambios o arquitecturas especiales en las aplicaciones.

- Escalabilidad horizontal

Consiste en aumentar la capacidad añadiendo más equipos o dispositivos. Es la opción más usada, requiere que las aplicaciones estén bien diseñadas pero en ese caso no hay limitaciones en cuanto al tamaño de carga a procesar.

El diseño de la escalabilidad horizontal en aplicaciones web afecta a dos componentes:

- Las bases de datos.
- Los servidores web y/o de aplicaciones

En el caso de la capa de datos como ya se ha comentado en el punto dedicado a [NoSQL](#) los problemas generados por el crecimiento de Internet propiciaron la aparición de las soluciones [NoSQL](#), que vinieron a paliar las carencias en ese sentido del modelo relacional.

En el caso de la capa de aplicación el problema viene por el uso de sesiones. Al ser HTTP un protocolo sin sesión, es decir cada petición es independiente de las anteriores, los programadores necesitan almacenar los datos de sesión en alguna parte, siendo el servidor la más habitual.

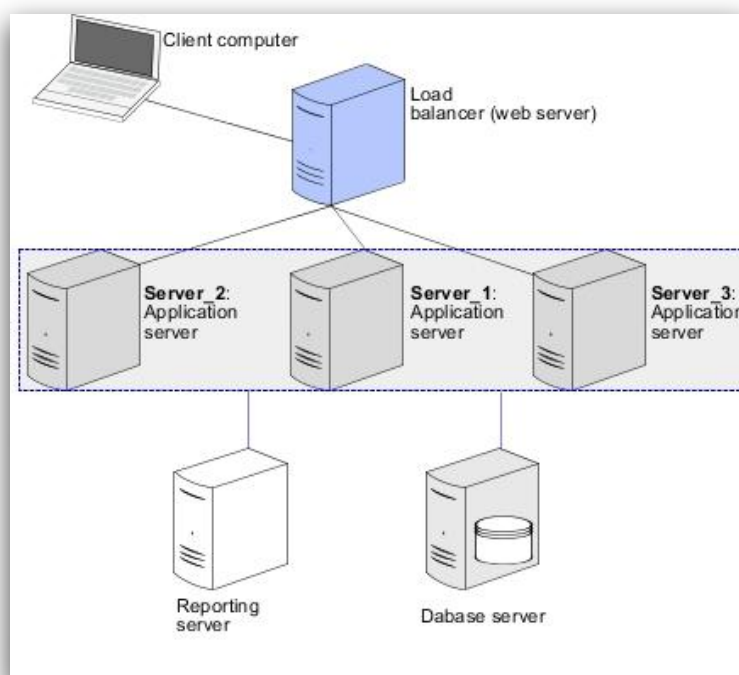


Figura 2.4 Esquema escalabilidad horizontal

Pero en caso de usar el servidor, se plantea el problema de que sucesivas peticiones de un mismo cliente pueden acabar en distintos servidores, con lo que los datos de una sesión pueden no ser accesibles para distintas peticiones del mismo usuario. Una solución habitual es usar las llamadas "Sticky Sessions", que consiste en que el balanceador se ocupa de enviar las peticiones de un mismo cliente al mismo servidor. Esta solución tan sencilla y que parece óptima no lo es tanto, ya que procediendo de este modo el balanceador no hace su función principal que es distribuir las peticiones proporcionalmente a todos los servidores, pudiendo llegarse a situaciones de sobrecarga de determinados servidores.

Una solución aceptable sería guardar la sesión en base de datos, de modo que siempre estuviera accesible independientemente del servidor que procesará la petición. Sin embargo, esta opción nunca ha sido muy popular debido principalmente a las limitaciones en cuanto a rendimiento de las bases de datos tradicionales. La aparición de las soluciones *NoSQL* con tiempos de respuesta muy rápidos están haciendo esta solución cada vez más popular, y eliminando poco a poco el uso de soluciones como los servidores "cache" que es otra de las opciones habituales para resolver este problema.

2.1.5.1 Aplicación en el proyecto

Para garantizar la escalabilidad de la aplicación diseñada en este proyecto se hace uso de las características de *MongoDB*. Para garantizar su rendimiento incluso con grandes cantidades de datos o ante un número muy alto de consultas por segundo, esta base de datos usa sus capacidades de distribución. Esta operación que se conoce como "sharding" consiste dividir los datos en particiones y distribuir estos trozos entre distintos servidores. De este modo cada servidor va a manejar una cantidad de datos razonable con la que puede ofrecer unas prestaciones mínimas, y por otra parte en caso de una frecuencia alta de operaciones sobre los datos, estas se distribuyen sobre varios servidores pudiendo mejorar la eficiencia y los tiempos de respuesta del sistema.

En cuanto a la escalabilidad a nivel de servidor de aplicaciones, en el proyecto se usa *MongoDB* para guardar los datos de sesión, de ese modo se pueden añadir tantos servidores como sean necesarios para atender al tráfico puntual, sin que haya problemas de ningún tipo, ya que todas las operaciones van a ser equivalentes independientemente del servidor que las procese. Las prestaciones de *MongoDB* en cuanto a tiempo de respuesta, garantizan además un rendimiento óptimo de la aplicación en cualquier circunstancia.

De hecho, una de las características más llamativas de *Django*, en comparación con herramientas más tradicionales, es que el modo de gestión de la sesión por defecto es almacenarla en base de datos. Incluso, teniendo en cuenta que *Django* en principio está diseñado para trabajar con bases de datos relacionales. Esto es una muestra de las tendencias que cada vez están ganando más terreno dentro de la industria.

2.2 Tecnologías

2.2.1 Geolocalización

Con este término se define la capacidad de obtener la localización geográfica de un dispositivo, como puede ser un dispositivo móvil o una computadora con conexión a Internet. Esta tecnología se ha popularizado rápidamente debido al auge de los distintos dispositivos móviles que en muchos casos suelen incorporar dispositivos GPS, unido a la potencia y facilidad de uso de la api del servicio de [Google Maps](#). Estos dos factores, combinados con las enorme posibilidades publicitarias de conocer la posición geográfica de los visitantes de un determinado sitio web, ha generado un escenario en que es habitual que la mayoría de nuevas aplicaciones incorporen y exploten capacidades de este tipo. De hecho, dada la creciente importancia de estas funcionalidades, se han desarrollado métodos para obtener una localización aproximada del usuario incluso sin contar con dispositivos GPS, como pueden ser la localización a través de la red móvil o a través de IP en el caso de equipos de sobremesa.

2.2.1.1 Aplicación en el proyecto

La geolocalización es una parte imprescindible en este proyecto, ya que una de sus principales funcionalidades es la posibilidad de almacenar la localización geográfica de un evento, y posteriormente realizar búsquedas dentro de una determinada zona.

Para implementar esta funcionalidad se usa otra de las características de [MongoDB](#), su excelente soporte para trabajar con datos geográficos. De hecho, uno de los mayores éxitos comerciales de esta herramienta y que sin duda contribuyo a su importante posición en el mercado actual, es su uso como base de datos de la aplicación "Foursquare". En efecto, [MongoDB](#) incorpora tipos de datos nativos para trabajar con posiciones geográficas e incluso se pueden definir figuras geométricas sobre la esfera terrestre. Además dispone de una api nativa muy potente con operaciones variadas para realizar búsquedas siguiendo parámetros geográficos.

Además se hace uso del servicio de mapas de google, a través de la librería javascript [gmap3](#), para ofrecer al usuario una mejor experiencia de uso, mostrando en un mapa la posición de los distintos eventos.

2.2.2 Cloud Computing

Con este término genérico se designa al conjunto de tecnologías que permiten la prestación de servicios en Internet. Esto incluye tanto servicios software como infraestructura hardware. Es una filosofía de prestación de servicios que está experimentando un enorme crecimiento en los últimos años. De hecho es la principal culpable del desmesurado aumento del volumen de datos que se ha registrado en Internet en los últimos tiempos. Dentro de esta tecnología se engloban también términos como *Infraestructure as a Service (IaaS)*, *Platform as a Service (PaaS)* y *Software as a Service (SaaS)*, que designan los principales servicios que se pueden ofrecer en Internet.

Como principales ventajas del *Cloud Computing* se pueden citar las siguientes:

- Bajo coste: Se elimina la necesidad de adquirir infraestructuras propias e incluso licencias, y se sustituye por pagos periódicos fijos o por utilización.
- Seguridad: Las empresas proveedoras ofrecen mecanismos de seguridad y replicación cuyo adquisición o mantenimiento sería muy costosa a nivel de usuario e incluso de empresa.
- Información en tiempo real independientemente del punto de acceso.
- Acceso desde cualquier lugar y en cualquier momento, sólo con una conexión a Internet.

En este nuevo escenario, el desarrollo y despliegue de una aplicación web han pasado a ser algo al alcance de cualquier individuo, al no tener que afrontar los gastos derivados de adquisición y mantenimiento de infraestructuras propias. Esto unido al importante ecosistema inversor generado en torno a las llamadas *startups* ha redundado en un importante incremento del número de aplicaciones que alberga Internet.

2.2.2.1 Aplicación en el proyecto

El despliegue de la aplicación desarrollada en un servidor de Internet junto con el diseño de la arquitectura necesaria para soportar y adaptarse a cualquier tamaño de carga de trabajo, datos o usuarios es el 50% de este proyecto.

En efecto, en este proyecto se ha usado la plataforma de *Cloud Computing* de Amazon, llamada *Amazon Web Services(AWS)*. Esta plataforma es probablemente la más madura del mercado y la que ofrece un rango más variado de servicios, siendo base de sitios web bien conocidos como Instagram o Reddit por ejemplo.

2.2.3 Ajax

Ajax es una técnica de desarrollo web que permite comunicarse asíncronamente con el servidor. Viene a solventar una de las limitaciones de HTML que es la necesidad de recargar completamente la página en cada petición. Gracias a esta técnica es posible descargar solo los datos necesarios en cada momento y actualizar solo la parte correspondiente de la página. De este modo se consigue mayor velocidad en la navegación y una experiencia de uso mucho más enriquecedora.

2.2.3.1 Aplicación en el proyecto

Uno de los requisitos de la aplicación es que el tiempo de respuesta sea lo más rápido posible. En un hipotético escenario con millones de datos y un número alto de búsquedas simultáneas es imprescindible controlar en todo momento los tiempos de todas las fases del proceso. Por ello es importante optimizar lo más posible las peticiones que se envíen al servidor, de este modo lo liberamos de generar código html que no es necesario volver a calcular, además reducimos la cantidad de datos que circulan por la red y por último ofrecemos al usuario una experiencia de uso más rica.

En el proyecto se ha usado *jQuery*, que actualmente es librería de referencia para trabajar con *Javascript* y *Ajax*. Se ha optimizado la navegación del sitio web, evitando siempre que fuera posible la recarga total de la página.

2.3 Herramientas

2.3.1 Django

Django es un framework de desarrollo web escrito en *Python*. Es de código abierto y se centra en automatizar todo lo posible. Además se adhiere al principio DRY (Don't Repeat Yourself). Su objetivo es permitir escribir aplicaciones web más rápido y con menos código. Sigue un paradigma conocido como Model-View-Template (MVT), que es similar al Model-View-Controller con unas pequeñas diferencias y cierta confusión en los términos. La vista en MVT no se refiere a la vista tradicional del modelo MVC que designa a la parte de presentación con la que interactúa el usuario. En este caso se refiere más bien al controlador del MVC, ya que engloba los métodos que dan respuesta a las distintas peticiones realizadas en combinación con el fichero *urls.py* que se ocupa de redirigir las peticiones al método correspondiente.

Por otra parte, *Django* usa un sistema de plantillas muy potente que incluye mecanismos de herencia y un lenguaje muy rico para su desarrollo. La parte Template del modelo MVT se puede entonces corresponder con la vista del MVC.

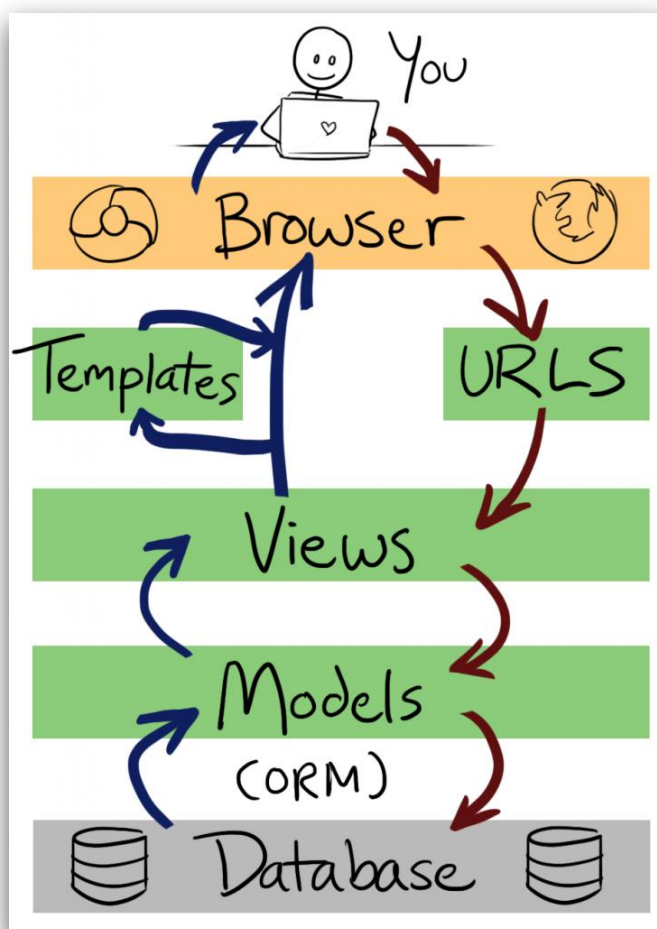


Figura 2.5. Arquitectura MVT de Django

Django incluye además las siguiente características:

- Una completa infraestructura de desarrollo incluyendo servidor web, base de datos y mapeador objeto-relacional.
- Una completa suite de herramientas para realizar pruebas y tests de todo tipo, incluyendo base de datos de prueba para evitar problemas con los datos reales.
- Una librería para desarrollar modelos de datos independientemente de la base de datos utilizada, que en un proyecto de desarrollo *Django* puede ser incluso transparente para el programador.
- Una serie de aplicaciones ya desarrolladas y listas para incorporar a cualquier proyecto y poder ofrecer servicios como gestión de usuarios, portal de administración, gestión de paso de mensajes entre peticiones, gestión de ficheros estáticos, etc.

- Un sistema "middleware" que incorpora características adicionales; por ejemplo, gestión de cache, compresión de la salida, normalización de URLs, protección CSRF y soporte de sesiones.
- Soporte de internacionalización.

2.3.1.1 Aplicación en el proyecto

Django ha sido la herramienta usada para desarrollar la aplicación web. La facilidad de uso y la potencia de características como la gestión de usuarios, el paso de mensajes entre peticiones o la gestión de sesiones han sido de especial utilidad en el proyecto.

El sistema de plantillas con sus capacidades de herencia y combinación resultó de gran utilidad para trabajar con la filosofía *Ajax*, ya que se pudo realizar una división de las partes de cada página de manera natural y organizada, y de esa manera gestionarlas separadamente mediante *Ajax*.

Su sistema de gestión de sesiones permite configurar donde se guardan los datos de sesión, pudiendo elegir entre servidor, cache o base de datos. Esta opción, unida a las prestaciones de *MongoDB* han permitido garantizar la escalabilidad de la aplicación a nivel de servidor.

Como nota negativa se puede mencionar que Django viene preparado para trabajar con bases de datos relaciones, eso obliga a usar la librería *MongoEngine* para poder trabajar con *MongoDB*.

2.3.2 MongoDB

MongoDB es actualmente una de las bases de datos *NoSQL* más usadas. Se dice que es orientada a documentos, ya que al no seguir el modelo relacional no guarda los datos en tablas sino que usa estructuras de datos con un formato llamado *BSON* que es similar a *JSON* pero usando codificación binaria. Esto facilita la integración de los datos en ciertas aplicaciones y herramientas que usan formatos similares, además de aportar una gran flexibilidad al tratamiento de los datos al no estar sujetos a una estructura determinada como en el caso del modelo relacional.

Además se pueden destacar las siguientes características:

- Sistema de replicación. De hecho a nivel empresarial se recomienda trabajar con grupos de tres servidores llamados Replica Sets.
- Sistema de distribución de los datos a través de distintos servidores. Se conoce como sharding y sirve para garantizar la escalabilidad horizontal de la base de datos.
- Soporte para operaciones Map/Reduce.
- GridFS para almacenar archivos de gran tamaño.

- Excelente soporte para geolocalización tanto a nivel de tipos de datos como de soporte para consultas geográficas.
- Las operaciones son siempre atómicas.

2.3.2.1 Aplicación en el proyecto

MongoDB es sin duda alguna la piedra angular de este proyecto. Las características comunes de los sistemas *NoSQL*, junto con las especiales capacidades de *MongoDB* la han hecho imprescindible para conseguir los objetivos propuestos. El uso de *MongoDB* en este proyecto se puede concretar en los siguientes puntos:

- Uso de la flexibilidad del esquema dinámico para representar los datos. En efecto, los eventos de todo tipo(cine, música, etc), con los que trabaja la aplicación constan de varios campos de datos, de los cuales no todos son obligatorios. La posibilidad de trabajar con datos heterogéneos sin los problemas derivados de trabajar con campos nulos habituales en el mundo relacional, es un valor añadido cuando se trabaja con datos de esta naturaleza.
- Uso del tipo de datos Point que ofrece *MongoDB* para almacenar posiciones geográficas. Cada evento contiene además de otros datos su localización geográfica de modo que el usuario pueda realizar búsquedas por posición.
- Uso de las funciones de la API para consultas geolocalizadas combinada con la posibilidad de indexar los datos geográficos para optimizar este tipo de búsquedas. En la aplicación web se permite buscar eventos dentro de una determinada zona geográfica.
- Uso de las prestaciones de *MongoDB* en cuanto a velocidad para almacenar los datos de sesión y de ese modo garantizar la escalabilidad horizontal a nivel de servidor.
- Uso de la opción de "sharding" que permite distribuir los datos en distintos servidores y de ese modo garantiza la escalabilidad a nivel de capa de datos.

2.3.3 Amazon Web Services

Aws es la plataforma de servicios de *Cloud Computing* elegida para desplegar la aplicación desarrollada. En la actualidad es la plataforma de referencia para este tipo de servicios y grandes empresas como Dropbox, Foursquare o Instagram por citar algunos ejemplos que alojan toda su infraestructura en sus servidores. *Aws* ofrece un conjunto de servicios facturados en función de su uso, que abarcan todas las necesidades que pueda tener una empresa para desplegar de modo seguro y eficiente sus aplicaciones en la nube. Entre sus principales servicios se pueden destacar:

- Alquiler de hardware bajo demanda cuyas características pueden ser configuradas a gusto del cliente. Esto incluye no solo maquinas virtuales(Amazon EC2), sino también dispositivos de red, balanceadores, dispositivos de almacenamiento virtuales, etc. En

definitiva, todo lo necesario para que cualquier empresa independientemente de su tamaño y posibilidades económicas puedan disponer de una infraestructura que se adapte a sus necesidades sin falta de acometer grandes inversiones económicas.

- Autoscaling. Con este servicio las aplicaciones web pueden aumentar o disminuir su número de servidores de forma automática, para poder adaptarse a los diferentes picos de uso que son habituales en la mayoría de las aplicaciones.
- Load Balancing. Amazon proporciona balanceadores para que funcionen como front-end de las aplicaciones, y se ocupen de distribuir las peticiones entre los distintos servidores del sitio web.
- Amazon Route 53. Es un servicio de DNS escalable y de alta disponibilidad que se usa para resolver direcciones DNS dentro de la infraestructura de Amazon.
- Amazon S3. Es un servicio de almacenamiento en Internet, altamente escalable, seguro y rápido. Se utiliza para almacenar y recuperar la cantidad de datos que desee, en cualquier momento y desde cualquier parte de la web.
- Amazon EBS proporciona volúmenes de datos diseñados para usarse con las máquinas virtuales creadas en Amazon. Funcionan como disco duros externos que pueden ser conectados a cualquier instancia de Amazon EC2.
- DynamoDB es el servicio de base de datos *NoSQL* de Amazon. Permite almacenar todo tipo de datos así como atender cualquier nivel de tráfico de solicitudes sin perder prestaciones.
- Amazon CloudWatch proporciona un servicio de supervisión de aplicaciones y recursos. Permite establecer alarmas para por ejemplo añadir automáticamente nuevos recursos ante aumentos de carga de trabajo.
- Diferentes regiones geográficas para poder distribuir los recursos optimizando los tiempos de acceso de acuerdo a la localización de los clientes y garantizando la disponibilidad incluso en circunstancias extremas.

2.3.3.1 Aplicación en el proyecto

AWS es la otra piedra angular de este proyecto junto con *MongoDB*. Es la plataforma elegida para el despliegue de la aplicación y gracias a sus numerosos servicios se consigue un despliegue que garantiza la escalabilidad de la aplicación además de ofrecer unas prestaciones muy competitivas. En concreto se usan los siguientes servicios de *AWS*:

- Instancias EC2 para alojar los distintos componentes de la aplicación.
- Elastic Load Balancer para funcionar como Front-End de la aplicación balanceando la carga de trabajo. También se ocupa en combinación con AutoScaling y CloudWatch de garantizar que la capa de aplicación siempre va a contar con un número adecuado de servidores en funcionamiento.
- AutoScaling para añadir o quitar servidores automáticamente dependiendo de la carga de trabajo del sistema.
- CloudWatch para monitorizar la carga del sistema.
- S3 para guardar y servir los ficheros estáticos.

2.3.4 MongoEngine

MongoEngine es una librería que realiza funciones de mapeador objeto-relacional pero adaptado a *MongoDB*, se puede decir entonces que es un mapeador objeto-documento ya que en el mundillo *NoSQL* se habla de documentos para referirse a la estructura de los datos. Usa *PyMongo* que es la librería de referencia para trabajar con *MongoDB* usando *Python*.

2.3.4.1 Aplicación en el proyecto

Django no está diseñado para trabajar con bases de datos no relacionales, por tanto para este proyecto es necesario usar una herramienta de este tipo que se integre de manera adecuada. Aunque no es plenamente compatible con todas las características de *Django*, si que aporta una funcionalidad suficiente para realizar cualquier proyecto profesional. En concreto *MongoEngine* soporta las siguientes funcionalidades de *Django*:

- Es compatible con el sistema de gestión de usuarios de Django.
- Es compatible con el sistema de gestión de sesiones.

Además permite trabajar con las siguientes características de *MongoDB*:

- Es compatible con GridFS
- Es compatible con los tipos de datos geográficos de *MongoDB*, así como con la API de geolocalización.

2.3.5 Boto

Es una librería realizada en *Python* que sirve como interface entre las aplicaciones y los servicios de *AWS*. Incluye soporte para la mayoría de los servicios ofrecidos por esta plataforma. Y por supuesto es totalmente compatible con el framework *Django*.

2.3.5.1 Aplicación en el proyecto

En este proyecto donde se realiza el despliegue de una aplicación *Django* en *AWS* es imprescindible el uso de esta herramienta. Concretamente es necesaria para gestionar el almacenamiento de ficheros estáticos en el servicio S3 de *AWS*. En este almacén de datos se almacenan las imágenes asociadas a cada evento y es necesario que la aplicación gestione la inserciones y borrados de estos archivos.

2.3.6 Git

En cualquier proyecto de software actual es casi imprescindible el uso de un sistema de gestión de versiones. A día de hoy, *Git* es probablemente la herramienta de ese tipo más usada y cada vez disfruta de más popularidad, apoyada por proyectos como la web GitHub que es uno de los repositorios de software más usados del mundo. GitHub usa las capacidades de *Git* para ofrecer sus servicios como almacén de software.

2.3.6.1 Aplicación en el proyecto

En cualquier proyecto sería adecuado el uso de esta herramienta, pero en este lo es más aún si cabe. En efecto, dado que se va a seguir una metodología basada en iteraciones, *Git* es una herramienta imprescindible para poder almacenar y gestionar las distintas versiones de software que se generarán durante el proyecto.

2.3.7 VirtualEnv

Es otra herramienta de referencia en el mundillo *Python*. Es un gestor de entornos virtuales que sirve para crear espacios de desarrollo completamente independientes. De esta manera se eliminan los problemas de versiones a la hora de trabajar a la vez en distintos proyectos. Cada entorno virtual está aislado de los demás, por tanto dentro de la misma máquina se puede trabajar en distintos proyectos con distintas versiones de la misma librería.

2.3.7.1 Aplicación en el proyecto

No tiene una aplicación específica en este proyecto, más allá de realizar su función principal. Esto es crear un entorno aislado donde realizar el desarrollo sin problemas de compatibilidad con otras versiones de librerías que puedan estar instaladas en el sistema.

2.3.8 Gmap3

Es un plugin de *JQuery* para trabajar con *Google Maps*. La librería para trabajar con mapas que ofrece Google está desarrollada en javascript. Actualmente una parte muy importante de los desarrolladores prefieren *JQuery* sobre javascript, dado su sencillez de uso y su potencia. Por

ello esta librería es de mucha utilidad para poder explotar toda la potencia de los mapas de Google sin renunciar a la facilidad de uso de *jQuery*.

2.3.8.1 Aplicación en el proyecto

En este proyecto se hace un uso bastante importante de la tecnología de *Google Maps*. Los eventos van a tener almacenada su localización geográfica y el uso de mapas para mostrar dichas localizaciones o para definir áreas de búsqueda va a ser habitual. Por ello el uso de esta librería es un valor añadido a la riqueza de este proyecto, ya que aporta un interfaz fácil y amigable para extraer todas las posibilidades de la API de *Google Maps*.

Capítulo 3. Planificación del Proyecto

3.1 Planificación inicial

En un proyecto de estas características donde se va abordar el aprendizaje y uso de nuevas herramientas, tecnologías y metodologías las dificultades para realizar una planificación inicial aceptablemente exitosa son muy grandes. En efecto, es difícil predecir la curva de aprendizaje necesaria para adquirir un dominio mínimo de esas nuevas capacidades, y tampoco es fácil estimar a continuación la velocidad en el desarrollo que se puede alcanzar. Además en este proyecto que está planteado para usar una metodología iterativa, es difícil estimar a priori el número de iteraciones necesarias para alcanzar los objetivos marcados, más teniendo en cuenta la falta de experiencia del projectante. De todos modos se ha planteado la siguiente planificación inicial, más como un punto de partida que como un documento formal que deba ser seguido de modo estricto. En principio se plantean tres iteraciones en el proyecto, siempre teniendo en cuenta que dicho número puede cambiar a lo largo del proyecto, simulando un escenario real donde un hipotético cliente puede variar sus requerimientos a lo largo del desarrollo. Otro aspecto a tener en cuenta es que estima mismo el tiempo de desarrollo para cada una de las iteraciones, condición que en la práctica no se va a cumplir, ya que cada uno tendrá una duración determinada por la complejidad de las historias de usuario que se incluyan. Por otra parte, con esta planificación se estima una fecha final que permitiría la defensa del proyecto en la convocatoria de Junio.

Inicio del proyecto	0 hrs	Mon 24/02/14	Mon 24/02/14
Estudio de nuevas tecnologías	120 hrs	Mon 24/02/14	Tue 25/03/14
⊕ Preparación de la infraestructura de desarrollo	0,38 days	Sat 01/03/14	<u>Sat 01/03/14</u>
⊖ Desarrollo de la aplicación web	17,63 days	Tue 25/03/14	Tue 29/04/14
⊕ Análisis preliminar	2,25 days	Tue 25/03/14	Sat 29/03/14
⊕ Diseño preliminar	0,63 days	Sat 29/03/14	Mon 31/03/14
Planificación de las iteraciones	3 hrs	Tue 25/03/14	Wed 26/03/14
⊕ Desarrollo de las iteraciones	17,63 days	Tue 25/03/14	Tue 29/04/14
⊖ Despliegue de la aplicación	4,75 days	Tue 29/04/14	Thu 08/05/14
Estudio de las plataformas Cloud Computing existentes	5 hrs	Tue 29/04/14	Wed 30/04/14
Diseño de la arquitectura	15 hrs	Wed 30/04/14	Sat 03/05/14
Preparación de la infraestructura	10 hrs	Sun 04/05/14	Tue 06/05/14
Despliegue	5 hrs	Tue 06/05/14	Thu 08/05/14
Pruebas de rendimiento	3 hrs	Thu 08/05/14	Thu 08/05/14
Documentación final	20 hrs	Fri 09/05/14	Tue 13/05/14
Elaboración de la memoria	30 hrs	Tue 13/05/14	Tue 20/05/14

Figura 3.1 Planificación inicial

3.2 Planificación intermedia

Esta planificación se aborda al final la primera iteración. En este momento ya se ha elaborado un producto software finalizado y listo para entregar, se ha trabajado con las nuevas herramientas y ya se puede realizar una estimación más acertada de los ritmos de trabajo.

Los resultados son que la curva de aprendizaje está resultando más empinada de lo que se estimo en un principio, especialmente en cuanto a la metodología *TDD*. En esta metodología la calidad de los tests elaborados es clave para la elaboración de un software de calidad. Sin embargo, esta tarea no es trivial y ha requerido muchas horas de estudio y análisis. Ha sido necesario por una parte aprender a usar las clases y métodos de la potente librería "unittest",

eso unido a una labor de investigación intensa para adquirir buenas prácticas a la hora de plantear los tests. Y todo ello junto la dificultad de trabajar con *Python* y *Django* que también son herramientas nuevas para el proyectante.

En resumen, de momento la metodología *TDD* está siendo un quebradero de cabeza y más que validar el código realizado con los tests, se están validando los tests con el código. De hecho, el tiempo dedicado a esta tarea dobla el estimado inicialmente. En la práctica el tiempo dedicado a la elaboración dobla al empleado en desarrollar el código. Esto ha generado un retraso estimado del proyecto que hace difícil su presentación para la convocatoria de junio, eso contando que la planificación estimada para la parte "Despliegue de la aplicación web" sea correcta.

Inicio del proyecto	0 hrs	Mon 24/02/14	Mon 24/02/14
Estudio de nuevas tecnologías	120 hrs	Mon 24/02/14	Tue 25/03/14
+ Preparación de la infraestructura de desarrollo	0,38 days	Sat 01/03/14	Sat 01/03/14
- Desarrollo de la aplicación web	28,13 days	Tue 25/03/14	Mon 19/05/14
+ Análisis preliminar	2,25 days	Tue 25/03/14	Sat 29/03/14
+ Diseño preliminar	0,63 days	Sat 29/03/14	Mon 31/03/14
Planificación de las iteraciones	3 hrs	Tue 25/03/14	Wed 26/03/14
+ Desarrollo de las iteraciones	28,13 days	Tue 25/03/14	Mon 19/05/14
- Despliegue de la aplicación	4,75 days	Mon 19/05/14	Thu 29/05/14
Estudio de las plataformas Cloud Computing existentes	5 hrs	Mon 19/05/14	Wed 21/05/14
Diseño de la arquitectura	15 hrs	Wed 21/05/14	Sat 24/05/14
Preparación de la infraestructura	10 hrs	Sat 24/05/14	Tue 27/05/14
Despliegue	5 hrs	Tue 27/05/14	Wed 28/05/14
Pruebas de rendimiento	3 hrs	Wed 28/05/14	Thu 29/05/14
Documentación final	20 hrs	Thu 29/05/14	Tue 03/06/14
Elaboración de la memoria	30 hrs	Tue 03/06/14	Tue 10/06/14

Figura 3.2 Planificación intermedia

3.3 Planificación final

Una vez finalizado el proyecto se incluye esta planificación indicando los plazos reales que finalmente se cumplieron. Una vez realizada la primera iteración con resultados no muy esperanzadores en cuanto a la cantidad de empleado y al éxito de la metodología, se comenzó el desarrollo de la segunda. En esta la experiencia de la primera iteración resultó de utilidad y la elaboración de los tests fue un poco más fluida, redundando en beneficio de todo el proceso. Poco a poco se empezaron a ver las ventajas de este modo de trabajo. Se mejoró un poco el tiempo estimado que fue inferior al empleado en la primera iteración.

En cuanto a la tercera y última aunque el desarrollo mediante *TDD* ya estaba más o menos asimilado y en ese aspecto no hubo problemas, la complejidad de las historias de usuario elegidas hizo que el tiempo de desarrollo fuera todavía bastante elevado.

En este punto dado las premuras de tiempo se decidió dar por concluida la parte del desarrollo.

La parte de despliegue de la aplicación también superó las estimaciones iniciales, cosa normal dado la inexperiencia del proyectante en esas tareas. Surgieron numerosos problemas no previstos y aspectos que no se habían tenido en cuenta en el análisis previo, lo que provocó un aumento del tiempo real dedicado.

Al final, se ha podido terminar el proyecto para su presentación en la convocatoria de Julio.

Inicio del proyecto	0 hrs	Mon 24/02/14	Mon 24/02/14
Estudio de nuevas tecnologías	120 hrs	Mon 24/02/14	Tue 25/03/14
+ Preparación de la infraestructura de desarrollo	0,38 days	Sat 01/03/14	<u>Sat 01/03/14</u>
- Desarrollo de la aplicación web	25,75 days	Tue 25/03/14	Thu 15/05/14
+ Análisis preliminar	2,25 days	Tue 25/03/14	Sat 29/03/14
+ Diseño preliminar	0,63 days	Sat 29/03/14	Mon 31/03/14
Planificación de las iteraciones	3 hrs	Tue 25/03/14	Wed 26/03/14
- Desarrollo de las iteraciones	25,75 days	Tue 25/03/14	Thu 15/05/14
+ Primera iteración	9,38 days	Tue 25/03/14	Sat 12/04/14
+ Segunda iteración	7,63 days	Sat 12/04/14	Sun 27/04/14
+ Tercera iteración	8,75 days	Sun 27/04/14	Thu 15/05/14
- Despliegue de la aplicación	9,75 days	Thu 15/05/14	Tue 03/06/14
Estudio de las plataformas Cloud Computing existentes	5 hrs	Thu 15/05/14	Fri 16/05/14
Diseño de la arquitectura	15 hrs	Sat 17/05/14	Tue 20/05/14
Preparación de la infraestructura	25 hrs	Tue 20/05/14	Mon 26/05/14
Despliegue	30 hrs	Mon 26/05/14	Mon 02/06/14
Pruebas de rendimiento	3 hrs	Mon 02/06/14	Tue 03/06/14
Documentación final	20 hrs	Tue 03/06/14	Sun 08/06/14
Elaboración de la memoria	30 hrs	Sun 08/06/14	Sun 15/06/14

Figura 3.3 Planificación final

3.4 Conclusiones finales

En un proyecto de estas características donde se incluía el uso de herramientas nuevas para el desarrollador las estimaciones iniciales nunca van a ser correctas ya que es imposible predecir el tiempo de aprendizaje que en todo caso es diferente para cada persona. En todo momento ha sido necesaria una labor de adecuación de las estimaciones a los ritmos de trabajo reales. Estas divergencias redundaron en el alcance final del proyecto, en el cual no se incluyeron todas las opciones que el proyectante hubiera deseado.

Capítulo 4. Desarrollo de la aplicación web

4.1 Descripción de la metodología empleada

Para el desarrollo de la aplicación se ha optado por seguir una filosofía ágil. Dado que la aplicación web no es el fin último del proyecto, sino más bien un medio para alcanzar los objetivos propuestos, y que las estimaciones de tiempo no van a ser muy precisas, en un proyecto donde se va a acometer el aprendizaje de un número importante de nuevas tecnologías y herramientas, se ha optado por seguir una metodología iterativa. De este modo, al igual que en un proyecto real, ya en fases tempranas del desarrollo se va contar con un producto terminado y con una funcionalidad mínima, que en caso de problemas de tiempo siempre permitirá tener una aplicación sobre la que trabajar la segunda parte del proyecto que es el diseño y despliegue de una arquitectura Cloud Computing.

Dado que se trata de un proyecto unipersonal tampoco se ha seguido fielmente una metodología específica, ya que por ejemplo algunas de las características más importantes de metodologías ya un poco maduras en este ámbito como *Scrum* o *XP* son precisamente las relativas al modo de organizar y trabajar con los equipos de desarrollo. Los únicos elementos que se han tenido en cuenta en este proyecto son los relativos a la planificación y desarrollo de las sucesivas iteraciones, y para ello se ha optado por seguir las directrices y la nomenclatura de *Scrum*, usando términos como sprint, *Product Backlog* o *Sprint Backlog* por citar algunos ejemplos.

Además se ha usado la filosofía *Test Driven Development* durante el desarrollo, que aunque en principio es independiente de las metodologías ágiles, su uso conjunto cada vez es más habitual en el mundo del desarrollo software. El uso de TDD+Ágil genera cierta controversia en cuanto al tema de la documentación. Uno de los principios del "Manifiesto Ágil" habla de primar siempre el código que funciona sobre la documentación exhaustiva, y el uso de *TDD* hace que muchos desarrolladores aboguen por minimizar las fases de análisis y diseño previas, llegando incluso a casos extremos como considerar a los tests como la única documentación necesaria para el proyecto. En el otro extremo están los que siguen los procesos tradicionales de realizar análisis y diseños previos exhaustivos con su correspondiente documentación, en cuyo caso no están usando "TDD+Ágil" y por tanto están perdiendo los beneficios del uso de estas nuevas tecnologías. Dado el carácter de auto aprendizaje de este proyecto se ha optado por experimentar e ir desarrollando una metodología propia y adecuada para este proyecto, a base de prueba y error. De cada iteración se han extraído conclusiones y aprendizajes que se han podido usar en las siguientes.

Finalmente, la metodología usada se ha concretado en las siguientes fases y tareas:

- Análisis previo.

- Elaboración del *Product Backlog*, que incluirá los requisitos funcionales de la aplicación en lenguaje no informático. Este es un documento que puede sufrir cambios durante las distintas fases de desarrollo. En efecto, una de las ventajas de las metodologías ágiles es que en cualquier momento los requisitos iniciales pueden sufrir cambios, y el desarrollo del proyecto se puede adaptar con cierta facilidad a estas modificaciones.
- Concreción de los requisitos mediante algún tipo de documento, puede ser textual o en el caso de este proyecto se usarán tablas de requisitos. En todo caso la descripción tampoco ha de ser excesivamente detallada, ya que una de las características del modo de trabajo de *Scrum* es la existencia de una comunicación fluida entre cliente y equipo de desarrollo, de este modo los requisitos detallados se pueden ir concretando durante este intercambio de información. Y en todo caso, se cuenta con un pruebas y una evaluación final por parte del cliente para validar el producto desarrollado. En este proyecto existe la ventaja de que cliente y desarrollador son la misma persona, con lo que la comunicación va a ser muy fluida.
- Diseño previo.

Aunque esta fase a priori podría no parecer necesaria, ya que cada sprint va a contar con su fase de diseño, la experiencia adquirida en la realización de este proyecto aconseja su inclusión. Aunque en principio cada sprint es independiente de los otros y los requisitos pueden sufrir cambios durante el ciclo de vida del proyecto, el partir de una visión general del diseño del futuro sistema siempre es positivo a la hora de acometer el desarrollo de las distintas partes. Por tanto, para esta fase se han definido las siguientes tareas:

 - Diseño previo de las clases. Esto incluye definir las clases necesarias dentro de la arquitectura *Django*. Es decir las *Views* y las rutas dentro del fichero *urls.py*.
 - Diseño de la interfaz de usuario. Esta tarea por si sola ya justifica la existencia de esta fase. En efecto, si no partimos de un diseño inicial y cada iteración diseñara su interfaz de usuario sin pensar en el futuro, los cambios a realizar en cada iteración serían muy profundos y complejos. Teniendo en cuenta además la existencia de aspectos como *Responsive Design* o *Ajax*, que tienen una gran influencia en este ámbito. Esta tarea ha de incluir el diseño de los *templates* en la arquitectura *Django*.
 - Diseño de la base de datos. Siguiendo el modo de trabajar de Django, la iteración entre desarrollador y base de datos no es necesaria en esta fase del desarrollo. La definición de los datos se hace a través de las clases *Model* de *Django*. Sin embargo, *Django* no está diseñado en principio para trabajar con bases de datos *NoSQL*, por ello en este caso es recomendable hacer un diseño teniendo en cuenta la base de datos a utilizar. En todo caso, más que

diseño aquí lo que se busca es una identificación preliminar de las entidades que existirán en el proyecto. Siempre sujetas a modificaciones posteriores.

- Desarrollo de las iteraciones. Para cada iteración se pueden definir las siguientes fases:
 - Elaboración del *Sprint Backlog*, es decir las funcionalidades del *Product Backlog* que se van a desarrollar en esta iteración.
 - Consideraciones previas. Esta fase comprende una descripción textual de las principales a tener en cuenta a la hora de realizar esta iteración, incluyendo posibles problemas que se pueden prever, necesidad de algún tipo de herramienta, y en general una especie de simulacro de reunión del equipo de desarrollo antes de iniciar el desarrollo de un sprint.
 - Análisis de los requisitos. Descripción un poco más detallada de los requisitos para esta iteración. A partir de esta documentación se generarán los tests que guiarán el desarrollo.
 - Diseño. En esta fase se incluye un diseño un poco más detallado que concrete las clases y las entidades de datos a desarrollar, así como la estructura de la interfaz de usuario.
 - Elaboración de los tests a partir del análisis y diseño previo.
 - Desarrollo del código guiado por los tests.
 - Pruebas funcionales y presentación al cliente para validar el producto realizado, esto conllevará modificaciones posteriores en caso de fallar alguna validación.
 - Refactorización del código generado, para su optimización.
- Documentación final. Aquí se incluye una descripción detallada de la aplicación ya finalizada, incluyendo las clases desarrolladas, el esquema de la base de datos y la interfaz de usuario.

4.2 Descripción del plan de pruebas

4.2.1 Pruebas unitarias y funcionales

Para la elaboración del plan de pruebas de esta aplicación hay que tener en cuenta que el uso de la metodología *TDD* ya proporciona un razonable seguridad en cuanto a la calidad del

código generado. De hecho una de las ventajas del uso de *TDD*, es que el desarrollo al estar en todo momento tutelado por los tests previos elaborados, normalmente genera un código libre de fallos y que cumple las especificaciones iniciales (siempre dependiendo de la calidad de los tests). La parte correspondiente a las pruebas unitarias está por tanto cubierta por este tipo de tests.

Dentro de estas metodologías, normalmente *TDD* trabaja en conjunto con otros métodos como *Behavior Driven Design (BDD)* o *Acceptance Test Driven Development (ATDD)*. Estos métodos entran dentro de las llamadas pruebas funcionales, que validan que el sistema cumple las funcionalidades requeridas. Estas se refieren a requerimientos como: "El usuario puede hacer login en el sistema" o "el usuario registrado puede añadir un comentario". Normalmente para este tipo de pruebas se usan herramientas que permiten abrir un navegador y realizar automáticamente las operaciones requeridas para validar que cumple una determinada funcionalidad. Para este proyecto se han realizado pruebas con *Selenium*, que es una de las herramientas de referencia para estas tareas dentro del mundo Django. Sin embargo, este tipo de utilidades presentan problemas al trabajar con peticiones asíncronas como es el caso de *Ajax*, y en durante el desarrollo de la aplicación, los inconvenientes han sido numerosos. Debido a ello, dado que los requisitos funcionales no son muy complejos ni numerosos se ha optado por abandonar el uso de *Selenium* y hacer este tipo de pruebas a mano. Es decir, interactuando con el navegador y comprobando que se cumple la funcionalidad requerida, de modo similar a como se desarrollan las pruebas de usabilidad. Además, se ha comprobado que los tests unitarios desarrollados siguiendo *TDD* aportan una importante capacidad de validación incluso de requisitos funcionales, relegando en muchos casos a los tests funcionales a una comprobación de si los enlaces son visibles en la página.

4.2.2 Pruebas de integración y rendimiento

En cuanto a las pruebas de integración, el escaso tamaño y la poca complejidad de la aplicación no hacen necesarios este tipo de pruebas. Las pruebas de rendimiento sí que son básicas en este tipo de proyectos, ya que uno de sus principales objetivos es que se pueda adaptar a distintos tamaños de carga de trabajo manteniendo un rendimiento aceptable. Hay que tener en cuenta las limitaciones de hardware con las que se cuenta en este proyecto, ya que el despliegue de una infraestructura de pruebas adecuada está fuera del alcance del proyectante. A pesar de ello se ha diseñado una batería de pruebas que analizan dentro de las limitaciones la adaptación del sistema ante distintos tamaños de datos con distintas configuraciones hardware.

4.2.3 Pruebas de usabilidad y accesibilidad

En este proyecto no se han tenido en cuenta estos conceptos. El objetivo del desarrollo de la aplicación es contar con un objeto de pruebas y estudio para estudiar nuevas técnicas y métodos de desarrollo. Además de servir como base para analizar los problemas y soluciones a

la hora de afrontar el crecimiento del tamaño de la aplicación. Dentro de este escenario los conceptos de usabilidad y accesibilidad no son relevantes, ya que no influyen de manera decisiva en las metodologías ni en las herramientas de desarrollo y no tienen ninguna influencia en el problema de la escalabilidad del sistema. Por ello no se han incluido pruebas en estos aspectos.

4.3 Análisis previo

4.3.1 Product Backlog

Para este proyecto se parte se ha elaborado el siguiente conjunto de requisitos como punto de partida. Puede sufrir modificaciones a lo largo del proyecto, debido a que en un proyecto como este donde se va a abordar el aprendizaje de nuevas tecnologías y herramientas, los plazos de tiempo nunca van a ser precisos.

El formato elegido incluye una columna con las historias de usuario que es el nombre típico de los requisitos en el ámbito ágil, y otra columna para medir la importancia de cada tarea y de ese modo empezar por las más prioritarias.

PRODUCT BACKLOG	
Historia de usuario	Prioridad
Como usuario no registrado quiero crear una cuenta	10
Como usuario registrado quiero poder loguearme	10
Como usuario registrado quiero poder cerrar sesion	10
Como usuario registrado quiero poder añadir un evento	10
Como usuario quiero poder ver una lista de eventos para poder seleccionar uno	9
Como usuario quiero poder ver el detalle de un evento seleccionado	9
Como usuario quiero poder ver la localizacion en un mapa de un evento seleccionado	9
Como usuario quiero poder buscar eventos por localización y titulo y/o categoria	8
Como usuario registrado quiero editar eventos creados por mi	8
Como usuario registrado quiero poder añadir fotos en los eventos creados por mi	7
Como usuario registrado quiero poder añadir comentarios en los eventos	7
Como usuario quiero poder ver la lista de eventos divididos en paginas	6
Como usuario registrado quiero borrar eventos creados por mi	8
Como cliente quiero que la aplicación responda rapido a las peticiones	N/A
Como cliente quiero que la aplicación se vea bien en todos los tamaños de pantalla	N/A
Como cliente quiero poder ver la pagina en ingles o español	N/A
Como cliente quiero que la aplicación se adapte al aumento de usuarios	N/A

Figura 4.1 Product Backlog

4.3.2 Tabla de requisitos

En esta tabla se concretan un poco más los requisitos de las historias de usuario incluidas en el [Product Backlog](#).

Nombre Requisito	Descripción del Requisito
Como usuario no registrado quiero crear una cuenta	El usuario crea una cuenta pasando un nombre de usuario y una password. El nombre de usuario no se puede repetir.
Como usuario registrado quiero poder loguearme	El usuario se loguea en el sistema introduciendo su nombre de usuario y su contraseña. Ha de estar previamente registrado.
Como usuario registrado quiero poder cerrar sesión	El usuario cierra sesión y deja de estar logueado. Ha de estar logueado previamente.
Como usuario registrado quiero poder añadir un evento	El usuario añade un evento en el sistema. El evento tendrá los siguientes campos: título, descripción, categoría y localización geográfica. El usuario ha de estar logueado.
Como usuario quiero poder ver una lista de eventos para poder seleccionar uno	El usuario puede ver en la página inicial una lista de los eventos que existen en el sistema. Esa lista incluirá el título y la categoría.
Como usuario quiero poder ver el detalle de un evento seleccionado	El usuario podrá ver todos los datos de un evento seleccionándolo de la lista.
Como usuario quiero poder ver la localización en un mapa de un evento seleccionado	El usuario podrá ver en un mapa la localización de un evento seleccionándolo en la lista.
Como usuario quiero poder buscar eventos por localización y título y/o categoría	El usuario podrá buscar eventos especificando un área geográfica y opcionalmente su título o categoría.
Como usuario registrado quiero editar eventos creados por mi	El usuario podrá editar la información solo de los eventos creados por él mismo. Requiere estar logueado.
Como usuario registrado quiero poder añadir fotos en los eventos creados por mi	Dentro de los datos de cada evento se podrá incluir una foto. Requiere estar logueado.
Como usuario registrado quiero poder añadir comentarios en los eventos	Dentro de la información de cada evento se puede incluir un comentario. Requiere estar logueado.
Como usuario quiero poder ver la lista de eventos divididos en paginas	Si la lista de eventos es muy grande se podrán visualizar en un sistema de páginas.
Como usuario registrado quiero borrar eventos creados por mi	El usuario podrá eliminar un evento creado por él mismo. Se requiere estar logueado.
Como cliente quiero que la aplicación responda rápido a las peticiones	La navegación debe ser rápida.
Como cliente quiero que la aplicación se vea bien en	Que se vea bien en móviles y tabletas, además de en ordenadores.

todos los tamaños de pantalla	
Como cliente quiero poder ver la pagina en inglés o español	Que la página actualice sus textos en función del idioma configurado en el navegador.
Como cliente quiero que la aplicación se adapte al aumento de usuarios	Que la aplicación no se caiga ante un aumento desmesurado de tráfico.

4.4 Diseño previo

4.4.1 Diseño de la jerarquía de desarrollo Django

Django incluye dentro de su jerarquía de desarrollo dos conceptos: proyecto y aplicación. Por aplicación se entiende una aplicación web que realiza alguna función, por ejemplo gestionar las cuentas de usuario, las nominas de una empresa, una red social, un sistema de mensajería, etc. Un proyecto se define como un conjunto de aplicaciones que funcionan en conjunto. En Django las aplicaciones se diseñan para poder ser reutilizadas en diversos proyectos. Por ejemplo se puede realizar un aplicación que ofrezca un servicio de chat entre los usuarios de un sitio web, dicha aplicación podrá ser instalada y usada por muchos sitios distintos.

Siguiendo esta filosofía, para este proyecto se va a desarrollar una aplicación llamada "EventsList" que incorporará toda la funcionalidad, y se incluirá dentro de un proyecto llamado "WorldEvents". De este modo la aplicación se podrá integrar en un futuro en cualquier proyecto.

4.4.2 Diseño previo de clases

Aquí se incluye un gráfico indicando las rutas y vistas correspondientes que sería necesario crear dentro de la arquitectura Django.

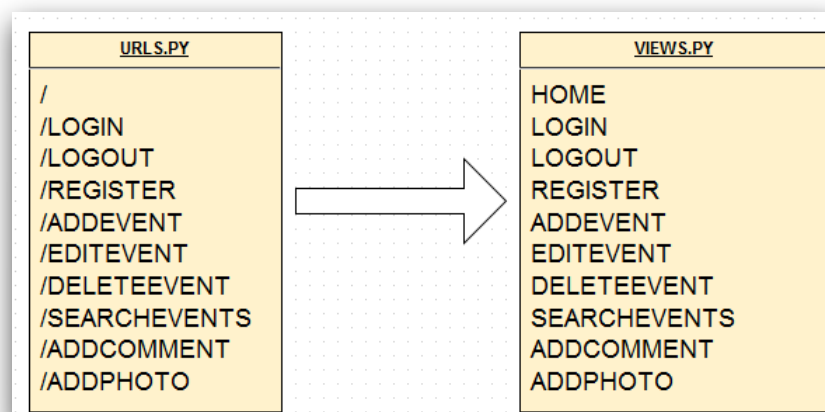


Figura 4.2 Diseño previo clases

4.4.3 Diseño previo de la base de datos

Para este diseño se han tenido ya en cuenta las especiales características de *NoSQL*. Al no existir relaciones los comentarios pertenecientes a un determinado evento se han incluido como una lista dentro del evento. De ese modo con una sola consulta se devolverá el evento y sus comentarios. En cuanto a las relaciones con las entidades *Category* y *User*, el método usado es incluir en las entidades relacionadas los campos *USERNAME* de *user* y *NAME* de *Category* que son claves candidatas dentro de sus entidades. La validación de las integridades referenciales en este caso queda como tarea para la capa de aplicación.

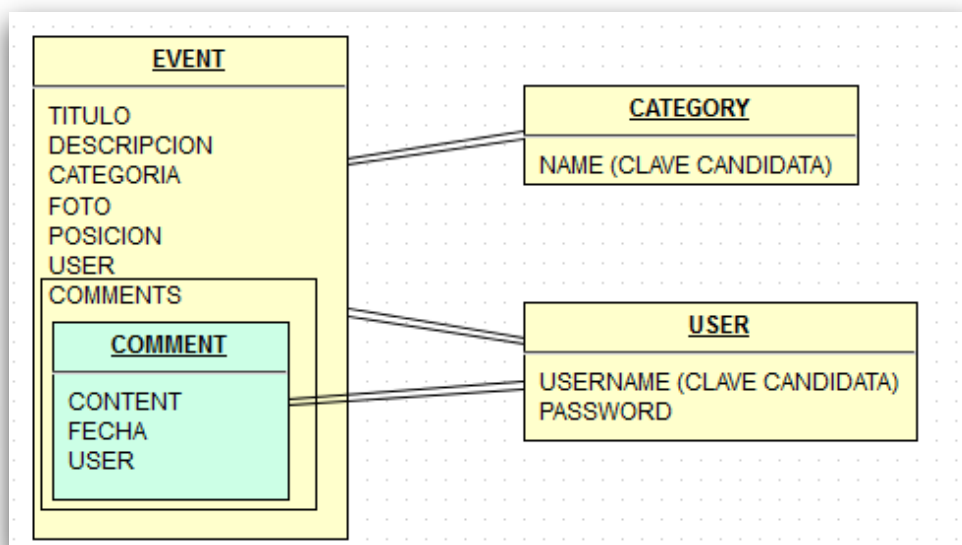


Figura 4.3 Diseño previo BD

4.4.4 Diseño previo de la interfaz de usuario

Los requerimientos listados en el *Product Backlog* sugieren que la aplicación va a mostrar tres tipos de información: una lista de eventos, un detalle de cada evento seleccionado y un mapa mostrando la posición del evento seleccionado. Además hay que tener en cuenta que el cliente quiere un diseño responsivo. Teniendo en cuenta estos factores se ha diseñado la siguiente interfaz, adecuándola para tres tamaños de pantalla típicos, menor que 768 píxeles(móviles), entre 768 y 1024 píxeles(tablets) y mayor de 1024 píxeles(ordenaador).

- Diseño para escritorio:

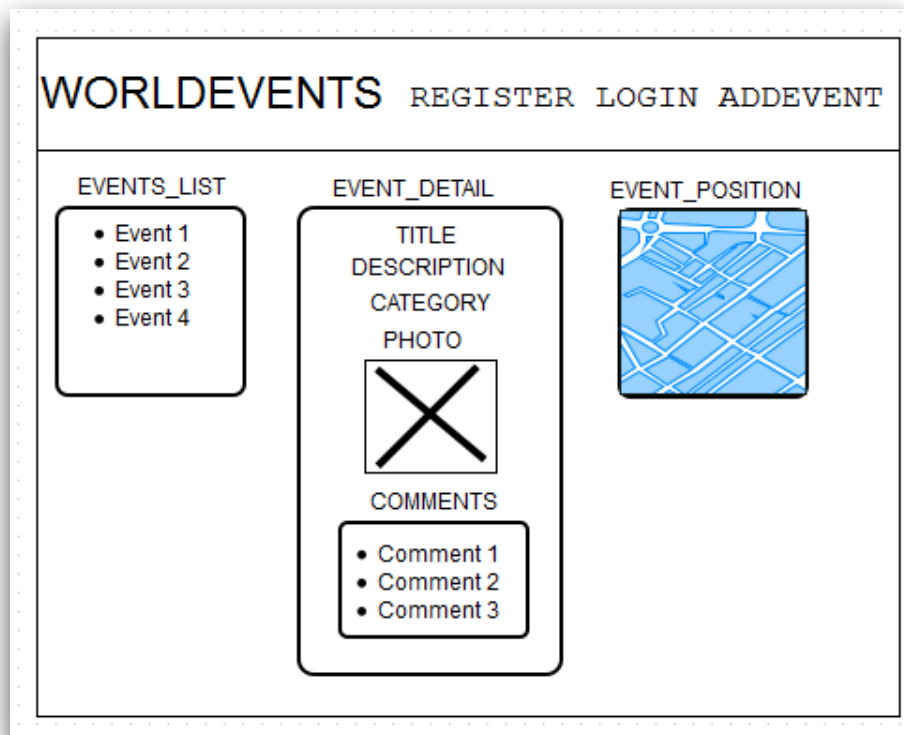


Figura 4.4 Diseño previo interfaz ordenador

- Diseño para tablet. Aquí se elimina una columna y se incluye un enlace para alternar la vista del detalle del evento y el mapa.

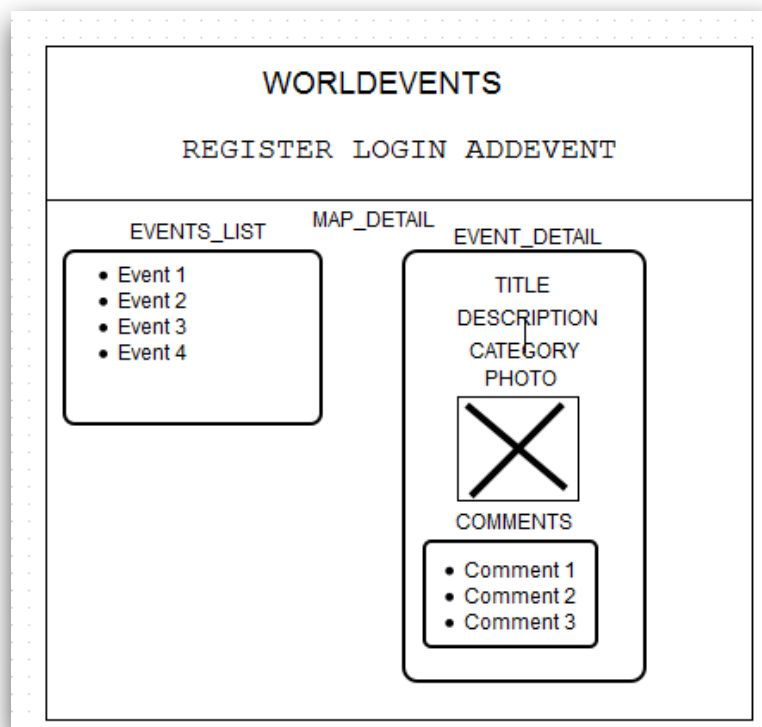


Figura 4.5 Diseño previo interfaz tablet 1

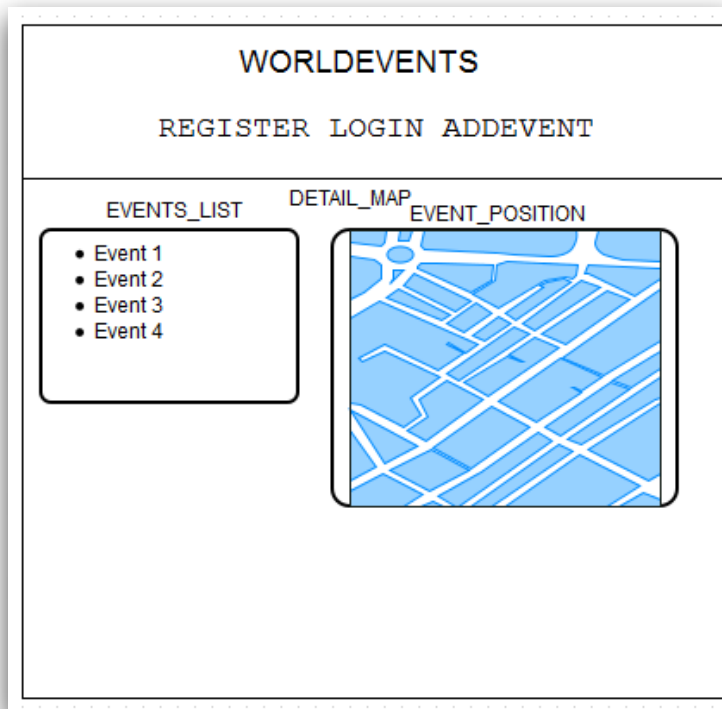


Figura 4.6 Diseño previo interfaz tablet 2

- Diseño para móvil. Aquí se muestra en pantalla la lista de eventos, seleccionando uno se pasa a una pantalla donde se muestra el detalle del evento y un enlace para alternar entre las vistas detalle y mapa, además de otro enlace para volver a la lista.

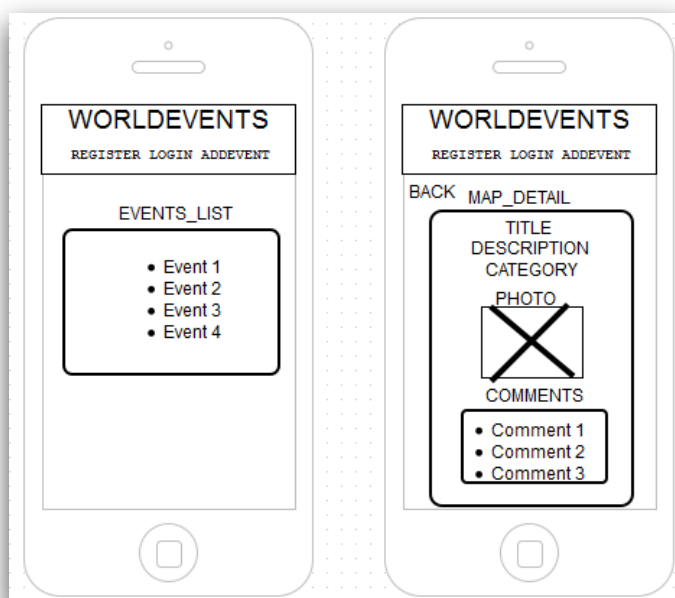


Figura 4.7 Diseño previo interfaz móvil

4.5 Sprint 1

4.5.1 Sprint Backlog

Para ese primer sprint se ha elaborado el siguiente *Backlog*. Se han seleccionado las historias más prioritarias de modo que el sistema pueda tener ya una funcionalidad básica. Esto incluye poder registrarse y luego loguearse para poder crear un evento y verlo en pantalla. Estas historias están resaltadas en amarillo. Las resaltadas en verde son historias transversales que habrá que tener en cuenta durante el desarrollo pero que no se corresponden directamente con una determinada funcionalidad.

SPRINT 1 BACKLOG
Historias de usuario
Como usuario no registrado quiero crear una cuenta
Como usuario registrado quiero poder loguearme
Como usuario registrado quiero poder cerrar sesion
Como usuario registrado quiero poder añadir un evento
Como usuario quiero poder ver una lista de eventos para poder seleccionar uno
Como usuario quiero poder ver el detalle de un evento seleccionado
Como cliente quiero que la aplicación responda rapido a las peticiones
Como cliente quiero que la aplicación se vea bien en todos los tamaños de pantalla
Como cliente quiero poder ver la pagina en ingles o español

Figura 4.8 Sprint1 Backlog

4.5.2 Consideraciones previas

Para la gestión de usuarios *Django* proporciona una aplicación ya operativa que siguiendo la filosofía de este framework se puede incluir ya en el proyecto aportando la funcionalidad requerida. *MongoEngine* es totalmente compatible con esta aplicación, luego el trabajo para desarrollar esta parte del sprint se reduce bastante.

En cuanto a la funcionalidad de añadir un evento, está requerirá la creación de los modelos *Event* y *Category*, así como las *views* necesarias. En principio, se incluirán como datos del evento el titulo, descripción, categoría, localización y fecha. La fecha se usara para ordenar los eventos y mostrar los añadidos más recientemente. En cuanto a la localización se usara el tipo *PointField* proporcionado por *MongoEngine*.

Este sprint requiere también la gestión de varios formularios, en este aspecto *Django* ofrece también un excelente soporte que simplifica mucho el trabajo.

Para seleccionar la posición de un nuevo evento se usará un mapa de Google apoyándose en la librería *gmaps3*.

En cuanto a la gestión de la navegación se va a usar *ajax* siempre que sea posible. En este caso se usará para descargar los formularios correspondientes sin recargar la página, ganando con ello en velocidad y mejorando la experiencia de usuario.

Para informar al usuario de diferentes eventos, como al alta de un nuevo usuario o de un nuevo evento, se usará la aplicación de mensajes de *Django*.

Para poder seleccionar un evento en el mapa y poder ver su detalle, se usará *JQuery*, de manera que al hacer click sobre un elemento de la lista se haga visible el detalle de dicho evento. Para optimizar la velocidad de la página se cargarán inicialmente todos los detalles y se mantendrán invisibles. Según el evento que seleccione el usuario se irán mostrando el adecuado y ocultando el resto.

En cuanto al tema del idioma solo es necesario marcar todas las cadenas de texto susceptibles de ser traducidas.

4.5.3 Análisis de historias de usuario

Crear cuenta	
Precondiciones	El usuario no debe estar logueado ni tener ya una cuenta
Poscondiciones	Debe existir un nuevo usuario en la base de datos con nombre de usuario único
Actores	Usuario no registrado
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click el enlace para registrarse 3. Rellena el formulario que contendrá los campos: username, password y comprobación de password 4. Enviará el formulario 5. Se le redigirá a la página inicial donde recibirá un mensaje indicando el éxito en la operación
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El nombre de usuario ya existe en el sistema <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo • Escenario Alternativo 2: Los datos introducidos en los campos de password no coinciden. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo • Escenario Alternativo 3: El nombre de usuario no satisface las condiciones previas. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Loguearse	
Precondiciones	El usuario no debe estar logueado y debe tener ya una cuenta
Poscondiciones	El usuario estará logueado en el sistema
Actores	Usuario registrado
Descripción	<p>El usuario registrado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click el enlace para loguearse 3. Rellena el formulario que contendrá los campos: username y password 4. Enviará el formulario 5. Se le redigirá a la página inicial donde recibirá un mensaje indicando el éxito en la operación
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: Los datos de login no son correctos <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Cerrar sesión	
Precondiciones	El usuario debe estar logueado
Poscondiciones	El usuario dejara de estar logueado en el sistema
Actores	Usuario registrado
Descripción	<p>El usuario registrado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click el enlace para hacer logout 3. Se le redigirá a la página inicial donde recibirá un mensaje indicando el éxito en la operación
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Añadir Evento	
Precondiciones	El usuario debe estar logueado
Poscondiciones	Se habrá añadido un nuevo evento en el sistema
Actores	Usuario registrado
Descripción	<p>El usuario registrado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click el enlace para añadir evento 3. Rellena el formulario que contendrá los campos: titulo, descripción y categoria 4. Seleccionara un radio de búsqueda y un punto en el mapa 5. Enviará el formulario 6. Se le redigirá a la página inicial donde recibirá un mensaje indicando el éxito en la operación y además podrá ver el nuevo evento en la lista de eventos
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: No se han introducido todos los campos obligatorios. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo • Escenario Alternativo 2: Los datos introducidos no cumplen las condiciones requeridas. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Ver lista de eventos	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. En ella puede ver una lista de eventos ordenados por fecha de inserción, mostrándose primero los más recientes.

Ver detalle de un evento	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none">3. Accede a la página de inicio4. En ella puede ver una lista de eventos donde el primero de ellos tiene un color de fondo naranja mostrando que está seleccionado.5. Se puede ver también el detalle con los datos del evento seleccionado.6. El usuario hace click en uno de los eventos de la lista7. El evento seleccionado cambiará su color de fondo para mostrar que está seleccionado.8. Se hará visible el detalle con los datos del evento seleccionado.

4.5.4 Diseño

- Diagrama de clases.

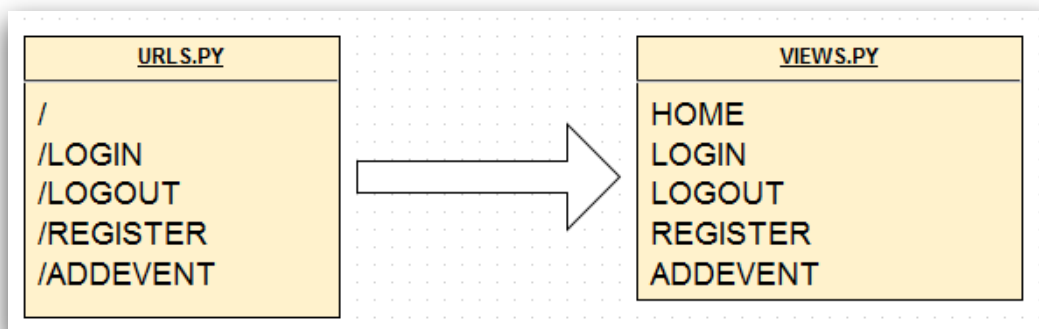


Figura 4.9 Sprint1 Diseño clases

- Diseño de los datos

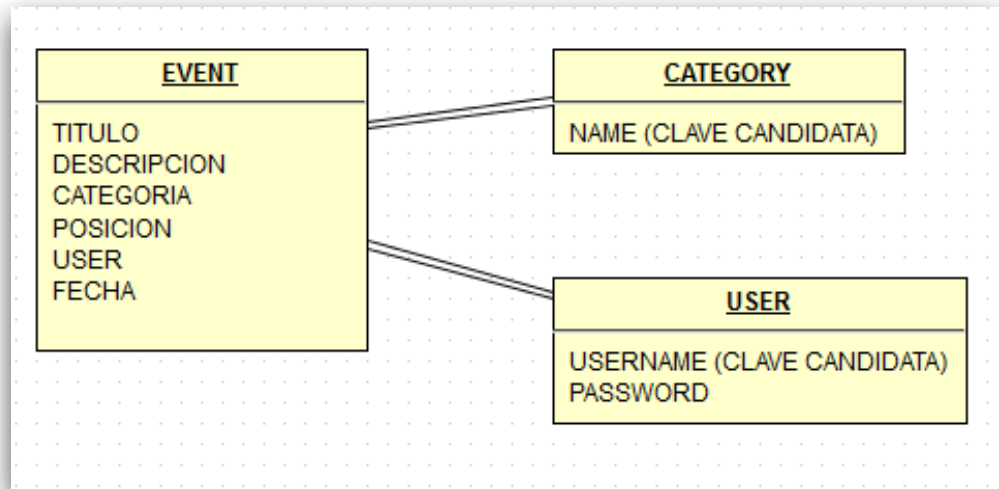


Figura 4.10 Sprint1 Diseño datos

- Estructura de *templates* y *forms*

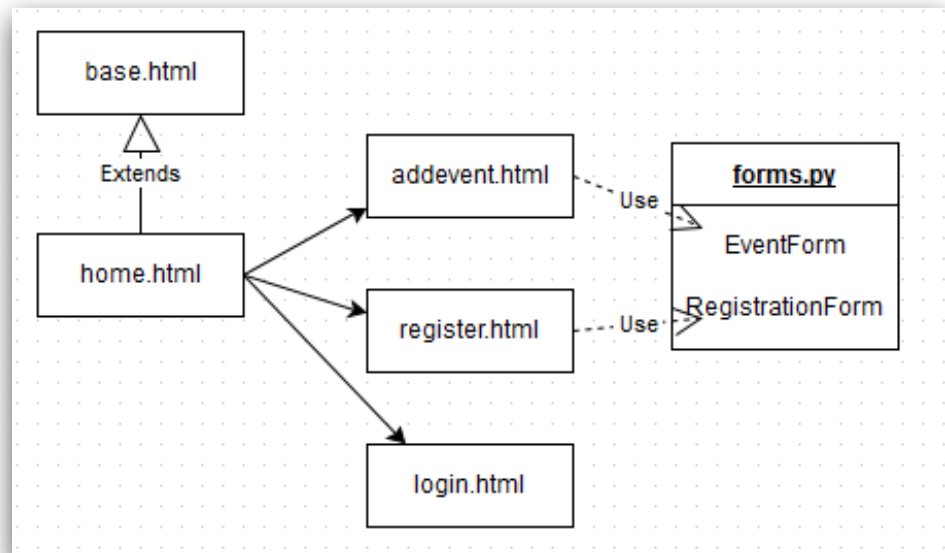


Figura 4.11 Sprint1 Diseño interfaz

4.5.5 Tests unitarios

Nombre del test
test_root_resolves_home_view
Descripción
Comprueba que la petición a la raíz de la aplicación es redireccionada a la view home.

Nombre del test
test_home_page_renders_home_template
Descripción
Comprueba que la respuesta a la petición a la raíz de la aplicación usa el template home.html

Nombre del test
test_home_page_show_events_list
Descripción
Comprueba que en la página inicial se muestra la lista de eventos

Nombre del test
test_events_details_are_in_home_page
Descripción
Comprueba que en la página inicial se muestra el detalle del evento seleccionado

Nombre del test
test_register_link_resolves_register_view
Descripción
Comprueba que en la página inicial se muestra la lista de eventos

Nombre del test
test_register_page_returns_correct_html
Descripción
Comprueba que la vista register devuelve la página correcta

Nombre del test
test_register_page_uses_registration_form
Descripción
Comprueba que la vista register usa el formulario RegisterForm

Nombre del test
test_register_page_save_user_to_database
Descripción
Comprueba el nuevo usuario registrado se guarda en la base de datos

Nombre del test
test_register_page_send_success_message_to_home
Descripción
Comprueba que el mensaje indicando que el usuario ha sido correctamente registrado se envía a la página inicial

Nombre del test
test_validation_errors_are_sent_back_to_register_template
Descripción
Comprueba que los errores en la validación se muestran en el mismo formulario

Nombre del test
test_for_invalid_input_passes_form_to_template
Descripción
Comprueba que ante errores de validación el formulario se pasa al template

Nombre del test
test_form_validation_for_blank_items
Descripción
Comprueba que si se dejan campos en blanco se devuelve un error

Nombre del test
test_form_validation_for_different_passwords
Descripción
Comprueba que si el contenido de los dos campos de password no coinciden se devuelve un error

Nombre del test
test_form_validation_for_duplicate_username
Descripción
Comprueba que si el nombre de usuario ya existe en el sistema se devuelve un error

Nombre del test
test_login_link_resolves_login_view
Descripción
Comprueba que el link login lleva a la view login

Nombre del test
test_login_page_returns_correct_html
Descripción
Comprueba que la view login devuelve el contenido esperado

Nombre del test
test_login_page_uses_authentication_form
Descripción
Comprueba que la vista login devuelve el formulario de autenticación

Nombre del test
test_login_page_login_user
Descripción
Comprueba que usando la view login el usuario consigue loguearse en el sistema

Nombre del test
test_login_form_validation_for_blank_items
Descripción
Comprueba que si se dejan los dos campos en blanco el formulario de login devuelve errores

Nombre del test
test_login_form_validation_for_blank_username
Descripción
Comprueba que si se dejan el campo username en blanco el formulario de login devuelve errores

Nombre del test
test_login_form_validation_for_blank_password
Descripción
Comprueba que si se dejan el campo password en blanco el formulario de login devuelve errores

Nombre del test
test_form_validation_for_wrong_data
Descripción
Comprueba que si los datos de login no existen en el sistema se devuelve un error

Nombre del test
test_logout_link_resolves_logout_view
Descripción
Comprueba que el link logout lleva a la view logout

Nombre del test
test_user_can_logout
Descripción
Comprueba que el usuario puede hacer logout

Nombre del test
test_addevent_page_requires_login_user
Descripción
Comprueba que para acceder a la página para añadir eventos es necesario estar logueado

Nombre del test
test_addevent_link_resolves_addevent_view
Descripción
Comprueba que el link addevent lleva a la view addevent

Nombre del test
test_addevent_page_uses_addevent_form
Descripción
Comprueba que la view addevent devuelve el formulario addevent_form

Nombre del test
test_addevent_page_save_event_to_database
Descripción
Comprueba que con la view addevent se añade el nuevo evento a la base de datos

Nombre del test
test_addevent_page_send_success_message_to_home
Descripción
Comprueba que al añadir un evento se envía un mensaje de éxito a la página inicial

Nombre del test
test_event_form_load_categories_from_db
Descripción
Comprueba que en el formulario se cargan las categorías de la base de datos

Nombre del test
test_event_validation_errors_are_sent_back_to_addevent_template
Descripción
Comprueba que los errores del formulario se pasan al template

Nombre del test
test_for_invalid_input_passes_event_form_to_template
Descripción
Comprueba que ante errores en los datos el formulario con los errores se pasa al template

Nombre del test
test_event_form_validation_for_blank_items
Descripción
Comprueba que ante datos en blanco en el formulario se muestra un error

Nombre del test
test_event_form_validation_for_blank_location
Descripción
Comprueba que si la localización se pasa en blanco se devuelve un error

4.6 Sprint 2

4.6.1 Sprint Backlog

Para ese segundo sprint se ha optado por añadir a la aplicación la opción de mostrar la localización de los eventos en el mapa así como la opción de buscar eventos por su localización y/o su título y categoría.

Además se completa la gestión de los eventos añadiendo la posibilidad de editarlo y borrarlos por parte de su creador.

Con estas funcionalidades la aplicación ya dispone de una operativa básica y se acerca a un producto terminado y susceptible de desplegar en un entorno de producción.

Historias de usuario
Como usuario quiero poder ver la localización de un evento seleccionado en un mapa
Como usuario quiero poder buscar eventos por localización y título y/o categoría
Como usuario registrado quiero editar eventos creados por mi
Como usuario registrado quiero borrar eventos creados por mi
Como cliente quiero que la aplicación responda rapido a las peticiones
Como cliente quiero que la aplicación se vea bien en todos los tamaños de pantalla
Como cliente quiero poder ver la pagina en ingles o español

Figura 4.12 Sprint2 Backlog

4.6.2 Consideraciones previas

Las opciones de editar y borrar eventos son tareas habituales que no necesitan ningún tipo de comentario. Únicamente si cabe, destacar la facilidad de uso de *Django* a la hora de realizar este tipo de tareas típicas dentro de una aplicación web.

Para la funcionalidad de mostrar la localización en un mapa se va a usar la potencia y simplicidad de la librería *gmaps3*. Usando *JQuery* se programará la acción de hacer click sobre un evento de la lista de modo que se muestre en un mapa de Google un indicador marcando la posición del evento.

En cuanto a la funcionalidad de búsqueda se usará la potencia de la API de geolocalización de *MongoDB* que permite realizar búsquedas dentro una zona geográfica. Para el formulario de búsqueda será necesario diseñar un interfaz de usuario que permita seleccionar una zona del mapa. Al incluir este tipo de consultas hay que plantearse ya la elección de los índices adecuados a incluir en la colección *Event* para optimizar las búsquedas. Como en el mundo relacional, la inclusión de los índices favorece las consultas pero penaliza las inserciones y borrados. En el caso de esta aplicación el escenario esperado es que el número de consultas sea muy superior al de inserciones, con lo cual es adecuado incluir un número alto de índices para agilizar las consultas.

4.6.3 Análisis de historias de usuario

Ver localización en un mapa	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none">1. Accede a la página de inicio2. En la página de inicio habrá una lista con los eventos del sistema, donde el primero tendrá un fondo de color naranja indicando que está seleccionado3. Se mostrará también un mapa de Google con una serie de marcadores de color azul indicando las posiciones de los eventos de la lista, excepto el del seleccionado que tendrá un marcador rojo.4. Hace click en uno de los eventos de la lista5. El elemento seleccionado cambiará su color de fondo para indicar que está seleccionado6. En el mapa de Google se mostrará un indicador de color rojo mostrando la localización del evento seleccionado.

Buscar eventos	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click el enlace search 3. Rellena el formulario que contendrá los campos: titulo, categoría. Estos campos son obligatorios. 4. El formulario tendrá un mapa donde se podrá seleccionar un punto y un control donde podrá indicar un radio de búsqueda. Es obligatorio que el usuario seleccione un punto y un radio. 5. Enviará el formulario 6. Se le redigirá a la página inicial donde se mostrará la lista de eventos resultado de la búsqueda.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: La búsqueda no devuelve ningún resultado. <ol style="list-style-type: none"> 1. Se redigirá al usuario a la página inicial donde se mostrará un mensaje indicando la ausencia de resultados.
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Editar evento	
Precondiciones	El usuario debe estar logueado y ser el creador del evento
Poscondiciones	El evento se guarda en el sistema reflejando los cambios realizados
Actores	Usuario registrado
Descripción	<p>El usuario registrado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Selecciona un evento que él ha creado 3. Hace click en el enlace edit del detalle del evento 4. Se mostrará un formulario con los datos del evento. 5. El usuario modifica los datos que quiera y envía el formulario 6. El usuario es redirigido a la página inicial donde puede comprobar que el evento refleja los cambios realizados
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: Se han introducido valores no válidos. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Borrar Evento	
Precondiciones	El usuario debe estar logueado y ser el creador del evento
Poscondiciones	El evento se elimina del sistema
Actores	Usuario registrado
Descripción	<p>El usuario registrado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Selecciona un evento que él ha creado 3. Hace click en el enlace delete del detalle del evento 4. Se mostrará un formulario para confirmar la operación. 5. El usuario confirma la eliminación 6. El usuario es redirigido a la página inicial donde puede comprobar que el evento ya no aparece
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: El usuario decide no confirmar la eliminación. <ol style="list-style-type: none"> 1. Se redirige a la página inicial sin realizar ningún cambio en el sistema.
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

4.6.4 Diseño

- Diagrama de clases.

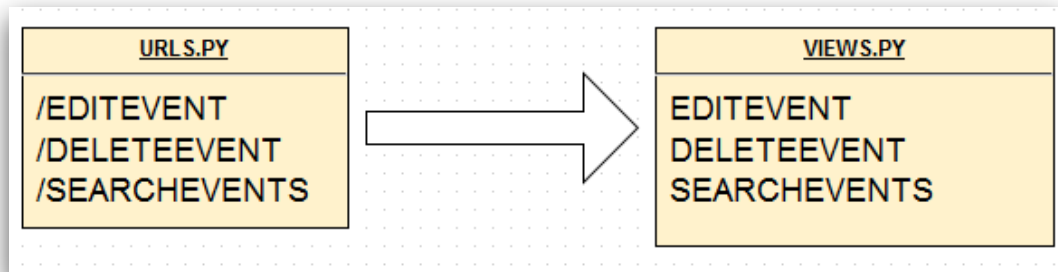


Figura 4.13 Sprint2 Diseño clases

- Diseño de los datos. El esquema no sufre cambios con respecto a la anterior iteración, sin embargo se van añadir los siguientes índices para optimizar las búsquedas:
 1. Dado que las listas de eventos siempre se ordenan por fecha de inserción se incluirá un índice simple descendente en el campo fecha.
 2. Para la consulta que se realiza en la *view* searchevents que comprende los campos POSICION, TITULO y CATEGORIA se va a crear un *COMPOUND INDEX* con esos tres campos en ese orden. Este índice garantizará un rendimiento óptimo para las consultas que incluyan las siguientes combinaciones de campos:
 - POSICION,TITULO,CATEGORIA
 - POSICION,TITULO
 - POSICION

Para las consultas con los campos POSICION,CATEGORIA, este índice también ofrece soporte, pero el rendimiento será algo peor que usando un índice con solo esos dos campos.

Por otra parte el resultado de la búsqueda se ha de devolver ordenado por fecha de inserción, en este caso aprovechando que ya existe un índice en el campo fecha, este tipo de consultas usarán la capacidad de *MongoDB* de realizar intersección de índices. De este modo los dos índices creados podrán trabajar en conjunto para devolver de modo eficiente los datos solicitados.

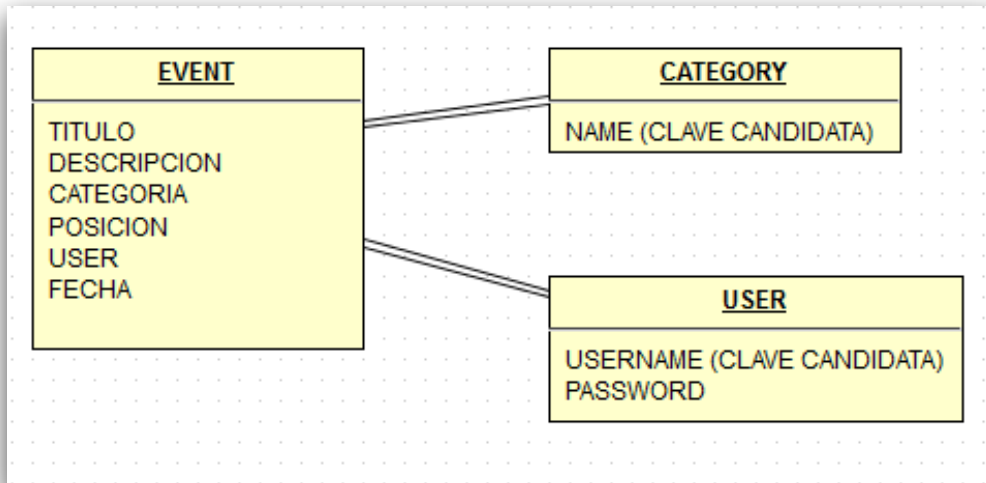


Figura 4.14 Sprint2 Diseño datos

- Estructura de *templates* y *forms*

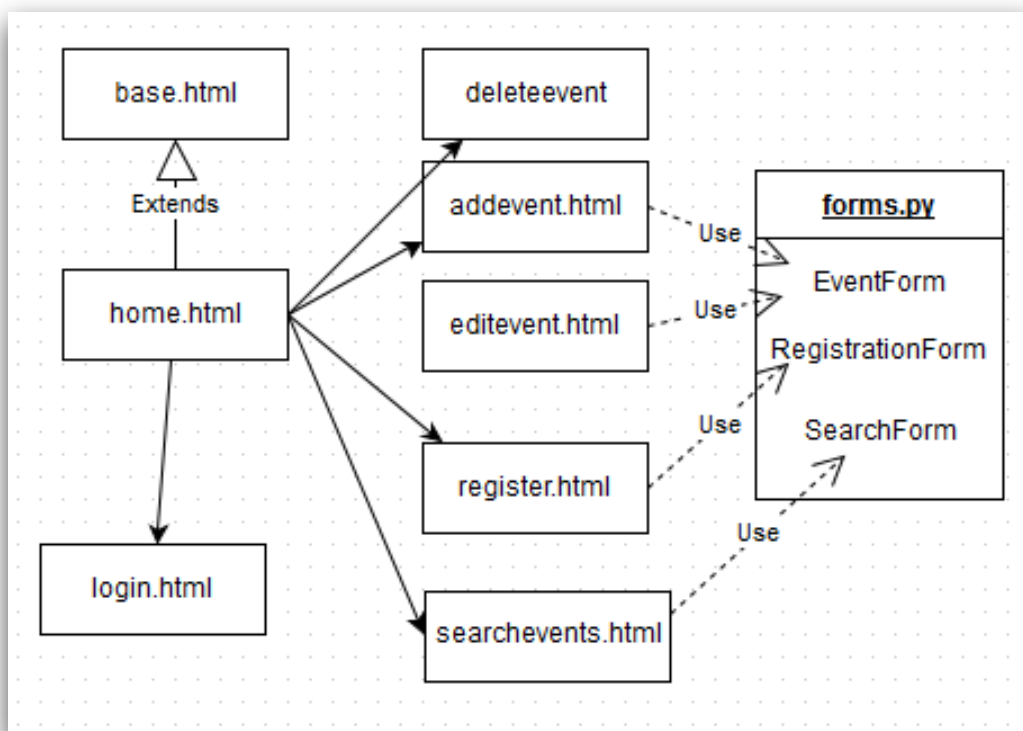


Figura 4.15 Sprint2 Diseño interfaz

4.6.5 Tests unitarios

Nombre del test
test_editevent_page_requires_login_user
Descripción
Comprueba que para editar eventos es necesario estar logueado.

Nombre del test
test_editevent_link_resolves_editevent_view
Descripción
Comprueba que el link edit lleva a la view editevent

Nombre del test
test_editevent_page_uses_editevent_form
Descripción
Comprueba que la vista editevent usa el formulario EventForm

Nombre del test
test_editevent_page_save_event_to_database
Descripción
Comprueba que la view editevent guarda las modificaciones en la base de datos

Nombre del test
test_editevent_page_send_success_message_to_home
Descripción
Comprueba que después de editar un evento se envía un mensaje de éxito a la página inicial

Nombre del test
test_edit_event_form_load_categories_from_db
Descripción
Comprueba que en el formulario de edición están las categorías de la base de datos

Nombre del test
test_validation_errors_are_sent_back_to_editevent_template
Descripción
Comprueba que los errores de validación se envían al template

Nombre del test
test_data_are_sent_to_editevent_template
Descripción
Comprueba que los datos del evento se muestran en el formulario

Nombre del test
test_deleteevent_page_requires_login_user
Descripción
Comprueba que para borrar un evento es necesario estar logueado

Nombre del test
test_deleteevent_link_resolves_deleteevent_view
Descripción
Comprueba que el link delete lleva a la view deleteevent

Nombre del test
test_deleteevent_page_delete_event_from_database
Descripción
Comprueba que la view deleteevent elimina el evento

Nombre del test
test_searchevents_link_resolves_searchevents_view
Descripción
Comprueba que el link search lleva a la view searchevents

Nombre del test
test_search_page_uses_search_form
Descripción
Comprueba que la view searchevent usa el formulario SearchForm

Nombre del test
test_search_form_return_correct_events
Descripción
Comprueba que la búsqueda devuelve los resultado correctos

Nombre del test
test_search_form_load_categories_from_db
Descripción
Comprueba que en el formulario se muestran las categorías de la base de datos

Nombre del test
test_validation_errors_are_sent_back_to_searchevents_template
Descripción
Comprueba que los errores de validación del formulario de búsqueda se envían al template

Nombre del test
test_for_invalid_input_passes_form_to_template
Descripción
Comprueba que ante una entrada invalida los datos se pasan al template

Nombre del test
test_form_validation_for_blank_location
Descripción
Comprueba que si se deja en blanco la localización se devuelve un error

Nombre del test
test_form_validation_for_right_distance
Descripción
Comprueba que si el radio de búsqueda no está dentro del rango permitido se devuelve un error

4.7 Sprint 3

4.7.1 Sprint Backlog

Una vez alcanzada una funcionalidad mínima en las anteriores iteraciones. En esta tercera iteración se incluyen mejoras que hacen la aplicación más rica y profesional. Esas mejoras son la posibilidad de añadir fotos y comentarios en los eventos. Además dado que el número de eventos puede llegar a ser elevado se incluye la capacidad de navegar por la lista de eventos mediante un sistema de paginación, que muestre en cada página un número de registros adecuado y manejable. Esta opción se usa tanto para moverse por la lista completa como para hacerlo por el resultado de una búsqueda.

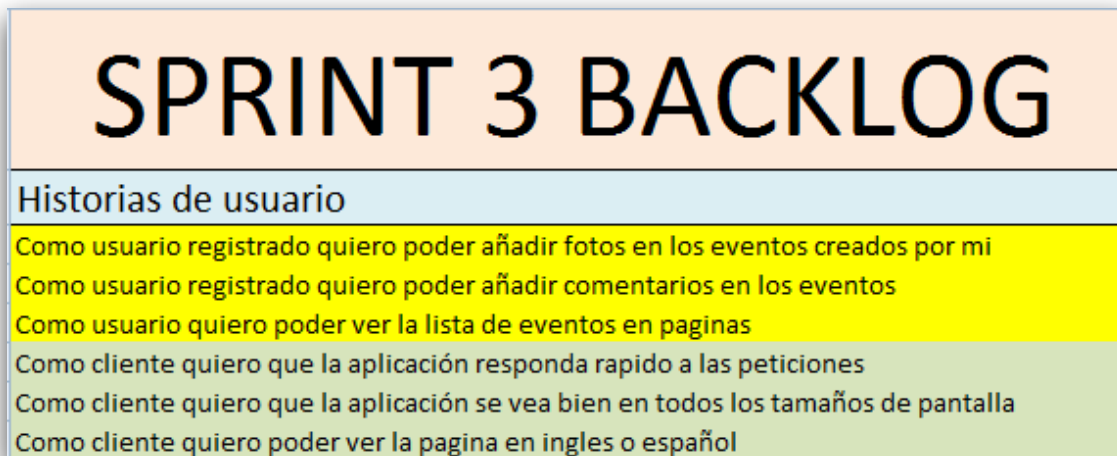


Figura 4.16 Sprint3 Backlog

4.7.2 Consideraciones previas

Esta iteración es probablemente la que incluye las operaciones más complejas de todas las realizadas hasta el momento. Todas ellas requieren un cuidadoso ejercicio de análisis previo para poder finalizarlas con éxito.

Para la gestión de fotos se plantea la duda de si guardarlas en base de datos o usarlas como archivos usando el sistema de gestión de archivos estáticos de *Django*. *MongoDB* ofrece la posibilidad de trabajar archivos con archivos binarios además de la especificación GridFS que permite trabajar con archivos binarios de gran tamaño. Por otra parte Django ofrece dentro de su estructura un sistema fácil para tratar con archivos estáticos dentro de una aplicación web. Dado que en este proyecto se busca rendimiento parece más adecuada la opción de trabajar directamente con los archivos mejor que hacerlo a través de la base de datos con el retardo que eso puede suponer. Por tanto se va a optar por guardar los archivos de imagen en el sistema de ficheros gestionado por *Django*, y guardar en la base de datos dentro de cada evento el nombre de su correspondiente fichero de imagen.

Para la opción de añadir comentarios, se va a seguir el diseño de datos realizado en el análisis previo. Es decir, se va a incluir dentro de cada evento su correspondiente lista de comentarios, siguiendo las prácticas habituales para el modelado de datos con soluciones *NoSQL*. Además esta opción ofrece beneficios de rendimiento ya que con cada consulta a los eventos se va a recuperar además sus comentarios eliminando la necesidad de realizar dos consultas para obtener la misma información.

En cuanto a la paginación de los eventos se va a usar la política de indicar en la url el intervalo de la lista de eventos que queremos. Por ejemplo la url "http://dominio/eventslist/30" partiendo de un tamaño de página 15 devolverá la lista con los eventos del 15 al 30. Además cuando se

navegue por los resultados de una búsqueda será necesario guardar los parámetros de dicha búsqueda durante las sucesivas peticiones. Para ello se guardarán dichos parámetros en la sesión, que en esta aplicación se guardará en base de datos. Por otra parte para seguir con la política de optimizar la carga de páginas usando *Ajax*, para la navegación por páginas se usará el método de cargar solo la nueva página con los eventos en lugar de recargar toda la página. Esto hará que aparezca una nueva *template* "eventslist.html" que contendrá los datos de la lista correspondiente.

4.7.3 Análisis de historias de usuario

Para la funcionalidad de añadir fotos será necesario actualizar las operaciones "Añadir Evento", "Editar Evento", "Borrar Evento" y "Ver Detalle del Evento" para incluir esta opción.

Añadir Comentario	
Precondiciones	El usuario debe estar logueado
Poscondiciones	Se habrá añadido un nuevo comentario a uno de los eventos
Actores	Usuario logueado
Descripción	<p>El usuario logueado:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Hace click en el enlace añadir comentario de un evento 3. Rellena el formulario que contendrá un campo texto para incluir el comentario 4. Enviará el formulario 5. En el detalle del evento aparecerá el comentario y el nombre del usuario que lo realizó en la lista de comentarios del evento.
Variaciones (escenarios secundarios)	<ul style="list-style-type: none"> • Escenario Alternativo 1: Se ha superado la longitud máxima del comentario. <ol style="list-style-type: none"> 1. Recargar el formulario indicando el fallo
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	

Navegar por las páginas de la lista de eventos	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Puede ver que la lista de eventos incluye enlaces para ir hacia adelante o hacia atrás 3. Usando dichos enlace puede avanzar o retroceder a través de las páginas de la lista de eventos.
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

Navegar por las páginas de una lista de eventos resultado de una búsqueda	
Precondiciones	N/A
Poscondiciones	N/A
Actores	Usuario
Descripción	<p>El usuario:</p> <ol style="list-style-type: none"> 1. Accede a la página de inicio 2. Realiza una búsqueda y es redirigido a la página principal donde se muestran los primeros 15 eventos del resultado de la búsqueda, además de dos enlaces para avanzar o retroceder en la lista. 3. Usando dichos enlace puede avanzar o retroceder a través de las páginas de la lista de eventos resultado de la búsqueda.
Excepciones	<ul style="list-style-type: none"> • La base de datos no está disponible <ol style="list-style-type: none"> 1. Enviar un mensaje indicando el problema
Notas	-

4.7.4 Diseño

- Diagrama de clases. Se incluye además de las nueva operación "Addcomment" las operaciones relacionadas con eventos que han de ser modificadas. La página inicial también ha de ser modificada para incluir las fotos en el detalle de los eventos. También se añade una nueva url que incluye un número indicando la página a mostrar y que redirecciona hacia la [view](#) "Home".

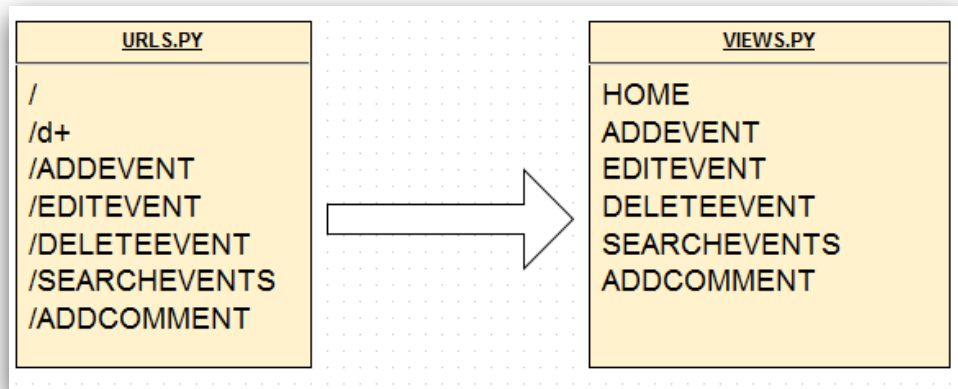


Figura 4.17 Sprint3 Diseño clases

- Diseño de los datos. Se incluye la entidad "Comment" y un nuevo campo "comments" en la entidad "Event" que incluye una lista con los comentarios asociados a dicho evento. También se incluye un campo foto que incluirá el nombre del archivo de imagen. Se incluye también la entidad "Sesion" que es gestionada por *Django*.

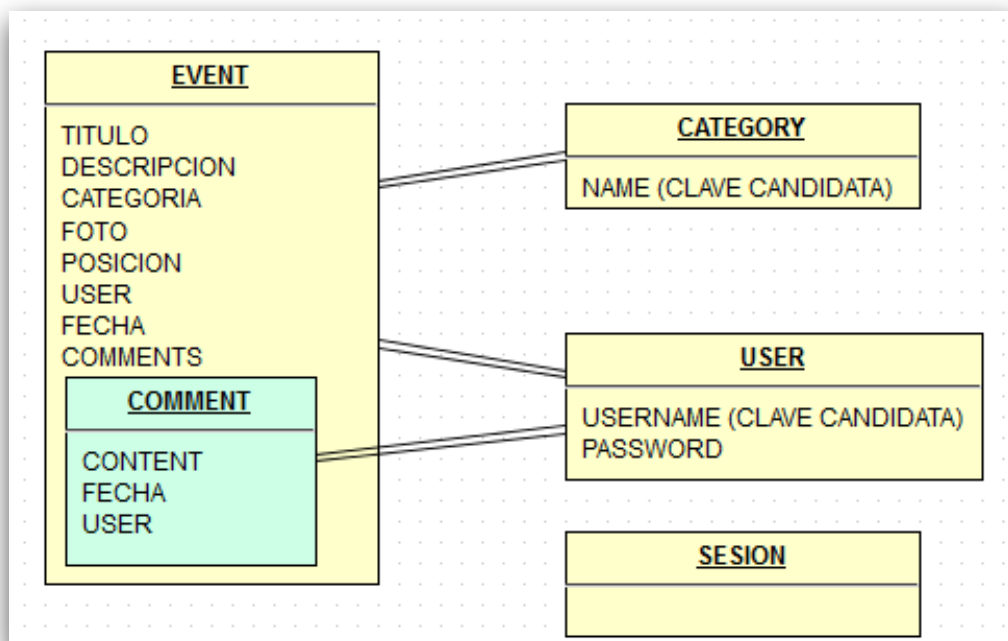


Figura 4.18 Sprint3 Diseño datos

- Estructura de *templates* y *forms*. En esta diseño se crea una nueva *template* llamada "eventslist.html". Dado que se va a trabajar con páginas y se va optimizar la navegación usando *Ajax* para cargar solo la página de la lista de eventos necesaria en

cada momento, se ha optado por separar los datos de los eventos en esa nueva *template*, que ira incluida en la *template* "home". Se ha creado también una *template* "comments.html" para almacenar la lista de comentarios de un evento. Irá incluida en la *template* "eventslist.html".

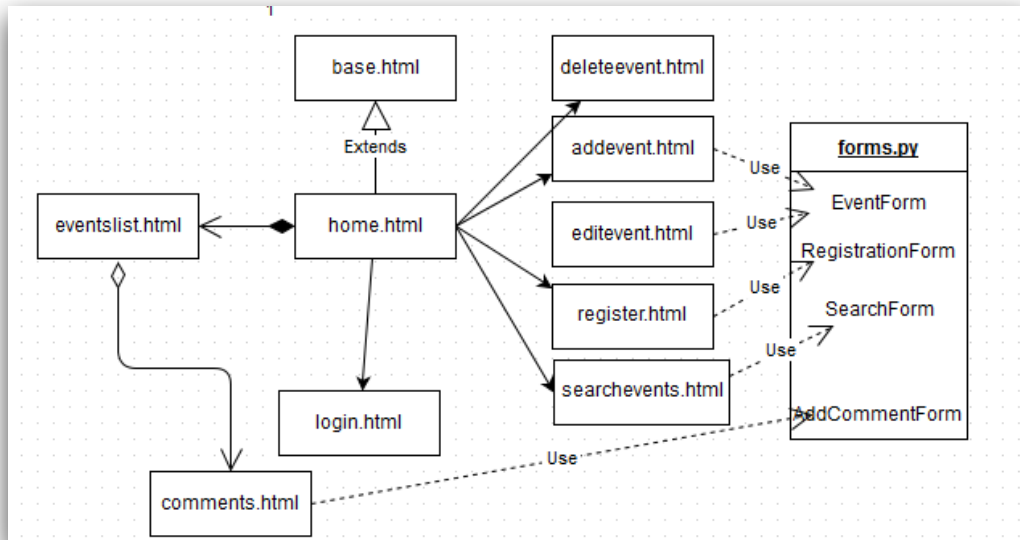


Figura 4.19 Sprint3 Diseño interfaz

4.7.5 Tests unitarios

Nombre del test
test_home_page_show_events_pages
Descripción
Comprueba que en la página principal se muestra una estructura de navegación para moverse por las páginas de la lista de eventos y que los enlaces funcionan

Nombre del test
test_events_photos_are_in_home_page
Descripción
Comprueba que en el detalle del evento seleccionado se muestra la foto

Nombre del test
test_addevent_page_save_event_to_database_and_create_photo_file
Descripción
Sustituye al test "test_addevent_page_save_event_to_database". Comprueba que la vista addevent añade el nuevo evento a la base de datos y que el fichero imagen se guarda en la carpeta adecuada

Nombre del test
test_editevent_page_save_event_to_database_save_photo_file
Descripción
Sustituye al test " test_editevent_page_save_event_to_database ". Comprueba que la view editevent guarda las modificaciones en la base de datos y actualiza el archivo de imagen en la carpeta adecuada

Nombre del test
test_deleteevent_page_delete_event_from_database_and_photo_file
Descripción
Sustituye al test " test_deleteevent_page_delete_event_from_database ". Comprueba que se borra el evento y además el fichero de imagen de la carpeta adecuada.

Nombre del test
test_search_show_events_pages
Descripción
Comprueba que después de una búsqueda los resultado se presentan con una estructura de navegación a través de páginas y que esta funciona.

Nombre del test
test_addcomment_requires_login_user
Descripción
Comprueba que para añadir un comentario hay que estar logueado

Nombre del test
test_addcomment_link_resolves_addcomment_view
Descripción
Comprueba el link addcomment lleva a la view addcomment

Nombre del test
test_addcomment_page_uses_addcomment_form
Descripción
Comprueba que la view addcomment usa el formulario adecuado

Nombre del test
test_addcomment_page_save_comment_to_database
Descripción
Comprueba que el comentario añadido se guarda en la base de datos

Nombre del test
test_comments_are_in_home_page
Descripción
Comprueba que los comentarios añadidos se muestran en el detalle del evento

4.8 Documentación final

4.8.1 Descripción detallada de clases y métodos

- Fichero "models.py"
En este fichero se incluyen las clases que modelan los datos y que *MongoEngine*, el mapeador usado, emplea para crear la estructura en *MongoDB*. Las entidades "User" y "Session" son modeladas automáticamente por las aplicaciones correspondientes de *Django*.

Nombre Clase	Descripción	Hereda de...
Event	Modela la entidad Evento.	Document
<u>Responsabilidades</u>		
Descripción		
Crea un modelo para almacenar eventos en la base de datos. Incluye un campo tipo lista donde se almacenan los comentarios asociados al evento, usando la entidad Comentario como tipo embebido.		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
queryset	search	title,category (String) lat,lng(Float) distance, num_events(Int)
string	__str__	N/A
<u>Atributos</u>		
Nombre		Tipo
title		StringField
photo		StringField
description		StringField
category		StringField
user		StringField
location		PointField
added_date		DateTimeField
comments		ListField
Observaciones		
El metodo search devuelve una lista con los eventos que cumplan las condiciones pasadas como parámetros.		

Nombre Clase	Descripción	Hereda de...
Comment	Modela la entidad Comentario.	Document
<u>Responsabilidades</u>		
Descripción		
Crea un modelo para almacenar comentarios de eventos en la base de datos. Se va a usar como campo embebido en la entidad Evento.		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
string	__str__	N/A
<u>Atributos</u>		
Nombre		Tipo
content		StringField
user		StringField
added_date		DateTimeField
Observaciones		
N/A		

Nombre Clase	Descripción	Hereda de...
Category	Modela la entidad Categoría.	Document
<u>Responsabilidades</u>		
Descripción		
Crea un modelo para almacenar las posibles categorías asociadas a los eventos.		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
string	__str__	N/A
<u>Atributos</u>		
Nombre		Tipo
name		StringField
Observaciones		
El valor del campo name ha de ser único en la colección		

- Fichero "views.py"
En este fichero se guardan los métodos responsables de atender las peticiones realizadas a la aplicación.

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)	home	request(HttpRequest) num_events(String)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición a la página de inicio de la aplicación "/". El parámetro num_events es opcional y su valor por defecto es 15. Devuelve la página principal con la lista de eventos asociada. Comprueba si en sesión existe una búsqueda actual y en ese caso trabaja sobre los resultados de esa búsqueda. En caso contrario trabaja sobre la lista completa de eventos.		
Observaciones		
El parámetro num_events indica el número de orden superior de la lista de eventos actualmente solicitada.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)	addcomment	request(HttpRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/addcomment" y procesa el formulario CommentForm. Añade al evento correspondiente en la base de datos el comentario pasado como dato del formulario. Devuelve la página comments.html asociada al evento.		
Observaciones		
Requiere estar logueado.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	searchevents	request(HttpRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/searchevents" y procesa el formulario SearchForm. Obtiene de la base de datos la primera página de resultados de la búsqueda solicitada y añade los datos en la sesión para poder seguir navegando a través de las páginas de la búsqueda. Si la petición es GET devuelve la página "searchevents.html" y si es POST hace una redirección a la página principal.		
Observaciones		
N/A.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	register	request(HttpServletRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/register" y procesa el formulario RegistrationForm. Añade un nuevo usuario a la base de datos con los datos pasados en el formulario. Si la petición es GET devuelve la página "register.html" y si es POST hace una redirección a la página principal.		
Observaciones		
El nombre del nuevo usuario no debe existir en la base de datos.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	loginpage	request(HttpServletRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/login" y procesa el formulario AuthenticationForm. Loguea en el sistema al usuario cuyos se han pasado en el formulario. Si la petición es GET devuelve la página "login.html" y si es POST hace una redirección a la página principal.		
Observaciones		
El usuario debe existir en la base de datos y sus datos ser correctos.		

Tipo de retorno	Nombre método	Parámetros y tipos
Redirección	logout	request(HttpServletRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/logout". Cierra la sesión del usuario actual y hace una redirección a la página principal.		
Observaciones		
El usuario debe estar logueado.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	addevent	request(HttpServletRequest)
<u>Responsabilidades</u>		
Descripción		
Atiende la petición "/addevent" y procesa el formulario EventForm. Añade un nuevo evento a la base de datos con los datos pasados en el formulario. Si la petición es GET devuelve la página "addevent.html" y si es POST hace una redirección a la página principal.		
Observaciones		
El usuario debe estar logueado.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	deletevent	request(HttpServletRequest) event_id(String)
<i>Responsabilidades</i>		
Descripción		
Atiende la petición "/addevent". Elimina de la base de datos el evento cuyo id se pasa como parámetro y elimina el archivo de imagen de la foto del evento si está incluida. Si la petición es GET devuelve la página "deleteevent.html" y si es POST hace una redirección a la página principal.		
Observaciones		
El usuario debe estar logueado.		

Tipo de retorno	Nombre método	Parámetros y tipos
Texto(html)/Redirección	editevent	request(HttpServletRequest) event_id(String)
<i>Responsabilidades</i>		
Descripción		
Atiende la petición "/editevent" y procesa el formulario EventForm. Modifica en la base de datos los datos del evento cuyo id se pasa como parámetro y si es necesario actualiza también el archivo de imagen, eliminando el anterior y creando uno nuevo. Si la petición es GET devuelve la página "editevent.html" con los datos del evento cargados en el formulario. Y si es POST hace una redirección a la página principal.		
Observaciones		
El usuario debe estar logueado.		

- Fichero "forms.py"
Aquí se almacenan las clases que permiten generar los formularios de la aplicación. El formulario "AuthenticationForm" que se usa para trabajar con la entidad "User" es generado automáticamente por la aplicación "auth" de Django.

Nombre Clase	Descripción	Hereda de...
SearchForm	Crea el formulario para realizar búsquedas de eventos.	Forms.Form
<u>Responsabilidades</u>		
Descripción		
Sirve como base para la creación automática por parte de Django de un formulario para realizar búsquedas de eventos		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
N/A	__init__	N/A
Dictionary	clean	
<u>Atributos</u>		
Nombre		Tipo
title		forms.CharField
category		forms.ChoiceField
lat		forms.CharField
lng		forms.CharField
distance		forms.CharField
Observaciones		
El metodo __init__ se ocupa de recuperar las categorías de la base de datos y precargarlas en el campo category.		
El metodo clean comprueba que los valores lat,lng y distance sean válidos.		

Nombre Clase	Descripción	Hereda de...
RegistrationForm	Crea el formulario para realizar el registro de un nuevo usuario.	Forms.Form
<u>Responsabilidades</u>		
Descripción		
Sirve como base para la creación automática por parte de Django de un formulario para realizar el registro de un nuevo usuario		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
Dictionary	clean	
<u>Atributos</u>		
Nombre		Tipo
username		forms.RegexField
password1		forms.CharField
password2		forms.CharField
Observaciones		
El metodo clean comprueba que los valores de los dos campos de contraseña coincidan y que el nombre de usuario no exista ya en el sistema.		

Nombre Clase	Descripción	Hereda de...
EventForm	Crea el formulario para trabajar con eventos.	Forms.Form
<u>Responsabilidades</u>		
Descripción		
Sirve como base para la creación automática por parte de Django de un formulario para realizar añadir o editar eventos.		
<u>Métodos</u>		
Tipo de Retorno	Nombre	Parámetros y tipos
N/A	__init__	N/A
Dictionary	clean	
<u>Atributos</u>		
Nombre		Tipo
title		forms.CharField
category		forms.ChoiceField
description		forms.CharField
photo		forms.CharField
lat		forms.CharField
lng		forms.CharField
Observaciones		
El metodo __init__ se ocupa de recuperar las categorías de la base de datos y precargarlas en el campo category.		
El metodo clean comprueba que los valores lat,lng sean válidos.		

4.8.2 Mapa de rutas del sistema.

Ruta(Expresión regular)	View asociada
^(\\d+)\$	home
^login\$	loginpage
^logout\$	logoutpage
^register\$	register
^addevent\$	addevent
^addcomment\$	addcomment
^deleteevent/(\\w+)\$	deleteevent
^editevent/(\\w+)\$	editevent

4.8.3 Descripción detallada del esquema de base de datos.

Nombre Entidad	Descripción	
Event	Almacena la información asociada a los eventos	
Estructura		
Campo	Tipo	Descripción
_id	Object id	Campo autogenerado que sirve de clave primaria
title	String	Título del evento
description	String	Descripción del evento
added_date	Date	Fecha de inserción del evento
photo	String	Nombre del archivo de imagen
user	String	Nombre del usuario que creó el evento
location	Point	Coordenadas geográficas del evento
comments	Array	Lista de los comentarios asociados a ese evento
Indices		
Campos		Tipo
_id		Single Field
added_date		Single Field(Descendente)
location		2dsphere
location,title,category		Compound Index
Observaciones		
El índice (location,title,category) sirve para dar soporte a la búsqueda de eventos independientemente de los parámetros que se usen.		

Nombre Entidad	Descripción	
Category	Almacena los nombres de las categorías.	
Estructura		
Campo	Tipo	Descripción
_id	Object id	Campo autogenerado que sirve de clave primaria
name	String	Nombre de la categoría
Indices		
Campos		Tipo
_id		Single Field
Observaciones		
N/A		

Nombre Entidad	Descripción	
Comment	Almacena la información asociada a los comentarios	
<i>Estructura</i>		
Campo	Tipo	Descripción
_id	Object id	Campo autogenerated que sirve de clave primaria
content	String	Texto del comentario
user	String	Nombre del usuario que hizo el comentario
added_date	Date	Fecha de inserción del comentario
<i>Indices</i>		
Campos		Tipo
_id		Single Field
Observaciones		
N/A		

4.8.4 Descripción de la navegabilidad.

Durante el desarrollo de la aplicación se ha intentado hacer uso de Ajax siempre que fuera posible para evitar de ese modo la recarga completa de la página. A continuación se incluyen diagramas mostrando las operaciones asíncronas incluidas.

- Página inicial sin estar logueado



Figura 4.20 Pagina inicial sin loguear

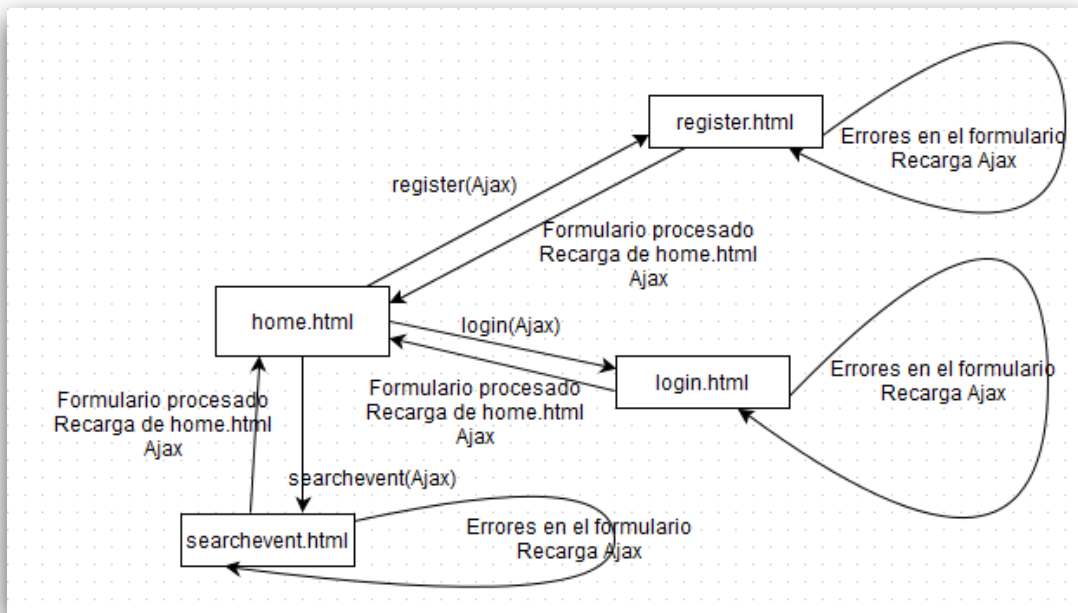


Figura 4.21 Diagrama navegabilidad sin loguear

- Página inicial logueado



Figura 4.22 Página inicial logueado

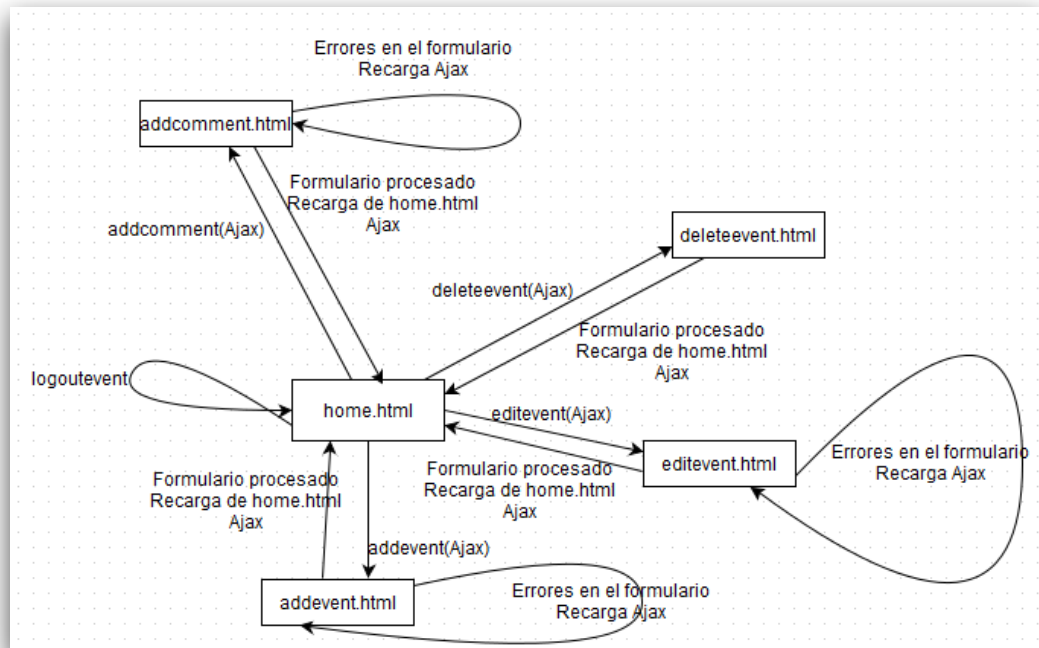


Figura 4.23 Diagrama navegabilidad logueado

- Navegación a través de la lista



Figura 4.24 Detalle navegación listas eventos

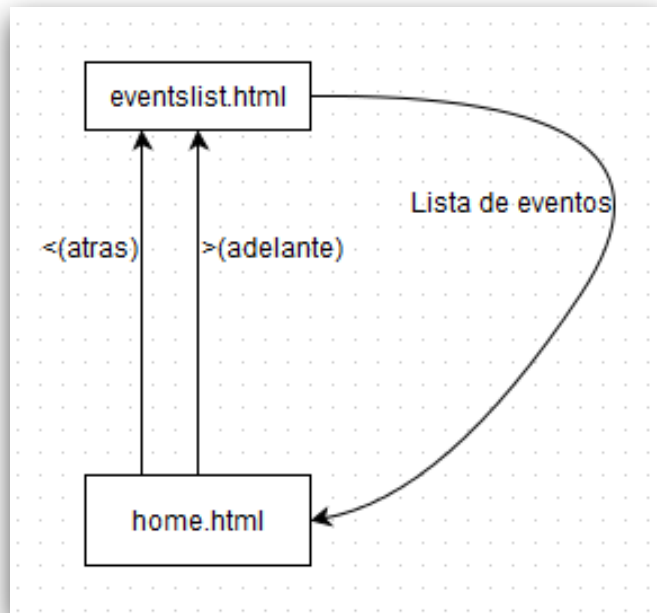


Figura 4.25 Diagrama navegabilidad listas eventos

4.8.5 Responsive design.

A continuación se incluyen capturas de pantalla de la aplicación mostrando el diseño realizado en cada uno de los tres tamaños tratados.

- Ordenador. Tamaño de pantalla mayor de 1024 pixeles. que 768 pixeles(móviles), entre 768 y 1024 pixeles(tablets) y mayor de 1024 pixeles(ordenador).



Figura 4.26 Responsive design tamaño ordenador

- Tablet. Entre 768 y 1024 píxeles.

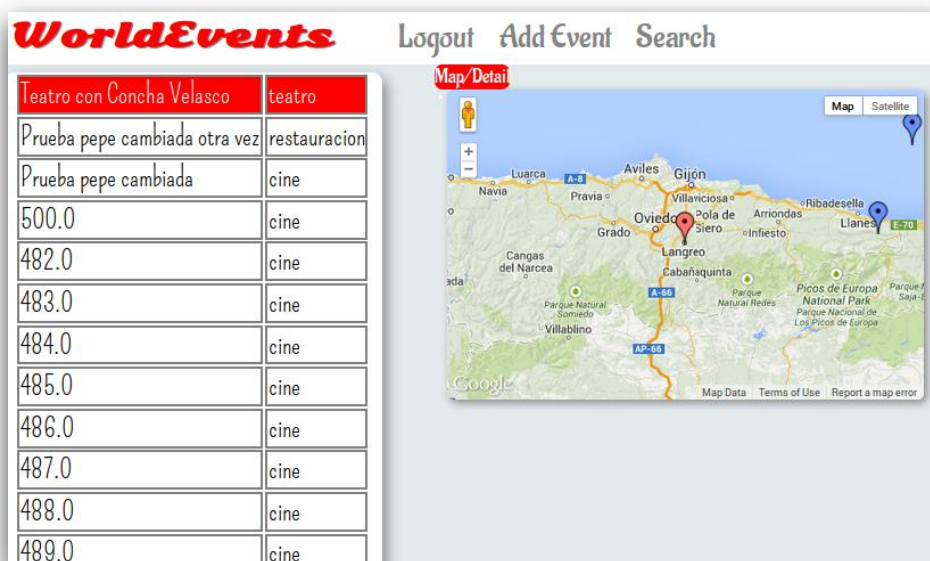


Figura 4.27 Responsive design tamaño tablet

- Móvil. Menor que 768 pixeles.



Figura 4.28 Responsive design tamaño móvil

Capítulo 5. Despliegue de la aplicación.

5.1 Elección de la plataforma

Para la elección de la plataforma se han evaluado un grupo significativo de proveedores de *Cloud Computing*.

- Google App Engine

Esta plataforma de Google ofrece un servicio de despliegue de aplicaciones además de un conjunto de servicios que ofrecen toda la funcionalidad requerida para el correcto funcionamiento de la aplicación. Sin embargo no ofrece servicios de *IaaS(Infrastructure as a Service)*, es decir no permite crear máquinas virtuales vacías donde instalar el software que se desee. En concreto ofrece servicios de *PaaS(Platform as a Service)*. Se está limitado a usar sus herramientas de desarrollo, además de que impone restricciones en ciertas operaciones, como por ejemplo prohíbe usar cierto tipo de consultas SQL.

- OpenShift

Similar a Google App Engine también ofrece un servicio de *PaaS*. Aunque tiene menos limitaciones, también tiene sus pegas, por ejemplo no permite el "Sharding" con *MongoDB* lo que lo descarta para este proyecto.

- Heroku

Similar a los anteriores, *PaaS* que en este caso ni soporta *MongoDB*.

- Windows Azure

Ofrece servicios tanto de *PaaS* como de *IaaS*. Permite la creación de máquinas virtuales así como el uso de paquetes con las herramientas más básicas de desarrollo y bases de datos para crear nuestra aplicación a la carta. Además ofrece todo tipo de servicios como herramientas de monitorización, escalado de máquinas, etc.

- Amazon Web Services

El servicio de *Cloud Computing* más usado en la actualidad y el que ofrece mayor número de servicios. Ofrece una amplia variedad de máquinas virtuales de todos los tamaños y optimizadas para diversos usos. Tiene también servicios de balanceo de carga, escalado automático, monitorización, gestión de almacenamiento en la nube y muchos más.

Como plataforma para el despliegue en este proyecto se ha optado por *AWS*. Sus servicios y capacidades son muy superiores a las de sus otros competidores. Solo *Azure* ofrece un rango

de servicios comparable, pero aún a mucha distancia en cuanto a variedad, facilidad de uso y madurez. De hecho, hoy en día **AWS** es la gran dominadora del este mercado con mucha diferencia sobre sus competidores.

5.2 Diseño de la arquitectura.

5.2.1 Capa de aplicación

Para el despliegue se ha diseñado la siguiente arquitectura a nivel de capa de aplicación:

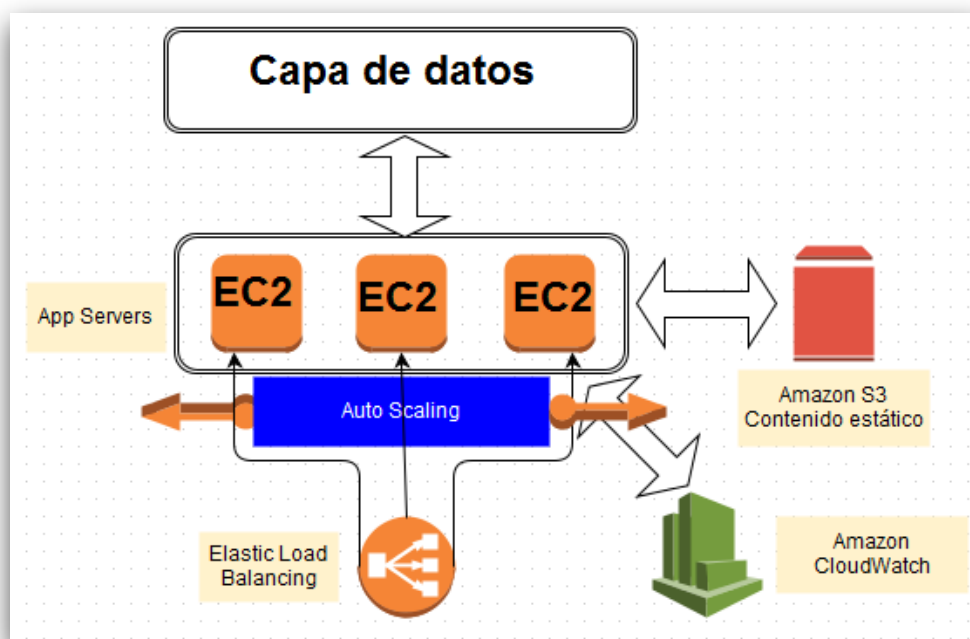


Figura 5.1 Diseño arquitectura

A continuación se describe cada componente:

- Amazon Elastic Compute Cloud (Amazon EC2) es un servicio web que proporciona la capacidad de crear y gestionar máquinas virtuales en la nube. En este caso se usa para albergar los servidores de aplicaciones de la aplicación web.
- Elastic Load Balancing.
Actúa como front-end de la aplicación y se ocupa de recibir las peticiones de los clientes y redirigirlas hacia los servidores distribuyendo la carga de trabajo de modo equilibrado.
- Amazon S3.

Es un almacén de datos escalable y de altas prestaciones donde se guardan los ficheros estáticos de la aplicación, de modo que estén disponibles para todos los servidores de aplicaciones. Los ficheros que se guardan en este almacén son los ficheros de hojas de estilos, javascript y las fotos de los eventos.

- Amazon CloudWatch.
Se encarga de monitorizar la carga de trabajo de los servidores de aplicaciones para poder adaptar los recursos del sistema a la carga de trabajo en cada momento. En la práctica, generará una serie de alarmas a partir de las cuales el servicio de AutoScaling añadirá o eliminará servidores según las necesidades.
- AutoScaling.
Permite escalar automáticamente la capacidad para aumentar o reducir el número de instancias EC2, de acuerdo con las condiciones que se defina. Trabaja en colaboración con Elastic Load Balancing y Amazon CloudWatch.

5.2.2 Capa de datos

El despliegue de *MongoDB* en sistemas en producción es una tarea compleja que requiere tener en cuenta aspectos como la gestión de replicas, las opciones de distribución, así como el hardware adecuado para obtener un óptimo funcionamiento. En la siguiente imagen se muestra un esquema habitual para un despliegue en producción de *MongoDB*:

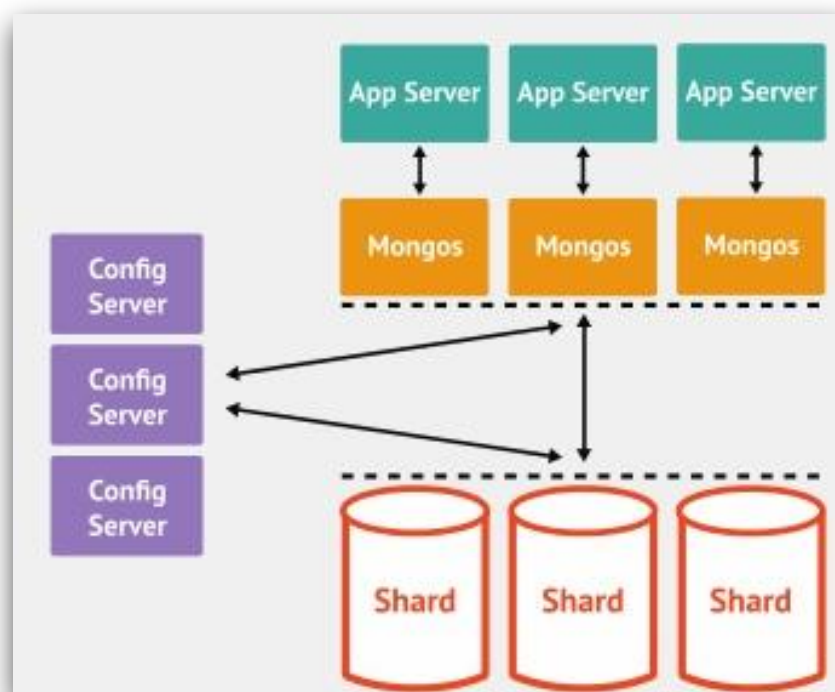


Figura 5.2 Despliegue MongoDB producción

A continuación se describe cada uno de los componentes:

- **Mongos.**
Son instancias de mongo que realizan las funciones de enrutamiento dentro de un sistema distribuido a través de varias particiones o "shards". Normalmente se alojan en los servidores de aplicaciones y reciben las peticiones de estos. Para procesar las peticiones usan la metainformación almacenada en los "Config Server" a partir de la cual averiguan en que partición están los datos deseados. A continuación solicitan dichos datos a la partición adecuada y se los pasan al servidor de aplicaciones que realizó la petición.
- **Config Server.**
Son instancias que almacenan la información correspondiente a la distribución de los datos, es decir en que partición o "shard" está cada dato. Si todos los "Config Servers" de un sistema se cayeran la base de datos quedaría inutilizable, por eso en entornos de producción es habitual que existan al menos tres de estos servidores.
- **Shard o partición.**
Son servidores *MongoDB* que albergan una partición de los datos del sistema. Este mecanismo es el usado por *MongoDB* para garantizar la escalabilidad. La distribución de los datos en una u otra partición depende una "Shard-Key" o clave de distribución, que determina en que servidor se guarda cada dato.

En entornos de producción es también habitual que cada una de las particiones o "shards" sea un "Replica Set". Con este término se denomina a un conjunto de servidores *MongoDB* que funcionan como un cluster, manteniendo todos ellos una copia de los datos de la réplica. De ese modo se garantiza la seguridad de los datos y su disponibilidad ante fallos hardware o de sistema.

Los "Replica Set" se componen de dos o más servidores, actuando uno de ellos como primario. Este recibirá por defecto todas las operaciones de lectura y escritura, mientras que los otros componentes actúan generalmente como meras copias de seguridad. En caso de que el servidor primario falle se activa el protocolo de sustitución mediante el cual los restantes componentes de la réplica eligen a un nuevo servidor primario de entre los secundarios existentes. En este caso puede darse el problema de que si el número de servidores es par es posible que la elección acabe en empate y que no se pueda elegir un nuevo servidor primario. Por eso, en caso de que el número de servidores sea par es necesario que exista en la réplica lo que se llama un "arbiter" o arbitro. Este es un servidor donde corre un proceso mongo ligero cuya única función es romper los empates en caso de igualdad en la elección de un nuevo servidor primario. A continuación se muestran esquemas habituales de "Replica Sets":

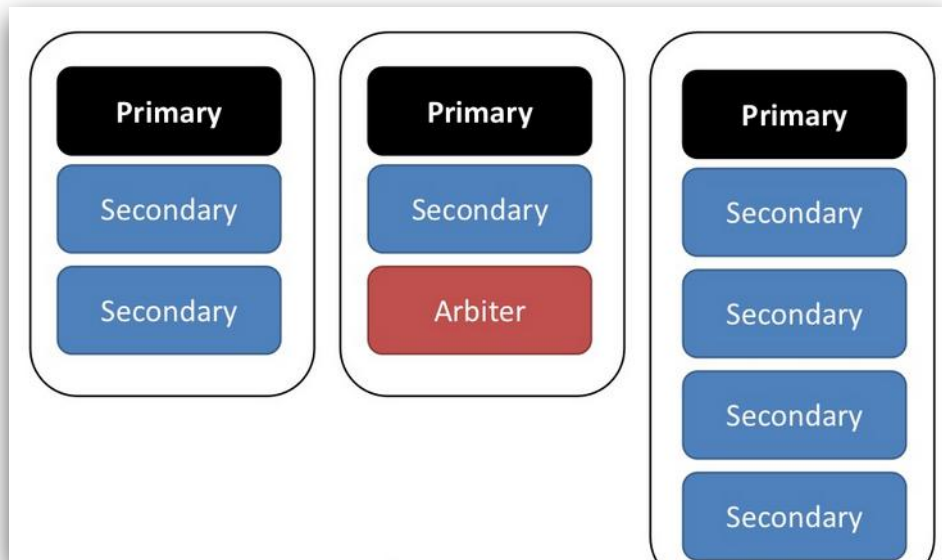


Figura 5.3 Replica Set

Hay que tener en cuenta no obstante, que en el caso de este proyecto al tratarse de un despliegue inicial sin grandes requerimientos en cuanto a volumen de datos o tráfico, no es necesario distribuir la base de datos inicialmente. De hecho, añadir una nueva partición es una tarea muy costosa para el sistema y el momento de realizar esta operación ha de ser cuidadosamente previsto para evitar en la medida de lo posible caídas del sistema. Por ello se ha optado por partir de un único "Replica Set" con tres servidores: uno primario y dos secundarios.

5.2.3 Arquitectura final

La arquitectura escogida para el despliegue inicial no incluye particiones ya que se supone que la cantidad de datos y de usuarios iniciales no será muy elevada, además de que hay que considerar también el aspecto económico. Los recursos cuestan dinero y por el bien de la hipotética empresa es preferible no usar recursos innecesariamente. Por tanto en el diseño elegido de la arquitectura la capa de datos se reduce a un "Replica Set".

Para la capa de aplicación se ha seguido el mismo principio y se ha optado por partir con dos servidores de aplicaciones. De este modo se obtiene una mínima tolerancia a fallos y disponibilidad. En efecto, si uno de los servidores fallará aún habría otro funcionando y con Amazon CloudWatch y AutoScaling programados para que al menos siempre haya dos servidores funcionando el sistema estaría razonablemente cubierto ante caídas inesperadas. En el peor de los casos el tiempo de recuperación sería de unos pocos minutos.

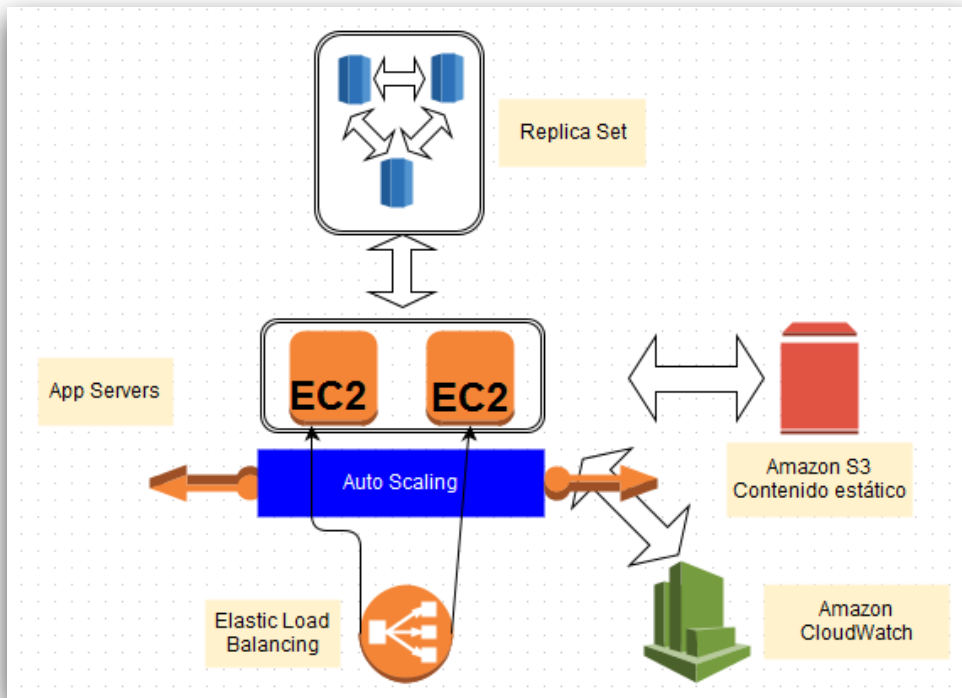
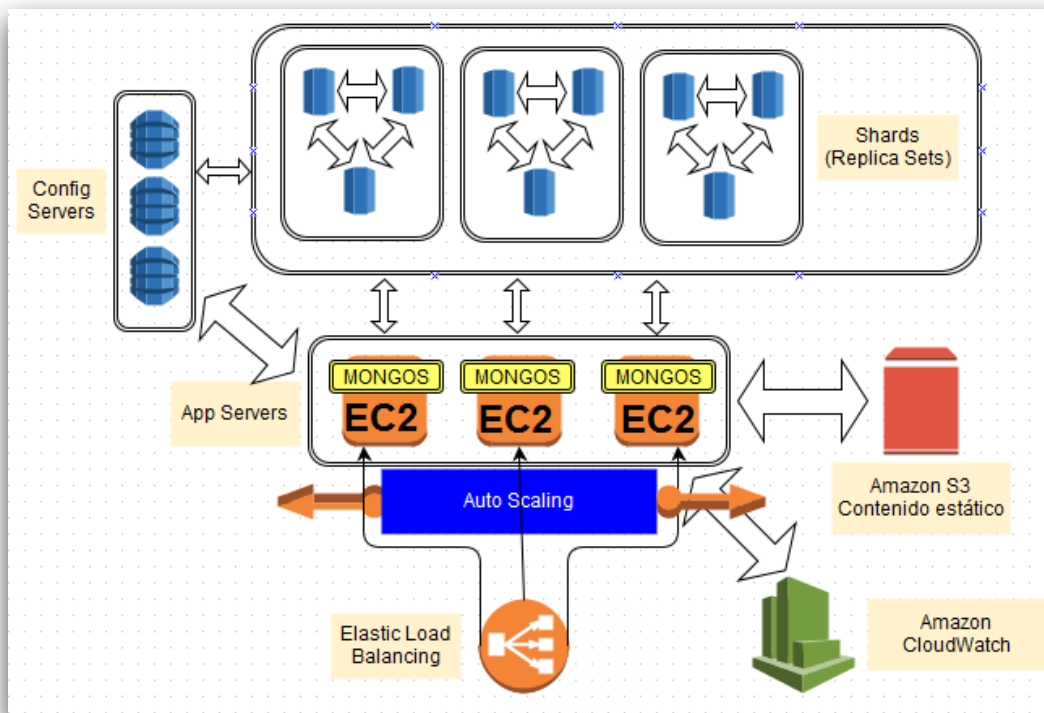


Figura 5.4 Despliegue final inicial

A continuación se presenta un diseño más habitual para sistemas más maduros y que ya han alcanzado un tamaño importante. En este la capa de datos ya incluye infraestructura para escalabilidad usando particiones o "Shards".



5.3 Adaptación de la aplicación a la plataforma elegida

En contraste con otras plataformas analizadas como por ejemplo Google App Engine, [AWS](#) no impone ningún tipo de restricción en cuanto a las estructuras, herramientas u operaciones que se incluyen en las aplicaciones a desplegar en su plataformas. Gracias a esto las modificaciones a la hora de desplegar la aplicación han sido mínimas. La mayoría se han solventado incluyendo estructuras condicionales en el fichero "settings.py", aplicando distintos parámetros de configuración dependiendo del valor de la variable "DEBUG".

El único punto que requirió modificar el código fue el relativo al almacenamiento de ficheros estáticos. Los ficheros estáticos de la aplicación(CSS, Javascript), así como las fotos asociadas con cada evento se almacenan en una estructura de directorios. Dicha estructura en modo desarrollo está dentro de la raíz de la aplicación, sin embargo en entorno de producción se sitúa en un almacén S3 de [AWS](#). Mientras en desarrollo se trabaja con el sistema de archivos local, en desarrollo debe hacerse con la API de [AWS](#), usando las claves de acceso proporcionadas. Esto requirió la instalación de la librería "Boto" que permite la gestión de los servicios de [AWS](#) a nivel de aplicación [Python](#).

Esta situación motivo modificaciones en "views.py" así como la inclusión de un nuevo fichero "storage.py" donde se almacenan los métodos para leer y escribir en ambos casos. Para usar uno u otro se usa el valor del parámetro de configuración "STORAGE", que definirá el nombre del método a usar en cada caso.

5.4 Estructura del servidor de aplicaciones

[Django](#) ofrece un entorno de desarrollo totalmente operativo con servidor web compatible con la especificación WSGI. Esta especificación define una interface ligera y simple entre servidores web y servidores de aplicaciones o frameworks. Sin embargo para desplegar la aplicación en un servidor en producción es necesario usar otras herramientas más potentes y profesionales. En este proyecto se ha optado por usar una de las arquitecturas más populares a la hora de trabajar con Django, la combinación del servidor web [Nginx](#) y del servidor WSGI [Gunicorn](#). La combinación de estas dos herramientas es cada vez más habitual en el mundo de las aplicaciones web y la conjunción de ambas aporta rendimiento y estabilidad a este tipo de proyectos. De este modo cada servidor de aplicaciones se compondrá de un servidor [Nginx](#) que actuará como front-end y que pasara las peticiones al servidor [Gunicorn](#) que es el interactúa con Django.

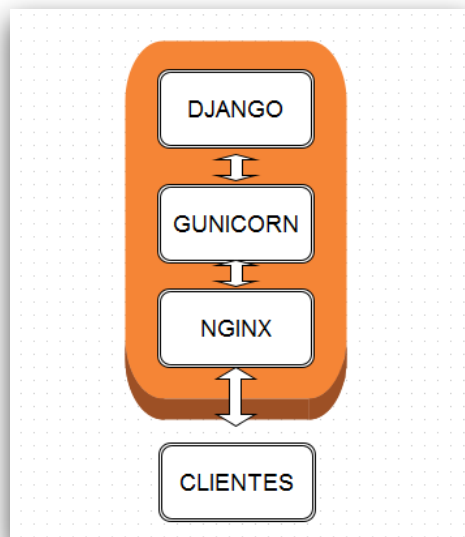


Figura 5.6 Esquema servidor aplicaciones

5.5 Consideraciones sobre el hardware

La elección del hardware a usar es también una tarea importante dentro del diseño, ya que es decisiva para garantizar el correcto rendimiento del sistema. Dado que esta elección también va a tener un impacto importante en el presupuesto del proyecto, el objetivo es encontrar un equilibrio entre rendimiento y coste, adaptando los recursos a las necesidades en cada momento evitando de este modo malgastarlos.

5.5.1 Capa de aplicación

La elección del hardware en la capa de aplicación no es crítica en cuanto a rendimiento ya que el uso de "AutoScaling" va a permitir que el sistema se adapte en todo momento a la carga de trabajo actual, independientemente del hardware escogido. La tarea está en buscar el mejor compromiso entre escalabilidad horizontal o vertical. Usando máquinas potentes reduciremos el número de instancias necesarias pero su coste unitario será superior, en cambio usando máquinas más livianas será necesario un número superior de instancias, pero con un menor coste unitario. En todo caso encontrar la configuración óptima requeriría un detallado estudio acompañado de pruebas del sistema en producción que escapan al ámbito de este proyecto.

En este caso se ha optado por partir de una configuración mínima siguiendo el principio de no malgastar recursos, además usando "AutoScaling" el sistema siempre va a estar cubierto ante cualquier variación de la carga de trabajo.

Dentro de la amplia oferta de Instancias EC2 que ofrece Amazon se ha optado por el tipo C3. Este tipo de máquinas cuyas características están indicadas para su uso como servidores web ofrecen una aceptable relación rendimiento/precio.

A continuación se incluye una tabla con los tipos y precios de este tipo de instancias:

Región: UE (Irlanda)					
	vCPU	ECU	Memoria (GiB)	Almacenamiento de instancias (GB)	Uso de Linux/UNIX
Optimizadas para informática – Generación actual					
c3.large	2	7	3.75	2 x 16 SSD	\$0.120 por hora
c3.xlarge	4	14	7.5	2 x 40 SSD	\$0.239 por hora
c3.2xlarge	8	28	15	2 x 80 SSD	\$0.478 por hora
c3.4xlarge	16	55	30	2 x 160 SSD	\$0.956 por hora
c3.8xlarge	32	108	60	2 x 320 SSD	\$1.912 por hora

Figura 5.7 Detalle precios instancias AWS C3

Para el despliegue inicial se ha optado por el tipo c3.large, el más modesto de la gama, siempre teniendo en cuenta que este tipo de sistemas requieren una continua monitorización para adaptar en todo momento los recursos a las necesidades del sistema.

5.5.2 Capa de datos

La elección del hardware de esta capa es una operación más delicada, ya que no existe la cobertura que nos proporciona el "AutoScaling" en la capa de aplicación. Ampliar el número de servidores en esta capa para adaptarlo a la carga del sistema no es una operación sencilla, ya que conlleva la configuración de replicas y particiones. Este tipo de operaciones son muy costosas para el sistema y el tiempo y recursos necesarios para alcanzar un estado estable son elevados. Por ello es de vital importancia realizar un diseño inicial acertado para limitar en la medida de lo posible este tipo de operaciones. Por otra parte el despliegue de un sistema *MongoDB* en producción comprende distintos tipos de máquinas("Config Servers", "Arbitrer", Servidores Mongo) y por consiguiente distintas configuraciones hardware.

En este caso, para los servidores de datos además de escoger una instancia adecuada es imprescindible el uso de Amazon Elastic Block Store(EBS). Este tipo de almacenamiento se replica automáticamente dentro de una zona de disponibilidad para protegerle frente a los fallos de componentes, ofreciéndole una alta disponibilidad y durabilidad. Los volúmenes de Amazon EBS ofrecen rendimiento constante y de baja latencia y son especialmente adecuados para trabajar con bases de datos. Además funcionan como discos externos por lo que en cada momento se pueden conectar a diferentes máquinas. Amazon ofrece un tipo especial de EBS llamado Amazon EBS Provisioned IOPS. Este tipo de volúmenes están especialmente indicados para trabajar con aplicaciones que requieran una tasa alta y consistente de operaciones de entrada y salida por segundo. Están especialmente indicados para trabajar con bases de datos como *MongoDB*.

En cuanto a los tipos de instancias EC2 a usar se han tomado las siguientes decisiones.

- Inicialmente se va a partir de un replica set con un servidor primario y dos secundarios. Amazon ofrece un tipo de instancias llamadas I2 optimizadas para almacenamiento y recomendadas para su utilización con *MongoDB* y otras soluciones *NoSQL* como *Cassandra*. A continuación se incluye una tabla detallada con las opciones y precios de este tipo de instancia.

Región: UE (Irlanda)

	vCPU	ECU	Memoria (GiB)	Almacenamiento de instancias (GB)	Uso de Linux/UNIX
Optimizadas para almacenamiento – Generación actual					
i2.xlarge	4	14	30.5	1 x 800 SSD	\$0.938 por hora
i2.2xlarge	8	27	61	2 x 800 SSD	\$1.876 por hora
i2.4xlarge	16	53	122	4 x 800 SSD	\$3.751 por hora
i2.8xlarge	32	104	244	8 x 800 SSD	\$7.502 por hora
hs1.8xlarge	16	35	117	24 x 2048	\$4.900 por hora

Figura 5.8 Detalle precios instancias AWS I2

Para el despliegue inicial se ha optado por usar instancias xlarge siguiendo la política de partir con los mínimos recursos. En todo caso usando EBS la opción de incrementar la capacidad de las instancias es sencilla y rápida.

- Para futuras ampliaciones que incluyan el uso de "Sharding" y por tanto de "Config Servers" se mantendrían las mismas características para todas las particiones. Es decir, un "Replica Set" de tres máquinas con instancias i2.xlarge. Para los "Config Servers" que serían tres en todo caso no son necesarios grandes recursos por tanto se optaría por la opción más económica: el uso de microinstancias.

Región: UE (Irlanda)

	vCPU	ECU	Memoria (GiB)	Almacenamiento de instancias (GB)	Uso de Linux/UNIX
Micro and Small Instances					
t1.micro	1	Variable	0.615	Solo EBS	\$0.020 por hora
m1.small	1	1	1.7	1 x 160	\$0.047 por hora

Figura 5.9 Detalle precios instancias AWS Micro

5.6 Consideraciones sobre la localización geográfica de los servidores

La infraestructura de [AWS](#) está dividida en regiones y dentro de estas en zonas de disponibilidad. Las regiones son localizaciones geográficamente aisladas. [AWS](#) ofrece ocho regiones geográficas distintas donde alojar recursos:

Code	Name
ap-northeast-1	Asia Pacific (Tokyo) Region
ap-southeast-1	Asia Pacific (Singapore) Region
ap-southeast-2	Asia Pacific (Sydney) Region
eu-west-1	EU (Ireland) Region
sa-east-1	South America (Sao Paulo) Region
us-east-1	US East (Northern Virginia) Region
us-west-1	US West (Northern California) Region
us-west-2	US West (Oregon) Region

Figura 5.10 Regiones AWS

Estas regiones al estar en distintas localizaciones geográficas ofrecen soporte para tolerancia a fallos y alta disponibilidad, ya que distribuyendo la infraestructura en distintas regiones se previenen las caídas totales del sistema, ya que aunque una región sufra problemas las otras seguirían funcionando y dando servicio. Además también ofrece beneficios como mejorar los tiempos de respuesta acercando los servidores a la localización geográfica de los clientes. En efecto, dependiendo de la distribución geográfica de los clientes se pueden ubicar más servidores en las regiones con más tráfico, mejorando de este modo las prestaciones del sistema. Sin embargo, un problema derivado del uso de esta característica puede ser el aumento de los tiempos de respuesta del tráfico interno del sistema. Por ejemplo, un servidor de aplicaciones en la zona asiática comunicándose con un servidor de base de datos en la zona europea puede no ser un escenario muy eficiente. Una posible solución para este problema manteniendo las ventajas de la tolerancia a fallos y la disponibilidad sería usar las zonas de disponibilidad. Estas albergan centros de datos que a pesar de estar en la misma región están aislados unos de otros. De ese modo se mantiene una cierta tolerancia a fallos y al mismo tiempo al estar en la misma región los tiempos de respuesta no son muy elevados.

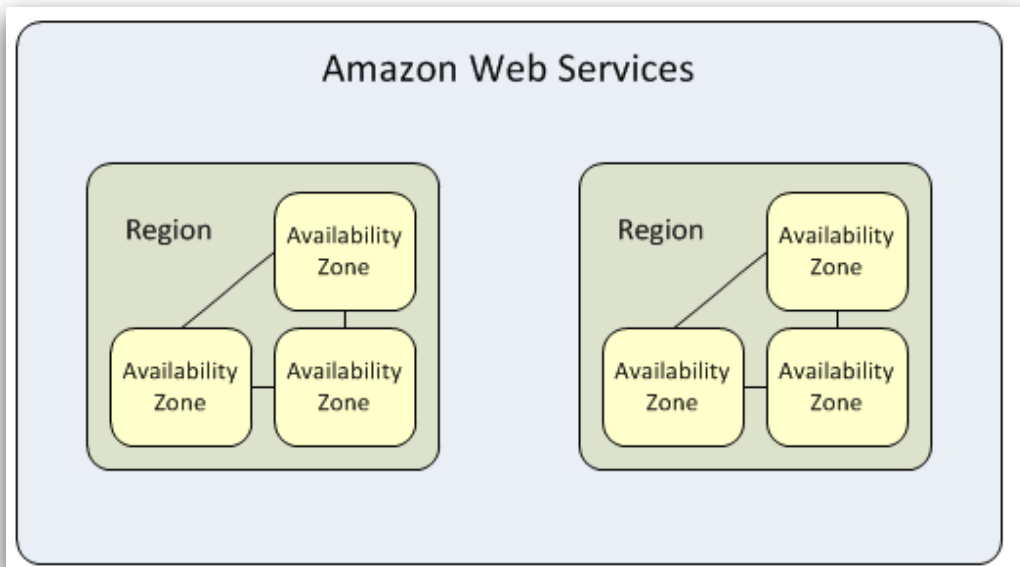


Figura 5.11 AWS Regiones y zonas

Para este proyecto se ha optado en primer lugar por usar la región europea, ya que al ser una aplicación realizada en España, se supone que inicialmente la mayoría de los usuarios van a pertenecer a esta región.

Para la capa de aplicación al constar inicialmente de dos servidores se ha optado por alojarlos en distintas zonas de disponibilidad, para de ese modo una posible caída de una de las zonas este cubierta.

Para la capa de datos que inicialmente va a contar con un "Replica Set" se usará el siguiente esquema:

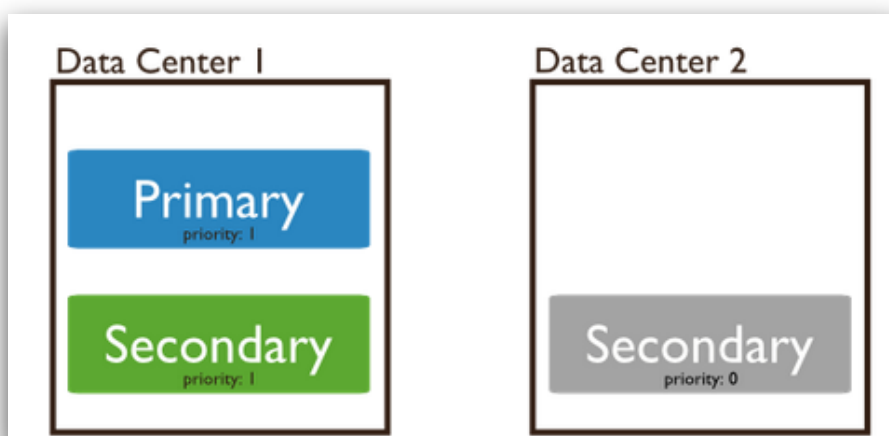


Figura 5.12 Despliegue Replica Set

De este modo las copias de los datos están distribuidos en dos localizaciones distintas. El primer centro de datos es el que realmente usa el sistema, el segundo cumple solo funciones de copia de seguridad y de respuesta ante fallo del primero. En efecto, en caso de caída del primer centro de datos el segundo pasaría a dar servicio al sistema mientras se restablece el primero.

Un problema añadido cuando usan estos métodos es la posibilidad de que ambos centros de datos se queden aislados ante un fallo de red. En ese caso el primer centro de datos cuenta una servidor secundario que en caso de fallo del primario pasaría a dar servicio. Debido a esto el servidor del segundo centro de datos está configurado como "Priority 0", esto significa que no puede ser primario, ya que en ese caso un fallo de red y una caída del segundo centro de datos provocaría la caída de todo el sistema.

En el caso de usar particiones, se seguiría el mismo esquemas para cada una de ellas.

Hay que recordar que en todo caso esto no sustituye ni elimina la necesidad de contar con un buen sistema de copias de seguridad.

5.7 Elección de la shard key

La "Shard Key" es la clave que determina la distribución de los datos a lo largo de las distintas particiones o "Shards". Una correcta elección es básica para obtener un buen rendimiento del sistema. Aunque como en muchos aspectos en el mundo informático va a haber objetivos contrapuestos. A la hora de escoger una "Shard Key" se deben tener en cuenta los siguientes puntos:

- La "Shard Key" ha de distribuir uniformemente los datos entre las distintas particiones. De ese modo se evitan diferencias de tamaño muy grandes entre los distintos servidores con los problemas que esto puede conllevar.
- La "Shard Key" debería ser capaz de distribuir los datos de manera que se minimice el número de servidores implicados en una determinada consulta. En efecto, las consultas serán más rápidas cuanto menos servidores estén implicados.
- En oposición al punto siguiente si el objetivo es optimizar las escrituras a base de datos, sería deseable que dichas operaciones se distribuyeran entre distintos servidores para favorecer el paralelismo de estos procesos.

En el caso de este proyecto, se supone que las consultas van a ser más numerosas que las inserciones, por ello lo ideal sería tener una distribución donde los datos que van a ser buscados en la misma consulta estén juntos. El campo de búsqueda principal en la aplicación web es la posición geográfica, es un campo requerido en todas las consultas y es el que figura en primer lugar en los índices de la base de datos. Este sería entonces el campo más adecuado para usarse como "Shard Key", por desgracia las [MongoDB](#) impone la limitación de que los índices geoespaciales no pueden usarse como "Shard Key". Ante esta situación, [MongoDB](#)

ofrece una alternativa aceptable que es usar un "Hash Index". Este es un tipo de índice que genera un hash del campo elegido, de este modo se garantiza una distribución uniforme y equilibrada de los datos independientemente de su valor. Para el proyecto se usará como "Shard Key" un índice "Hash Index" del campo "_id" de la colección "Events". Este tipo de campos se comportan muy bien con este tipo de índices, ya que se incrementan uniformemente y la aplicación de la función hash garantiza una distribución equilibrada.

5.8 Tareas de despliegue.

Como en todos los proyectos de software, en este son necesarias ciertas modificaciones para desplegar las aplicaciones en un entorno de producción. Esto incluye tareas como adecuar los archivos de configuración, la estructura de archivos, pasar a usar servidores o bases de datos más adecuadas, eliminar información de desarrollo que no debería subirse a producción, instalar software necesario en el servidor, etc. A continuación se describen los pasos y operaciones necesarios para la puesta en producción del sistema diseñado.

5.8.1 Despliegue de la capa de aplicación

- Crear maquina virtual.
Para este proyecto se usarán maquinas Ubuntu Server 14.04 de 64 bits.
- Configurar el cortafuegos de [AWS](#).
Hay que abrir el puerto 80 para poder recibir tráfico web.
- Instalar el software necesario.
Eso incluye los siguientes paquetes: [Nginx](#), [Gunicorn](#), [Git](#), [Virtualenv](#) y [Pip](#).
- Subir al servidor los ficheros necesarios.
Aprovechando que en el proyecto se ha usado [Git](#) como herramienta de gestión de versiones, se puede aprovechar para clonar en el servidor el repositorio con los archivos del proyecto.
- Instalar los paquetes [Python](#) requeridos en el proyecto.
Para ello usamos el fichero requirements.txt incluido en la aplicación.
- Crear la estructura de ficheros estáticos en Amazon S3.
Esto incluye crear la misma estructura de carpetas que existe en el proyecto y subir los ficheros a este almacenamiento de [AWS](#).
- Modificar el fichero de configuración de [Django](#) para adaptarlo al nuevo entorno.
- Configurar [Nginx](#) para que pase las peticiones a [Gunicorn](#).

- Iniciar *Nginx* y *Gunicorn* y comprobar que todo funciona correctamente.
- Configurar la máquina para que tanto *Nginx* como *Gunicorn* se ejecuten al inicio.
- Crear una imagen de la máquina para poder usarla posteriormente al añadir nuevos servidores.
- Crear una nueva instancia a partir de la imagen creada anteriormente.
- Crear un "Load Balancer" e incluir los dos servidores creados.
- Configurar "AutoScaling" para que gestione el número de servidores automáticamente y adapte el tamaño del sistema a la carga de trabajo existente en cada momento.
- En caso de que la capa de datos use particiones habría que instalar *MongoDB* en cada servidor de aplicaciones para poder ejecutar los procesos "mongos" que posibilitarían acceder a la base de datos.

5.8.2 Despliegue de la capa de datos

El despliegue de la capa de datos incluye la creación de las máquinas así como la configuración de la estructura elegida que puede incluir "Replica Sets" y "Sharding". En el caso de la configuración inicial propuesta, esta capa se va a reducir a un "Replica Set" formado por tres servidores. El despliegue de esta infraestructura con la configuración propuesta comprende las siguientes tareas:

- Creación de una instancia EC2 xlarge y con un volumen EBS provisioned IOPS optimizado para bases de datos
- Instalación e inicio de mongod en esa instancia con la opción de "Replica Set".
- Crear y añadir datos iniciales a la colección "category"
- Configurar la máquina para que mongod se ejecute al inicio.
- Crear una imagen de la máquina y a partir de esta crear otras dos instancias.
- Configurar la "Replica Set".

En el caso de que se opte por añadir "Sharding" habría que realizar los siguientes pasos:

- Creación de una instancia EC2 de tipo microinstance.
- Instalación e inicio de mongod en esa instancia con la opción "configsvr", ya que va a funcionar como "Config Server".
- Configurar la máquina para que mongod se ejecute al inicio.
- Crear una imagen de la máquina y a partir de esta crear otras dos instancias. Estas tres instancias serán los "Config Servers".
- Instalar y arrancar mongos en cada uno de los servidores de aplicaciones pasando como parámetros los tres "Config Servers".
- Conectarse a uno de los procesos mongos y añadir la "Replica Set" ya creada como "Shard".
- Para añadir más particiones habría que crear primero otro "Replica Set" y realizar la misma operación.
- Para que el sistema empiece a distribuir hay que conectarse de nuevo a un proceso mongos y habilitar el "Sharding" en la base de datos elegida. A continuación hay que habilitar el "Sharding" para la colección o colecciones que hayamos elegido indicando al mismo tiempo la "Shard Key" que se va a usar. En este caso se va a particionar la colección Events y como "Shard Key" se usará un "Hash Index " del campo "_id".

5.9 Pruebas de rendimiento.

En un proyecto de esta naturaleza donde se intenta desarrollar un sistema preparado para adaptarse a diferentes cargas de trabajo este tipo de pruebas son sin duda las más importantes. No obstante para realizar estas pruebas de modo correcto sería necesario contar con un sistema desplegado en un entorno de producción, y por desgracia por motivos económicos no es posible disponer de esta infraestructura de pruebas.

Teniendo en cuenta estas limitaciones se ha realizado una batería de pruebas sobre varios despliegues más modestos en [AWS](#).

5.9.1 Volumen de datos.

La mayor parte del éxito de este desarrollo se apoya sobre las supuestas prestaciones de [MongoDB](#). Para estas pruebas se han desplegado un servidor de aplicaciones y un servidor [MongoDB](#) en la nube de [AWS](#), y usando el programa "ab" se han realizado ensayos con distintas cantidades de datos almacenados en [MongoDB](#).

Los resultados obtenidos muestran los tiempos de respuesta promedio para distintas cantidades de eventos almacenados en la base de datos, con distinto número de peticiones simultaneas a la página inicial. En este página se carga una lista con los primeros eventos ordenados por fecha de inserción. Esta consulta está soportado por un índice en el campo fecha. Se puede ver que los tiempos de respuesta del sistema no varían a pesar del incremento del número de datos. Se ha pasado de 5.000 eventos a 5.000.000 sin prácticamente alteración de los tiempos de respuesta.

Peticiones	Simultaneas	Nº Eventos	Test1	Test2	Test3	Promedio
1000	5	500	368	398	390	385,33
1000	10	500	787	653	665	701,67
1000	5	5000	338	351	379	356,00
1000	10	5000	669	685	701	685,00
1000	5	50000	354	366	389	369,67
1000	10	50000	632	649	663	648,00
1000	5	500000	332	401	329	354,00
1000	10	500000	673	623	693	663,00
1000	5	5000000	386	375	319	360,00
1000	10	5000000	688	713	662	687,67

Figura 5.13 Datos rendimiento por volumen de datos

5.9.2 Balanceador.

En estas pruebas se ha comparado el rendimiento del sistema trabajando con un solo servidor como en el punto anterior, en contraposición con un despliegue con dos servidores y un balanceador.

En primer lugar se ha añadido un balanceador para que funcione como font-end, pero se ha mantenido solo un servidor. Aprovechando los datos del punto anterior se ha realizado una comparativa para ver la influencia del uso del balanceador en los tiempos de respuesta.

Se puede ver que la influencia del balanceador es inapreciable.

SIN BALANCEADOR						
Peticiones	Simultaneas	Nº Eventos	Test1	Test2	Test3	Promedio
1000	5	5000000	386	375	319	360,00
1000	10	5000000	688	713	662	687,67

CON BALANCEADOR						
Peticiones	Simultaneas	Nº Eventos	Test1	Test2	Test3	Promedio
1000	5	5000000	371	350	367	362,67
1000	10	5000000	691	625	630	648,67

Figura 5.14 Datos rendimiento balanceador

A continuación se ha añadido otro servidor de modo que el balanceador distribuya las peticiones entre ambos y se ha realizado otra batería de pruebas variando el número de peticiones simultaneas. Los resultados han sido los esperados y se ve que cuando aumenta el número de peticiones simultaneas el rendimiento mejora con el trabajo del balanceador.

UN SERVIDOR						
Peticiones	Simultaneas	Nº Eventos	Test1	Test2	Test3	Promedio
1000	5	5000000	371	350	377	366,00
1000	10	5000000	691	625	630	648,67
1000	100	5000000	5944	5820	6363	6042,33

DOS SERVIDORES						
Peticiones	Simultaneas	Nº Eventos	Test1	Test2	Test3	Promedio
1000	5	5000000	346	361	367	358,00
1000	10	5000000	347	349	382	359,33
1000	100	5000000	3375	3282	3174	3277,00

Figura 5.15 Datos rendimiento por número servidores

5.9.3 Conclusiones.

Las pruebas realizadas han confirmado la capacidad del sistema para adaptarse a cambios ante aumentos de carga de trabajo. La posibilidad de añadir nuevos servidores automáticamente listos para dar servicio sin ningún otro tipo de modificación y la constatación del correcto funcionamiento del sistema gestionado por el balanceador mejorando los tiempos de respuesta, confirma la validez del diseño. Además se ha comprobado el buen funcionamiento de *MongoDB* independientemente de la cantidad de datos. Faltaría sin embargo realizar pruebas para comprobar el comportamiento del sistema ante una capa de datos con distinto número de particiones. Por desgracia, los requerimientos de infraestructura para pruebas de ese tipo está fuera del alcance del proyecto.

5.10 Plan de crecimiento.

Este plan es una pequeña guía para el hipotético cliente de este proyecto de modo que en un futuro tenga unas pautas para intentar optimizar y adaptar el sistema ante aumentos de tamaño. Aunque el sistema está preparado para adaptarse a cambios y en los manuales se indican los procedimientos para ampliar la capacidad de la capa de datos, hay muchas más opciones que se pueden usar para mejorar el rendimiento sin apostar todo a añadir recursos.

5.10.1 Capa de aplicación

El diseño realizado permite que en esta capa no sean necesarios grandes cambios ante un hipotético aumento del volumen de la carga de trabajo. El hecho de guardar la sesión en base de datos hace que las peticiones se puedan enviar indistintamente a cualquier servidor de esta capa. El uso del servicio de "AutoScaling" de *Django* hace que esta capa se adapte automáticamente a cualquier cambio en el tráfico. De este modo el número de servidores podría crecer indefinidamente siempre que el resto del sistema, en especial la capa de datos, se pueda adaptar también a esta situación. El único problema que se podría plantear en este caso sería que el balanceador de *AWS* no pudiera dar respuesta al aumento de tráfico. Sin embargo, el servicio ELB(Elastic Load Balancer) de *AWS* es escalable y se adapta automáticamente a la demanda de servicio en cada momento.

Sería interesante una vez el sistema esté en producción realizar unas pruebas para intentar buscar la mejor relación entre precio y prestaciones para las instancias EC2. En definitiva se trata de encontrar el perfecto equilibrio entre escalabilidad horizontal y vertical, decidir si es mejor muchas máquinas pequeñas o unas pocas más potentes. Eso requeriría realizar pruebas con distintas configuraciones de hardware intentan buscar la configuración óptima.

También es interesante realizar un análisis de prestaciones de la red interna, es decir medir tiempos de respuesta entre las distintas máquinas del sistema, para poder detectar cuellos de botella que puedan estar afectando negativamente al rendimiento.

5.10.2 Capa de datos.

En esta capa la adaptación al aumento de tráfico se deja a cargo de las capacidades en ese aspecto de *MongoDB*. De hecho esta base de datos fue creada para resolver este tipo de problemas para los que las bases de datos relacionales no ofrecían rendimientos óptimos. La respuesta de *MongoDB* a este problema es la distribución o partición de los datos a través de varios servidores. Al contrario que en la capa de aplicación donde el añadido o la eliminación de servidores está automatizado usando los servicios de Amazon, en esta capa no se sigue ese método. El añadir una nueva partición a un sistema *MongoDB* es una tarea muy delicada que debe hacerse bajo una estricta supervisión. Esta operación implica que los datos existentes en el sistema se distribuyan uniformemente, lo que conlleva que el resto de las particiones cedan datos a la nueva partición. Este proceso sobrecarga mucho el sistema reduciendo sus prestaciones durante un periodo importante de tiempo. De hecho en el manual de *MongoDB* se advierte de que no se realice esta operación durante un periodo de mucha carga de trabajo del sistema. Por tanto, la decisión de cuando ampliar el sistema ha de ser meditada en profundidad.

En la arquitectura inicial planteada en este proyecto no se incluyen particiones, principalmente por motivos económicos ya que la inclusión de una partición conllevaría la necesidad de usar seis nuevas máquinas, tres para los "Config Server" y otras tres para el nuevo "Replica Set". Esta situación en el caso de un despliegue inicial no parece muy necesaria, sin embargo si el presupuesto lo permite iniciar el sistema con dos particiones no sería mala opción pensando en problemas futuros.

En todo caso tampoco la partición es la única solución para mejorar el rendimiento. Antes de llegar a esta opción se pueden hacer otras optimizaciones. Por ejemplo, una de las características de las bases de datos *NoSQL* es que no existen relaciones. Aprovechando esa característica, una posible optimización sería distribuir las colecciones entre distintos servidores, ya que al ser colecciones aisladas no se pierde ninguna capacidad por separarlas. Por ejemplo en el caso de este proyecto dado que la colección que va requerir mayores prestaciones es la que almacena los eventos, parece una buena medida dedicarle una base de datos a ella sola.

Otra posible optimización sería usar una sola base de datos para almacenar las sesiones. En efecto las sesiones se están usando continuamente en el proyecto, incluso aunque los usuarios no estén logueados, y todo ese tráfico se dirige a la base de datos. De hecho en arquitecturas actuales es bastante habitual dedicar una base de datos solo para trabajar con las sesiones. Este es un trabajo que habitualmente realizaban los sistemas de cache, pero actualmente ese tipo de trabajo está siendo realizado cada vez en mayor medida por los sistemas *NoSQL*.

Con este planteamiento se podría incluso considerar una optimización de hardware, ya que ambos sistemas no requerirían las mismas prestaciones. En efecto, la colección "Event"

necesitará una capacidad de almacenamiento muy alto, por contra la colección "Session" no tendrá grandes requerimientos de almacenamiento pero si necesitará un tiempo de respuesta muy elevado.

A continuación se presenta un posible esquema futuro con estas modificaciones incluyendo dos particiones para la colección "Event". Las colecciones "Category" y "User" dado su escaso tamaño se ha optado por incluirlas con la "Session". Con este planteamiento se puede ir analizando de manera separada la demanda en cada una de las bases de datos y optimizando independientemente cada una de ellas.

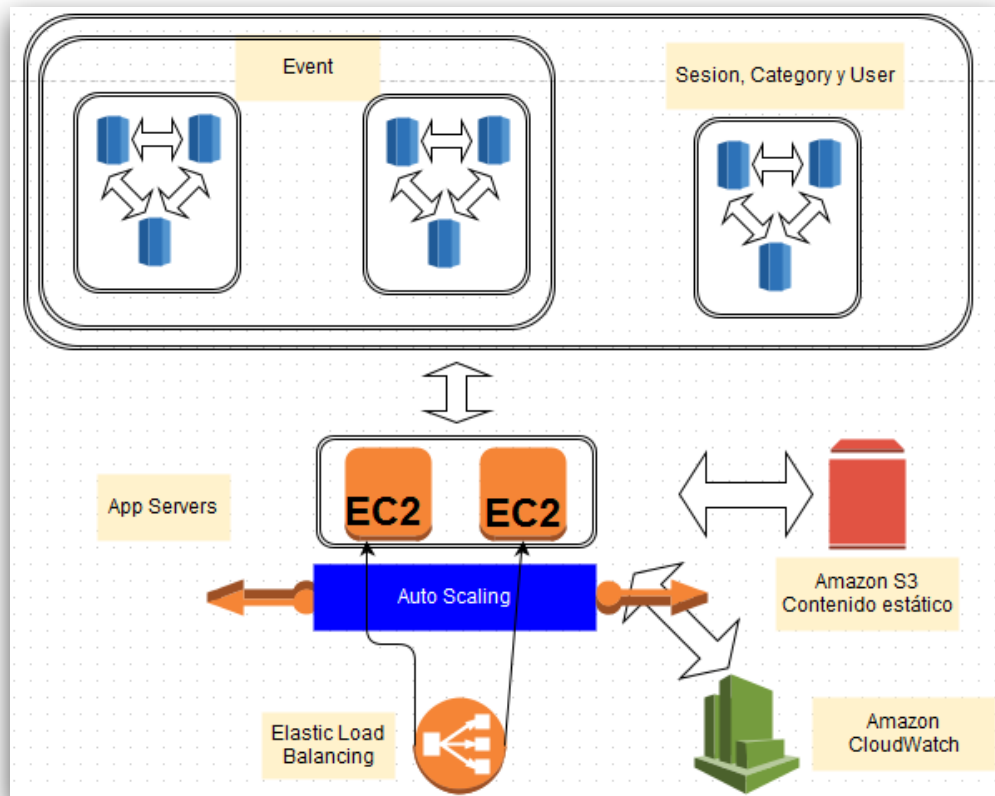


Figura 5.16 Esquema arquitectura evolucionada

Capítulo 6. Manuales del Sistema

6.1 Manual de despliegue

Este manual comprende las tareas necesarias para desplegar la aplicación y crear la infraestructura necesaria en AWS. Generalmente para estas tareas son de mucha utilidad herramientas como *Fabric*, que posibilitan realizar conexiones a máquinas remotas y ejecutar tareas en éstas. Sin embargo, en este caso las tareas de este manual incluyen tanto interactuar con máquinas virtuales linux a través de terminal, como usar el entorno web de *AWS* por ello se ha descartado la creación de scripts de ese tipo y se ha optado por realizar un manual detallado con los pasos a seguir.

6.1.1 Crear instancia EC2 para base de datos en AWS.

En esta tarea se va a crear una máquina virtual para que realice funciones de servidor de aplicaciones. Para ello debemos estar logueados en *AWS*.

- Accedemos a la consola de *AWS*(AWS Management Console).
- Seleccionamos la opción EC2 dentro de la lista de servicios.
- Hacemos click en el botón "Launch Instance" para crear una nueva instancia.
- En el paso 1 escogemos una máquina Ubuntu Server 14.04 de 64 bits.
- En el paso 2 escogemos una máquina i2.xlarge de tipo Storage Optimized.
- En el paso 3 dejamos las opciones por defecto.
- En el paso 4 seleccionamos la opción de almacenamiento "Provisioned IOPS".

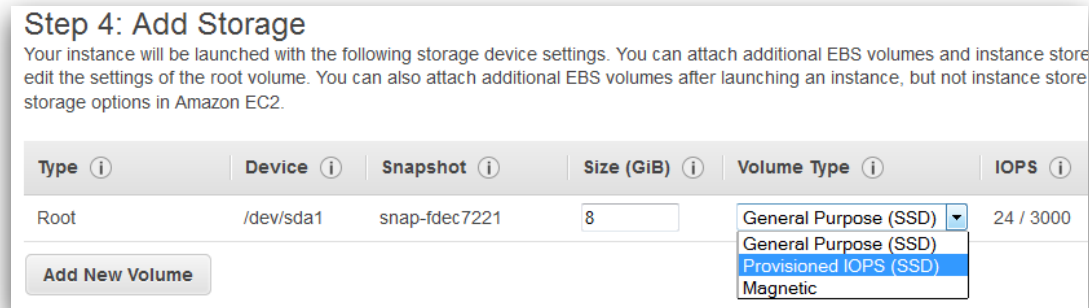


Figura 6.1 Creación instancia AWS - Add Storage

- En la paso 5 añadimos una etiqueta que describa su función, por ejemplo "MongoServer".
- En el paso 6 añadimos una nueva regla al grupo de seguridad para que se permita el tráfico entrante en el puerto 27017.
- En el paso 7 confirmamos los cambios y lanzamos la instancia.
- A continuación nos dará la opción de crear las claves de acceso para SSH y descargarnos un fichero con ellas.
- En la página de Instancias comprobamos que después de unos pocos minutos ya está funcionando.

6.1.2 Instalar y configurar MongoDB en la instancia.

- La instalación de MongoDB se describe detalladamente en el siguiente enlace de la página oficial de la herramienta:

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/>

- Dado que se va a crear un "Replica Set" se debe iniciar mongod con la opción "--replSET".

```
mongod --replSET "rep0"
```

- A continuación creamos la base de datos

```
use worldevents
```

- Y los índices necesarios

```
db.event.ensureIndex({added_date: -1})
db.event.ensureIndex({location:1,title:1,category:1})
```

- Hay que precargar las categorías en la colección "category". Esto se puede hacer de varias maneras, a mano, elaborando un script a partir de una lista o como en este caso usando las herramientas "mongodump" y "mongorestore" que facilita mongo. Con la orden:

```
mongodump --collection category --db worldevents
```

se exportan los datos de la colección "category" a un fichero. A continuación en la máquina destino se recuperan mediante la orden:

```
mongorestore --collection category --db worldevents fichero
```

- Incluimos en el fichero "/etc/rc.local" la orden `mongod --replSET "rep0"` para que mongod se ejecute siempre al arrancar la máquina.

6.1.3 Crear una imagen base para añadir nuevos servidores.

En la consola de AWS seleccionamos la opción "Create Image" dentro de las opciones de la instancia:

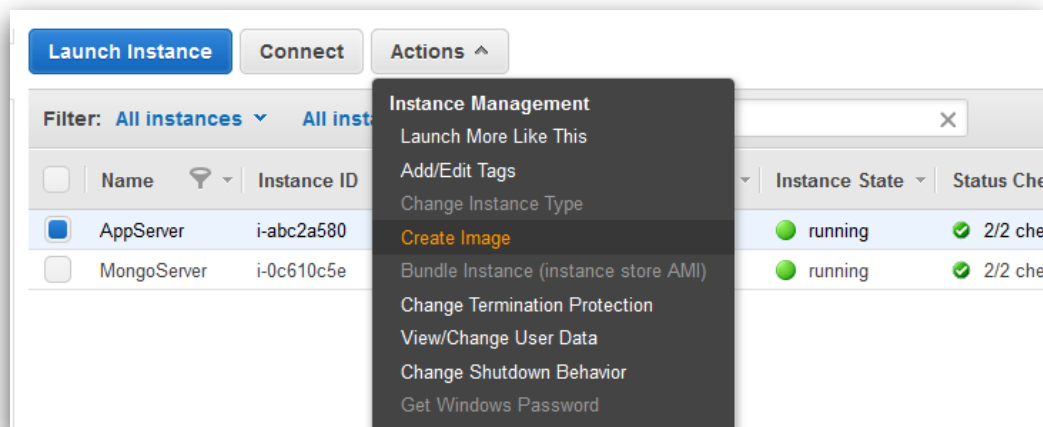


Figura 6.2 Crear imagen instancia AWS

En la siguiente pantalla le damos un nombre descriptivo y creamos la imagen. A continuación comprobamos que se ha creado correctamente accediendo a la opción "AMIs" en el submenú "IMAGES":

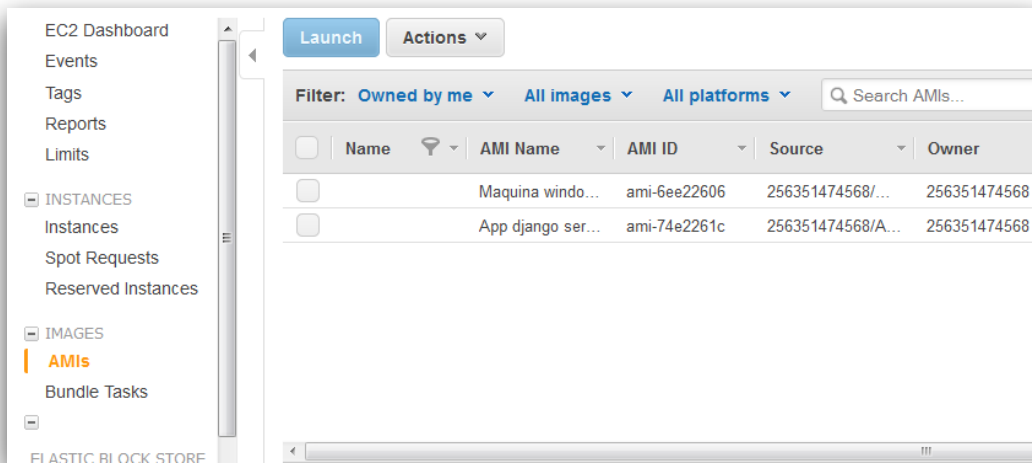


Figura 6.3 Submenú AMIs AWS

6.1.4 Crear dos nuevas instancias a partir de la imagen.

De este modo tendremos los componentes del "Replica Set". El proceso es similar al realizado para crear la primera máquina, simplemente hay que seleccionar como base la imagen creada, para ello en el primer paso del proceso se seleccionan las imágenes del usuario con la opción "My AMIs" .

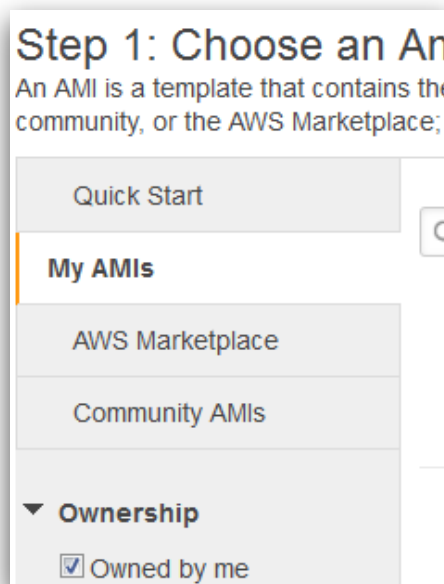


Figura 6.4 Opción My AMIs AWS

Otro aspecto a tener en cuenta es que para seguir el diseño planteado una de las máquinas ha de estar en una zona de disponibilidad distinta, para garantizar la disponibilidad del sistema en caso de que alguna zona se caiga.

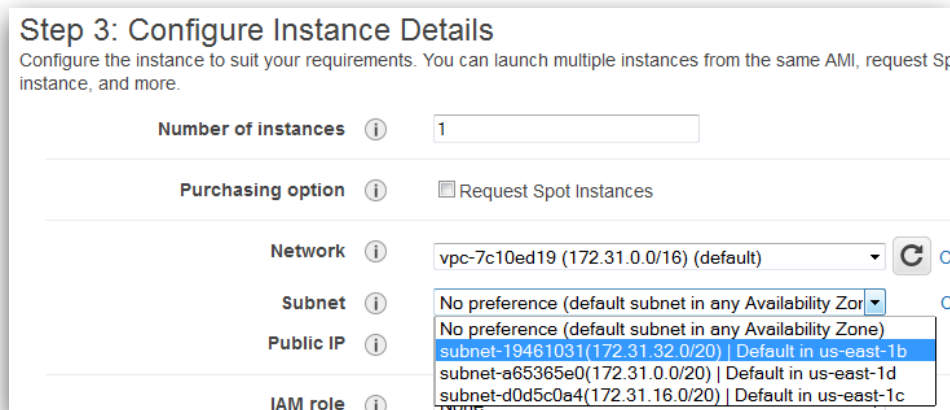


Figura 6.5 Selección Subnet AWS

6.1.5 Configurar el "Replica Set".

- En uno cualquiera de los servidores ejecutamos "mongo" para conectarnos a la base de datos.
- Iniciamos el "Replica Set" mediante la orden:

```
rs.initiate()
```

- Añadimos los otros dos miembros mediante la orden:

```
rs.add("nombre DNS del servidor")
```

- Mediante la orden "rs.status()" podemos comprobar el estado del "Replica Set".

6.1.6 Crear la estructura de ficheros estáticos en AWS S3.

Esto incluye crear la misma estructura de carpetas que existe en el proyecto y subir los ficheros a este almacenamiento de AWS. Para ello:

- Accedemos a la consola de AWS(AWS Management Console)

- Seleccionamos la opción S3 dentro de la lista de servicios
- Dentro de la página de gestión seleccionamos la opción "Create Bucket" para crear el volumen donde almacenaremos los ficheros.
- Una vez creado entramos en el volumen y mediante la opción "Create Folder" creamos la estructura de carpetas deseada y a continuación subimos los archivos mediante la opción "Upload".

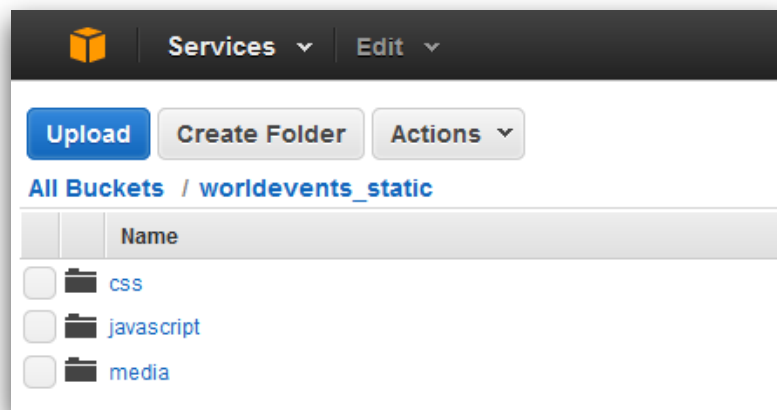


Figura 6.6 Detalle S3 AWS

6.1.7 Crear instancia EC2 para servidor de aplicaciones en AWS.

En esta tarea se va a crear una máquina virtual para que realice funciones de servidor de aplicaciones. Para ello debemos estar logueados en [AWS](#).

- Accedemos a la consola de AWS(AWS Management Console).
- Seleccionamos la opción EC2 dentro de la lista de servicios.
- Hacemos click en el botón "Launch Instance" para crear una nueva instancia.
- En el paso 1 escogemos una máquina Ubuntu Server 14.04 de 64 bits.
- En el paso 2 escogemos una máquina c3.large de tipo Compute Optimized.
- En los pasos 3 y 4 dejamos las opciones por defecto.
- En la paso 5 añadimos una etiqueta que describa su función, por ejemplo "AppServer".

- En el paso 6 añadimos una nueva regla al grupo de seguridad para que se permita el tráfico entrante en el puerto 80.
- En el paso 7 confirmamos los cambios y lanzamos la instancia.
- A continuación nos dará la opción de crear las claves de acceso para SSH y descargarnos un fichero con ellas.
- En la página de Instancias comprobamos que después de unos pocos minutos ya está funcionando.

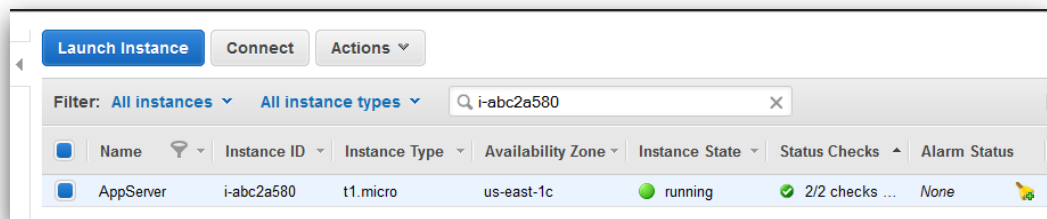


Figura 6.7 Detalle instancias AWS

6.1.8 Instalar el software necesario en la instancia

Primero debemos conectarnos mediante SSH a la ip pública de la instancia creada usando el fichero que nos descargamos en el paso anterior.

A continuación se instalan los siguientes paquetes Nginx, Gunicorn, Git, Virtualenv y Pip, mediante la orden siguiente orden:

```
apt-get install nginx gunicorn git python-virtualenv python-pip
```

6.1.9 Clonar en el servidor el repositorio

En este paso hay que copiar la carpeta del proyecto Django en la instancia. Se puede hacer de varias formas. En el caso de este proyecto, aprovechando que se usó la herramienta git se subió a un repositorio en github.com y se clonó partir de este en la instancia.

6.1.10 Instalar los paquetes python necesarios

Instalamos en el sistema las librerías necesarias usando la herramienta pip, mediante la orden:

```
pip install -r requeriments.txt
```

6.1.11 Modificamos el fichero de configuración de python.

El fichero de configuración de la aplicación "settings.py" ya está preparado para funcionar en entorno de producción. Únicamente hay que cambiar la opción "DEBUG" y darle el valor "False". Esto realiza automáticamente los siguientes cambios en las opciones por defecto del fichero:

- Cambiar el parámetro TEMPLATE_DEBUG a False, de este modo evitamos que la información de los fallos del servidor sea tan rica como en modo desarrollo, mejorando con ello la seguridad.
- Modifica los datos de conexión a la base de datos adaptándolos al entorno de producción.
- Cambiar los parámetros STATIC_URL, STATICFILES_DIR y MEDIA_ROOT para que usen el valor del almacén de datos que hemos creado en S3. Y añade las claves necesarias para acceder a dicho almacén.

6.1.12 Configurar Nginx

Para ello editamos su fichero de configuración "/etc/nginx/sites-available/default" e incluimos las siguientes opciones en la etiqueta "server":

```
server {  
    listen 80;  
    index index.html index.htm;  
    server_name localhost;  
    location / {  
        proxy_pass http://127.0.0.1:8000;  
    }  
}
```

6.1.13 Iniciar Nginx y Gunicorn

Se usan la siguientes ordenes:

```
python manage.py run gunicorn  
sudo service nginx start
```

Comprobamos que todo funciona correctamente accediendo al servidor web a través de la IP pública de la instancia.

6.1.14 Configurar la máquina para que nginx y gunicorn se ejecuten al iniciarla

Incluimos en el archivo "/etc/rc.local" las siguientes líneas:

```
python /home/ubuntu/proyecto/worldevents/manage.py run_gunicorn
sudo service nginx start

exit 0
```

6.1.15 Creamos una imagen para usarla como base .

En la consola de AWS seleccionamos la opción "Create Image" dentro de las opciones de la instancia:

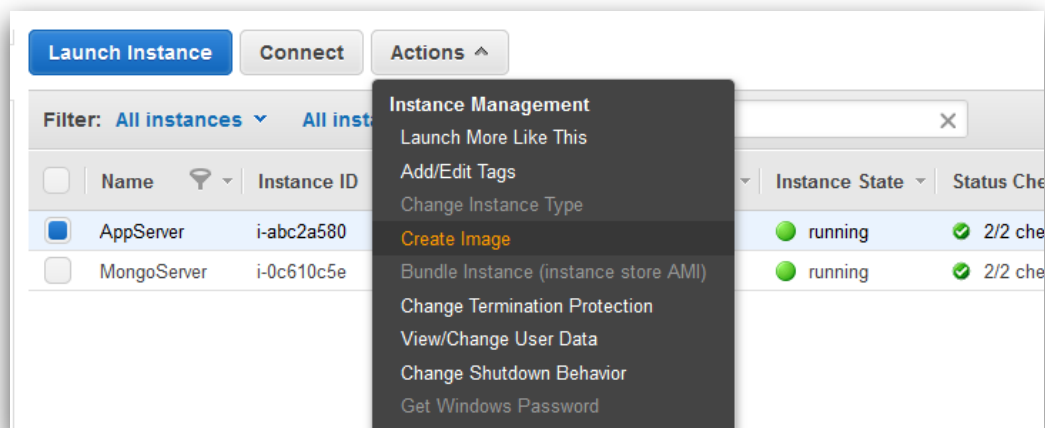


Figura 6.8 Opción Create Image AWS

En la siguiente pantalla le damos un nombre descriptivo y creamos la imagen. A continuación comprobamos que se ha creado correctamente accediendo a la opción "AMIs" en el submenú "IMAGES":

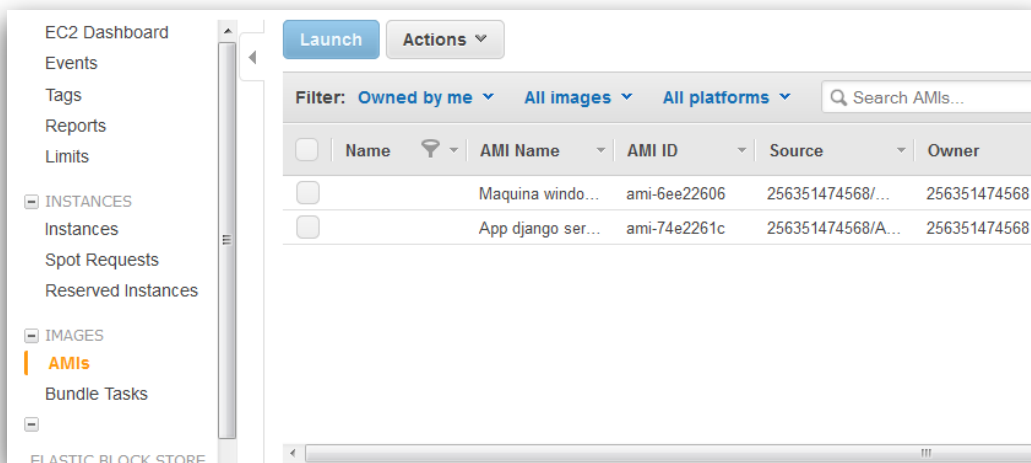


Figura 6.9 Lista imágenes AWS

6.1.16 Creamos un Elastic Load Balancer en AWS.

- Seleccionamos la opción "Load Balancers" del submenú "NETWORK & SECURITY"

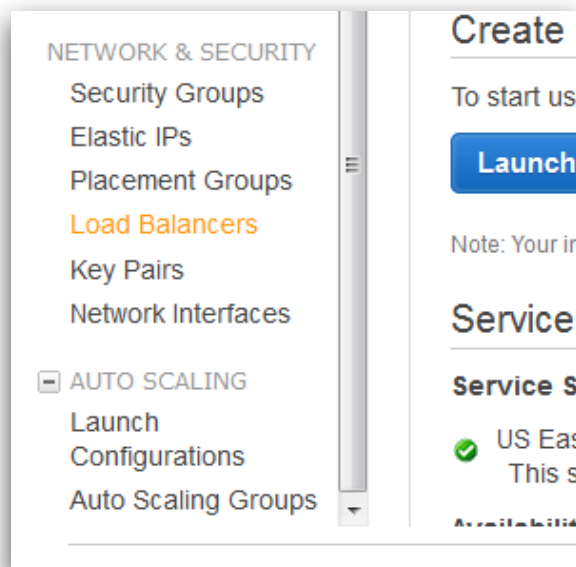


Figura 6.10 Submenú "Load Balancers" AWS

- Usamos la opción "CREATE LOAD BALANCER".
- En el paso uno introducimos un nombre para el balanceador y dejamos el resto de opciones como están.

Create Load Balancer

1. Define Load Balancer | 2. Configure Health Check | 3. Add EC2 Instances | 4. Review

This wizard will walk you through setting up a new load balancer. Begin by giving your new load balancer a unique name so that you can identify it from other load balancers you might create. You will also need to configure ports and protocols for your load balancer. Traffic from your clients can be routed from any load balancer port to any port on your EC2 instances. By default, we've configured your load balancer with a standard web server on port 80.

Load Balancer name:

Create LB Inside:

Create an internal load balancer: (what's this?)

Enable advanced VPC configuration:

Listener Configuration:

Load Balancer Protocol	Load Balancer Port	Instance Protocol	Instance Port
<input type="text" value="HTTP"/>	<input type="text" value="80"/>	<input type="text" value="HTTP"/>	<input type="text" value="80"/>

Figura 6.11 Crear Load Balancer AWS Paso 1

- En el paso dos configuramos el "Ping Path" para que coincida con la página de inicio de la aplicación. En nuestro caso "/". En la sección "Advanced Details" se puede configurar el intervalo de tiempo en el que el balanceador comprueba que los servidores que gestiona están en funcionamiento. La opción por defecto es válida en este caso.

Create Load Balancer

1. Define Load Balancer | 2. Configure Health Check | 3. Assign Security Groups | 4. Add EC2 Instances | 5. Review

Configure Health Check

Your load balancer will automatically perform health checks on your EC2 instances and only route traffic to instances that pass the health check. If an instance fails the health check, it is automatically removed from the load balancer. Customize the health check to meet your specific needs.

Ping Protocol:

Ping Port:

Ping Path:

Advanced Details

Response Timeout: seconds

Health Check Interval: seconds

Unhealthy Threshold:

Healthy Threshold:

Figura 6.12 Crear Load Balancer AWS Paso 1

- En el paso tres se selecciona el grupo de seguridad, que son las opciones del cortafuegos de [AWS](#). Podemos dejar el que sugiere por defecto.
- En el paso cuatro se añaden las instancias con las que el balanceador va a trabajar. En este caso se va a usar un "Auto Scaling Group" que se configurará posteriormente con lo que podemos saltar al paso siguiente.
- El paso cinco es un resumen de las opciones seleccionadas. Si está todo correcto usamos la opción "CREATE" para crear el balanceador.
- Comprobamos que se ha creado correctamente.

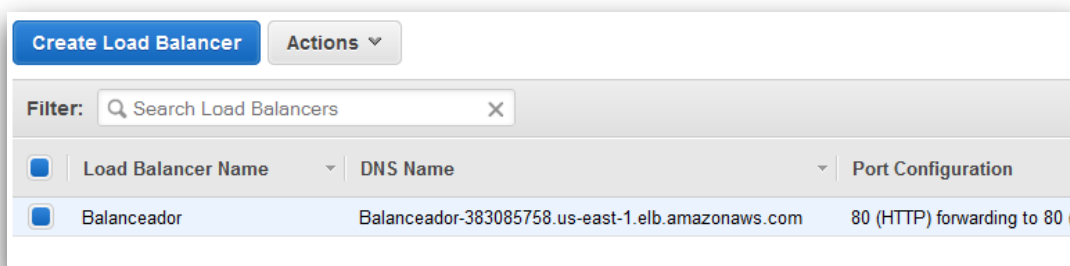


Figura 6.13 Lista balanceadores AWS

El nombre DNS será el que se use para acceder a la aplicación web. En un proyecto real habría que configurar el dominio en el servidor DNS de [AWS](#).

6.1.17 Creamos un Auto Scaling Group AWS.

Este servicio sirve para monitorizar un grupo de servidores de modo que siempre haya un número mínimo en funcionamiento y además se pueda adaptar a los cambios en la densidad del tráfico añadiendo o eliminando servidores automáticamente.

- Usamos la opción "Launch Configurations" del submenú "AUTO SCALING"

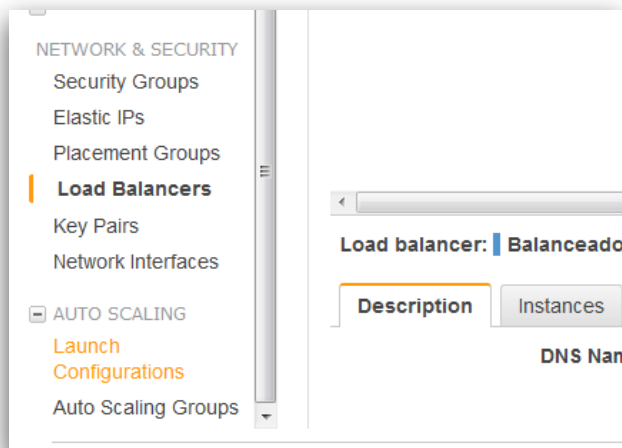


Figura 6.14 Submenú "Launch Configuration"

- Seleccionamos la opción "Create Launch Configuration".
- En el paso uno seleccionamos la imagen del servidor de aplicaciones que habíamos creado, que servirá como base para la creación de nuevas instancias.

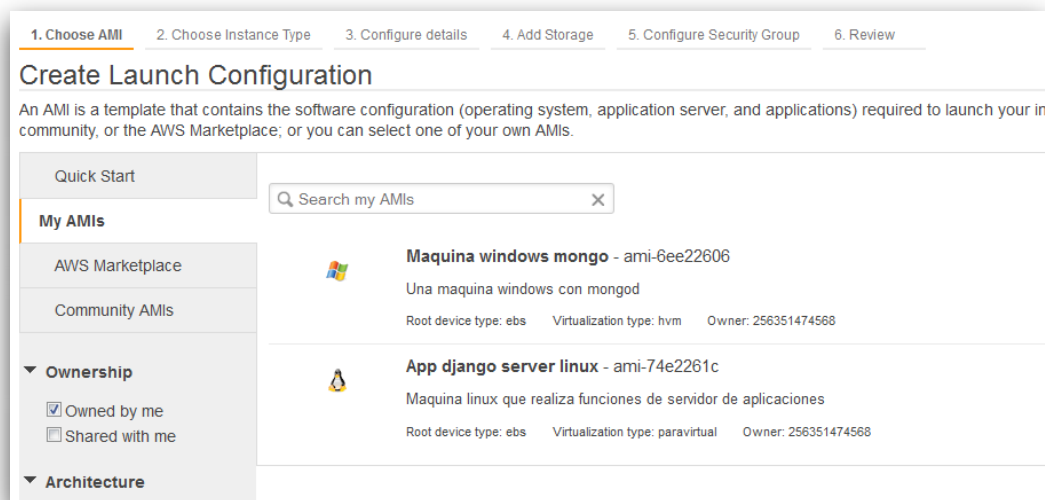


Figura 6.15 Lista imágenes AWS

- En el paso dos escogemos el tipo de instancia que en el caso del servidor de aplicaciones se trataba de una máquina c3.large de tipo Compute Optimized.
- En el paso tres damos un nombre a la configuración.
- El paso cuatro es para añadir los volúmenes de almacenamiento. Dejamos la opción por defecto.

- En el paso cinco se configura el cortafuegos de AWS. En este caso es necesario añadir una regla que permita el tráfico entrante HTTP hacia los servidores.

Create Launch Configuration

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create about Amazon EC2 security groups.

Assign a security group: Create a **new** security group
 Select an **existing** security group

Security group name: AutoScaling-Security-Group-2

Description: AutoScaling-Security-Group-2 created on Thursday, July 3, 2014 11:10:52 AM UTC+2

Protocol <small>(i)</small>	Type <small>(i)</small>	Port Range (Code) <small>(i)</small>
SSH	TCP	22
HTTP	TCP	80

Add Rule

Figura 6.16 Create Launch Configuration AWS paso 5

- El paso seis es un resumen de la configuración realizada. Si todo está correcto creamos la nueva configuración usando la opción "Create Launch Configuration". A continuación se abrirá una ventana donde debemos seleccionar las claves que usaremos para acceder vis SSH a las instancias.
- Si todo ha ido bien llegaremos a la siguiente pantalla donde escogeremos la opción de crear un AutoScaling Group para este configuración.

Launch configuration creation status

✓ Successfully created launch configuration: Capa de aplicaci?n
[View creation log](#)

View
[View your launch configurations](#)
[View your Auto Scaling groups](#)

Here are some helpful resources to get you started

[Create an Auto Scaling group using this launch configuration](#) [Close](#)

Figura 6.17 Launch Configuration AWS creada

- En el paso uno damos un nombre al nuevo "AutoScaling Group". En "Group Size" indicamos que se arranque con dos instancias, y en subnet debemos escoger la subred

donde queremos que funcione se arranquen las instancias. Cualquiera de las existentes es válida. Por último debemos seleccionar el balanceador que creamos anteriormente para que trabaje con este grupo.

Figura 6.18 Create Auto Scaling Group AWS paso 1

- En el paso dos se configuran las opciones de auto escalado. Debemos añadir dos reglas, una para indicar cuándo se deben crear nuevos servidores y otra para indicar cuándo se deben eliminar.

Figura 6.18 Create Auto Scaling Group AWS paso 1

- Es necesario crear nuevas alarmas para indicar las condiciones a partir de las cuales se modificará el número de servidores. Seleccionamos "Add new alarm".
- En este caso se ha creado una alarma para alertar de que el uso de CPU ha superado el 80%.

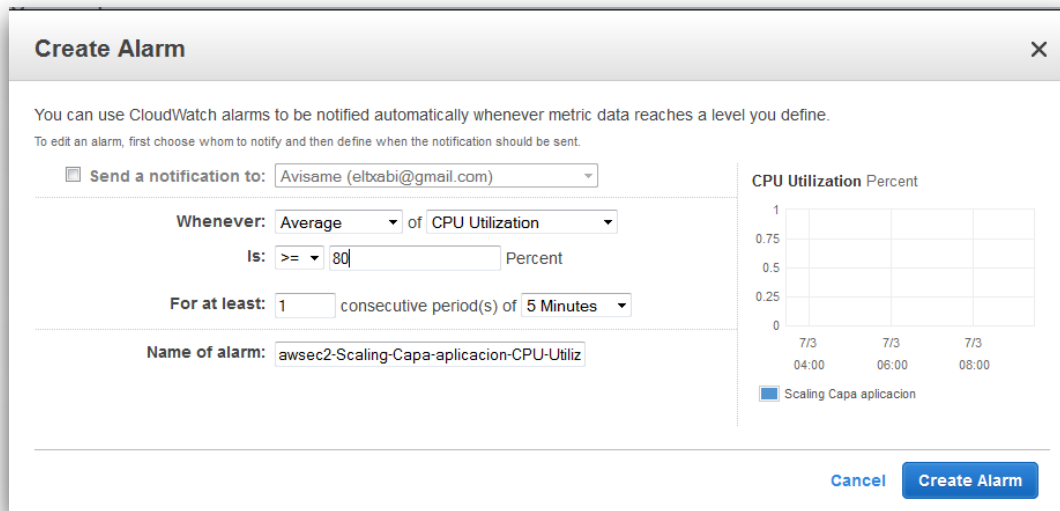


Figura 6.19 Create Alarm AWS

- Usando esa alarma se indica que en caso de saltar se añada un nuevo servidor, y que siempre se mantengan como mínimo dos en funcionamiento.

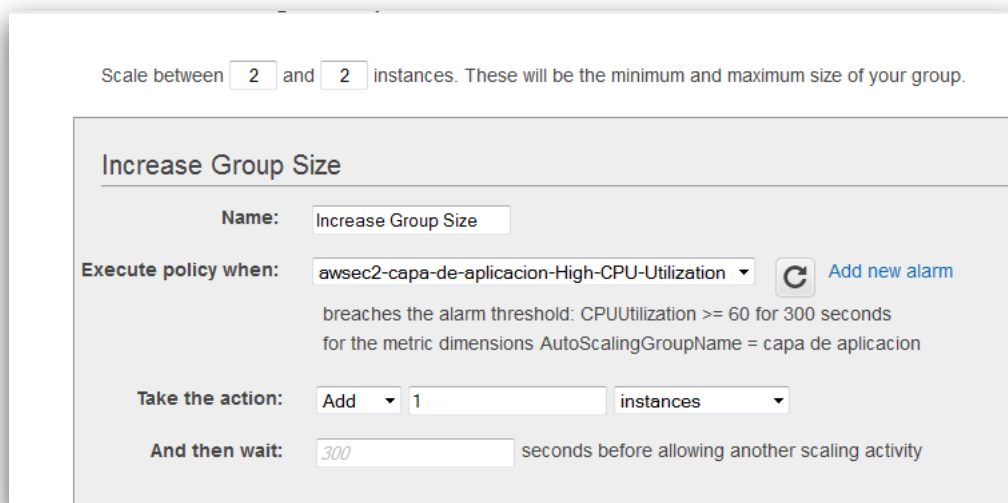
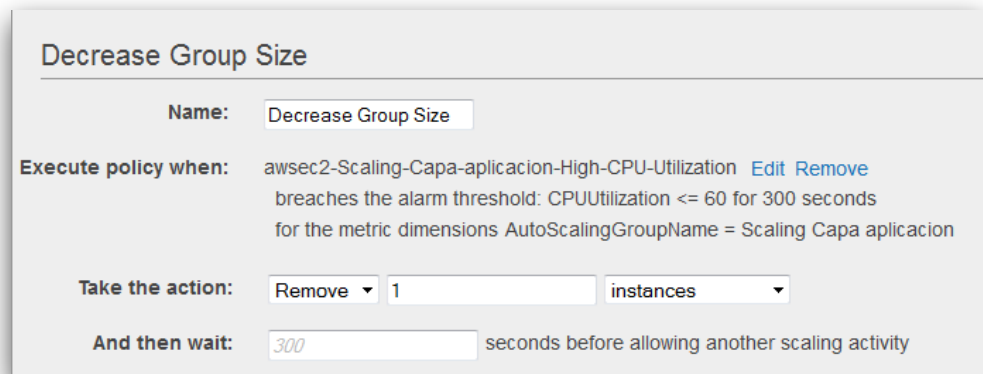


Figura 6.20 Create Alarm AWS Increase condition

- Se repite el mismo paso para crear una regla que disminuya el número de servidores cuando la utilización media de las CPU sea del 60%.



Decrease Group Size

Name: Decrease Group Size

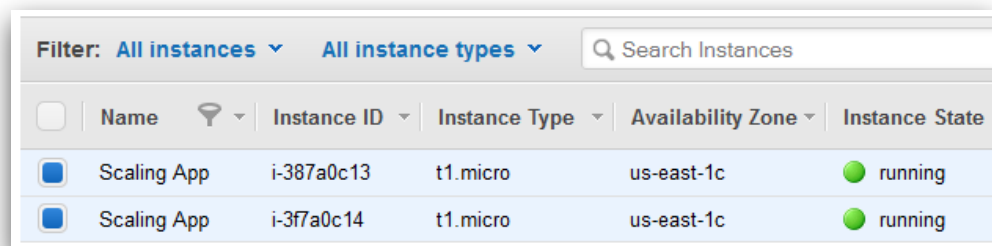
Execute policy when: [awsec2-Scaling-Capa-aplicacion-High-CPU-Utilization](#) [Edit](#) [Remove](#)
breaches the alarm threshold: CPUUtilization <= 60 for 300 seconds
for the metric dimensions AutoScalingGroupName = Scaling Capa aplicacion

Take the action: Remove 1 instances

And then wait: 300 seconds before allowing another scaling activity

Figura 6.21 Create Alarm AWS decrease condition

- En el paso tres podemos configurar el envío de notificaciones.
- En el paso cuatro podemos añadir etiquetas a las nuevas instancias.
- El paso cinco es el resumen de las opciones seleccionadas. Si todo está en orden podemos crear el "AutoScaling Group".
- Si todo está correcto se habrán creado dos nuevas instancias pertenecientes al "AutoScaling Group" creado y el balanceador distribuirá las peticiones a dichas instancias.



<input type="checkbox"/>	Name	Instance ID	Instance Type	Availability Zone	Instance State
<input checked="" type="checkbox"/>	Scaling App	i-387a0c13	t1.micro	us-east-1c	running
<input checked="" type="checkbox"/>	Scaling App	i-3f7a0c14	t1.micro	us-east-1c	running

Figura 6.22 Lista instancias Auto Scaling Group AWS

6.2 Manual de Sharding

En este manual se explica detalladamente el proceso de configuración para habilitar el sistema de particionado en la capa de datos y añadir nuevas particiones. Este proceso comprende realizar cambios en el servidor de aplicaciones, con lo cual probablemente haya que parar el sistemas y reconfigurar tanto la imagen de los servidores de aplicaciones como el "AutoScaling Group".

6.2.1 Crear "Config Servers".

En primer lugar se han de crear los "Config Servers" que almacenarán los metadatos de las particiones. No son necesarias instancias muy potentes, por tanto usaremos máquinas de tipo "microinstance".

- Accedemos a la consola de [AWS](#)(AWS Management Console).
- Seleccionamos la opción EC2 dentro de la lista de servicios.
- Hacemos click en el botón "Launch Instance" para crear una nueva instancia.
- En el paso 1 escogemos una máquina Ubuntu Server 14.04 de 64 bits.
- En el paso 2 escogemos una máquina t2.micro.
- En los pasos 3 y 4 dejamos las opciones por defecto.
- En la paso 5 añadimos una etiqueta que describa su función, por ejemplo "ConfigServer".
- En el paso 6 añadimos una nueva regla al grupo de seguridad para que se permita el tráfico entrante en el puerto 27019, que es el puerto por defecto de los "Config Server".
- En el paso 7 confirmamos los cambios y lanzamos la instancia.
- A continuación nos dará la opción de crear las claves de acceso para SSH y descargarnos un fichero con ellas.
- En la página de Instancias comprobamos que después de unos pocos minutos ya está funcionando.
- Accedemos a ella a través de SSH usando las claves descargadas.

- Instalamos MongoDB siguiendo las instrucciones del siguiente enlace en la página oficial de la herramienta:

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/>

- Creamos el directorio donde se almacenarán los datos.

```
mkdir /data/configdb
```

- Arrancamos el "Config Server"

```
mongod --configsvr --dbpath /data/configdb
```

- Incluimos el comando anterior en el archivo "/etc/rc.local" para que se arranque el servidor al iniciar la instancia.
- Creamos una imagen de la instancia para poder usarla como base para añadir nuevos servidores. Para ello:
 - En la consola de [AWS](#) seleccionamos la opción "Create Image" dentro de las opciones de la instancia:
 - En la siguiente pantalla le damos un nombre descriptivo y creamos la imagen. A continuación comprobamos que se ha creado correctamente accediendo a la opción "AMIs" en el submenú "IMAGES".
- Usando la imagen anterior creamos dos nuevas instancias.

6.2.2 Arrancar y configurar procesos "mongos".

Estos procesos servirán de intermediarios entre los servidores de aplicaciones y el sistemas de base de datos. Han de arrancarse en el servidor con lo cual habrá que reconfigurar el sistema de AutoScaling y la imagen usada por este servicio.

- Instalamos [MongoDB](#) en el servidor de aplicaciones siguiendo las instrucciones del siguiente enlace en la página oficial de la herramienta:

<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-ubuntu/>

- Arrancamos el proceso "mongos" pasando como parámetros los tres "Config Servers" creados anteriormente.

```
mongos --configdb server1:27019,server2:27019,server3:27019
```

- Nos conectamos a un proceso "mongos" usando la siguiente orden:

```
mongo --host server --port 27017
```

- Añadimos los "Replica Set" como "Shards" mediante la orden siguiente, siendo "rs1" el nombre del "Replica Set" a añadir y "Servidor" el nombre DNS de uno de los servidores

del "Replica Set". Usamos la misma orden para añadir todos los "Replica Set" que queramos.

```
sh.addShard( "rs1/Servidor:27017" )
```

6.2.3 Habilitar Sharding en el sistema.

Finalmente debemos habilitar la opción de "Sharding" tanto en la base de datos como en las colecciones que se van a particionar, en nuestro caso la colección "Event", además de incluir la "Shard Key".

- Nos conectamos a una instancia de "mongos" como en el paso anterior.
- Habilitamos "Sharding" en la base de datos mediante la orden:

```
sh.enableSharding("worldevents")
```

- Habilitamos "Sharding" en la colección "event" usando como "Shard Key" un índice tipo "hashed" del campo "_id".

```
sh.shardCollection("worldevents.event", { "_id": "hashed" } )
```

- En caso de tener varias particiones habrá que esperar un periodo de tiempo hasta que el sistema equilibre la cantidad de datos de cada partición. Esta es una tarea que consume muchos recursos y que puede ralentizar el rendimiento del sistema. Debido a esto es recomendable realizar este tipo de operaciones en periodos de poca actividad del sistema.

6.3 Manual de uso de la aplicación web.

La página inicial de la aplicación es la siguiente:



Figura 6.23 Página inicial WorldEvents

Existen tres zonas bien diferenciadas, la lista de eventos con uno de ellos seleccionado, la zona de detalle del evento que muestra la información asociada al evento seleccionada y el mapa donde se muestra la posición de los eventos de la lista. El evento seleccionado se muestra con un marcador rojo.

Haciendo click en los eventos de la lista se puede cambiar el evento seleccionado y la información asociada se mostrará instantáneamente en la zona de detalle. El mapa se centrará en el marcador del nuevo evento seleccionado.



Figura 6.24 Mapa página inicial WorldEvents

Mediante los controles en la parte inferior de la lista podemos navegar por los distintos eventos del sistema.

En la barra superior aparecen tres enlaces "Register", "Login" y "Search".

- El enlace "Register" sirve para dar de alta a usuarios en el sistema mediante el siguiente formulario:



Figura 6.24 Formulario Create Account

- El enlace "Login" sirve para loguearse mediante el siguiente formulario:

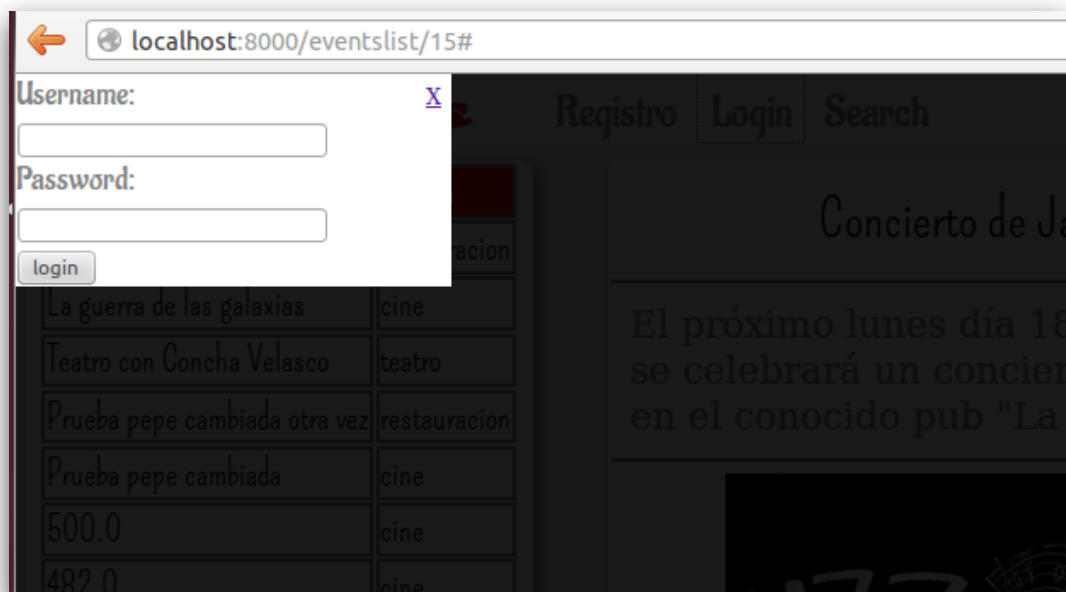


Figura 6.25 Formulario Login

- El enlace "Search" permite realizar una búsqueda dentro de la lista de eventos siguiendo tres criterios: "title", "category" y zona geográfica. Para definir la zona geográfica es necesario seleccionar un punto en el mapa y definir un radio de búsqueda mediante el control "slider".

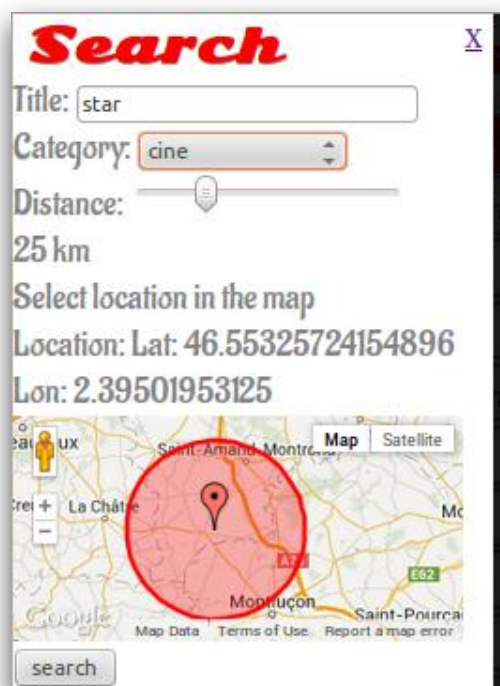


Figura 6.26 Formulario Search

Los resultados de la búsqueda se mostrarán en la lista de la página y se podrá navegar por ellos usando los controles de la lista.

Si el usuario se loguea en el sistema en el detalle del evento se muestra un enlace para añadir comentarios en el evento. Además si el usuario logueado es el creador del evento se muestran dos enlaces para editar o borrar el evento.

Concierto de Jazz	musica
Pizzeria La Gondola	restauracion
La guerra de las galaxias	cine
Teatro con Concha Velasco	teatro
Prueba pepe cambiada otra vez	restauracion
Prueba pepe cambiada	cine
500.0	cine
482.0	cine
483.0	cine
484.0	cine
485.0	cine
486.0	cine
487.0	cine
488.0	cine

Concierto de Jazz
musica

El próximo lunes día 18 de Agosto se celebrará un concierto de jazz en el conocido pub "La ostra azul"



[Edit](#)
[Delete](#)

Add comment

Figura 6.27 Detalle links Edit, Delete y Add comment

Para añadir un comentario se usa el enlace "Add comment" que abrirá un pequeño formulario donde se puede insertar el texto del comentario:

Figura 6.28 Detalle formulario Add Comment

Una vez añadido se mostrará en el detalle del evento junto con el nombre del autor:

La guerra de las galaxias	cine
Teatro con Concha Velasco	teatro
Prueba pepe cambiada otra vez	restauracion
Prueba pepe cambiada	cine
500.0	cine
482.0	cine
483.0	cine
484.0	cine
485.0	cine
486.0	cine
487.0	cine
488.0	cine
489.0	cine

< 1-15 >

El próximo lunes día 18 de Agosto se celebrará un concierto de jazz en el conocido pub "La ostra azul"

Edit Delete

Add comment

pepe
No faltare!!!

Figura 6.29 Detalle comentario

En caso de trabajar con un tamaño de pantalla entre 768 y 1024 pixeles de ancho, el cual entra dentro del rango de las tabletas el interfaz cambia ligeramente. El mapa no se muestra por defecto y aparece un nuevo enlace "Map/Detail" para alternar entre el mapa y el detalle del evento seleccionado.

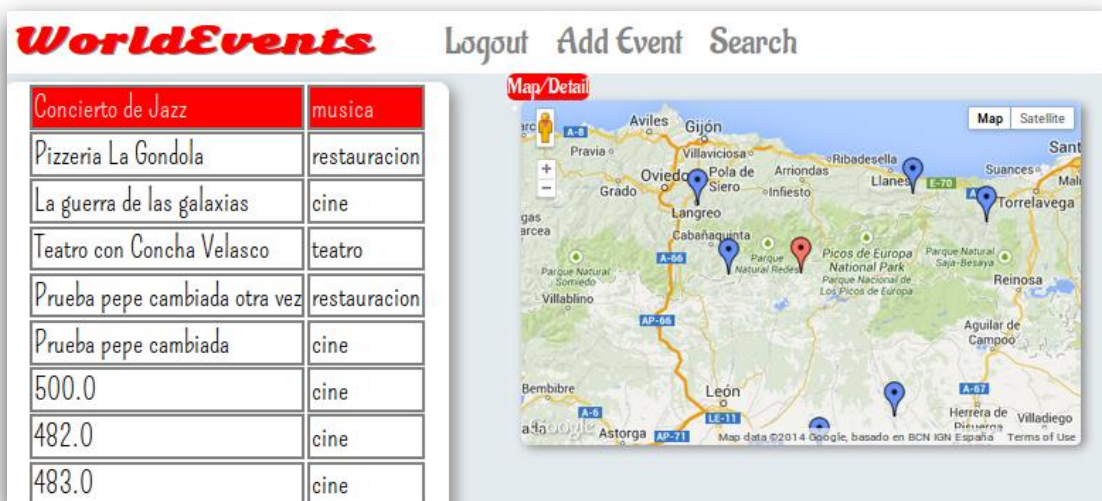
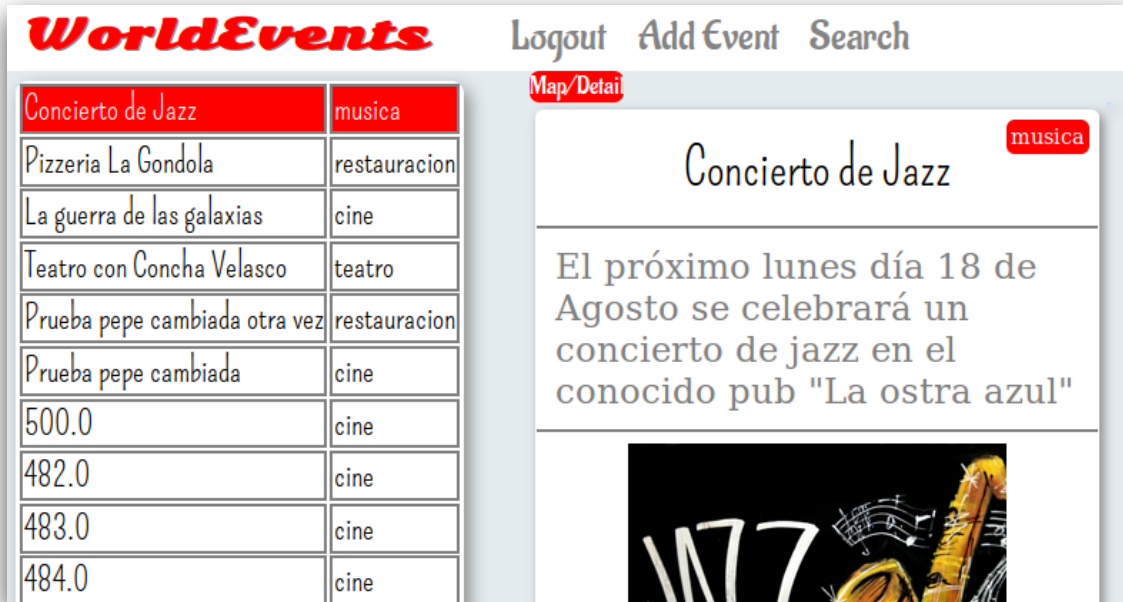


Figura 6.30 Opción Map/Detail

En el caso de pantallas menores de 768 pixeles de ancho, dentro del rango de dispositivo móviles, inicialmente se mostrará solo la lista de eventos:

The screenshot shows the 'WorldEvents' mobile application interface. At the top, there is a navigation bar with the title 'WorldEvents' in red, and three menu items: 'Logout', 'Add Event', and 'Search'. Below the navigation bar is a table listing various events and their categories.

Concierto de Jazz	musica
Pizzeria La Gondola	restauracion
La guerra de las galaxias	cine
Teatro con Concha Velasco	teatro
Prueba pepe cambiada otra vez	restauracion
Prueba pepe cambiada	cine
500.0	cine
482.0	cine
483.0	cine
484.0	cine
485.0	cine

Figura 6.31 Lista eventos pantalla móvil

En caso de seleccionar un evento, se pasará a otra pantalla donde se mostrará el detalle del evento. Además habrá dos enlaces:

- Map/Detail para alternar entre la vista detalle o la vista mapa.
- Back to list para volver a la lista de eventos.



Figura 6.32 Link Back to list

Capítulo 7. Conclusiones y Ampliaciones

7.1 Conclusiones

En primer lugar se puede decir que los objetivos planteados inicialmente se han cumplido. Se ha desarrollado una aplicación web con una funcionalidad mínima, que ha servido como base para aprender y usar nuevos conocimientos y diseñar una arquitectura escalable en un sistema de *Cloud Computing* empresarial.

Las herramientas elegidas han sido bastante acertadas y se han adaptado bien a los requerimientos del proyecto. Únicamente se puede mencionar como nota negativa la falta de integración por defecto de Django con sistemas *NoSQL*, lo que ha obligado al uso de librerías externas como *MongoEngine* y ha planteado algunos problemas de compatibilidad.

Por otra parte el uso de *MongoDB* ha demostrado ser una elección acertada, ya que ha demostrado su buena fama actual dentro del ámbito informático, ayudando con sus capacidades a alcanzar los objetivos del proyecto.

En cuanto a Amazon Web Services, también se ha revelado como una garantía a la hora de afrontar la creación de una infraestructura empresarial para cualquier tipo de proyecto de software. Su variedad de servicios y su facilidad de uso han sido de gran ayuda durante la realización de este proyecto.

Como nota negativa hay que destacar la mala planificación inicial y los problemas de tiempo sufridos, debido a la dificultad a la hora de asimilar nuevos conceptos y trabajar con nuevas herramientas. La curva de aprendizaje fue más pronunciada de lo esperado y eso hizo que el ritmo de desarrollo fuera muy lento sobre todo en las etapas iniciales. Como resultado, aunque al final se alcanzaron los objetivos mínimos, la satisfacción no fue completa ya que se desearía haber tenido más tiempo para obtener un mayor dominio de los nuevos conocimientos y poder ofrecer más funcionalidades.

7.2 Ampliaciones.

La aplicación web desarrollada incluye una funcionalidad mínima para poder alcanzar los objetivos del proyecto, pero es susceptible de ampliaciones. Se pueden plantear por ejemplo, las siguientes:

- Inclusión de datos temporales en los eventos. De ese modo el sistema podría trabajar con preguntas del tipo: "Conciertos en mi zona durante el próximo fin de semana".
- Uso de WebSockets para poder ofrecer un servicio de información de eventos en tiempo real.
- Opción de crear y gestionar listas de eventos favoritos.

- Opción de incluir alertas para recibir información cuando se añada al sistema un evento con unas determinadas características.

En cuanto al despliegue de la aplicación, la inclusión de ampliaciones y modificaciones siempre vendrá marcada por las necesidades y comportamientos futuros del sistema. Será necesaria una labor de monitorización continua para poder detectar las nuevas necesidades y poder reaccionar lo más rápido posible. Por ello, la ampliación más urgente probablemente sería la inclusión de un sistema de monitorización.

Capítulo 8. Presupuesto

Dado el carácter de este proyecto enfocado más bien al auto aprendizaje y donde se han usado herramientas desconocidas para el proyectante no se incluyó un presupuesto de desarrollo del proyecto. De hecho una parte muy importante del tiempo empleado ha estado dedicado al estudio de nuevas tecnologías y herramientas. No parece muy coherente cargar a un hipotético cliente con los gastos de aprendizaje del desarrollador. Por otra parte dada la gran variedad de tareas de las que se compuso el proyecto parece difícil encontrar un método de facturación adecuado.

Sin embargo sí que parece coherente incluir un presupuesto detallado de despliegue y mantenimiento del proyecto. Aunque para las pruebas y desarrollo de este proyecto se usó la capa gratuita que ofrece Amazon, todos los servicios incluidos tienen en realidad un coste. Otro de los problemas asociados al aumento repentino de usuarios, además de que el sistema se vea saturado, es también que los costes necesarios para adaptarse al nuevo escenario sean prohibitivos. Por ello se incluye un presupuesto detallado del coste de mantenimiento de esta infraestructura, así como una estimación del aumento de costes ante posibles ampliaciones del sistema.

Para la elaboración de este presupuesto se han tenido en cuenta además de los costes asociados al uso de AWS, los derivados del uso de Google Maps, que en caso de sobrepasar ciertos límites de tráfico aplica una cuota mensual siguiendo la siguiente tabla:

Daily Map Loads	Cost for JavaScript v3, Staticmaps, Streetview (per day)
5,000	\$0
15,000	\$0
25,000	\$0
35,000	\$5
45,000	\$10
75,000	\$25
100,000	\$37.50

Figura 8.1 Precios Google Maps

8.1 Presupuesto inicial

Se ha calculado este presupuesto inicial para los primeros meses de vida del sistema, suponiendo una carga de trabajo baja. Se incluye un volumen de transferencias de 1000 GB mensuales y un almacenamiento de 1 TB en el almacén S3. Se puede observar que casi el 90% del coste corresponde al Replica Set, principalmente debido al alto coste de las instancias optimizadas para almacenamiento. En este escenario, el uso de un "Replica Set" es innegociable ya que garantiza rendimiento, disponibilidad y seguridad de los datos. Una solución para ahorrar costes sería sustituir uno de los servidores por un "arbitrer" que podría ser ejecutado en una instancia de bajas prestaciones. O incluso mientras el sistema no tenga una carga muy alta, usar instancias más económicas. En todo como ya se ha mencionado en otras partes de esta documentación ese tipo de decisiones entran ya dentro del proceso de monitorización y mantenimiento del sistema, que en todo caso escapa al ámbito de este proyecto.

En cuanto a los costes variables que podrían derivarse de picos en la carga de trabajo que supondrían la creación automática de nuevos servidores o el incremento de las transferencias del sistema no parecen muy elevados, ya [AWS](#) factura el uso de los recursos por horas. De ese modo, mientras la carga media mensual se mantenga y no sea necesario añadir nuevos recursos los costes estarán controlados.

Concepto	PU Mensual	Cantidad	Precio Total
Instancias EC2 c3.large	65,42	2	130,84
Instancias EC2 i2.xlarge	511,33	3	1533,99
Elastic Load Balancer			
<i>Tarifa por tiempo(0,02/hora)</i>	14,88	1	14,88
<i>Tarifa por transferencias(1000GB)</i>	8	1	8
Almacenamiento S3 (1 TB)	21,98	1	21,98
Uso de Google Maps	0	0	0
		Precio Total	1709,69

Figura 8.2 Presupuesto inicial

A continuación se incluye el presupuesto modificado para usar máquinas más asequibles para los servidores de bases de datos. Una vez el sistema este en producción es necesario realizar un continuo análisis del rendimiento para decidir cuál sería la mejor opción e introducir los cambios necesarios en el momento adecuado.

Concepto	PU Mensual	Cantidad	Precio Total
Instancias EC2 c3.large	65,42	2	130,84
Instancias EC2 c3.large	65,42	3	196,26
Elastic Load Balancer			
<i>Tarifa por tiempo(0,02/hora)</i>	14,88	1	14,88
<i>Tarifa por transferencias(1000GB)</i>	8	1	8
Almacenamiento S3 (1 TB)	21,98	1	21,98
Uso de Google Maps	0	0	0
		Precio Total	371,96

Figura 8.3 Presupuesto inicial modificado

8.2 Presupuesto final

En este punto se estima un presupuesto para una infraestructura suponiendo que el proyecto ha tenido un gran éxito y que la carga de trabajo del sistema ha sufrido un importante incremento. Se incluyen instancias más potentes y se amplía el número de servidores de la capa de aplicación a 10, además se distribuye la base de datos en 4 "Shards".

Concepto	PU Mensual	Cantidad	Precio Total
Instancias EC2 c3.8xlarge	1041,6	10	10416
Instancias EC2 t1.micro	14,88	3	44,64
Instancias EC2 i2.8xlarge	2670	12	32040
Elastic Load Balancer			
<i>Tarifa por tiempo(0,02/hora)</i>	14,88	1	14,88
<i>Tarifa por transferencias(10TB)</i>	80	1	80
Almacenamiento S3 (10 TB)	219,8	1	219,8
Uso de Google Maps(75.000 Loads per day)	775	1	775
		Precio Total	43590,32

Figura 8.4 Presupuesto Final

Capítulo 9. Referencias Bibliográficas

9.1 Libros y Artículos

Holovaty, Adrian; Kaplan-Moss, Jacob. The Django Book. 2009.

Percival, Harry J.W. Test Driven development with Python. 2014.

9.2 Referencias en Internet

MongoDB, Inc. <https://www.mongodb.org>

Django Software Foundation. <https://www.djangoproject.com>

Amazon Web Services, Inc. <http://aws.amazon.com>

MongoEngine Organisation. <http://mongoengine.org>

Mitch Garnaat. <http://docs.pythonboto.org>

<http://git-scm.com>

Ian Bicking, The Open Planning Project, PyPA. <https://virtualenv.pypa.io>

<http://gmap3.net>

The jQuery Foundation. <http://jquery.com>

Todd Hoff. <http://highscalability.com>

Capítulo 10. Apéndices

10.1 Contenido Entregado en el anexo

En el anexo se incluyen los siguientes componentes:

- Copia del proyecto Django de la aplicación web.
- Una copia de la base de datos con datos de ejemplo.
- Un fichero LEEME.txt con instrucciones para el despliegue de la aplicación en un sistema linux. El fichero tiene el siguiente contenido:

En este anexo se incluyen dos carpetas:

`CopiaBaseDatos:` Incluye una copia de la base de datos con datos de prueba.

`CodigoFuente:` Incluye el proyecto Django con todos sus archivos y su estructura de carpetas

Para instalar la aplicación en un sistema linux siga los siguientes pasos:

1. Instale MongoDB en el sistema. Se pueden encontrar instrucciones detalladas en la página:

```
http://docs.mongodb.org/manual/administration/install-on-linux/
```

2. Restaure la copia de seguridad de la base de datos mediante la orden:

```
mongorestore -db worldevents CopiaBasedatos/worldevents
```

3. Instale en el sistema los paquetes necesarios que están listados en el fichero `requeriments.txt` mediante la orden:

```
pip install -r requeriments.txt
```

4. Inicie el servidor mediante la siguiente orden desde la carpeta `'worldevents'`

```
python manage.py runserver
```

5. La aplicación estará disponible en la dirección `https://localhost:8000`

10.2 Índice Alfabético

A

AWS, 2, 3, 25, 30, 31, 95, 101, 103, 104, 105, 106, 108, 110, 113, 117, 118, 119, 120, 121, 122, 123, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 147, 148, 159, 160

C

Cloud Computing, 2, 12, 13, 15, 25, 29, 41, 95, 145

D

Django, 2, 3, 12, 13, 19, 23, 26, 27, 28, 31, 37, 42, 44, 47, 51, 52, 63, 72, 75, 79, 84, 85, 101, 108, 113, 123, 145, 150

M

MongoDB, 2, 3, 14, 17, 23, 24, 28, 29, 30, 31, 63, 67, 72, 79, 95, 97, 98, 103, 104, 107, 109, 110, 113, 114, 118, 135, 145, 150

MongoEngine, 3, 28, 31, 51, 79, 145, 150

N

NoSQL, 2, 12, 13, 14, 17, 22, 23, 28, 29, 30, 31, 42, 48, 72, 104, 114, 145

P

Product Backlog, 41, 42, 43, 45, 48
Python, 2, 13, 19, 26, 31, 32, 37, 101, 108, 150

R

Responsive Design, 2, 3, 12, 13, 14, 19, 42

T

TDD, 3, 18, 19, 36, 37, 38, 41, 43, 44

10.3 Código Fuente

10.3.1 Fichero "views.py"

```

from eventslis.forms import RegistrationForm,EventForm,SearchForm,CommentForm
from eventslis.models import Category,Event,Comment
from django.http import HttpResponseRedirect
from django.shortcuts import render
from mongoengine.queryset import Q
from mongoengine.django.auth import User
from django.contrib import messages
from django.contrib.auth import login,logout,authenticate
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth.decorators import login_required
from mongoengine import *
from django.conf import settings
from time import time
import os
import pymongo
from storages import storage
from django.utils.translation import gettext as _

#####

def home(request,num_events='15'):
    if "event_list" not in request.session: # If there aren't results of a search in
session
        if "search_query" in request.session: # If there is a current search

            event_list=Event.search(request.session['search_query']['title'],request.session[
'search_query']['category'],request.session['search_query']['lat'],request.session['sear
ch_query']['lng'],request.session['search_query']['distance'],int(num_events))
            else:
                event_list = Event.objects.order_by('-added_date')[int(num_events)-
15:int(num_events)]
            else:
                event_list=request.session['event_list']
                del request.session['event_list']

        form = CommentForm()
        return render(request, "eventslis/home.html",
{'event_list':event_list,'num_events':num_events,'form':form,})

#####

@login_required
def addcomment(request):
    if request.method == 'POST':
        event_id=request.POST['event_id']
        form = CommentForm(request.POST)
        if form.is_valid():
            content=form.cleaned_data['content']
            user=request.user.username
            comment=Comment(content=content,user=user)
            Event.objects(id=event_id).update_one(push__comments=comment)
            form = CommentForm()
        else:
            form = CommentForm()

    event=Event.objects.get(id=event_id)
    # Return only comments of modified event
    return render(request, "eventslis/comments.html", {'form':form,'event':event})

#####

def searchevents(request):
    if request.method == 'POST':

```

```

form = SearchForm(request.POST)
if form.is_valid():
    title=form.cleaned_data['title']
    category=form.cleaned_data['category']
    lat=form.cleaned_data['lat']
    lng=form.cleaned_data['lng']
    distance=form.cleaned_data['distance']

    event_list=Event.search(title,category,lat,lng,distance,15)

    if event_list: # Put search query and results in session
        request.session["search_query"]={'title':title,'category':category,'lat':lat,'lng':lng,'distance':distance,'num_events':15}
        request.session["event_list"]=list(event_list)
    else:
        messages.success(request,_( 'No search results'))

        return HttpResponseRedirect("/")
else:
    form = SearchForm()
    return render(request, "eventslist/searchevents.html", {'form':form,})

#####

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():

            User.create_user(form.cleaned_data['username'],form.cleaned_data['password1'])
            messages.success(request, form.cleaned_data['username'] + _(' you have
been successfully registered'))
            return HttpResponseRedirect("/")
        else:
            form = RegistrationForm()
            return render(request, "eventslist/register.html", {
                'form': form,
            })

#####3333333

def loginpage(request):
    if request.method == 'POST':
        form = AuthenticationForm(request.POST)
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username,password=password)
        if user is not None:
            login(request,user)
            messages.success(request,_( 'Login Ok ') + user.username)
            return HttpResponseRedirect("/")
        else:
            messages.success(request,_( 'Login Fail'))
    else:
        form = AuthenticationForm()

    return render(request,'eventslist/login.html', {
        'form': form,
    })

#####

@login_required
def logoutpage(request):
    logout(request)
    return HttpResponseRedirect("/")

#####

@login_required
def addevent(request):
    if request.method == 'POST':

```

```

form = EventForm(request.POST, request.FILES)

if form.is_valid():
    title=form.cleaned_data['title']
    description=form.cleaned_data['description']
    category=form.cleaned_data['category']
    lat=form.cleaned_data['lat']
    lng=form.cleaned_data['lng']
    user=request.user.username

    event =
Event(title=title,description=description,category=category,user=user)

    if (request.FILES): #If photo is included store in static folder
        photo_name=request.user.username+"-"+str(int(time()))+".jpeg"
        f=request.FILES["photo"]
        strg=storage()
        getattr(strg,settings.STORAGE+"_write")(photo_name,f)
        event.photo=photo_name

    event.location=[float(lat),float(lng)]
    event.save()
    messages.success(request, title + _(' has been created'))
    return HttpResponseRedirect("/")

else:
    form = EventForm()

return render(request,'eventslist/addevent.html', {
    'form': form,
})

#####

@login_required
def deleteevent(request,event_id):
    if request.method == 'POST':
        event=Event.objects(id=event_id)[0]
        if event.photo is not None: # If event has photo it must be deleted
            strg=storage()
            getattr(strg,settings.STORAGE+"_remove")(event.photo)

        event.delete()
        return HttpResponseRedirect("/")

return render(request,'eventslist/deleteevent.html', {
    'id': event_id,
})

#####

@login_required
def editevent(request,event_id):
    if request.method == 'POST':
        form = EventForm(request.POST, request.FILES)
        if form.is_valid():
            title=form.cleaned_data['title']
            description=form.cleaned_data['description']
            category=form.cleaned_data['category']
            photo=Event.objects(id=event_id)[0].photo
            lat=form.cleaned_data['lat']
            lng=form.cleaned_data['lng']
            user=request.user.username

            event =
Event(id=event_id,title=title,description=description,category=category,photo=photo,user
=user)

            if (request.FILES): # If a photo must be added to event
                strg=storage()
                if photo is not None: # If event had photo it must be deleted
                    getattr(strg,settings.STORAGE+"_remove")(photo)

                photo_name=request.user.username+"-"+str(int(time()))+".jpeg"
                f=request.FILES["photo"]
                getattr(strg,settings.STORAGE+"_write")(photo_name,f)
                event.photo=photo_name

```

```

        event.location=[float(lat),float(lng)]
        event.save()
        messages.success(request, title + _(' has been updated'))
        return HttpResponseRedirect("/")
    else:
        form = EventForm()
        event=Event.objects(id=event_id)[0]
        form.fields["title"].initial = event.title
        form.fields["description"].initial = event.description
        form.fields["category"].initial = event.category
        form.fields["lat"].initial = event.location['coordinates'][0]
        form.fields["lng"].initial = event.location['coordinates'][1]

    return render(request,'eventslist/editevent.html', {
        'form': form, 'photo': Event.objects(id=event_id)[0].photo, 'id': event_id,
    })

```

10.3.2 Fichero "models.py"

```

from mongoengine.queryset import QuerySet
from mongoengine import *
from django.conf import settings
import datetime

connect('worldevents')

#####

class Comment(EmbeddedDocument):
    content = StringField()
    user = StringField()
    added_date = DateTimeField(default=datetime.datetime.now)

    def __str__(self):
        return self.content

#####

class Event(Document):

    title = StringField(max_length = 64)
    photo = StringField(max_length = 64)
    location = PointField()
    description = StringField()
    category = StringField()
    added_date = DateTimeField(default=datetime.datetime.now)
    comments = ListField(EmbeddedDocumentField(Comment))
    user = StringField()

    # Search query
    @queryset_manager
    def search(doc_cls, queryset, title, category, lat, lng, distance, num_events):
        if title and category:
            return
            queryset(Q(location__geo_within_sphere=[(float(lat), float(lng)), float(distance)/6371]) &
            Q(title__icontains=title) & Q(category=category)).order_by('-added_date')[num_events-
            15:num_events]

            elif title and not category:
                return
            queryset(Q(location__geo_within_sphere=[(float(lat), float(lng)), float(distance)/6371]) &
            Q(title__icontains=title)).order_by('-added_date')[num_events-15:num_events]

            elif not title and category:
                return
            queryset(Q(location__geo_within_sphere=[(float(lat), float(lng)), float(distance)/6371]) &
            Q(category=category)).order_by('-added_date')[num_events-15:num_events]

            elif not title and not category:

```

```

        return
        queryset(Q(location__geo_within_sphere=(float(lat),float(lng)),float(distance)/6371)).
        order_by('-added_date')[num_events-15:num_events]

    def __str__(self):
        return self.title +'-
'+str(self.location['coordinates'][0])+str(self.location['coordinates'][1])+'-
'+self.description +'-'+self.category

    meta = {'collection':'event'}

#####
##333

class Category(Document):
    name = StringField(max_length = 64, required = True, unique = True)

    def __str__(self):
        return self.name

    meta = {'collection':'category'}

```

10.3.3 Fichero "forms.py"

```

from django import forms
from django.contrib.auth import authenticate, get_user_model
from django.utils.text import capfirst
from django.forms.widgets import Input
from mongoengine.django.auth import User
from eventslist.models import Category
from django.utils.translation import ugettext_lazy as _

#####

class RangeInput(Input):
    input_type='range'

#####

class SearchForm(forms.Form):
    error_messages = {
        'location_not_selected': _("You must select a location"),
        'distance_out_of_range': _("Distance must be between 1 and 100"),
    }

    title = forms.CharField(label=_("Title"), max_length=30, required=False)
    category = forms.ChoiceField(label=_("Category"), required=False,
        widget=forms.Select)
    lat = forms.CharField(required=False,widget=forms.HiddenInput)
    lng = forms.CharField(required=False,widget=forms.HiddenInput)
    distance =
forms.CharField(label=_("Distance"),widget=RangeInput(attrs={'min':'1','max':'100','valu
e':'1'}))

    # Load categories from database
    def __init__(self, *args, **kwargs):
        super(SearchForm, self).__init__(*args, **kwargs)
        choices = [(unicode(pt), unicode(pt)) for pt in Category.objects.all()]
        self.fields['category'].choices = choices
        self.fields['category'].choices.insert(0, ('','_('Select category'))))

    # Check location data
    def clean(self):
        cleaned_data = super(SearchForm,self).clean()
        lat = cleaned_data.get("lat")
        lng = cleaned_data.get("lng")
        distance = cleaned_data.get("distance")

        if not lat or not lng:

```

```

        raise forms.ValidationError(self.error_messages['location_not_selected'])

    if (int(distance)<1) or (int(distance)>100):
        raise forms.ValidationError(self.error_messages['distance_out_of_range'])

    return cleaned_data

#####
####33

class RegistrationForm(forms.Form):
    error_messages = {
        'duplicate_username': _("A user with that username already exists."),
        'password_mismatch': _("The two password fields didn't match."),
    }
    username = forms.RegexField(label=_("Username"), max_length=30,
        regex=r'^[\w.@+-]+$',
        help_text=_("Required. 30 characters or fewer. Letters, digits and "
            "@/./+/-/_ only."),
        error_messages={
            'invalid': _("This value may contain only letters, numbers and "
                "@/./+/-/_ characters.")})
    password1 = forms.CharField(label=_("Password"), widget=forms.PasswordInput)
    password2 = forms.CharField(label=_("Password confirmation"),
        widget=forms.PasswordInput,
        help_text=_("Enter the same password as above, for verification.))

    # Check passwords are equal and username don't exist
    def clean(self):
        cleaned_data = super(RegistrationForm, self).clean()
        user_name = cleaned_data.get("username")
        password1 = cleaned_data.get("password1")
        password2 = cleaned_data.get("password2")
        if password1!=password2:
            raise forms.ValidationError(self.error_messages['password_mismatch'])

        if User.objects(username=user_name).count()==1:
            raise forms.ValidationError(self.error_messages['duplicate_username'])

        return cleaned_data

#####
####

class CommentForm(forms.Form):
    content = forms.CharField(label="Comment", widget=forms.Textarea)

#####
###

class EventForm(forms.Form):
    error_messages = {
        'location_not_selected': _("You must select a location"),
    }

    title = forms.CharField(label=_("Title"), max_length=30)
    description = forms.CharField(label=_("Description"), widget=forms.Textarea)
    category = forms.ChoiceField(label=_("Category"), widget=forms.Select)
    photo = forms.ImageField(required=False)
    lat = forms.CharField(required=False, widget=forms.HiddenInput)
    lng = forms.CharField(required=False, widget=forms.HiddenInput)

    # Load categories from database
    def __init__(self, *args, **kwargs):
        super(EventForm, self).__init__(*args, **kwargs)
        choices = [(unicode(pt), unicode(pt)) for pt in Category.objects.all()]
        self.fields['category'].choices = choices
        self.fields['category'].choices.insert(0, ('', 'Select category'))

    # Check if location is selected
    def clean(self):
        cleaned_data = super(EventForm, self).clean()
        lat = cleaned_data.get("lat")
        lng = cleaned_data.get("lng")

        if not lat or not lng:
            raise forms.ValidationError(self.error_messages['location_not_selected'])

```

```
return cleaned_data
```

10.3.4 Fichero "urls.py"

```
from django.conf.urls import patterns, include, url
from eventslist import views

urlpatterns = patterns('',
    url(r'^deleteevent/(\w+)$', views.deleteevent, name='deleteevent'),
    url(r'^deleteevent$', views.deleteevent, name='deleteevent'),
    url(r'^editevent/(\w+)$', views.editevent, name='editevent'),
    url(r'^editevent$', views.editevent, name='editevent'),
    url(r'^searchevents$', views.searchevents, name='searchevents'),
    url(r'^addevent$', views.addevent, name='addevent'),
    url(r'^addcomment$', views.addcomment, name='addcomment'),
    url(r'^register$', views.register, name='register'),
    url(r'^login$', views.loginpage, name='login'),
    url(r'^logout$', views.logoutpage, name='logout'),
    url(r'^(\d+)$', views.home, name='home'),
    url(r'^$', views.home, name='home'),
)
```

10.3.5 Fichero "storages.py"

```
from django.conf import settings
import os
import boto

class storage:
    def local_remove(self, photo_name):
        os.remove(settings.MEDIA_ROOT+photo_name)

    def local_write(self, photo_name, f):
        with open(settings.MEDIA_ROOT+photo_name, 'wb+') as destination:
            for chunk in f.chunks():
                destination.write(chunk)

    def s3_write(self, photo_name, f):
        s3 =
boto.connect_s3(settings.AWS_ACCESS_KEY_ID, settings.AWS_SECRET_ACCESS_KEY)
        bucket = s3.get_bucket(settings.BUCKET_NAME)
        key = bucket.new_key('media/'+photo_name)
        key.set_contents_from_file(f)
        key.set_acl('public-read')

    def s3_remove(self, photo_name):
        print photo_name
        s3 =
boto.connect_s3(settings.AWS_ACCESS_KEY_ID, settings.AWS_SECRET_ACCESS_KEY)
        key = s3.get_bucket(settings.BUCKET_NAME).get_key('media/'+photo_name)
        print key
        if key.exists:
            key.delete()
```

10.3.6 Fichero "settings.py"

```
# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
```

```
import os
BASE_DIR = os.path.dirname(os.path.dirname(__file__))

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'dkh!=y2d5@odkfngnz$(xotn=r-9(!gyddqo&u*#-!4f%t^ae!'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

if DEBUG:
    TEMPLATE_DEBUG = True
else:
    TEMPLATE_DEBUG = False

ALLOWED_HOSTS = ['localhost']

# Application definition

INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'eventslist',
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.locale.LocaleMiddleware'
)

ROOT_URLCONF = 'worldevents.urls'

LOGIN_URL = '/eventslist/login'

WSGI_APPLICATION = 'worldevents.wsgi.application'

#Test
TEST_RUNNER = 'worldevents.tests.MongoTestSuiteRunner'
_MONGODB_TEST_NAME = 'db_test'
TEST_MODE = False

AUTHENTICATION_BACKENDS = (
    'mongoengine.django.auth.MongoEngineBackend',
)

# Internationalization
# https://docs.djangoproject.com/en/1.6/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

_ = lambda s: s

LANGUAGES = (
    ('es', _('Español')),
    ('en', _('English')),
)
```



```

LOCALE_PATHS = (
    os.path.join(BASE_DIR, "locale"),
)

#S3 STORAGE

AWS_ACCESS_KEY_ID = ''
AWS_SECRET_ACCESS_KEY = ''
BUCKET_NAME = 'worldevents_static'

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.6/howto/static-files/

if DEBUG:

    STATIC_URL = '/static/'

    STATICFILES_DIRS = (os.path.join(BASE_DIR, "static"),)

    MEDIA_ROOT=os.path.join(BASE_DIR, "static/media/")

    STORAGE = "local"

else:

    STATIC_URL = 'https://s3.amazonaws.com/worldevents_static/'

    STATICFILES_DIRS = 'https://s3.amazonaws.com/worldevents_static/'

    MEDIA_ROOT='https://s3.amazonaws.com/worldevents_static/media/'

    STORAGE = "s3"

# Database
# https://docs.djangoproject.com/en/1.6/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': ''
    }
}

#log
LOGGING = {
    'version': 1,
}

#MongoDatabase

if DEBUG:
    _MONGODB_HOST = 'localhost'
else:
    _MONGODB_HOST = ''

_MONGODB_NAME = 'worldevents'
_MONGODB_DATABASE_HOST = \
    'mongodb://%s/%s' \
    % (_MONGODB_HOST, _MONGODB_NAME)

from mongoengine import connect
connect(_MONGODB_NAME, host=_MONGODB_DATABASE_HOST)

SESSION_ENGINE = 'mongoengine.django.sessions'

```

10.3.7 Fichero "tests.py"

```

from django.test import LiveServerTestCase,Client
from django.core.urlresolvers import resolve

```

```

from eventslist.views import
home, register, loginpage, logoutpage, addevent, searchevents, editevent, deleteevent, addcommen
t
from eventslist.models import Event, Category
from django.http import HttpRequest
from django.template.loader import render_to_string
from eventslist.forms import RegistrationForm, EventForm, SearchForm, CommentForm
from django.contrib.auth import login
from django.contrib.auth.forms import AuthenticationForm
from django.utils.html import escape
from mongoengine.django.auth import User
from pymongo import Connection
import unittest
from auth import PasswordUtils
import os
from django.conf import settings

class HomeTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_root_resolves_home_view(self):
        page=resolve('/')
        self.assertEqual(page.func, home)

    def test_home_page_renders_home_template(self):
        response = self.client.get('/')
        self.assertTemplateUsed(response, 'eventslist/home.html')

    def test_home_page_show_events_list(self):
        response = self.client.get('/')
        num_events = response.content.count('</tr>')
        self.assertEqual(Event.objects().count(), num_events)

    def test_home_page_show_events_pages(self):
        for i in range(45):
            response = self.client.post('/eventslist/addevent', data={'title':
'Concierto'+str(i), 'description': 'Es un concierto', 'category':
'Musica', 'lat': '49.8', 'lng': '4.7'})
            response = self.client.get('/')
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)
            expected_input = escape('<td>Concierto44</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<td>Concierto30</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/30">')
            self.assertIn(expected_input, escape(response.content))
            response = self.client.get('/eventslist/30')
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)
            expected_input = escape('<td>Concierto29</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<td>Concierto15</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/45">')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/15">')
            self.assertIn(expected_input, escape(response.content))
            response = self.client.get('/eventslist/15')
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)

```

```

expected_input = escape('<td>Concierto44</td>')
self.assertIn(expected_input,escape(response.content))
expected_input = escape('<td>Concierto30</td>')
self.assertIn(expected_input,escape(response.content))
expected_input = escape('<a href="/eventslist/30">')
self.assertIn(expected_input,escape(response.content))
expected_input = escape('<a href="/eventslist/30">')
self.assertIn(expected_input,escape(response.content))

def test_events_details_are_in_home_page(self):
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7'})
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto2','description': 'Es un concierto2','category':
'Musica','lat':'49.8','lng':'4.7'})
    self.assertEqual(Event.objects(title='Concierto').count(),1)
    self.assertEqual(str(Event.objects(title='Concierto')),'[<Event: Concierto-
49.84.7-Es un concierto-Musica>]')
    self.assertEqual(Event.objects(title='Concierto2').count(),1)
    self.assertEqual(str(Event.objects(title='Concierto2')),'[<Event: Concierto2-
49.84.7-Es un concierto2-Musica>]')
    self.assertEqual(Event.objects().count(),2)
    response = self.client.get('/')
    expected_input = escape('<td>Concierto</td>')
    self.assertIn(expected_input,escape(response.content))
    expected_input = escape('<td>Concierto2</td>')
    self.assertIn(expected_input,escape(response.content))
    expected_input = escape('<li>Concierto</li>')
    self.assertIn(expected_input,escape(response.content))
    expected_input = escape('<li>Concierto2</li>')
    self.assertIn(expected_input,escape(response.content))

def test_events_photos_are_in_home_page(self):
    myphoto = open('static/media/prueba.jpeg','r')
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7','photo':myphoto})
    self.assertEqual(Event.objects(title='Concierto').count(),1)
    self.assertEqual(str(Event.objects(title='Concierto')),'[<Event: Concierto-
49.84.7-Es un concierto-Musica>]')
    response = self.client.get('/')
    expected_input = escape(Event.objects(title='Concierto')[0].photo)
    self.assertIn(expected_input,escape(response.content))
    os.remove(settings.MEDIA_ROOT+str(Event.objects(title='Concierto')[0].photo))

class RegisterTest(LiveServerTestCase):
    def _fixture_setup(self):
        pass
    def _fixture_teardown(self):
        User.objects(username='john').delete()
        User.objects(username='bob').delete()

    def test_register_link_resolves_register_view(self):
        page=resolve('/eventslist/register')
        self.assertEqual(page.func, register)

    def test_register_page_returns_correct_html(self):
        request = HttpRequest()
        response = register(request)
        expected_html = render_to_string('eventslist/register.html',{'form':
RegistrationForm()})
        self.assertEqual(response.content.decode(), expected_html)

    def test_register_page_uses_registration_form(self):
        response = self.client.get('/eventslist/register')
        self.assertIsInstance(response.context['form'], RegistrationForm)

    def test_register_page_save_user_to_database(self):
        response = self.client.post('/eventslist/register',data={'username':
'john','password1': 'john','password2': 'john'})
        self.assertEqual(User.objects(username='john').count(),1)

```

```

def test_register_page_send_success_message_to_home(self):
    response = self.client.post('/eventslist/register', data={'username':
'bob', 'password1': 'bob', 'password2': 'bob'}, follow=True)
    expected_html='you have been successfully registered'
    self.assertIn(expected_html, response.content)

class RegistrationFormTest(LiveServerTestCase):
    def _fixture_setup(self):
        pass
    def _fixture_teardown(self):
        User.objects(username='john').delete()

    def test_validation_errors_are_sent_back_to_register_template(self):
        response = self.client.post('/eventslist/register')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'eventslist/register.html')
        expected_error = escape("This field is required.")
        self.assertContains(response, expected_error)

    def test_for_invalid_input_passes_form_to_template(self):
        response = self.client.post('/eventslist/register', data={'username': '$'})
        self.assertIsInstance(response.context['form'], RegistrationForm)
        expected_input = escape('<input id="id_username" maxlength="30" name="username"
type="text" value="$" />')
        self.assertIn(expected_input, escape(response.content))

    def test_form_validation_for_blank_items(self):
        form = RegistrationForm(data={'username': '', 'password1': '', 'password2': ''})
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['username'], ["This field is required."])
        self.assertEqual(form.errors['password1'], ["This field is required."])
        self.assertEqual(form.errors['password2'], ["This field is required."])

    def test_form_validation_for_different_passwords(self):
        form = RegistrationForm(data={'username': 'John', 'password1': 'a', 'password2':
'b'})
        self.assertFalse(form.is_valid())
        self.assertEqual(form.non_field_errors(), ["The two password fields didn't
match."])

    def test_form_validation_for_duplicate_username(self):
        response = self.client.post('/eventslist/register', data={'username':
'john', 'password1': 'john', 'password2': 'john'})
        response = self.client.post('/eventslist/register', data={'username':
'john', 'password1': 'john', 'password2': 'john'})
        expected_input = escape('A user with that username already exists.')
        self.assertIn(expected_input, escape(response.content))

class LoginTest(LiveServerTestCase):
    def _fixture_setup(self):
        user=User.create_user('john', 'john')
    def _fixture_teardown(self):
        User.objects(username='john').delete()

    def test_login_link_resolves_login_view(self):
        page=resolve('/eventslist/login')
        self.assertEqual(page.func, loginpage)

    def test_login_page_returns_correct_html(self):
        request = HttpRequest()
        response = loginpage(request)
        expected_html = render_to_string('eventslist/login.html', {
'form': AuthenticationForm()})
        self.assertEqual(response.content.decode(), expected_html)

    def test_login_page_uses_authentication_form(self):
        response = self.client.get('/eventslist/login')
        self.assertIsInstance(response.context['form'], AuthenticationForm)

    def test_login_page_login_user(self):

```

```

        response = self.client.post('/eventslist/login',data={'username':
'john','password': 'john'})
        self.assertIn(response.content.decode(),'john')

class LoginFormTest(LiveServerTestCase):
    def _fixture_setup(self):
        pass
    def _fixture_teardown(self):
        pass

    def test_form_validation_for_blank_items(self):
        response = self.client.post('/eventslist/login',data={'username': '', 'password':
''})
        expected_input = escape('Login incorrecto')
        self.assertIn(expected_input,escape(response.content))

    def test_form_validation_for_blank_username(self):
        response = self.client.post('/eventslist/login',data={'username': '', 'password':
'111'})
        expected_input = escape('Login incorrecto')
        self.assertIn(expected_input,escape(response.content))

    def test_form_validation_for_blank_password(self):
        response = self.client.post('/eventslist/login',data={'username':
'111','password': ''})
        expected_input = escape('Login incorrecto')
        self.assertIn(expected_input,escape(response.content))

    def test_form_validation_for_wrong_data(self):
        response = self.client.post('/eventslist/login',data={'username':
'111','password': '111'})
        expected_input = escape('Login incorrecto')
        self.assertIn(expected_input,escape(response.content))

class LogoutTest(LiveServerTestCase):
    def _fixture_setup(self):
        User.create_user('john','john')
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        User.objects(username='john').delete()

    def test_logout_link_resolves_logout_view(self):
        page=resolve('/eventslist/logout')
        self.assertEqual(page.func, logoutpage)

    def test_user_can_logout(self):
        self.client.logout()
        response = self.client.get('/')
        expected_input = escape('Login')
        self.assertIn(expected_input,escape(response.content))

class AddEventTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_addevent_page_requires_login_user(self):
        self.client.logout()
        response = self.client.get('/eventslist/addevent',follow=True)

```

```

self.assertEqual(response.status_code,200)
self.assertRedirects(response,'/eventslist/login?next=/eventslist/addevent')

def test_addevent_link_resolves_addevent_view(self):
    page=resolve('/eventslist/addevent')
    self.assertEqual(page.func, addevent)

def test_addevent_page_uses_addevent_form(self):
    response = self.client.get('/eventslist/addevent')
    self.assertIsInstance(response.context['form'], EventForm)

def test_addevent_page_save_event_to_database_and_create_photo_file(self):
    myphoto = open('static/media/prueba.jpeg','r')
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7','photo':myphoto})
    self.assertEqual(Event.objects(title='Concierto').count(),1)
    self.assertEqual(str(Event.objects(title='Concierto')),'[<Event: Concierto-
49.84.7-Es un concierto-Musica>]')
    self.assertTrue(str(Event.objects(title='Concierto')[0].photo).find('john')==0)
    self.assertTrue(os.path.isfile(settings.MEDIA_ROOT+str(Event.objects(title='Conci
erto')[0].photo)))
    os.remove(settings.MEDIA_ROOT+str(Event.objects(title='Concierto')[0].photo))

def test_addevent_page_send_success_message_to_home(self):
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un
concierto','category':'Musica','lat':'49.8','lng':'4.7'},follow=True)
    expected_html='has been created'
    self.assertIn(expected_html,response.content)

class EventFormTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_event_form_load_categories_from_db(self):
        response = self.client.get('/eventslist/addevent')
        for c in Category.objects().all() :
            self.assertIn('<option
value="'+c.name+'"'>'+c.name+'</option>',response.content.decode())

    def test_validation_errors_are_sent_back_to_addevent_template(self):
        response = self.client.post('/eventslist/addevent')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'eventslist/addevent.html')
        expected_error = escape("This field is required.")
        self.assertContains(response, expected_error)

    def test_for_invalid_input_passes_form_to_template(self):
        response = self.client.post('/eventslist/addevent',data={'title':
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'})
        self.assertIsInstance(response.context['form'], EventForm)
        expected_input = escape('<input id="id_title" maxlength="30" name="title"
type="text" value="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" />')
        self.assertIn(expected_input,escape(response.content))

    def test_form_validation_for_blank_items(self):
        response = self.client.post('/eventslist/addevent',data={'title':
'', 'description': '', 'category': '', 'lat': '', 'lng': ''})
        expected_input = escape('This field is required.')

```

```

self.assertIn(expected_input, escape(response.content))

def test_form_validation_for_blank_location(self):
    response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un concierto', 'category': 'Musica', 'lat': '', 'lng': ''})
    expected_input = escape('You must select a location')
    self.assertIn(expected_input, escape(response.content))

class EditEventTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_editevent_page_requires_login_user(self):
        self.client.logout()
        response = self.client.get('/eventslist/editevent', follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertRedirects(response, '/eventslist/login?next=/eventslist/editevent')

    def test_editevent_link_resolves_editevent_view(self):
        page=resolve('/eventslist/editevent')
        self.assertEqual(page.func, editevent)

    def test_editevent_page_uses_editevent_form(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        event_id=Event.objects(title='Concierto')
        response = self.client.get('/eventslist/editevent/'+str(event_id[0].id))
        self.assertIsInstance(response.context['form'], EventForm)

    def test_editevent_page_save_event_to_database_save_photo_file(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        event_id=Event.objects(title='Concierto')
        event_id_old=event_id[0].id
        myphoto = open('static/media/prueba.jpeg', 'r')
        response =
self.client.post('/eventslist/editevent/'+str(event_id[0].id), data={'title':
'ConciertoModificado', 'description': 'Es un concierto
Modificado', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7', 'photo': myphoto}, follow=True)
        self.assertEqual(Event.objects(title='Concierto').count(), 0)
        self.assertEqual(Event.objects(title='ConciertoModificado').count(), 1)
        self.assertEqual(str(Event.objects(title='ConciertoModificado')), '<Event:
ConciertoModificado-49.84.7-Es un concierto Modificado-Musica>')
        event_id_modificado=Event.objects(title='ConciertoModificado')
        self.assertEqual(event_id_old, event_id_modificado[0].id)
        self.assertTrue(str(Event.objects(title='ConciertoModificado')[0].photo).find('jo
hn') != -1)
        self.assertTrue(os.path.isfile(settings.MEDIA_ROOT+str(Event.objects(title='Conci
ertoModificado')[0].photo)))
        os.remove(settings.MEDIA_ROOT+str(Event.objects(title='ConciertoModificado')[0].p
hoto))

```

```

def test_editevent_page_send_success_message_to_home(self):
    response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'}, follow=True)
    event_id=Event.objects(title='Concierto')
    response =
self.client.post('/eventslist/editevent/'+str(event_id[0].id), data={'title':
'ConciertoModificado', 'description': 'Es un concierto
Modificado', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'}, follow=True)
    expected_html='has been updated'
    self.assertIn(expected_html, response.content)

class EditEventFormTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john', 'john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john', password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_edit_event_form_load_categories_from_db(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        event_id=Event.objects(title='Concierto')
        response = self.client.get('/eventslist/editevent/'+str(event_id[0].id))
        for c in Category.objects().all():
            self.assertIn('<option value="'+c.name+"'", response.content.decode())

    def test_validation_errors_are_sent_back_to_editevent_template(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        event_id=Event.objects(title='Concierto')
        response =
self.client.post('/eventslist/editevent/'+str(event_id[0].id), data={'title':
'', 'description': '', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'}, follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'eventslist/editevent.html')
        expected_error = escape("This field is required.")
        self.assertContains(response, expected_error)

    def test_data_are_sent_to_editevent_template(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un
concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        event_id=Event.objects(title='Concierto')
        event_id_old=event_id[0].id
        response = self.client.get('/eventslist/editevent/'+str(event_id[0].id))
        self.assertIn('Concierto', escape(response.content))
        self.assertIn('Es un concierto', escape(response.content))
        self.assertIn('49.8', escape(response.content))
        self.assertIn('4.7', escape(response.content))

class DeleteEventTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()

```



```

c=Category(name='Restauracion')
c.save()
user=User.create_user('john','john')
request = HttpRequest()
user=User.objects(username='john')
self.client.login(username='john',password='john')

def _fixture_teardown(self):
    Category.objects.all().delete()
    Event.objects.all().delete()
    self.client.logout()
    User.objects(username='john').delete()

def test_deleteevent_page_requires_login_user(self):
    self.client.logout()
    response = self.client.get('/eventslist/deleteevent',follow=True)
    self.assertEqual(response.status_code,200)
    self.assertRedirects(response,'/eventslist/login?next=/eventslist/deleteevent')

def test_deleteevent_link_resolves_deleteevent_view(self):
    page=resolve('/eventslist/deleteevent')
    self.assertEqual(page.func, deleteevent)

def test_deleteevent_page_delete_event_from_database_and_photo_file(self):
    myphoto = open('static/media/prueba.jpeg','r')
    response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7','photo':myphoto})
    self.assertEqual(Event.objects(title='Concierto').count(),1)
    self.assertEqual(str(Event.objects(title='Concierto')),'[<Event: Concierto-
49.84.7-Es un concierto-Musica>]')
    event_id=Event.objects(title='Concierto')
    photo_file=event_id[0].photo
    response = self.client.post('/eventslist/deleteevent/'+str(event_id[0].id))
    self.assertEqual(Event.objects(title='Concierto').count(),0)
    self.assertFalse(os.path.isfile(settings.MEDIA_ROOT+str(photo_file)))

class SearchTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_searchevents_link_resolves_searchevents_view(self):
        page=resolve('/eventslist/searchevents')
        self.assertEqual(page.func, searchevents)

    def test_search_page_uses_search_form(self):
        response = self.client.get('/eventslist/searchevents')
        self.assertIsInstance(response.context['form'], SearchForm)

    def test_search_form_return_correct_events(self):
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')
        '''AnADIR MAS PRUEBAS DE BUSQUEDA'''
        response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7'})
        self.assertEqual(Event.objects(title='Concierto').count(),1)
        self.assertEqual(str(Event.objects(title='Concierto')),'[<Event: Concierto-
49.84.7-Es un concierto-Musica>]')

```

```

        response = self.client.post('/eventslist/searchevents', data={'title':
'Concierto', 'category': 'Musica', 'lat': '49.8', 'lng': '4.7', 'distance': '4'}, follow=True)
        self.assertContains(response, '<li>Concierto</li>')
        self.assertContains(response, '<li>Es un concierto</li>')
        self.assertContains(response, '<li>Musica</li>')
        User.objects(username='john').delete()

    def test_search_show_events_pages(self):
        user=User.create_user('john', 'john')
        user=User.objects(username='john')
        self.client.login(username='john', password='john')
        for i in range(45):
            response = self.client.post('/eventslist/addevent', data={'title':
'Concierto'+str(i), 'description': 'Es un concierto', 'category':
'Musica', 'lat': '49.8', 'lng': '4.7'})
            response =
self.client.post('/eventslist/searchevents', data={'lat': '49.8', 'lng': '4.7', 'distance': '4
'}, follow=True)
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)
            expected_input = escape('<td>Concierto44</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<td>Concierto30</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/30">')
            self.assertIn(expected_input, escape(response.content))
            response = self.client.get('/eventslist/30')
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)
            expected_input = escape('<td>Concierto29</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<td>Concierto15</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/45">')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/15">')
            self.assertIn(expected_input, escape(response.content))
            response = self.client.get('/eventslist/15')
            num_events = response.content.count('</tr>')
            self.assertEqual(num_events, 15)
            expected_input = escape('<td>Concierto44</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<td>Concierto30</td>')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/30">')
            self.assertIn(expected_input, escape(response.content))
            expected_input = escape('<a href="/eventslist/30">')
            self.assertIn(expected_input, escape(response.content))
            User.objects(username='john').delete()

class SearchFormTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john', 'john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john', password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_search_form_load_categories_from_db(self):
        response = self.client.get('/eventslist/searchevents')
        for c in Category.objects().all():
            self.assertIn('<option
value="'+c.name+'"'>'+c.name+'</option>', response.content.decode())

    def test_validation_errors_are_sent_back_to_searchevents_template(self):

```

```

        response = self.client.post('/eventslist/searchevents')
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'eventslist/searchevents.html')
        expected_error = escape("This field is required.")
        self.assertContains(response, expected_error)

    def test_for_invalid_input_passes_form_to_template(self):
        response = self.client.post('/eventslist/searchevents', data={'title':
'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'})
        self.assertIsInstance(response.context['form'], SearchForm)
        expected_input = escape('Ensure this value has at most 30 characters (it has
34)')
        self.assertIn(expected_input, escape(response.content))

    def test_form_validation_for_blank_location(self):
        response = self.client.post('/eventslist/searchevents', data={'title':
'', 'category': '', 'lat': '', 'lng': '', 'distance': ''})
        expected_input = escape('You must select a location')
        self.assertIn(expected_input, escape(response.content))

    def test_form_validation_for_right_distance(self):
        response = self.client.post('/eventslist/searchevents', data={'title':
'', 'category': '', 'lat': '2', 'lng': '2', 'distance': '200'})
        expected_input = escape('Distance must be between 1 and 100')
        self.assertIn(expected_input, escape(response.content))

class AddCommentTest(LiveServerTestCase):
    def _fixture_setup(self):
        c=Category(name='Musica')
        c.save()
        c=Category(name='Restauracion')
        c.save()
        user=User.create_user('john','john')
        request = HttpRequest()
        user=User.objects(username='john')
        self.client.login(username='john',password='john')

    def _fixture_teardown(self):
        Category.objects.all().delete()
        Event.objects.all().delete()
        self.client.logout()
        User.objects(username='john').delete()

    def test_addcomment_requires_login_user(self):
        self.client.logout()
        response = self.client.get('/eventslist/addcomment', follow=True)
        self.assertEqual(response.status_code, 200)
        self.assertRedirects(response, '/eventslist/login?next=/eventslist/addcomment')

    def test_addcomment_link_resolves_addcomment_view(self):
        page=resolve('/eventslist/addcomment')
        self.assertEqual(page.func, addcomment)

    def test_addcomment_page_uses_addcomment_form(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un concierto', 'category':
'Musica', 'lat': '49.8', 'lng': '4.7'})
        response = self.client.post('/eventslist/addcomment', data={'event_id':
str(Event.objects(title='Concierto')[0].id)})
        self.assertIsInstance(response.context['form'], CommentForm, )

    def test_addcomment_page_save_comment_to_database(self):
        response = self.client.post('/eventslist/addevent', data={'title':
'Concierto', 'description': 'Es un concierto', 'category':
'Musica', 'lat': '49.8', 'lng': '4.7'})
        response = self.client.post('/eventslist/addcomment', data={'content': 'Esto es un
comentario', 'event_id': str(Event.objects(title='Concierto')[0].id)})
        self.assertEqual(Event.objects(title='Concierto').count(), 1)
        self.assertEqual(str(Event.objects(title='Concierto')), '<Event: Concierto-
49.84.7-Es un concierto-Musica>')
        self.assertEqual(str(Event.objects(title='Concierto')[0].comments[0]), 'Esto es un
comentario')

    def test_comments_are_in_home_page(self):

```

```

        response = self.client.post('/eventslist/addevent',data={'title':
'Concierto','description': 'Es un concierto','category':
'Musica','lat':'49.8','lng':'4.7'})
        response = self.client.post('/eventslist/addcomment',data={'content':'Esto es un
comentario','event_id': str(Event.objects(title='Concierto')[0].id)})
        self.assertEqual(Event.objects(title='Concierto').count(),1)
        self.assertEqual(str(Event.objects(title='Concierto')),' [<Event: Concierto-
49.84.7-Es un concierto-Musica>]')
        response = self.client.get('/')
        expected_html='Esto es un comentario'
        self.assertIn(expected_html,response.content)
    
```

10.3.8 Fichero "addevent.html"

```

{% load i18n %}
<h1>{% trans "Add Event" %}</h1>

        <form enctype="multipart/form-data" action="{% url 'addevent' %}"
method="post">
            {{ form.as_p }}
            
            <p>{% trans "Select location in the map" %}</p>
            <div id="map_addevent"></div>
            {% csrf_token %}
            <input type="submit" value="{% trans "add" %}">
        </form>

<script type="text/javascript">
// Draw map
$( document ).ready(function() {
red_icon="http://maps.google.com/mapfiles/ms/icons/red-dot.png"
$("#map_addevent").width("300px").height("150px").gmap3({
map:{
options:{
center:[46.578498,2.457275],
zoom: 3
},
events:{
click: function(map,event) {
$("#id_lat").val(event.latLng.lat());
$("#id_lng").val(event.latLng.lng());
drawMarker(event.latLng.lat(),event.latLng.lng())
}
}
}
});

//if location already selected
if (($("#id_lat").val())&&($("#id_lng").val())){
drawMarker($("#id_lat").val(),$("#id_lng").val())
}

//Put selected image in form
$("#input[type='file']").bind("change",function(){
file=$(this).prop("files")[0]
img = $("#form img[alt='preview image']")
var reader = new FileReader();
reader.onload = function(e){
img.attr('src',e.target.result)
}
reader.readAsDataURL(file);
});

//Draw marker when user has clicked
function drawMarker(lat,lng){
$("#map_addevent").gmap3({
clear:{
tag:["selected"]
}
}
}
    
```

```

        },
        marker: {
            latLng: [lat, lng],
            tag: "selected",
            options: {
                draggable: true,
                icon: red_icon
            }
        }
    });
}

});
</script>

```

10.3.9 Fichero "base.html"

```

{% load staticfiles %}
{% load i18n %}
<!DOCTYPE html>
<html>
  <head>
    <title>{% trans "WorldEvents" %}</title>
    <meta charset="uft-8">
    {% block stylesheets %}
    {% endblock %}
    {% block javascript %}
    {% endblock %}
  </head>
  <body>
    <header>
      <h1>{% trans "WorldEvents" %}</h1>
      <nav>
        {% block nav %}
        {% endblock %}
      </nav>

      <div id='messages'>
        {% if messages %}
          <ul class="messages">
            {% for message in messages %}
              <li{% if message.tags %}
                class="{{ message.tags }}"
              {% endif %}>{{ message }}
            </li>
            {% endfor %}
          </ul>
        {% endif %}
      </div>
    </header>
    {% block content %}
    {% endblock %}

    {% block scripts %}
    {% endblock %}
  </body>
</html>

```

10.3.10 Fichero "comments.html"

```

{% load i18n %}
        {% if user.is_authenticated %}

```

```

        <a href='comment'>{% trans "Add comment" %} </a>

        <form action="{% url 'addcomment' %}" method="post">
            {{ form.as_p }}
            {% csrf_token %}
            <input type="submit" value="{% trans "add" %}">
        </form>
    {% endif %}

    {% if event.comments %}
        <ul>
            {% for comment in event.comments%}
                <li>
                    <p>{{comment.user}}</p>
                    <p>{{comment.content}}</p>
                </li>
            {% endfor %}
        </ul>
    {% endif %}

```

10.3.11 Fichero "deleteevent.html"

```

{% load i18n %}
<h1>{% trans "Delete Event" %}</h1>

    <form action="{% url 'eventslist.views.deleteevent' %}/{{ id }}" method="post">
        <p>{% trans "Delete Event, ¿are you sure?" %}</p>
        {% csrf_token %}
        <input type="submit" value="{% trans "Delete" %}">
        <input type="button" value="{% trans "Cancel" %}">
    </form>

```

10.3.12 Fichero "editevent.html"

```

{% load staticfiles %}
{% load i18n %}
<h1>{% trans "Edit Event" %}</h1>

    <form enctype="multipart/form-data" action="{% url 'eventslist.views.editevent' %}/{{ id }}" method="post">
        {{ form.as_p }}
        
        <p>{% trans "Select location in the map" %}</p>
        <div id="map_addevent"></div>
        {% csrf_token %}
        <input type="submit" value="{% trans "Save" %}">
    </form>

<script type="text/javascript">

    $( document ).ready(function() {
        red_icon="http://maps.google.com/mapfiles/ms/icons/red-dot.png"
        $("#map_addevent").width("300px").height("150px").gmap3({
            map: {
                options: {
                    center: [46.578498, 2.457275],
                    zoom: 3
                },
                events: {
                    click: function(map, event) {
                        //$("#Location").text("Location: Lat: "+event.latLng.lat()+" Lon: "+event.latLng.lng());
                        $("#id_lat").val(event.latLng.lat());
                    }
                }
            }
        });
    });

```

```

        $("#id_lng").val(event.latLng.lng());
        drawMarker(event.latLng.lat(),event.latLng.lng())
    }
}
});

//if location already selected
if (($("#id_lat").val())&&$("#id_lng").val()){
    drawMarker($("#id_lat").val(),$("#id_lng").val())
}

//Put selected image in form
$("#input[type='file']").bind("change",function(){
    file=$(this).prop("files")[0]
    img = $("form img[alt='preview image']")
    var reader = new FileReader();
    reader.onload = function(e){
        img.attr('src',e.target.result)
    }
    reader.readAsDataURL(file);
});

//Draw marker when user has clicked
function drawMarker(lat,lng){
    $("#map_addevent").gmap3({
        clear:{
            tag:["selected"]
        },
        marker:{
            latLng:[lat,lng],
            tag:"selected",
            options:{
                draggable:true,
                icon: red_icon
            }
        }
    });
}

});
</script>

```

10.3.13 Fichero "eventslist.html"

```

{% load staticfiles %}
{% load i18n %}
<div id="events_list">
    <table>
        {% for event in event_list%}
            <tr>
                <td style="display:none;">{{ event.id }}</td>
                <td style="display:none;">{{ event.location }}</td>
                <td>{{ event.title }}</td>
                <td>{{ event.category }}</td>
            </tr>
        {% endfor %}
    </table>
    <div id="events_list_control">
        {% with event_list|length as events %}
        {% if num_events == '15' %}
        <p>&lt;</p>
        {% else %}
        <p><a href="{% url 'eventslist.views.home' %}">{{ num_events|add:'-
15' }}&lt;</a></p>
        {% endif %}
    </div>

```

```

    }}</p>
    <p>{{ num_events|add:'-14' }}-{{ num_events|add:'-15'|add:events
    }}</p>
    {% if events < 15 %}
    <p>&gt;</p>
    {% else %}
    <p><a href="{% url 'eventslist.views.home' %}"{{
num_events|add:'15' }}">&gt;</a></p>
    {% endif %}
    {% endwith %}
</div>
</div>
<div id="back_to_list"><a href="#">{% trans "Back to list" %}</a></div>
<div id="toggle_map_detail"><a href="#">{% trans "Map/Detail"
%}</a></div>
{% for event in event_list%}
<ul class="event_detail" id="{{ event.id }}">
<li>{{ event.title }}</li>
<li>{{ event.description }}</li>
<li>{{ event.category }}</li>
{% if event.photo %}
<li></li>
{% endif %}
{% if user.is_authenticated and user.username == event.user%}
<a class="form" href="{% url 'eventslist.views.editevent'
%}"{{ event.id }}">{% trans "Edit" %}</a>
<a class="form" href="{% url 'eventslist.views.deleteevent'
%}"{{ event.id }}">{% trans "Delete" %}</a>
{% endif %}
<div class="comments">
{% include "eventslist/comments.html" %}
</div>
</ul>
{% endfor %}

```

10.3.14 Fichero "home.html"

```

{% extends "eventslist/base.html" %}
{% load staticfiles %}
{% load i18n %}
{% block stylesheets %}
<link rel="stylesheet" type="text/css" href="{% static "css/base.css" %} >
<link href='http://fonts.googleapis.com/css?family=Sonsie+One' rel='stylesheet'
type='text/css'>
<link
href='http://fonts.googleapis.com/css?family=Merriweather:400,300,400italic'
rel='stylesheet' type='text/css'>
<link href='http://fonts.googleapis.com/css?family=Pompiere' rel='stylesheet'
type='text/css'>
<link href='http://fonts.googleapis.com/css?family=Aladin' rel='stylesheet'
type='text/css'>
{% endblock %}
{% block javascript %}
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js" ></script>
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript" src="{% static "javascript/gmap3.min.js"
%}"></script>
<script type="text/javascript" src="{% static "javascript/home.js" %}"></script>
{% endblock %}
{% block nav %}
<ul>
{% if user.is_authenticated %}

```



```

        <li><a href="{% url 'eventslist.views.logoutpage' %}">{% trans "Logout"
%}</a></li>
        <li><a class="form" href="{% url 'eventslist.views.addevent' %}">{% trans "Add
Event" %}</a></li>

{% else %}
        <li><a class="form" href="{% url 'eventslist.views.register' %}">{% trans
"Registro" %}</a></li>
        <li><a class="form" href="{% url 'eventslist.views.loginpage' %}">{% trans
"Login" %}</a></li>

{% endif %}

        <li><a class="form" href="{% url 'eventslist.views.searchevents' %}">{% trans
"Search" %}</a></li>
</ul>
{% endblock %}

{% block content %}

        {% if search_parameters %}
        <ul>
{% for key, value in search_parameters %}
        <li>{{key}} - {{value}}</li>
        {% endfor %}
</ul>

        {% endif %}
        <div id="container">
        {% if event_list %}
                {% include "eventslist/eventslist.html" %}
                <div id="map_detail"></div>
        {% endif %}
        <div>
        <div id="form_container">
        </div>

{% endblock %}
{% block scripts %}
<script type="text/javascript">
        $( document ).ready(init());
</script>
{% endblock %}

```

10.3.15 Fichero "login.html"

```

{% load i18n %}
<div id='messages'>
        {% if messages %}
                <ul class="messages">
                        {% for message in messages %}
                                <li{% if message.tags %}class="{{ message.tags }}"
                                {% endif %}>{{ message }}</li>
                        {% endfor %}
                </ul>
        {% endif %}
</div>
{% if form %}
        <form action="{% url 'login' %}" method="post">
                {{ form.as_p }}
                {% csrf_token %}
                <input type="submit" value="{% trans "login" %}">
        </form>
{% endif %}

```

10.3.16 Fichero "register.html"

```
{% load i18n %}
<h1>{% trans "Create an account" %}</h1>

<form action="{% url 'register' %}" method="post">
  {{ form.as_p }}
  {% csrf_token %}
  <input type="submit" value="{% trans "Create the account" %}" >
</form>
```

10.3.17 Fichero "searchevents.html"

```
{% load i18n %}

<h1>{% trans "Search" %}</h1>
<form action="{% url 'searchevents' %}" method="post">
  {{ form.as_p }}
  <div id="distance">1 km</div>
  <p>{% trans "Select location in the map" %}</p>
  <div id="Location">{% trans "Location" %}</div>
  <div id="my_map"></div>
  {% csrf_token %}
  <input type="submit" value="{% trans "search" %}" >
</form>

<script type="text/javascript">
  // Draw map
  $( document ).ready(function() {
    red_icon="http://maps.google.com/mapfiles/ms/icons/red-dot.png"
    $("#my_map").width("300px").height("150px").gmap3({
      map:{
        options:{
          center:[46.578498,2.457275],
          zoom: 10
        },
        events:{
          click: function(map,event) {
            $("#Location").text("Location: Lat: "+event.latLng.lat()+" Lon:
"+event.latLng.lng());
            $("#id_lat").val(event.latLng.lat());
            $("#id_lng").val(event.latLng.lng());
            drawMarker(event.latLng.lat(),event.latLng.lng())
            drawCircle(event.latLng.lat(),event.latLng.lng(),$("#id_distance").val())
          }
        }
      }
    });

    //if location already selected
    if (($("#id_lat").val())&&($("#id_lng").val())){
      drawMarker($("#id_lat").val(),$("#id_lng").val())
    }

    //if distance changes
    $("#id_distance").change(function() {
      $("#distance").text($(this).val()+" km")
      drawCircle($("#id_lat").val(),$("#id_lng").val(),$("#id_distance").val())
    });

    //Draw marker when user has clicked
    function drawMarker(lat,lng){
      $("#my_map").gmap3({
        clear:{
          tag:["selected"]
        },
        marker:{
```

```

        latLng: [lat, lng],
        tag: "selected",
        options: {
            draggable: true,
            icon: red_icon
        }
    });
}

//Draw distance circle
function drawCircle(lat, lng, rad) {
    $("#my_map").gmap3({
        clear: {
            tag: ["circle"]
        },
        circle: {
            tag: "circle",
            options: {
                center: [lat, lng],
                radius : parseFloat(rad)*1000,
                fillColor : "red",
                strokeColor : "red"
            }
        }
    });
}
});
</script>

```

10.3.18 Fichero "home.js"

```

red_icon="http://maps.google.com/mapfiles/ms/icons/red-dot.png"
blue_icon="http://maps.google.com/mapfiles/ms/icons/blue-dot.png"

function init() {
    //Select first event and show detail
    set_first_event()

    //Create map and center in first event
    create_map_detail(first_lat(), first_lng())

    //Add pointers in the map
    addMarkers()

    //Show or hide comments form
    toggle_comments_form()

    //show form
    show_form()

    //process comment form
    process_comment_form()

    //process event form
    process_event_form()

    //change event detail
    changeEvent()

    //delete messages
    delete_messages()

    //toggle map/detail link
    toggle_map_detail()

    //back to list link
    back_to_list()

    //clear inline style

```

```

        manage_resize()
    }

    //Select longitude of first event
    function first_lng(){
    return
    $("tr:first").find("td:eq(1)").html().substring($("tr:first").find("td:eq(1)").html().lastIndexOf(",")+1,$("tr:first").find("td:eq(1)").html().indexOf(","))
    }

    //Select latitude of first event
    function first_lat(){
    return
    $("tr:first").find("td:eq(1)").html().substring($("tr:first").find("td:eq(1)").html().indexOf(",")+1,$("tr:first").find("td:eq(1)").html().lastIndexOf(","))
    }

    //Create map detail
    function create_map_detail(lat,lng){
    $("#map_detail").gmap3({
        map:{
            options:{
                center:[lat,lng],
                zoom: 8
            }
        }
    });
    }

    //Add class 'selected_event' to first event
    function set_first_event(){
    $("tr:first").addClass("selected_event")
        id=($(".selected_event").find("td:first").html())
        $("#"+id).toggle()
    }

    //Show and hide add_comment form
    function toggle_comments_form(){
    $("a[href='comment']").on("click",function(e){
        e.preventDefault()
        if ($ (this).text()=="Add comment"){
            $(this).text("Close")
            $(".event_detail[style*='display: block;'] form").show()

        }else{
            $(this).text("Add comment")
            $(".event_detail[style*='display: block;'] form").hide()
        }
    });
    }

    //Load form in form_container
    function show_form(){
    $(".form").on("click",function(e){
        e.preventDefault()
        $("#form_container").load($(this).attr("href"),function(){
            show_form_container()
        });
    });
    }

    //Process comment form using ajax
    function process_comment_form(){
        $(".ul.event_detail").on("submit","form",function(e){
            e.preventDefault();
            var postData = new FormData()

            $(".textarea",this).add("input:not([type='submit'])",this).each(function(i) {
                postData.append($(this).attr("name"), $(this).val());
            });
            postData.append("event_id",$(".ul.event_detail[style='display: block;']").attr("id"))
        }
    }

```

```

        var formUrl = $(this).attr("action");

        $.ajax({
            url:formUrl,
            type:"POST",
            data:postData,
            processData: false,
            contentType: false,
            success:function(data,textStatus,jqXHR)
            {
                $(".event_detail[style='display: block;']"
                .comments").html(data);
            },
            error:function(jqXHR,textStatus,errorThrown)
            {alert(errorThrown)}
        });
    });
}

//Process event form using ajax
function process_event_form(){
    $("#form_container").on("submit","form",function(e){
        e.preventDefault();

        var postData = new FormData()
        $("#form_container form input[type='file']").each(function(i) {
            postData.append($(this).attr("name"), $(this).get(0).files[0]);
        });

        $(".form textarea, form select, form
        input:not([type='submit']):not([name='photo'])").each(function(i) {
            postData.append($(this).attr("name"), $(this).val());
        });

        var formUrl = $(this).attr("action");
        $.ajax({
            url:formUrl,
            type:"POST",
            data:postData,
            processData: false,
            contentType: false,
            success:function(data,textStatus,jqXHR)
            {
                if (data.indexOf("<html>")!=-1){
                    document.write(data)
                    document.close()

                }else{
                    $(".#form_container").html(data);
                    show_close_link()
                }
            },
            error:function(jqXHR,textStatus,errorThrown)
            {}
        });
        e.unbind();
    });
}

//Change selected event when user clicks on it
function changeEvent(){
    $(".tr").click(function(){
        map=$("#map_detail")
        id($(".selected_event").find("td:first").html())

        lat($(".selected_event").find("td:eq(1)").html().substring($(".selected_event").find("td:eq(1)").html().indexOf("[")+1,$(".selected_event").find("td:eq(1)").html().lastIndexOf(",")
        lng($(".selected_event").find("td:eq(1)").html().substring($(".selected_event").find("td:eq(1)").html().lastIndexOf(",")+1,$(".selected_event").find("td:eq(1)").html().indexOf("]"))
        map.gmap3({clear:{tag:[id]}});
        addMarker(map,id,blue_icon,lat,lng)
    });
}

```

```

    $("#a[href='comment']").text("Add comment")
    $("#"+id+" form").hide()

    $("#"+id).toggle()
    $(".selected_event").removeClass("selected_event")
    $(this).addClass("selected_event")
    id=$(this).find("td:first").html()
    $("#"+id).toggle()

    lat=$(this).find("td:eq(1)").html().substring($(this).find("td:eq(1)").html().indexOf("[")+1,$(this).find("td:eq(1)").html().lastIndexOf(", "))

    lng=$(this).find("td:eq(1)").html().substring($(this).find("td:eq(1)").html().lastIndexOf(",")+1,$(this).find("td:eq(1)").html().indexOf("]"))
    map.gmap3({clear:{tag:[id]}});
    addMarker(map,id,red_icon,lat,lng)
    map.gmap3("get").setCenter(new google.maps.LatLng(lat,lng))

    if ($(window).width()<800){
        $("#events_list").css("width","0px")
        $("#events_list").css("height","0px")
        $("#events_list").css("overflow","hidden")
        $(".event_detail").css("width","100%")
        $(".event_detail").css("height","auto")
        $("#toggle_map_detail").css("display","inline")
        $("#back_to_list").css("display","inline")
    }

});
}

//Hide messages
function delete_messages(){
    $("#messages").fadeOut(5000, "linear",function(){
        $(".messages").remove()
    });
}

//Manage Map/Detail link
function toggle_map_detail(){
    $("#toggle_map_detail").click(function(){

        detail$(".event_detail")
        map=$("#map_detail")
        if ($(window).width()<768){
            width="100%"
        }else{
            width="50%"
        }

        if (detail.width()<=1){
            detail.css("width",width)
            detail.css("height","auto")
            map.css("width","0px")
            map.css("height","0px")
        }else{
            detail.css("width","0px")
            detail.css("height","0px")
            map.css("width",width)
            map.css("height","550px")
            map.gmap3({trigger:"resize"});
            center_map()
        }

    });
}

//Manage Back_to_list link
function back_to_list(){
    $("#back_to_list").click(function(){
        detail$(".event_detail")
        map=$("#map_detail")

```

```

        events_list=$("#events_list")
        detail.css("width","0px")
        detail.css("height","0px")
        map.css("width","0px")
        map.css("height","0px")
        events_list.css("width","100%")
        events_list.css("height","auto")
        $("#back_to_list").css("display","none")
        $("#toggle_map_detail").css("display","none")
    });
}

//Set sizes when windows is resized
function manage_resize(){
    $( window ).resize(function() {
        detail=$("#.event_detail")
        map=$("#map_detail")
        events_list=$("#events_list")
        map.css("width","")
        map.css("height","")
        detail.css("width","")
        detail.css("height","")
        events_list.css("width","")
        events_list.css("height","")
    });
}

//Add markers to map showing events positions
function addMarkers(){
    map=$("#map_detail")
    $("tr").each(function(){

        lat=$(this).find("td:eq(1)").html().substring($(this).find("td:eq(1)").html().indexOf("[")+1,$(this).find("td:eq(1)").html().lastIndexOf(",") )

        lng=$(this).find("td:eq(1)").html().substring($(this).find("td:eq(1)").html().lastIndexOf(",")+1,$(this).find("td:eq(1)").html().indexOf("]"))
        if ($(this).hasClass("selected_event")){
            icon_image=red_icon
        }else{
            icon_image=blue_icon
        }
        tag_label=$(this).find("td:first").html()

        addMarker(map,tag_label,icon_image,lat,lng)
    });
}

//Add marker to map
function addMarker(map,tag_label,icon_image,lat,lng){
    map.gmap3({
        marker:{
            latLng:[lat,lng],
            tag:tag_label,
            options:{
                draggable:true,
                icon: icon_image
            }
        }
    });
}

//Show shade div and form container
function show_form_container()
{
    $("body").append("<div id='shade'></div>")
    $("#form_container").css("display","block")
    show_close_link()
    $("input[value='Cancel']").click(function(){
        $("#shade").remove();
        $("#form_container").html("");
    });
}
}

```

```
//Show close link in form container
function show_close_link()
{
    $("#form_container").append("<div class='close'><a href='#'>X</a></div>");
    $(".close").click(function(){
        $("#shade").remove();
        $("#Form_container").html("");
    });
}

//Center map in selected event position
function center_map()
{
    map=$("#map_detail")
    id=$("#selected_event").find("td:first").html()
    lat=$("#selected_event").find("td:eq(1)").html().substring($("#selected_event").find("td:eq(1)").html().indexOf("[")+1,$("selected_event").find("td:eq(1)").html().lastIndexOf(", "))
    lng=$("#selected_event").find("td:eq(1)").html().substring($("#selected_event").find("td:eq(1)").html().lastIndexOf(",")+1,$("selected_event").find("td:eq(1)").html().indexOf("]"))
    map.gmap3("get").setCenter(new google.maps.LatLng(lat,lng))
}
```