Proyecto Final de Carrera – Eventlayer.com

Barcelona Tech 2011

Título:	***Creación de la web Eventlayer.com***
Alumno:	***Pablo Guevara Núñez***
Director/Ponente:	***Antonio Cañabate Carmona***
Departamento:	***Organización de Empresas***
Fecha:	***2 de Julio de 2011***

DATOS DEL PROYECTO

Título del Proyecto	Creación del sitio web Eventlayer.com
Nombre del estudiante	Pablo Guevara Núñez
Titulación	Ingeniería Informática
Créditos	37,5
Director/Ponente	Antonio Cañabate Carmona
Departamento	Organización de Empresas

MIEMBROS DEL TRIBUNAL

Presidente	Ferran Sabate Garriga
Vocal	M. Del Carme Álvarez Faura
Ponente	Antonio Cañabate Carmona

CALIFICACIÓN

Calificación numérica _____

Calificación descriptiva _____

A fecha 2 de Julio de 2011

Prólogo

Dicen que la vida es aquello que te va sucediendo mientras te empeñas en hacer otros planes. Seguramente esa sea la única explicación por la que este proyecto haya podido aportarme tanto como persona. Es por eso que este prólogo va dedicado a todo lo que nos ha ido ocurriendo, a mí y al resto de rebeldes casuales: Sergi, Adri, y Joan; mientras nos ocupábamos de otras cosas.

El proyecto Eventlayer.com ha durado 2 años en los que el mundo no ha parado de girar. Los ordenadores se han hecho más pequeños, los móviles vuelven a ser grandes, los políticos nos engañaron más que nunca y los pobres callaron, una vez más. Pero mientras los cautelosos dudaban, los valientes fueron, triunfaron y volvieron. Es por eso que hoy, más que nunca, me alegro de haberme atrevido a abrazar la incertidumbre cuando Sergi me propuso que emprendiéramos.

Durante este tiempo hemos conocido a mucha gente, algunos nos inspiraron como nunca antes, algunos de ellos fueron tan sólo un sinsentido. Pero todos con su ejemplo nos han curado en humildad, demostrándonos que cualquier persona, cualquiera, puede enseñarnos algo. Muchos nos han ayudado porque han creído en nuestro trabajo, nuestras familias nos han apoyado porque creen en nuestro esfuerzo y nosotros como equipo siempre creímos los unos en los otros. A todos vosotros, gracias.

Una de las personas que nos brindó su apoyo fue Toni. Todo empezó en una mesa que cojeando aguantaba las cervezas que escuchaban atentamente los consejos de aquel profe “tan guai”. Me atrevo a decir por la pasión en sus memorias que en aquel momento él mismo hubiese dejado atrás todos sus logros por poder volver a intentarlo. Hubiese olvidado sus triunfos, aún a riesgo de fracasar, por ser nosotros y escuchar aquellas palabras que nos servirían de guía y como un comienzo. A él también, gracias.

La verdad es que nos ocurrieron muchas cosas positivas mientras trabajábamos, tuvo que ser así, porque trabajábamos mucho. Fueron muchas horas dedicadas a equivocamos y a fallar, pero de todo, todos aprendimos. Esa es la razón de que valiese la pena, porque si aprendes sin estudiar y trabajas en lo que te gusta, no volverás a tener que trabajar ni un minuto más en tu vida.

Tras este proyecto nada volverá a ser lo mismo, incluso esta universidad, nuestra Alma mater, dejó atrás su nombre, y ahora se llama Barcelona Tech, a ti también gracias. El cambio y la reflexión nos hizo más sabios y pidiéndonos a cambio tan solo un pedazo de nuestra inocencia. Algo que supimos compensar con más ilusión. Lo que aprendimos nos empujó a trabajar aún con más ganas, porque cada vez sabíamos hacerlo mejor. Es por eso que ahora, después de todo este largo camino, me apetece más que nunca continuar, y llegar hasta el final.

Hasta aquí esta oda a todo aquello que nos ha ocurrido sin planearlo, sin pedirlo de antemano. Porque yo siempre digo que las cosas buenas, las cosas realmente buenas, nos ocurren por casualidad. Y es que nadie es lo suficientemente bueno para merecérselas y nuestra tarea está simplemente, en saber aprovecharlas.

Sergi, Adri, Joan y yo nos conocimos por casualidad en una de estas aulas. Lo que seamos capaces de hacer juntos, está por demostrar.

Índice de contenido

1. Introducción

Este documento es la memoria de un Proyecto Final de Carrera, a partir de ahora PFC, para la obtención del título de Ingeniero en Informática por la Facultat d'Informàtica de Barcelona (FIB) de la Universitat Politècnica de Catalunya (UPC). Este PFC consiste en especificar, diseñar e implementar una aplicación web que hemos llamado Eventlayer y que a grandes rasgos podría definirse como una guía de ocio online con funcionalidades de una red social (explicación completa de la aplicación en el capítulo *1.2. Producto)*.

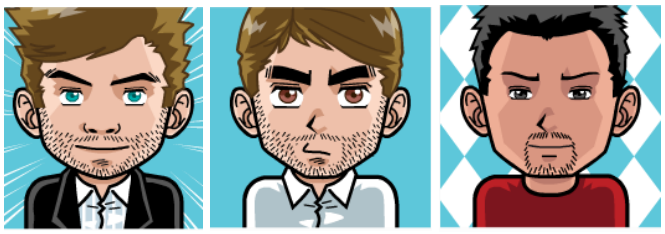


Ilustración 1 - Pablo Guevara, Sergi Pérez, Adrià Heredia. Los desarrolladores de Eventlayer

Este PFC forma parte de un conjunto de 3 PFCs. Todos con el mismo objetivo, la construcción del sistema Eventlayer. Los nombres de cada uno de los estudiantes responsables de los citados PFCs son Sergi Pérez, Adrià Heredia Rius y yo mismo, Pablo Guevara. Para saber cuál ha sido concretamente la división den trabajo entre cada uno de nosotros se pueden consultar estos 2 capítulos:

- Capítulo *1.1.1.* División de responsabilidades en la redacción de la memoria. Se explican cuáles son las partes comunes e individuales en las memorias de los PFCs.

- *Capítulo 1.3.2. División de responsabilidades en el proyecto Eventlayer.* Se aclaran las responsabilidades de cada uno de nosotros dentro del proyecto Eventlayer (construcción del software).

Por último comentar que nuestra intención es que después de concluir el PFC el proyecto continúe como proyecto empresarial bajo el dominio de internet www.eventlayer.com y se conozca nuestro producto como *Eventlayer, la guía de ocio fácil y social*.

Es por ese motivo que el alcance de este PFC es por una parte la construcción del sistema Eventlayer, que es el objetivo principal. Y por otra, como objetivo secundario, obtener un proyecto empresarial viable. Para tal fin se han incluido en esta memoria algunos capítulos relativos al futuro Plan de negocio de Eventlayer (En el capítulo *1.1. Descripción de este documento* se concreta qué capítulos están destinados al citado plan de negocio).

1.1. Descripción de este documento

Este documento representa la memoria de mi PFC. El objetivo de su redacción es la de tener por escrito la máxima información de cómo se ha desarrollado el proyecto y sobretodo, tener una documentación más o menos exhaustiva del software construido.

Básicamente debe servir para que alguien ajeno al proyecto tenga una idea bastante real de como lo hemos llevado a cabo y que además, pueda entender las diferentes partes del sistema desarrollado. Este último punto implica tener conocimientos de ingeniería del software por lo que habrán partes en esta memoria sólo entendibles por lectores familiarizados con esta rama de la ingeniería, especialmente los *capítulos 4.* Diseño del sistema *y 5.* Implementación del sistema.

La estructura principal del documento es la siguiente:

- **Introducción al proyecto**. Explicar de qué trata este PFC, sus objetivos y la metodología utilizada. Capítulos:
    1. Introducción

- **Proceso de construcción del software**. Este PFC se basa en la creación de un sistema software y siguiendo la metodología estándar escogida (ver capítulo *1.4. Metodología utilizada*) hay ya unas pautas de documentación a seguir. Es por eso que los capítulos de esta sección coinciden con las fases típicas de un proyecto de ingeniería: análisis y especificación, planificación, diseño, implementación y pruebas. En cada capítulo de este bloque habrá un comentario inicial donde se explicará la manera en que se ha planteado la fase de construcción en cuestión. Además se puede consultar el capítulo *1.4.3. Documentación* donde se justifica la elección de la manera de documentar proyectos de software que hemos utilizado. Capítulos:

    2. Análisis de requisitos y especificación
    3. Planificación y análisis económico
    4. *Diseño del sistema*
    5. *Implementación del sistema*
    6. *Pruebas*

- **Anexos.** En la parte de anexos se ha redactado un documento que puede ser interesante para la visión más empresarial del PFC, el Plan de Marketing para Eventlayer. Además se incluyen, el manual de usuario, la bibliografía y el glosario de rigor. Capítulos:

    Anexo I – Plan de Marketing
    Anexo II – Manual de usuario
    Anexo III – Bibliografía
    Anexo IV – Glosario

1.1.1. División de responsabilidades en la redacción de la memoria

En el capítulo anterior (1. *Introducción*) hemos comentado que el proyecto Eventlayer engloba 3 PFCs. Es decir, que se han redactado 3 memorias, incluyendo esta misma, que contienen partes comunes y partes individuales de cada alumno.

Se decidieron estructurar así las memorias para que cada PFC fuese lo más auto-contenido posible (se pudiese leer como un todo) y a la vez fuese fácil la corrección de los 3 PFCs (para la corrección de los PFCs sólo hace falta leerse una vez las partes comunes y a continuación leerse cada una de las partes individuales)

- Los **capítulos comunes** son (entre paréntesis el responsable):

  1. Introducción (Pablo)
  2.Análisis de requisitos y especificación
    3.1. Actores (Pablo)
    3.2. Historias principales (Pablo)
    3.3. Requisitos no funcionales (Sergi)
  *3.*Planificación y análisis económico (Sergi) (cada uno aportó su planificación individual)
  *4.*Diseño del sistema
    *4.1. Patrones de software (*Pablo*)*
  6. Pruebas (Sergi) (cada uno aportó sus planes de pruebas)

  Anexo I – Plan de Marketing(Pablo)
  Anexo II – Manual de usuario (Sergi)
  Anexo III – Bibliografía (Pablo, Sergi, Adrià)
  Anexo IV – Glosario (Pablo, Sergi, Adrià)

- El capítulo de diseño del sistema es **común entre Pablo y Sergi** (los dos son responsables del sistema web) mientras que **en la memoria de Adrià es diferente**. Por lo tanto las responsabilidades del capítulo 5 quedan así:

  4.2. Diseño del sistema web (Sergi)
  4.2. Diseño del sistema móvil (Adrià)

- El resto de **capítulos son individuales**:

  Prólogo (individual)
  1.3.2.1.Objetivos específicos de mi PFC (individual)

  5. Implementación del sistema (individual)

1.2. Producto

¿Cómo serían las guías de ocio si se hubieran inventado hoy?

A partir de esta pregunta nace **Eventlayer**, una **guía de ocio** local, simple y social. Centrada en ofrecerte lo que más te interese de **tu ciudad**. Destaca por ser s**imple**, nuestra obsesión es poner la tecnología al servicio del usuario y hacer un producto increíblemente cómodo de usar. Se diferencia por ser **social** porque ¿verdad que no vamos solos a los conciertos o al cine? entonces una guía de ocio debería tener en cuenta también a nuestros amigos.

En Eventlayer el usuario tiene la información de todos los eventos que le interesan como son conciertos, obras de teatro, cine; tienes descuentos grupales (del 50%, 60%...) y rankings con los 10 mejores restaurantes, bares o lugares de interés. También puede ver dónde van a ir sus amigos.

Cuando algo le gusta al usuario dice "me gusta" y se le añade a su calendario. Si algo no le gusta dice "no me gusta" y desaparece del listado. Si encuentra un concierto, una exposición, etc. a la que le gustaría ir nos dice Asistiré y a continuación selecciona con qué amigos le gustaría asistir y a partir de ese momento se les crea su "layer", un espacio privado donde puede chatear con ellos para decidir cómo van a quedar, compartir fotos, utilizar nuestra herramienta para escoger entre todos el mejor día y la mejor hora, votaciones para decidir dónde irán a cenar, compartir fotos, o escoger aleatoriamente quién se encarga de llevar el coche, etc.

Ilustración 2 - Organiza como vas a quedar con tus amigos usando el chat, la herramienta para decidir el día y la hora, encuestas, selector aleatorio, comparte fotos...

Todo conectado con su cuenta de Facebook, tu email y también accesible desde su teléfono móvil.

Ilustración 3 - Utiliza Eventlayer con tu Facebook, tu email y tu teléfono móvil

Cuando un amigo te pregunte ¿Qué hacemos? o ¿Cómo quedamos? Tu respuesta **Eventlayer.com, la guía de ocio simple y social.**

Se puede ver el vídeo oficial de Eventlayer en

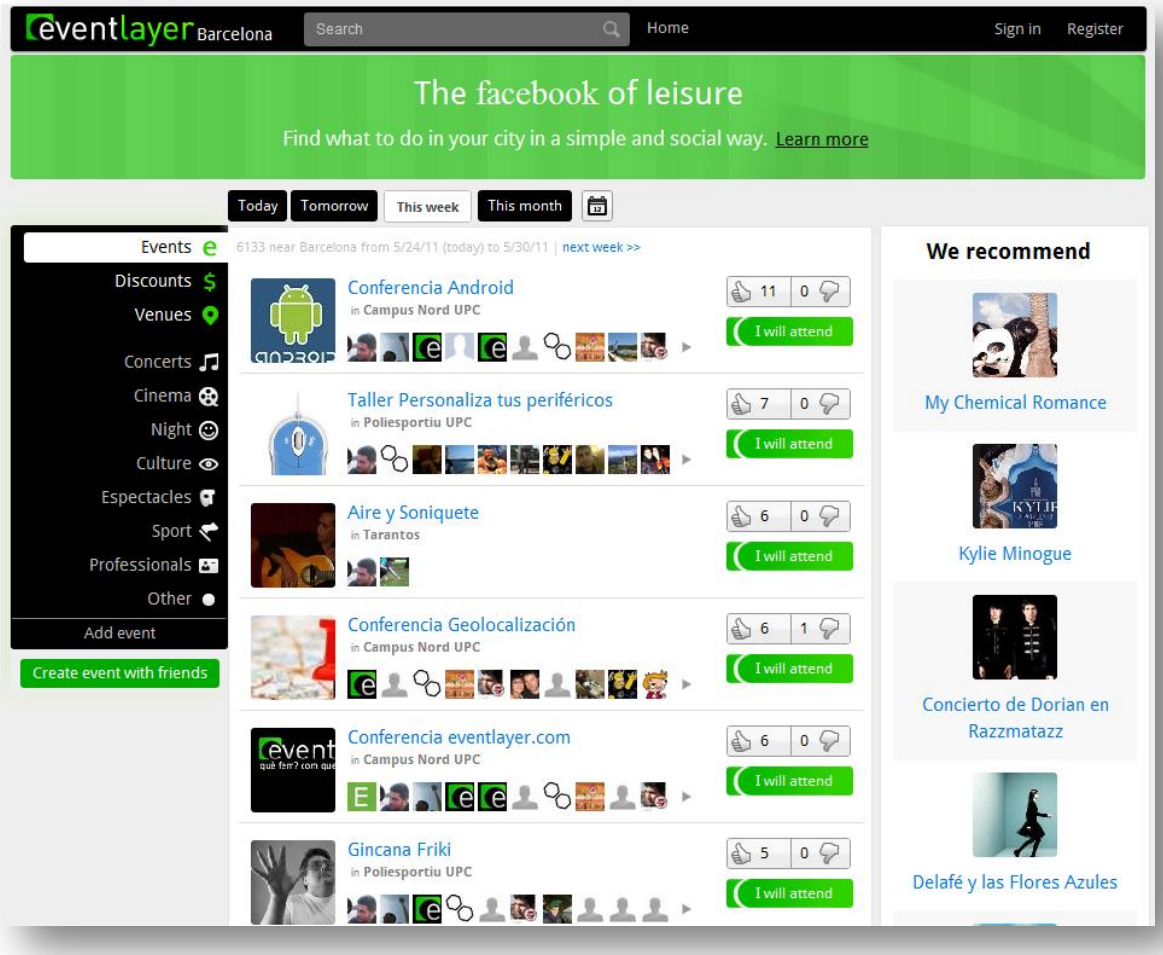
http://player.vimeo.com/video/22843727

1.2.1. Eventlayer en imágenes

Vamos a incluir aquí las principales pantallas para ayudar a describir Eventlayer en imágenes.

1. La **pantalla principal** de Eventlayer es un listado de los eventos de la semana corriente ordenados por popularidad. Podemos filtrar el listado por fecha (hoy, mañana, esta semana, este mes…) o bien por el tipo de eventos (conciertos, cine, noche, cultura…).

   Cuando el usuario clica a me gusta en un evento queda destacado (color verde) y se añade a su calendario. Cuando un evento no te gusta se oculta.



2. Cuando se hace click en uno de los elementos de la lista se abre la información del evento en un popup, de esta manera el usuario se ahorra el tener que utilizar a cada momento el botón “atrás” y nunca pierde la perspectiva de donde se encuentra respecto a la lista. Además a medida que el usuario va haciendo scroll la lista automáticamente va cargando más eventos para ahorrar distracciones al usuario.

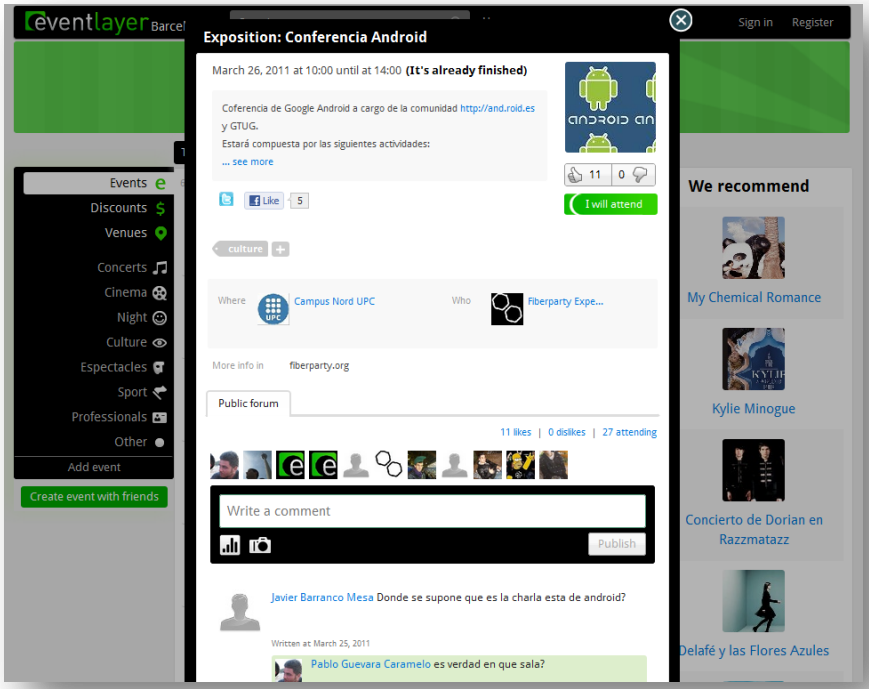


Ilustración 5 – Vista de un evento en eventlayer.com

3. En Eventlayer, el usuario también puede consultar la información de los lugares donde se organizan los eventos que le interesan y también puede hacer lo mismo con los artistas que actúan en los eventos. A continuación capturas del perfil de un lugar y de un artista u organización.

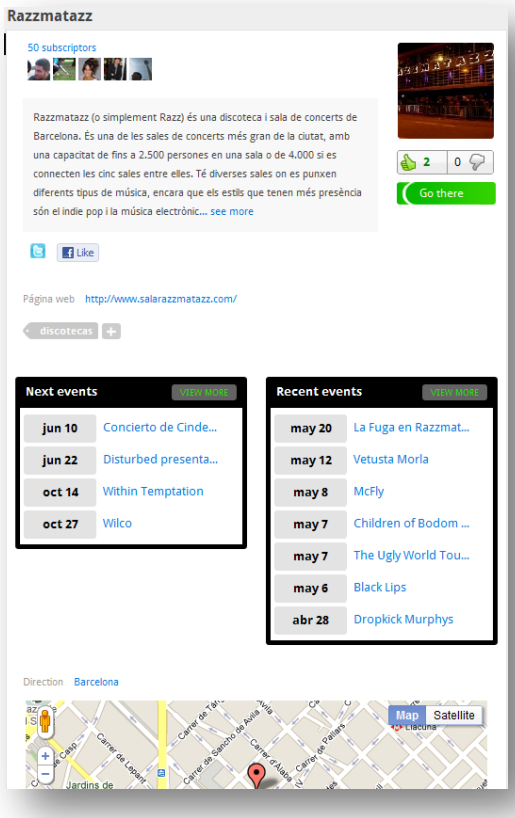


Ilustración 7 - Perfil de un lugar en eventlayer.com

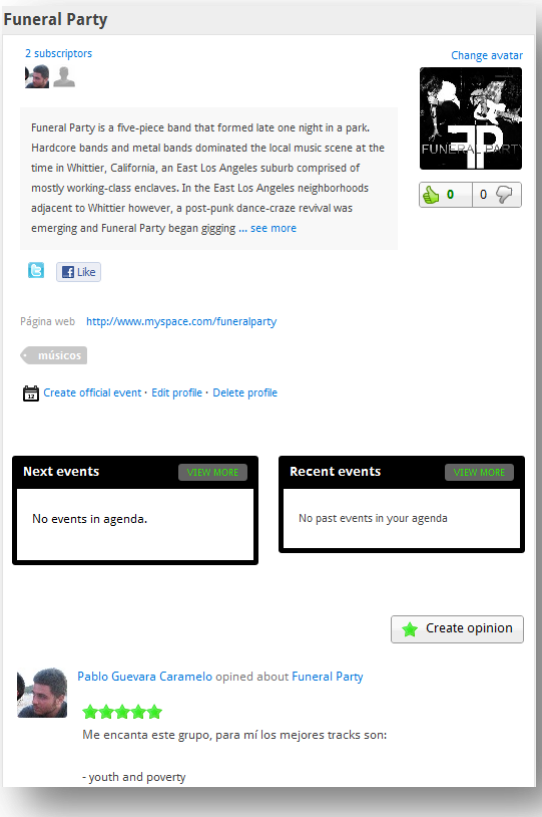


Ilustración 6 - Perfil de un artista en eventlayer.com

4. A parte del listado de eventos se ofrece un listado de **descuentos**. Son descuentos del tipo ofertas del día, lo que actualmente se conoce como modelo Groupon[1], Eventlayer agrega todas las ofertas de diferentes páginas que siguen este modelo (Offerum.com, Groupalia.com, Letsbonus.com…).

Ilustración 8 - Descuentos en eventlayer.com

El usuario también puede inscribirse en las listas de invitados de Eventlayer para entrar gratis o con descuento a los clubes de su ciudad.

Ilustración 9 - Listas de invitados en eventlayer.com

[1] Modelo Groupon hace referencia al modelo de negocio de groupon.com una web de ofertas de más del 50% que se basa en ofrecer una oferta cada día que sólo es válida si la compran un número mínimo de usuarios.

5. Otra sección principal de Eventlayer es el listado de **lugares** de interés. En este listado se muestran top dieces (top10), es decir los 10 mejores lugares en tu ciudad para por ejemplo cenar barato, llevar a una amiga a tu primera cita, salir a tomar un café… Los usuarios pueden dejar opiniones en los lugares y proponer sus propios top10.

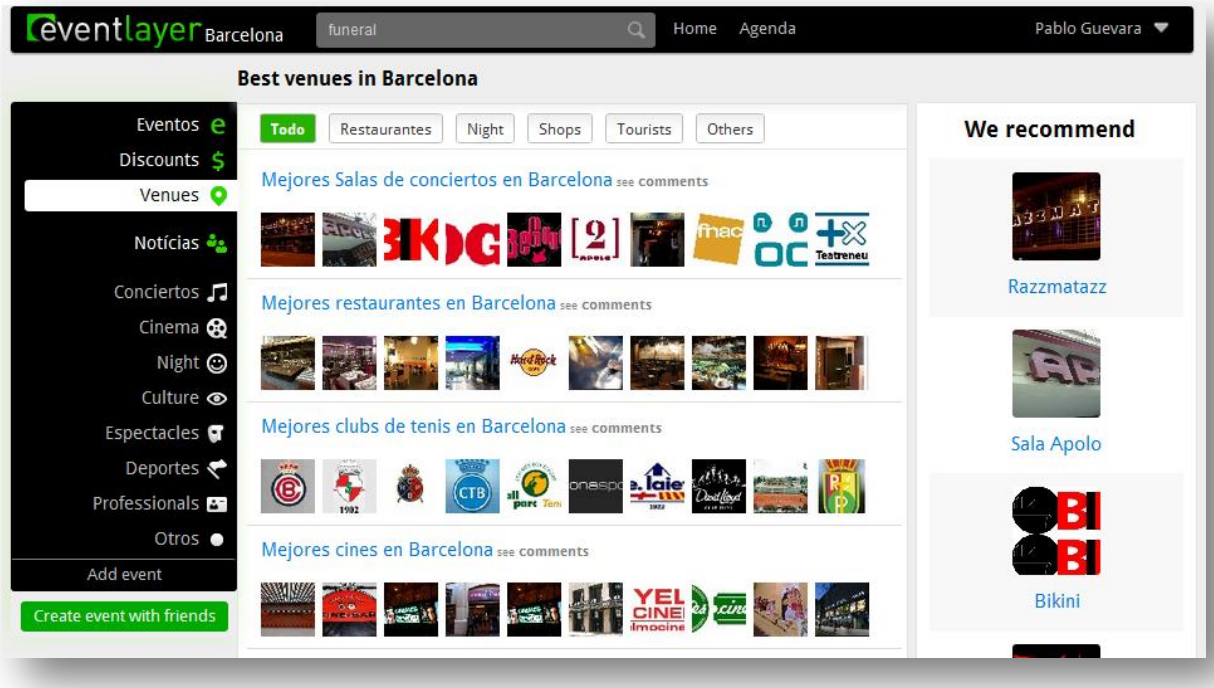


Ilustración 10 - Página de lugares de interés en eventlayer.com

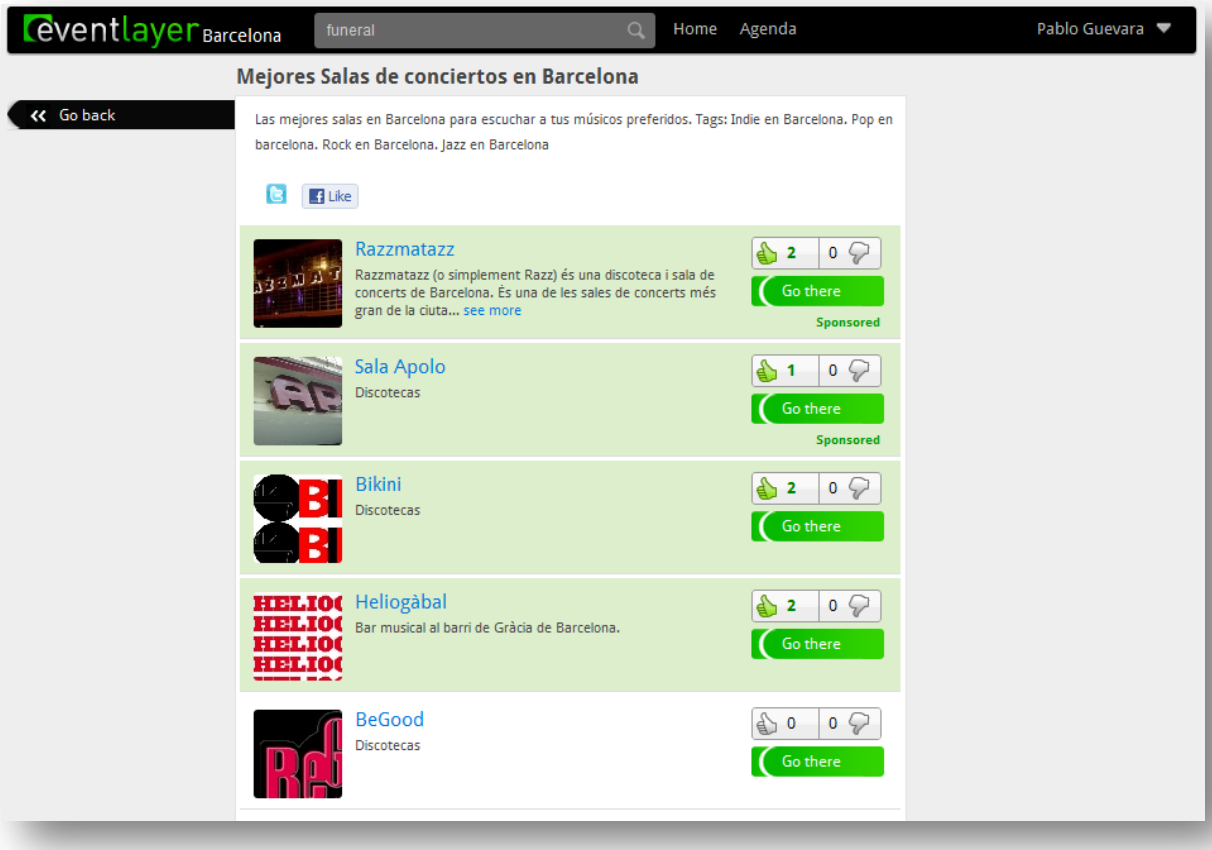


Ilustración 11 - Top10 de salas de concierto en eventlayer.com

6. En Eventlayer también ofrecemos una sección de **cine** con una cartelera muy simple para buscar todas las proyecciones de esa película en los cines de la ciudad del usuario. Se incluyen horarios, comentario de la película y valoración de IMDB (Internet Movie Data Base).

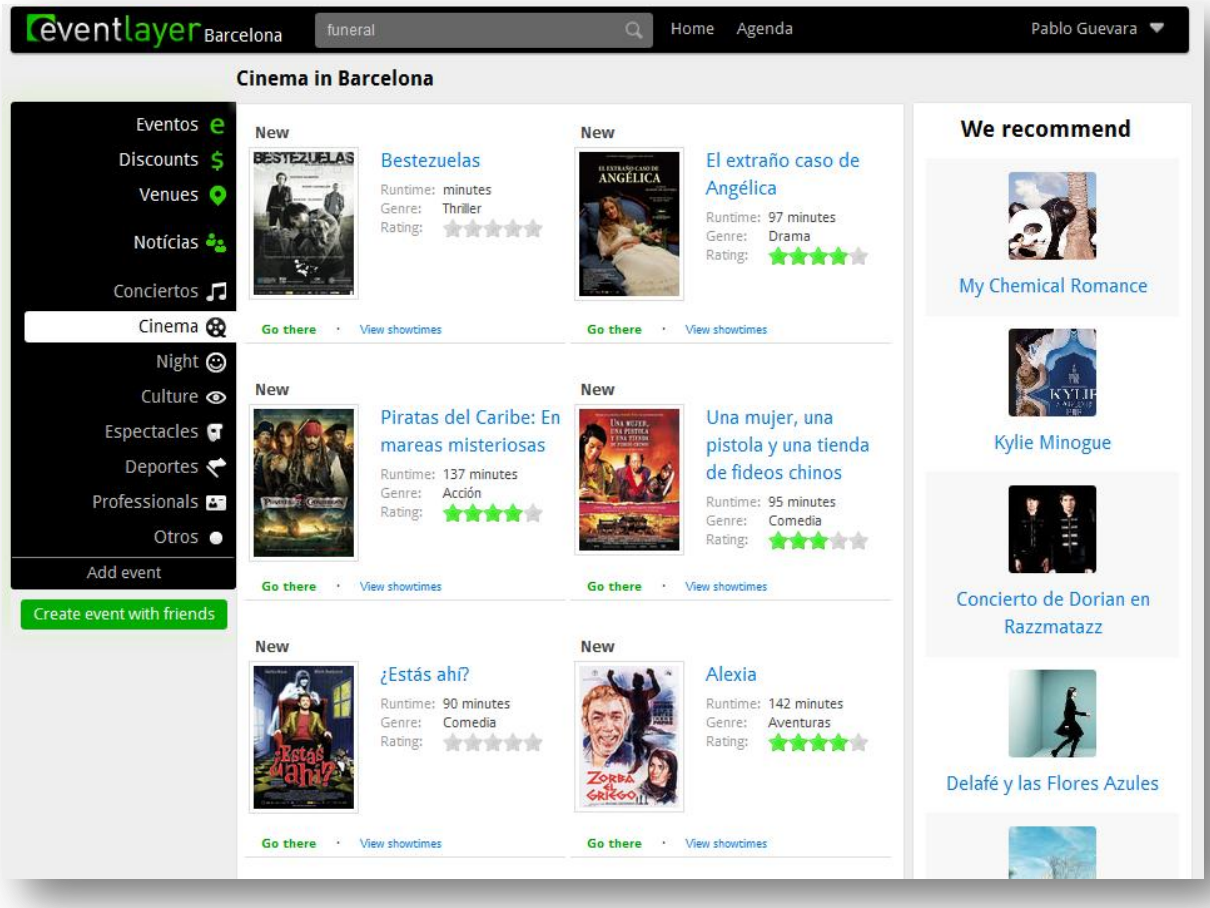


Ilustración 12 - Sección de cine en eventlayer.com

Ilustración 14 - Información de una película en eventlayer.com

Ilustración 13 - Horarios de una película en eventlayer.com

7. La siguiente pantalla es lo que llamamos el **muro.** En el muro puedes ver de manera cronológica los eventos que interesan a tus amigos, a qué eventos van a ir, qué lugares les gustan, a qué artistas se suscriben … **Además cada vez que un artista o lugar que "te gusta" tenga un concierto cerca de tu ciudad se añadirá un aviso en este muro**.

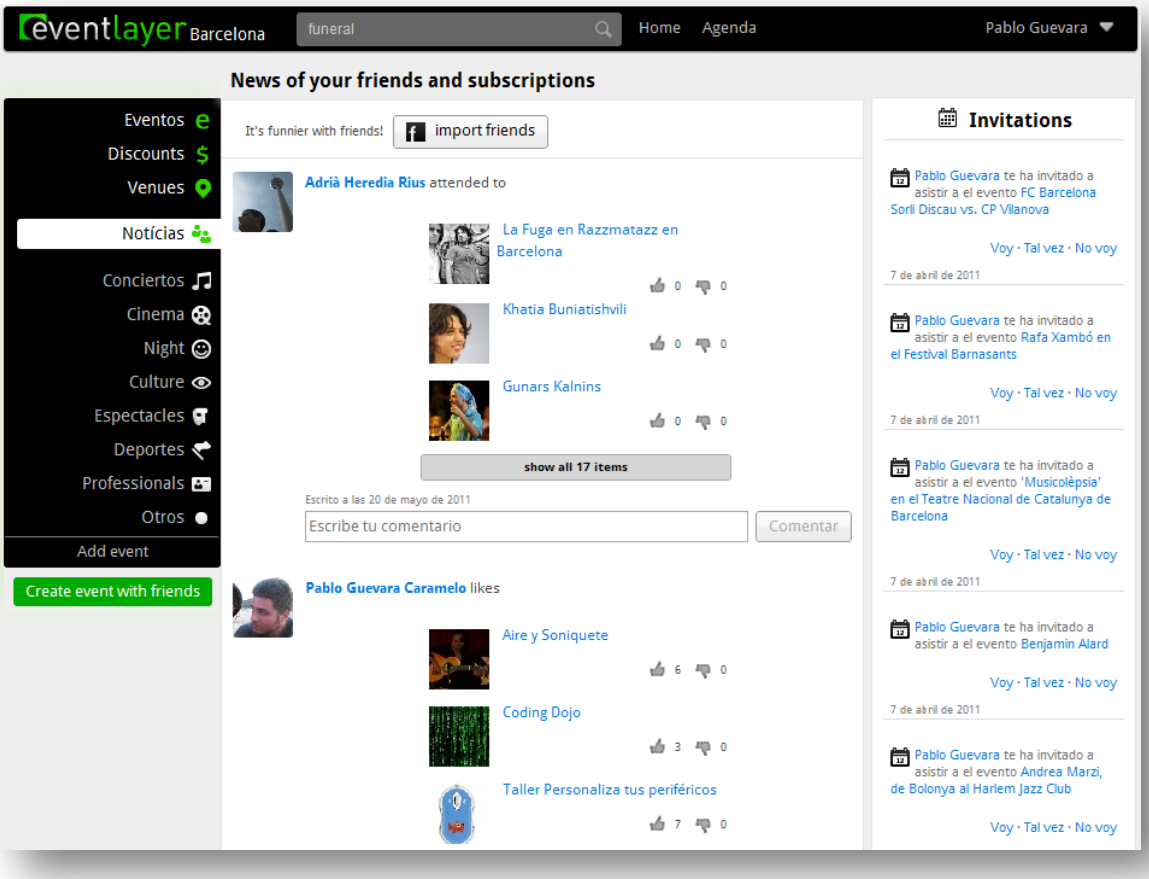


Ilustración 15 - Muro en eventlayer.com

Si el usuario quiere consultar el muro de un usuario en concreto o el suyo propio visita su agenda.

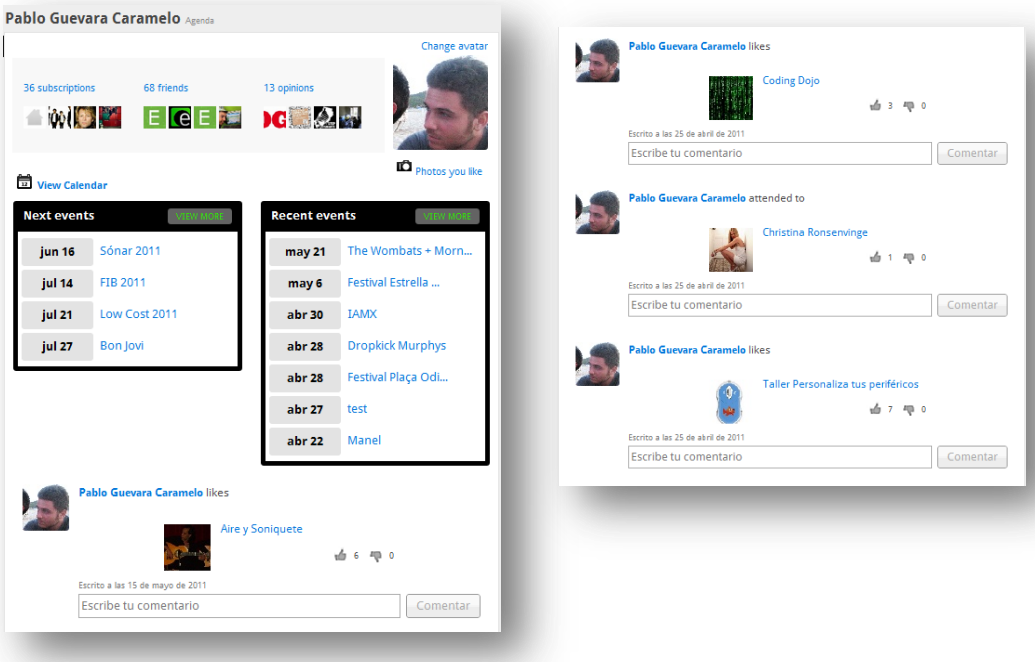


Ilustración 16 - Agenda usuario en eventlayer.com

8. Una vez el usuario tiene pensado asistir a uno de los eventos, descuentos o lugares que has encontrado dice:

Escoge los amigos con los que quiere asistir. La mayor ventaja es que los puede seleccionar directamente de su Facebook para ahorrar tiempo.

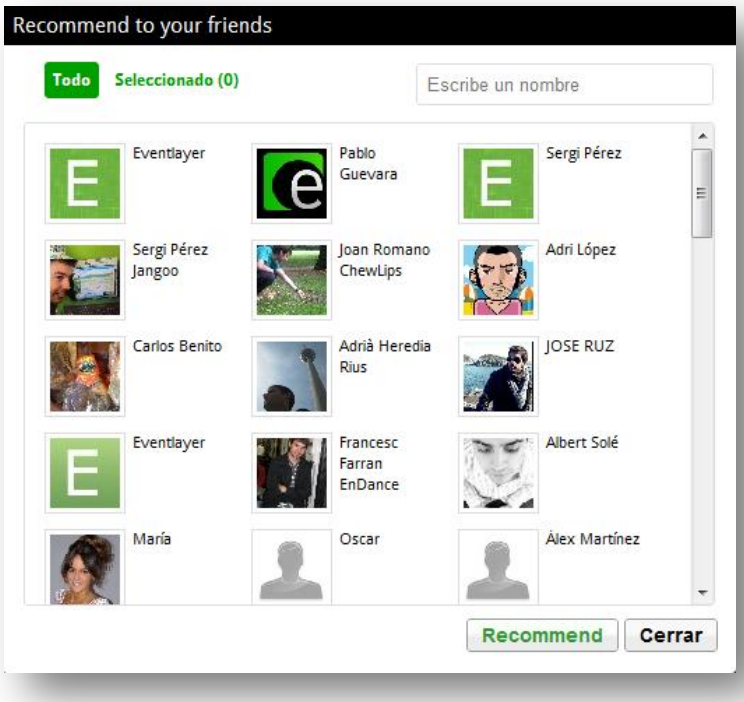


Ilustración 17 - Selector de amigos en eventlayer.com

A partir de ese momento los amigos del usuario y él mismo disponen de un espacio privado en el mismo evento para organizar como van a quedar. A continuación se pueden ver dos pestañas, la gris donde se puede intercambiar mensajes y preguntas con el resto de usuarios y una verde que es privada para el usuario y la gente que va a asistir con él. Lo que llamamos event layer. Por último comentar que los botones verdes son los que permiten añadir los mensajes para quedar más fácilmente. A continuación un ejemplo.

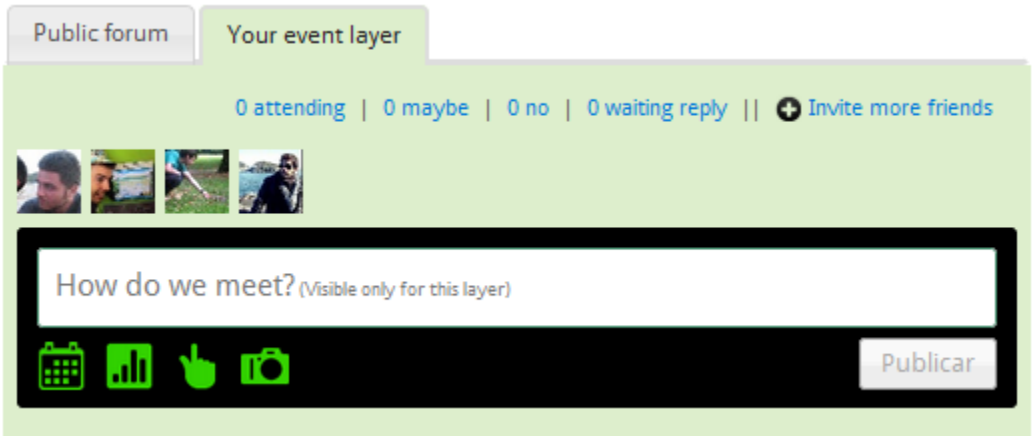


Ilustración 18 - Layer de un evento en eventlayer.com

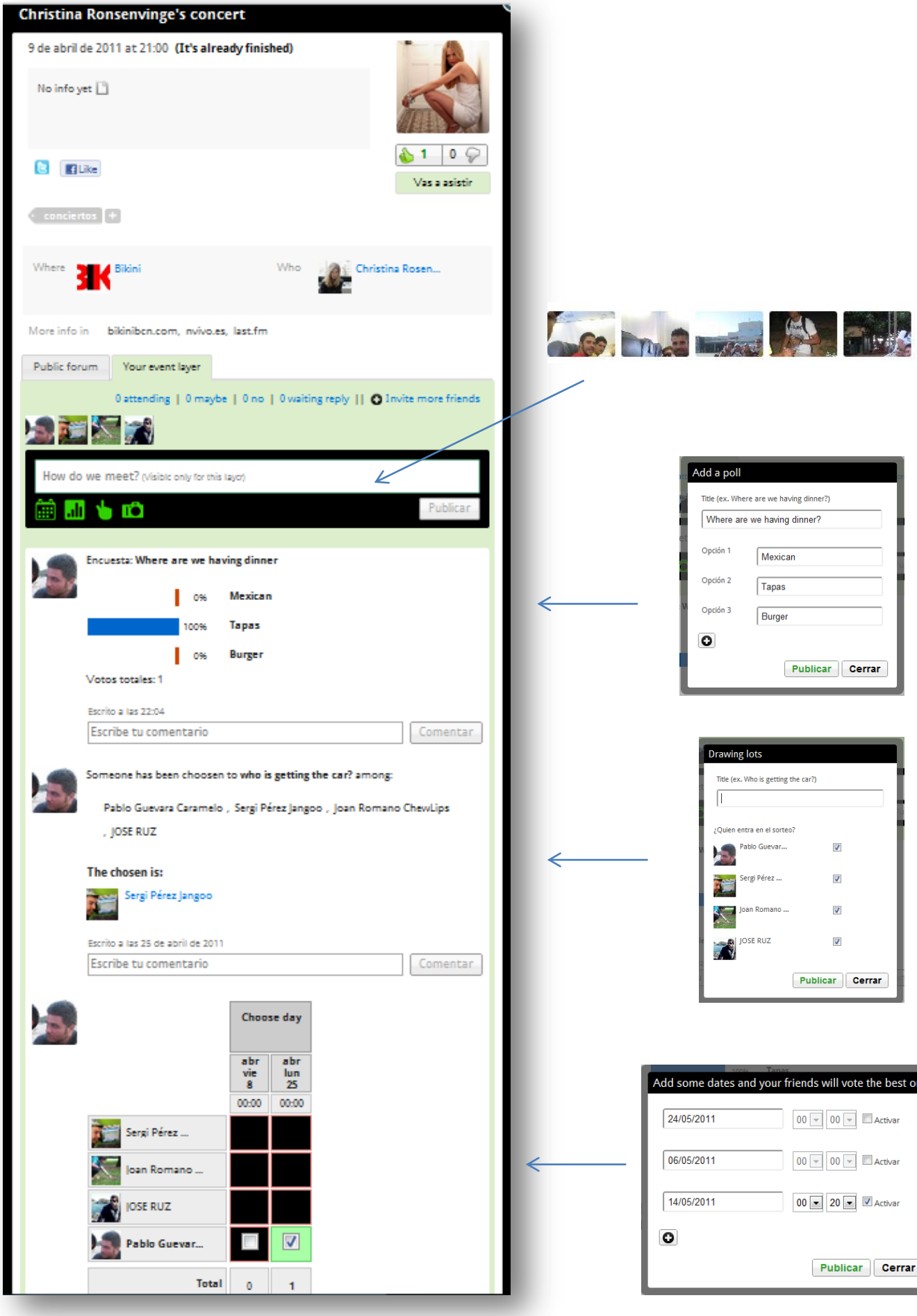


Ilustración 19 - Ejemplo de layer en eventlayer.com

9. Destacar el calendario que pueden usar los usuarios registrados para tener a mano todos sus próximos eventos.

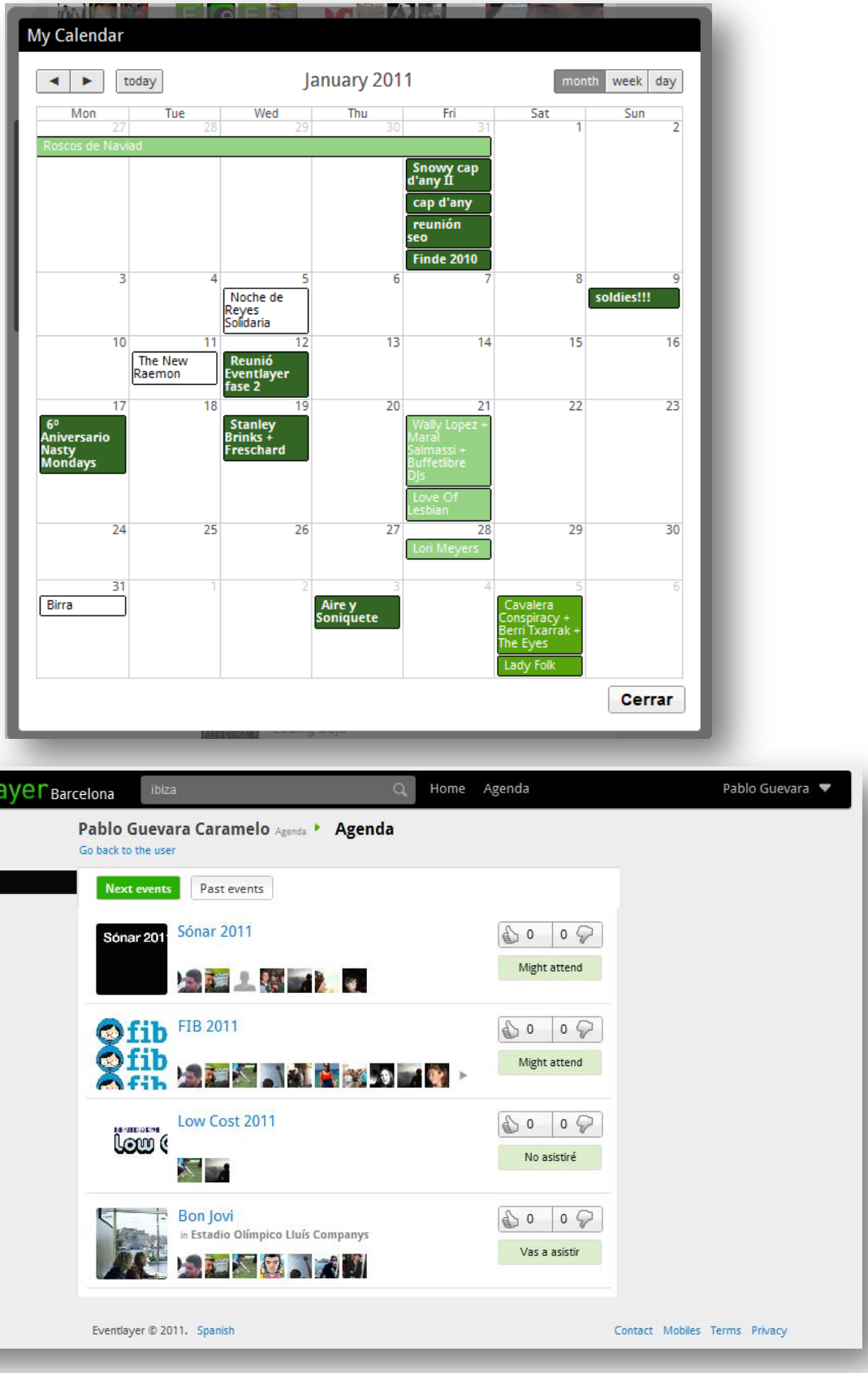


Ilustración 20 - Calendario y agenda en eventlayer.com

10. Cuando el usuario se registra en Eventlayer utilizando la conexión de Facebook se le añade una aplicación de Eventlayer en su cuenta de Facebook. Las ventajas de esa aplicación son que las notificaciones de Eventlayer están totalmente integradas con las de Facebook y además el usuario puede consultar su agenda y una selección de las mejores propuestas de la guía de ocio, todo sin salir de su cuenta de Facebook. A continuación unas capturas la aplicación de Eventlayer en Facebook

Ilustración 21 - Integración con Facebook de eventlayer.com

11. Finalmente las capturas de pantalla de las aplicaciones para el móvil. En el PFC de Adrià se pueden ver detalladamente todas y cada una de las pantallas en el apartado de implementación.

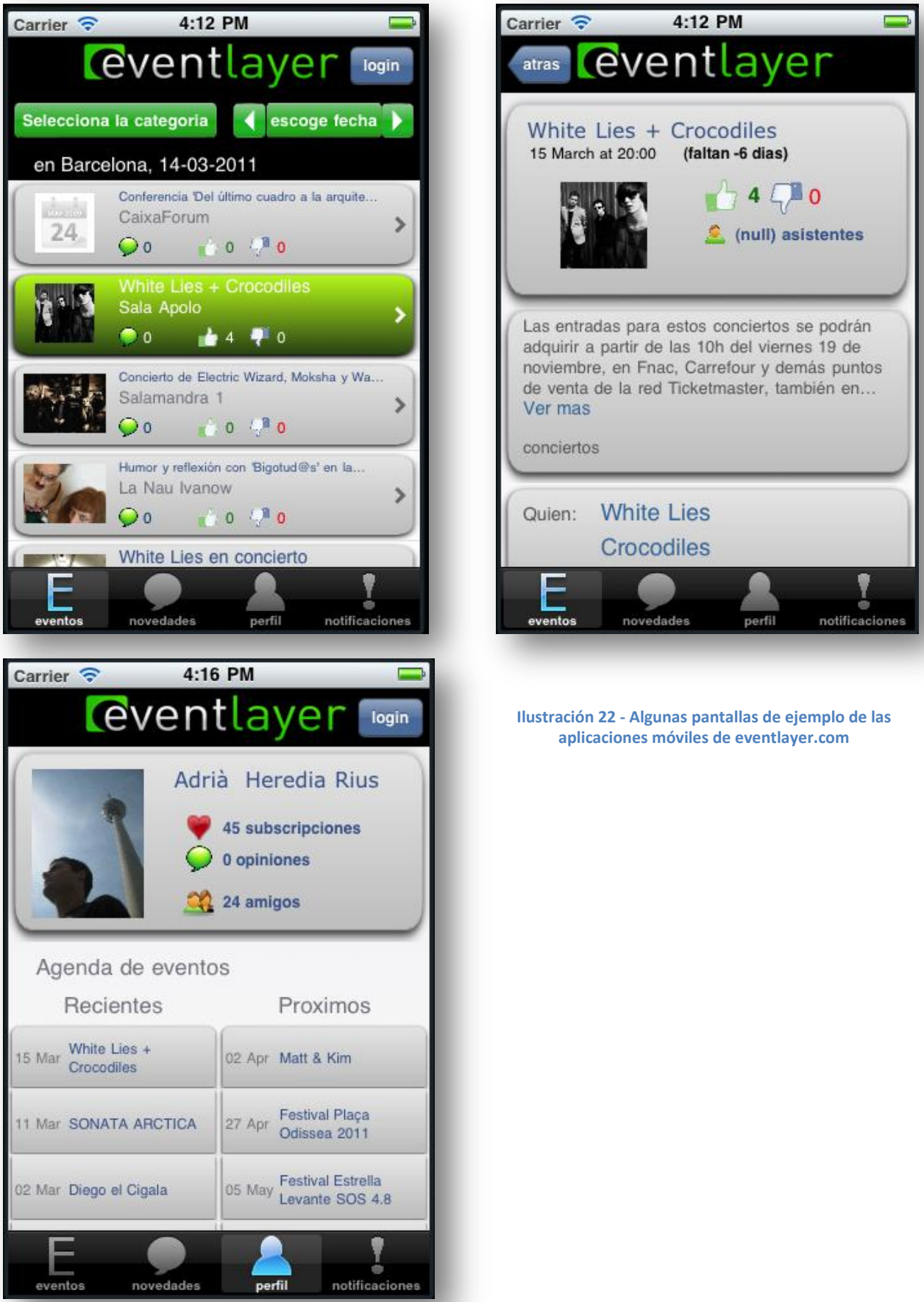


Ilustración 22 - Algunas pantallas de ejemplo de las aplicaciones móviles de eventlayer.com

1.2.2. Diferenciación (propuesta de valor)

Eventlayer es una guía de ocio local, simple y social. Intenta diferenciarse de las guías tradicionales utilizando las nuevas tecnologías y un diseño muy intuitivo. Los 2 puntos que conforman la propuesta de valor son:

1. El diseño de Eventlayer es **muy fácil de utilizar**; además, se mantiene una misma estructura durante toda la aplicación. De derecha a izquierda
   - Columna navegación. Siempre está visible aunque el usuario vaya haciendo scroll.
   - Contenido. Siempre se personaliza de la misma manera: para guardar y para esconder. El usuario no tiene que pasar de página, los elementos se van cargando automáticamente y la información de cada evento, descuento o lugar se abre en popups por lo que no se tiene que volver atrás a cada momento como pasa en las webs tradicionales.
   - Recomendado y publicidad. Columna reservada para contenido destacado y publicidad.

Todo **integrado** con el email, Facebook y accesible desde el **móvil** (aplicaciones nativas para iPhone y Android).

2. Además, Eventlayer es la primera guía de ocio social, el usuario puede **conocer gente y quedar con sus amigos**. Porque nadie va a los conciertos ni al cine sólo, las guías de ocio por lo tanto deben ser sociales y permitir al usuario opinar, preguntar a otros usuarios, compartir fotos, ver dónde van a ir sus amigos y quedar con ellos.

Ilustración 23 - Foto de un concierto, un ámbito naturalmente social

1.2.3. Tabla comparativa de funcionalidades

Hemos elaborado una tabla de las funcionalidades de Eventlayer agrupadas en 3 tipos:

- **Funcionalidades de guía de ocio**
- **Funcionalidades de red social**
- **Funcionalidades de aplicación móvil**

Además hemos escogido 8 productos reales que pueden verse como competencia, directa o indirecta de Eventlayer y hemos hecho un análisis de sus funcionalidades. Por lo tanto la siguiente tabla puede servir como un análisis comparativo entre Eventlayer y el resto de ofertas existentes en el mercado.

Funcionalidad	eventlayer.com	atrapalo.com	lanetro.com	timeout.com	salir.com
					Guía de ocio
Buscador de eventos	sí	sí	no	no	no
Descuentos	sí	sí	sí	no	sí
Opiniones	sí	sí	sí	sí	sí
Cartelera	sí	no	sí	no	sí
Top10 de lugares	sí	no	sí	sí	sí
Buscador de restaurantes	casi	sí	sí	casi	sí
Compra de entradas	no	con descuento	casi	casi	casi
					Funcionalidades sociales
Eventos entre amigos	sí	no	no	no	no
Widgets colaborativos (votaciones, elegir día...)	sí	no	no	no	no
Suscribirte a los eventos de tus artistas favoritos	sí	no	no	no	no
Usar tus amigos de Facebook	sí	no	no	no	no
Notificaciones en eventos	sí	no	no	no	no
Fotos en eventos	album colaborativo	no	no	no	no
					Aplicación móvil
Aplicación android	sí	no	no	no	no
Aplicación Iphone	sí	sí	no	sí	no
Aplicación BB	no	no	no	no	no
Avisos de eventos en móvil	sí	no	no	no	no

Funcionalidad	dooplan.com	kedin.es	nvivo.es	facebook.com
Buscador de eventos	sí	sí	sólo conciertos	no
Descuentos	sí	no	no	no
Opiniones	sí	sí	sí	no
Cartelera	no	no	no	no
Top10 de lugares	no	no	no	no
Buscador de restaurantes	no	no	no	no
Compra de entradas	no	no	sí	no
sociales				
Eventos entre amigos	no	no	no	sí
Widgets colaborativos (votaciones, elegir día...)	no	no	no	no
Suscribirte a los eventos de tus artistas favoritos	casi	no	sí	no
Usar tus amigos de Facebook	no	no	no	sí
Notificaciones en eventos	no	no	no	sí
Fotos en eventos	sí	sí	sí	sí
Aplicación android	no	no	no	sí
Aplicación Iphone	no	no	no	sí
Aplicación BB	no	no	no	sí
Avisos de eventos en móvil	no	no	no	sí

Ilustración 24 - Cuadro comparativo de Eventlayer frente a los productos de sus más directos competidores

1.2.4. Modelo de negocio

El modelo de negocio de un proyecto empresarial es la manera en la empresa va a generar ingresos. Eventlayer se sitúa en el sector del ocio y turismo (ver el apartado 1. Sector en el Plan de Marketingl). El modelo de negocio de Eventlayer se desarrolla por medio de la web www.eventlayer.com y se basa en los siguientes 3 puntos:

a) Venta de entradas

Venta de entradas de conciertos, obras de teatro, espectáculos, descuentos en actividades, menús de restaurantes…

En una fase inicial se empezará con la participación en programas de afiliados. Esto significa que, de momento, no nos dedicaremos a contactar directamente con los representantes de los artistas ni con los representantes de los restaurantes. Ya hay otras empresas actualmente que se dedican a ello y tienen como hemos dicho **programas de afiliados.** La afiliación consiste en que Eventlayer se lleva un porcentaje de cada una de las ventas que se realizan a través de nuestra web. En concreto, actualmente contamos con las siguientes posibilidades:

I. Empresas de venta de entradas de conciertos. Ejemplo: Ticketmaster.com
II. Empresa de reserva de restaurantes. Ejemplo: Restalo.com
III. Empresas de venta de descuentos. Ejemplo: Groupon.com

b) Venta de publicidad micro segmentada

Este es el punto fuerte de nuestro modelo de negocio, el hecho de que los usuarios utilicen en Eventlayer los botones de “me gusta” y “no me gusta” nos permite saber de cada usuario, no sólo los datos típicos como edad o sexo sino también sus gustos y sus patrones de asistencia a eventos y a lugares (ver el capítulo *2.3. Requisitos no funcionales* donde se trata la confidencialidad de sus datos).

Esto permite a Eventlayer mostrar un anuncio sólo a los usuarios de una determinada franja de edad pero que además acostumbren, por ejemplo, a asistir a eventos de Jazz. Esto nos permite cobrar la publicidad a un precio más elevado y además aportar un valor al usuario ya que el anuncio también le será interesante.

c) Patrocinio

Ofreceremos la posibilidad de un patrocinio con rediseño temporal de la web, mostrando la imagen de su marca en primer plano, a cualquier empresa relacionada con el ocio o la cultura. Un ejemplo de patrocinio como el que se llevó a cabo entre la marca de vodka Smirnoff y la guía de ocio TimeOut en Londres.

1.3. Objetivos del proyecto Eventlayer

El objetivo principal del proyecto Eventlayer es la construcción de un sistema de información que dé cobertura a los requisitos del producto que hemos planteado (Véase el capítulo *1.2. Producto)*. Más adelante, se detallan los objetivos específicos de este PFC que se derivan de las responsabilidades concretas asumidas por el alumno en la construcción del sistema.

Los objetivos generales de la construcción de Eventlayer son conseguir un:

- Sistema de información accesible vía internet por medio de un navegador que cumpla los estándares de la W3C[2] y que ofrezca funcionalidades de una guía de ocio como son
    1. Acceso a un listado personalizable de eventos (Conciertos, obras de teatro, exposiciones…)
    2. Acceso a un listado personalizable de descuentos
    3. Acceso a un listado personalizable de lugares de interés

- El sistema además debe ofrecer las funcionalidades sociales de
    1. Registro y administración de contactos
    2. Histórico de acciones de tus contactos y artistas favoritos (Muro[3])
    3. Herramientas para que el usuario organice fácilmente como va a asistir a los eventos con sus contactos

- El sistema debe ser de estilo RIA (Rich Internet Application)[4]

- El sistema debe ser accesible desde móviles de tipo Smartphone a través de aplicaciones específicas para 2 plataformas: iPhone y Android, que permitan acceder a un rango reducido de funcionalidades de la web vía móvil.

Además, como ya hemos comentado en el capítulo 1*. Introducción,* además de la construcción del sistema Eventlayer, que es el objetivo principal del PFC, nos hemos propuesto como objetivo secundario que www.eventlayer.com sea un proyecto empresarial viable. Los capítulos dedicados a este objetivo son los siguientes:

- *3.Planificación y análisis económico.* Se incluye una tabla con proyecciones financieras a 1 año de los gastos e ingresos de la futura empresa.
- *Anexo I - Plan de Marketing y apartado 1. Sector.* Planificación de lo que será el plan de marketing estratégico y operativo del producto.

[2] Estándares web definidos por la World Wide Web Consortium como son el Html, el Css…
[3] Facebook popularizó el uso del "Muro" una vista donde podemos ver de manera cronológica un segmento de las acciones realizadas por nuestros contactos dentro de la aplicación.
[4] Las aplicaciones RIA se caracterizan porque son aplicaciones web pero que no necesitan de recargas de página para ejecutar las acciones y por lo tanto son mucho más cómodas de utilizar, tanto como las aplicaciones de escritorio.

1.3.1. Esquema general del sistema

A continuación un diagrama en el que se pueden observar todos los subsistemas que componen la aplicación que hemos desarrollado. El diagrama nos servirá para dar al lector una visión general del sistema y además será útil a la hora de explicar que partes son responsabilidad de cada uno en el siguiente capítulo *1.3.2. División de responsabilidades en el proyecto Eventlayer* de este PFC.

Eventlayer es una aplicación web, por lo tanto el primer subsistema es el servidor donde está alojada. Además se puede acceder bien desde el navegador (Internet Explorer, Firefox…), segundo subsistema, o bien desde dispositivos móviles Android e iPhone que representan el tercer subsistema. A estos dos últimos subsistemas se les llama clientes, ya que son los que acceden a nuestro servidor. Los clientes se comunican con el servidor por medio de internet.

En cada subsistema encontramos diferentes bloques con esquinas redondeadas que según el patrón de software (patrón de capas) engloban diferentes tipos de subsistemas según responsabilidad. Los llamaremos por lo tanto capas. Por defecto tenemos la capa de controladores y vistas, la de clases de modelo y la de datos. Además en el diagrama encontramos otros bloques blancos pero sin bordes redondeados que representan las librerías de código externo que utilizamos en cada subsistema. Ejemplos de ello son Zend Framework o Jquery. Finalmente tenemos unos bloques en forma de cilindro que representan las diferentes bases de datos que utiliza nuestro sistema.

Además de todo esto, se pueden identificar en el diagrama unos bloques de color azul. Estos bloques representan lo que llamamos bloques de implementación. No se han representado todos, sólo los más importantes. Un bloque de implementación consta de una serie de líneas de código que actúan conjuntamente para resolver una tarea dentro del sistema. Por ejemplo todas las líneas de código que se ocupan de la gestión de errores del sistema pertenecen al bloque de implementación “gestión de errores” representado en el diagrama.

Los bloques de implementación nos sirven para dividirnos las responsabilidades durante el proceso de construcción de la aplicación. De esta forma cada uno de los miembros del equipo nos ocupamos de ciertos módulos de implementación concretos.

Por último, también encontramos otros símbolos que representan al usuario, el sistema de correo, cron (es un sistema de ejecución de tareas por intervalos), la api (Application interface) de Facebook… No son relevantes en este momento, el objetivo de este diagrama es que se tenga durante todo el capítulo de la introducción una idea de lo que representa el sistema del que estamos hablando (Eventlayer).

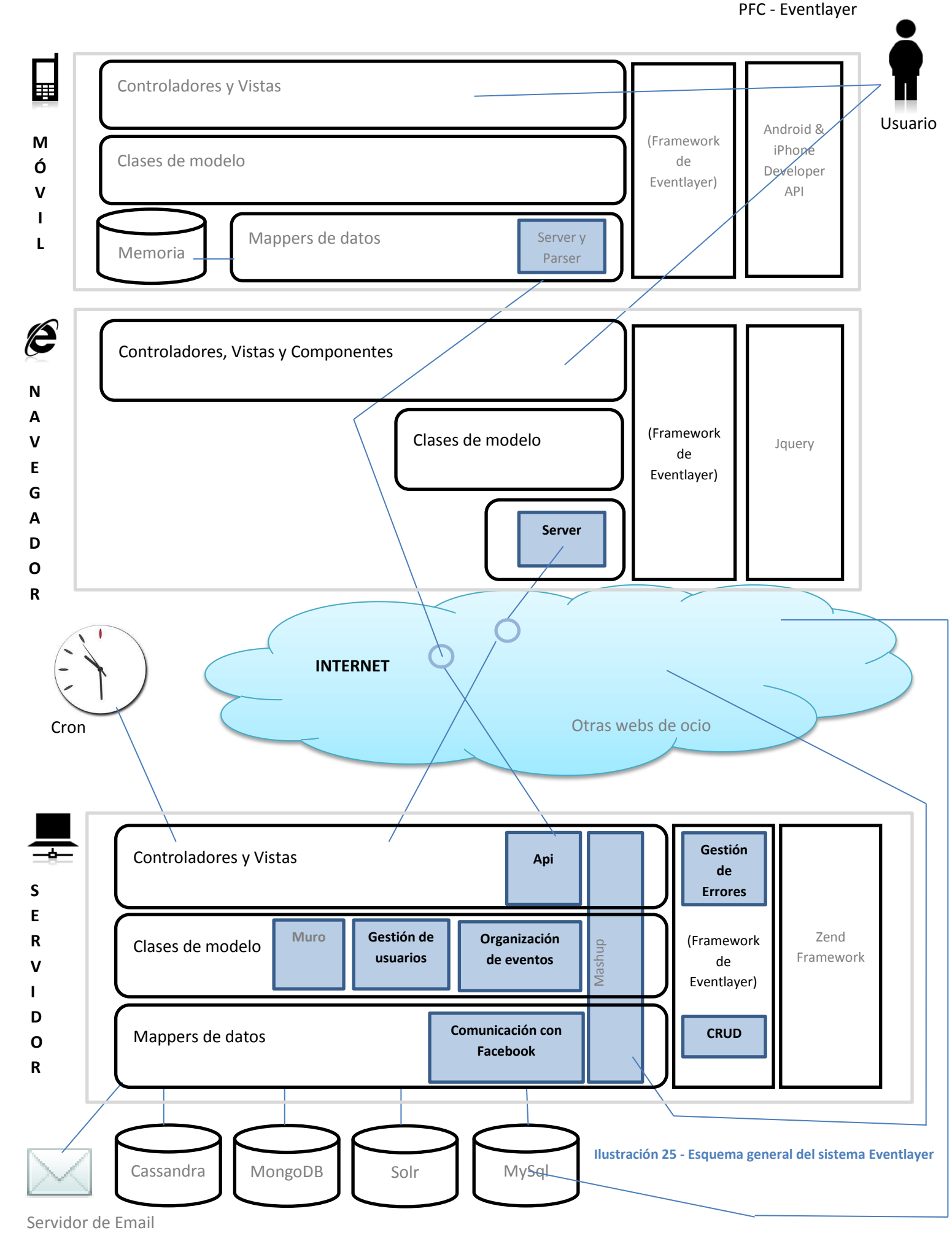


Ilustración 25 - Esquema general del sistema Eventlayer

1.3.2. División de responsabilidades en el proyecto Eventlayer

Como ya hemos dicho en el capítulo 1. Introducción, este PFC forma parte de un conjunto de tres. En este apartado vamos a describir la división de responsabilidades entre los diferentes miembros del equipo que permiten definir los objetivos específicos de cada uno de los tres PFCs

Una descripción general de esta división podría ser la siguiente:

- **Pablo Guevara.** Responsable del desarrollo de la **metodología de trabajo**, la **arquitectura de la aplicación web** y aparte de **implementar los módulos** que se describen en su memoria en el capítulo de Implementación. En resumen son:
    - Arquitectura
    - Gestión de errores
    - CRUD
    - Gestión de usuarios
    - Notificaciones
    - Gestión de asistencia a eventos
    - Integración con Facebook

Además se ha ocupado del plan de Marketing.

- **Sergi Pérez.** Responsable de la **elección de las tecnologías web**, **diseño y administración de las bases de datos, pruebas** y de **implementar los módulos** que se describen en su memoria en el capítulo *6. 1. Implementación sistema web (Parte Sergi Pérez).* En resumen son:
    - Muro
    - Sistema de búsquedas
    - Recomendador
    - Sistema estadístico
    - Mashup
    - Vistas

Además se ha ocupado del análisis de costes y evaluación de la planificación.

- **Adrià Heredia.** Responsable del diseño e implementación de las **aplicaciones para móviles.** En concreto 2 aplicaciones nativas una para Android y otra para IPhone.

1.3.2.1. Objetivos específicos de mi PFC

A continuación explicaré más en profundidad cuales han sido **mis objetivos concretos**.

En el diagrama del capítulo 1.3.1. Esquema general del sistema se puede ver de manera general cuales son mis responsabilidades en cuanto a los subsistemas de la aplicación.

- Texto en **negrita** representa subsistema de mi responsabilidad
- Texto en negro representa subsistema del que comparto responsabilidad
- Texto en gris representa subsistema del que no me responsabilizo

Ilustración 26 - Leyenda del gráfico 1.3.1. Esquema general del sistema - Partes de las que me responsabilizo

Mis objetivos concretos son por lo tanto:

a. Diseñar e implementar una arquitectura eficiente que soporte un sistema de información de acceso web alojado en un servidor dedicado adaptando funcionalidades de un framework existente en el mercado. En nuestro caso ha sido Zend Framework.

b. Diseñar e implementar una arquitectura eficiente que soporte un sistema cliente de tipo RIA (Rich Internet Application) en navegador (siguiendo W3C) adaptando funcionalidades de un framework existente en el mercado. En nuestro caso ha sido Jquery. El sistema debe ejecutarse en un navegador web (será en lenguaje JavaScript) y desarrollar parte de su lógica en el cliente (Mozilla, Chrome, Explorer…). Para que la aplicación sea muy usable esta debe minimizar las peticiones al servidor para servir una nueva página o una recarga de la misma, de aquí la importancia de este punto.

c. Diseñar e implementar una arquitectura eficiente que por medio de una interfaz estándar dé acceso al sistema a aplicaciones para teléfonos móviles. Utilizamos el protocolo REST[5] y las aplicaciones para móvil lo utilizan actualmente para comunicarse con el sistema web.

d. Diseñar e implementar un desarrollo eficiente para los siguientes módulos de implementación (Ver el capítulo *de Implementación*)
   - Gestión de errores
   - CRUD
   - Gestión de usuarios
   - Notificaciones
   - Gestión de asistencia a eventos
   - Integración con Facebook

e. Elaborar una buena documentación del proyecto y de la aplicación (esta memoria), comprensible y estructurada.

f. Proponer una metodología de desarrollo de software estándar y adaptarla a nuestras necesidades para el proyecto en su conjunto y coordinar su aplicación por parte del equipo.

Finalmente la parte relativa a Eventlayer como proyecto de negocio:

[5] Es un protocolo de comunicación entre aplicaciones que es la evolución del antiguo SOAP

g. Elaborar las partes de un Business Plan relativas a idea de negocio y plan de marketing.

1.4. Metodología utilizada

La metodología utilizada en la gestión de este proyecto ha sido Scrum, una metodología de gestión de proyectos de software de las llamadas "metodologías ágiles".

Este grupo de metodologías son la alternativa al Relational Unified Process (RUP). La diferencia principal es que en RUP las fases del proyecto son secuenciales, primero se realiza el análisis previo, una vez acabado se empieza con la especificación, a continuación el diseño y por último la implementación y pruebas. Una etapa no comienza hasta que la anterior ha acabado. Por el contrario las metodologías ágiles se basan en un proceso iterativo y de prototipado. Esto significa que se realizan todas las fases citadas pero de manera parcial y se repiten una y otra vez (iteración) en cadena. De esta manera el producto va convergiendo un poco más en cada iteración hasta lograr el alcance [6] que se desea.

Hemos preferido el uso de Scrum frente a RUP, primero de todo, porque es el estándar de facto en este tipo de proyectos web/startup. El motivo principal es la "volatilidad en los requisitos". Esto significa que en este tipo de proyectos los requisitos no están totalmente definidos desde el principio, el producto final no está claro y por lo tanto ese proceso iterativo del que hablábamos nos permitirá ir construyendo el sistema poco a poco a medida que vayamos descubriendo los nuevos requisitos.

Por ejemplo, en un principio no nos habíamos planteado tener descuentos en Eventlayer. Por lo tanto no hubiese sido contemplado en la primera fase de especificación de RUP y por lo tanto nunca lo hubiésemos incluido en el sistema final. Esto no es compatible con una startup, un tipo de empresas en las que hay que mantenerse "ágiles" para sobrevivir. La única ventaja de una startup frente a las grandes compañías es precisamente su tamaño reducido que les permite adaptarse a las oportunidades del mercado más rápidamente. Es por eso que en reacción al gran boom que están siendo hoy, a día 28 de Abril del 2011, empresas como LetsBonus, Grupalia o Groupon, Eventlayer como producto y por lo tanto como sistema debe sacar partido a esa oportunidad de negocio. Esto supone cambiar los requisitos iniciales y funcionalidades que planteábamos al principio ni siquiera llegan a especificarse porque deben dejar paso a otras nuevas y más importantes.

Otra razón importante de la elección de Scrum frente a RUP es el coste de recursos que requiere una metodología frente a la otra. RUP es una gran metodología ya que ofrece un camino muy guiado. Establece unos pasos que si se siguen reducen el margen de error del proyecto y aumentan su trazabilidad. A cambio requiere de la dedicación de más horas a tareas que no contribuyen directamente en la creación del sistema en cuestión, con esto me refiero a que no aportan código al software que hay que desarrollar. Las metodologías ágiles por el contrario no son tan guiadas y el equipo debe utilizar más su pericia. A cambio permiten desarrollar el sistema más rápidamente y haciéndolo de una manera más directa. Esto es evidente en el número de artefactos que plantea

[6] Nos referimos a alcance como al alcance de un proyecto, es decir, qué se contempla en los requisitos y qué no.

RUP, lo que viene a ser documentación, diagrama de casos de uso, diagrama de clases, documentos de análisis de requisitos… frente a Scrum que sólo cuenta con Product Backlog y Sprint Backlog.

1.4.1. Agile development & Scrum

Ilustración 27 - Cabecera del Scrum Body of Knowledge

Hemos dicho que utilizaremos la metodología Scrum y que pertenece a la familia de metodologías ágiles. A continuación expondremos las características más importantes de Scrum que serán las que seguiremos. Se desprenden parte del Agile Manifesto[7] y parte del Scrum Body of Knowledge[8]. Empezaremos por las primeras que son comunes a todas las metodologías ágiles.

- Se minimizan los riesgos desarrollando software en cortos lapsos de tiempo. El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

- Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación.

- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

A parte de estas pautas comunes en todas las metodologías ágiles, Scrum también define las suyas particulares.

[7] Es un documento colgado en http://agilemanifesto.org/ [2011, 3 de Junio] y que es la base de la que parten todos las metodologías ágiles

[8] Es un documento colgado en http://www.scrum.org/scrumguides/ [2011,3 de Junio] y que redacta las directrices principales que conforman la metodología Scrum

Los roles[9] principales en Scrum son el ScrumMaster, que mantiene los procesos y trabaja de forma similar al director de proyecto, el ProductOwner, que representa a los stakeholders (interesados externos o internos), y el Team que incluye a los desarrolladores.

Durante cada sprint, un periodo entre 15 y 30 días (la magnitud es definida por el equipo), el equipo crea un incremento de software potencialmente entregable (utilizable). El conjunto de características que forma parte de cada sprint viene del Product Backlog, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Estos requisitos se redactan en forma de lo que se denomina **historias de usuario** y vienen a ser explicaciones en lenguaje natural del requisito. En el capítulo *2.2 Historias principales* hay algo más de teoría al respecto de las historias de usuario en Scrum.

Los elementos del Product Backlog que forman parte del sprint se determinan durante la reunión de Sprint Planning. Durante esta reunión, el Product Owner identifica los elementos del Product Backlog que quiere ver completados y los hace saber al equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Scrum permite la creación de equipos auto-organizados impulsando la co-localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan (a menudo llamado requirements churn), y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas "postit" y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar.

1.4.2. Fases del proyecto

Hasta aquí hemos explicado las características de la metodología escogida y también hemos justificado su elección, ahora es el momento de explicar cómo la hemos utilizado en este proyecto y como afecta esto a la memoria.

La base de toda la gestión del proyecto es el Product Backlog, es el artefacto en el que se escriben en forma de lista todas las historias (lo que vienen a ser requisitos) que hay que realizar en el sistema. La metodología Scrum también habla de otros artefactos como el gráfico Burn down para medir la velocidad de los programadores pero no lo vimos necesario.

[9] Tipos de participantes en un proyecto

Dicho esto, definimos que la **primera fase** del proyecto fue la creación de un primer Product Backlog que se fue modificando a lo largo de todos los sprints que vinieron a continuación.

Por otra parte como ya hemos dicho la base de Scrum es la iteración, a continuación se adjuntan las sub-fases que se repiten para cada iteración, llamada Sprint. Nuestras iteraciones oscilaban entre 30 y 60 días según los problemas que se presentaran en su implementación. Por lo tanto esta serie de iteraciones puede considerarse la **segunda fase** de nuestro proyecto.

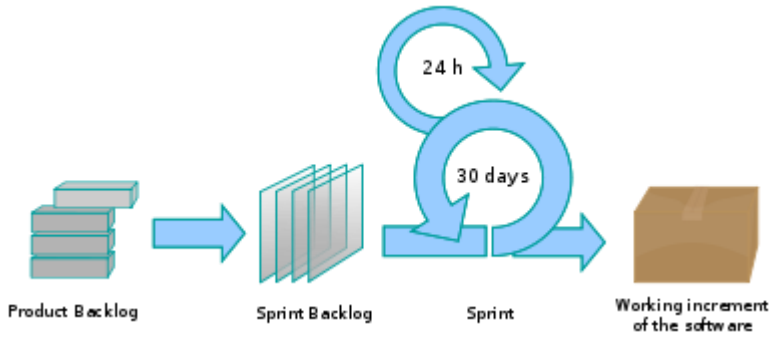


Ilustración 28 - Fases de Scrum

- **Sprint Planning Meeting**: Esta reunión da inicio a un sprint y básicamente consiste en priorizar las historias del Product Backlog y escoger cuales se van a realizar en esa iteración, se agrupan y se crea el Sprint Backlog.

- **Especificación:** En este tipo de proyectos se suele utilizar lo que se llama TDD (Test Driven Development)[10]. Esto significa que la especificación consiste en definir una serie de juegos de prueba que el sistema debe pasar. Nosotros hemos optado por el DDD (Design Driven Development)[11] que se basa en que la especificación son las pantallas finales del sistema. Por lo tanto nuestra especificación ha consistido en definir como serían las pantallas y las acciones relativas a cada historia.

- **Aprendizaje**: Para muchas historias nos hacía falta documentarnos sobre ciertas tecnologías necesarias para su implementación. Por lo tanto esta fase era a menudo necesaria.

- **Diseño:** En este apartado hay que aclarar que a la hora de priorizar las historias, como es lógico, se empezaron por las relativas a la arquitectura global del sistema y a la elección de las tecnologías que íbamos a utilizar. Por lo tanto para el resto de historias, en la fase de diseño se buscaban soluciones que cumplieran con la especificación en base a las tecnologías que habíamos elegido y a la arquitectura desarrollada.

- **Implementación:** Como en cualquier proyecto de software trasladar las decisiones tomadas en la fase de diseño a código fuente.

[10] Visitar la web http://en.wikipedia.org/wiki/Test-driven_development [2011, 6 de Junio]
[9] Visitar la web: http://www.designdrivendevelopment.org/ [2011, 6 de Junio]

- **Documentación:** Todo el código debía estar comentado, además para las historias más importantes se desarrollaba una documentación extra que es la que se ha incluido en esta memoria. En el punto 1.4.3 Documentación se amplía este párrafo.

- **Pruebas:** En el TDD las pruebas representan la primera tarea, no obstante en el DDD las pruebas se realizan una vez implementadas las historias y se hacen a la manera tradicional. Consultar el punto 6 *Pruebas* para más información.

Por último la **tercera fase** de este PFC ha sido la redacción de esta memoria. La memoria vendría a ser una extensión de la sub-fase "Documentación" incluida en cada una de las iteraciones.

1.4.3. Documentación

Podemos entender como documentación todo aquel material que sin ser parte del resultado final del sistema a desarrollar es necesario para el desarrollo del proyecto. En esta definición entran por una parte, los artefactos que exige la metodología y por otra, la documentación del código típica en cualquier sistema de software.

En cuanto a los artefactos de Scrum, hemos utilizado los principales el Product Backlog y el Sprint Backlog. En el apartado 0

Análisis de requisitos y especificación se incluye un resumen del Product Backlog utilizado y los Sprint Backlogs al ser una división del Product Backlog no son relevantes y no se han incluido.

En cuanto a la documentación del código, Scrum no establece ningún patrón a seguir. En RUP por ejemplo se hace un uso intensivo del lenguaje UML (Unified Modeling Language), pero no es así en las metodologías ágiles.

a) Documentación UML

No obstante, una de las críticas a las metodologías ágiles es la falta de documentación y nosotros personalmente creemos en la importancia de un código bien documentado. Es por eso que recurrimos a la opinión de uno de los mayores expertos en lo que a desarrollo de software se refiere, el señor Martin Fowler (martinfowler.com), autor de libros tan famosos como "Patterns of Enterprise Application Architecture" o "UML Distilled: A Brief Guide to the Standard Object Modeling Language".

Fowler habla de los 3 UML Modes[12]:

[12] Visitar Martin Fowler. http://martinfowler.com/bliki/UmlMode.html [2011, 6 de Junio]. Simplemente se refiere a un tipo de sintaxis de este lenguaje.

Ilustración 29 - Logo de UML

- **UmlAsSketch.** Consiste en utilizar UML para compartir ideas y documentar. En este modo los desarrolladores usan el UML para unificar la comunicación de diferentes aspectos de un sistema. Se pueden utilizar estos esbozos en dos direcciones. La primera es la dirección natural, primero el esbozo y a partir del esbozo se crea el sistema. La segunda, consiste en que a partir del código del sistema se realiza el esbozo en UML a modo de documentación y se denomina ingeniería inversa.

- **UmlAsBlueprint**. Consiste en utilizar UML para diseñar tu software antes de implementarlo. Durante mucho tiempo la ingeniería tradicional ha influido en los procesos de desarrollo de software. Un ejemplo de esto es la cantidad de esfuerzos que se han realizado para obtener técnicas que nos permitieran experimentar con los diseños antes de llevarlos a cabo, algo así como los planos de un puente. Si bien es fácil ver que llevar a cabo nuestras pruebas en un plano en lugar de hacerlo con el puente en sí es un ahorro significativo en el desarrollo de software no está tan clara esta ventaja. Es por eso que hay grandes críticos sobre este modo de UML.

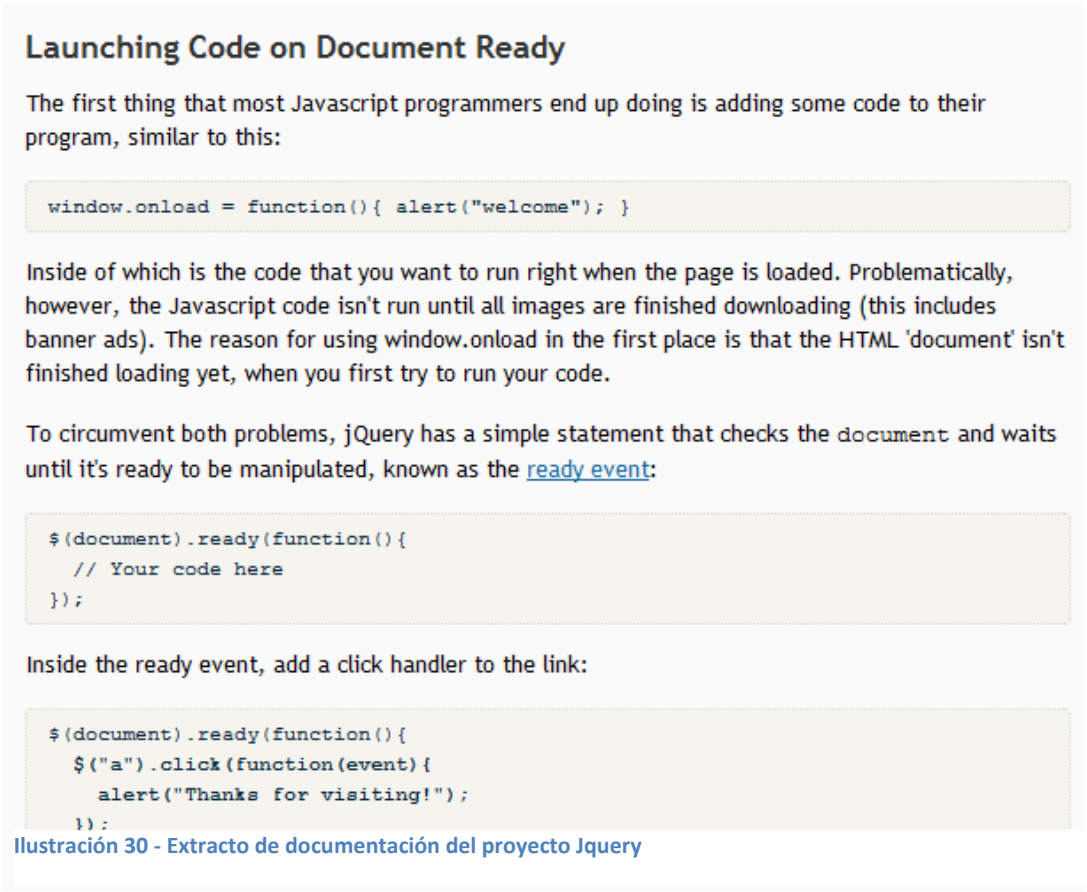
- **UmlAsProgrammingLanguage.** Consiste en utilizar UML como tu lenguaje de programación. Si eres capaz de detallar suficiente tus diagramas UML y proveer suficiente semántica para las necesidades de tu software este modo de UML podría ser tu lenguaje de programación. Existen herramientas que pueden compilar tus diagramas UML directamente a programas ejecutables. La pregunta, por supuesto, es si esta promesa es real o no. Según Fowler no lo es, la idea de un lenguaje de programación gráfico (el UML lo es) en lugar de los sistemas actuales que trabajan en formato algorítmico no tiene comparación.

La metodología RUP aconseja como documentación el UmlAsSketch y el UmlAsBlueprint. Scrum no habla de ninguna de ellas. Nosotros siguiendo las reflexiones de Fowler optaremos por documentar nuestro código con UmlAsSketch y lo haremos a modo de ingeniería inversa.

Ejemplo de ello serán los diagramas de clases de diseño del sistemay de otros diagramas que veamos necesarios para compartir ideas en los capítulos de 5 *Implementación del sistema.*

b) Documentación con ejemplos de código fuente

Por otro lado también hemos tomado como ejemplo el formato de documentación pública de varios proyectos de gran envergadura como son Jquery (www.jquery.com) y Zend Framework (www.zendframework.com). Su documentación se basa en una mezcla entre código y explicaciones relacionadas con ese código. A continuación un ejemplo:



Launching Code on Document Ready

The first thing that most Javascript programmers end up doing is adding some code to their program, similar to this:

```
window.onload = function(){ alert("welcome"); }
```

Inside of which is the code that you want to run right when the page is loaded. Problematically, however, the Javascript code isn't run until all images are finished downloading (this includes banner ads). The reason for using window.onload in the first place is that the HTML 'document' isn't finished loading yet, when you first try to run your code.

To circumvent both problems, jQuery has a simple statement that checks the `document` and waits until it's ready to be manipulated, known as the ready event:

```
$(document).ready(function(){
  // Your code here
});
```

Inside the ready event, add a click handler to the link:

```
$(document).ready(function(){
  $("a").click(function(event){
    alert("Thanks for visiting!");
  });
```

Ilustración 30 - Extracto de documentación del proyecto Jquery

Nuestra idea es enfocar los apartados de 4 *Diseño del sistema* y 5 *Implementación del sistema* de esta manera pero como hemos dicho antes incluyendo además, fragmentos de diagramas UML cuando sea necesario.

c) Documentación con phpDocumentor

Por último existe la buena práctica[13] de documentar en el mismo código. Se trata de añadir, además de los típicos comentarios en ciertas líneas para describir nuestro código, otro tipo de comentarios especiales en los que describimos los contratos (Precondición y Postcondición[14]) de cada función. La ventaja de esta documentación es que gracias a programas como phpDocumentor (www.phpdoc.org) se consigue unir esa información a la del propio código generando una documentación muy completa de todo el sistema. Nosotros hemos creído conveniente utilizarla, no con el fin de incluirla en la memoria pero sí porque resulta muy útil para mantener el código fuente del proyecto en versiones futuras.

[13] Se conoce como buena práctica, es una recomendación casi un estándar de facto

[14] Es una especie de contrato que se utiliza en ingeniería de software que define que espera una función y que debe retornar

```
/**
   * Deletes the entities
   *
   * @param array $collection
   * @param boolean $persist
   *
   * @return null
   */
  public function delete(array $collection = null, $persist = true) {
      $this->_preDelete($collection);
      $this->_postDelete($collection,$persist);
  }
```

Ilustración 31 - Ejemplo de comentario phpDoc (color azul)

```
$writer = new Zend_Log_Writer_Stream('php://output');
$formatter = new Zend_Log_Formatter_Simple('hello %message%' . PHP_EOL);
// Here we set the formater
$writer->setFormatter($formatter);
```

Ilustración 32 - Ejemplo de comentario normal (color verde)

1.4.4. Colaboración y coordinación

Como ya hemos dicho, este PFC lo hemos llevado a cabo 3 estudiantes. En este contexto la coordinación y un buen entorno colaborativo es clave para tirar adelante el proyecto.

Por suerte Scrum, influido por el Agile Development tiene pautas muy útiles al respecto. Básicamente la idea principal viene a ser:

"*Los métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación.*"
Agile Manifesto

Además establece el concepto de "timeboxed" que viene a decir que cualquier tarea de comunicación debe tener un tiempo delimitado previamente. Esto es así para evitar reuniones maratonianas en las que se pierde el norte de la discusión.

a) Reuniones

Hay 2 reuniones básicas a seguir:

- **Scrum Planning Meeting.** De esta ya hemos hablado antes, es una reunión cada 3 o 4 semanas en la que se concreta una especie de planning para el próximo sprint.

- **Daily Scrum.** Es una reunión diaria entre todos los miembros del equipo que se realiza a primera hora. El timebox es de 10 minutos máximo y según el Agile Manifesto todos los miembros tienen que estar de pie. Básicamente para que a esas horas de la mañana nadie se duerma en la silla. En esta reunión cada integrante del equipo contesta a 3 preguntas:

    - ¿Qué tenía que hacer ayer y qué es lo que acabé haciendo?
    - ¿Por qué no hice todo lo que me propuse hacer? Problemas que ocurrieron.
    - ¿Qué haré hoy?

La reunión de Scrum Planning era importante para que no ocurrieran desviaciones importantes de tiempo y para priorizar las tareas mientras que el Daily Scrum era básico para saber si existía algún problema puntual.

Estas reuniones y otras de tipo consulta las llevábamos a cabo usando el software de videoconferencia Skype. Este programa es gratuito y permite conversaciones individuales y también conversaciones en grupo. Además se le puede añadir el plugin Teamviewer que permite compartir la pantalla.

Para compartir una pizarra en la que hacer algún diagrama, no ocurrió demasiadas veces, utilizábamos un Google Docs de tipo “drawing”.

b) Artefactos

Como ya he comentado en varias ocasiones, sólo hemos utilizado 2 artefactos. Product Backlog y Sprint Backlog. Estos artefactos son básicamente listas priorizadas de tareas, se llaman historias. Hay quien utiliza postits, otros prefieren software especializado, cualquier formato puede servir. Nosotros hemos utilizamos una hoja de cálculo de Google Docs para el Product backlog y Teambox para el Sprint Backlog.

Teambox es una herramienta de gestión de proyectos muy simple, es parecido a un Google Groups pero más usable y con alguna que otra función como la de tener listas de tareas y poderlas asignar a diferentes usuarios. Es esta última funcionalidad la que más nos ha servido.

A continuación incluyo un ejemplo de Sprint Backlog que ejecutamos entre los meses de Marzo y Abril. Es un Sprint de 40 días en el que se han completado 24 historias y aún quedan 13 por acabar. Recuerdo que calculamos bastante mal y no lo acabamos a tiempo. Como se puede observar hay historias de diferentes cargas de trabajo desde un bug como es,

- “ group counts in event”: básicamente la tarea era revisar que no se mostraban correctamente la cantidad de asistentes en un evento. Aproximadamente 2 horas de trabajo.

Hasta una historia tan completa como:

- " fb": realizar la integración con Facebook de nuestra aplicación. Aproximadamente 50 horas de trabajo.

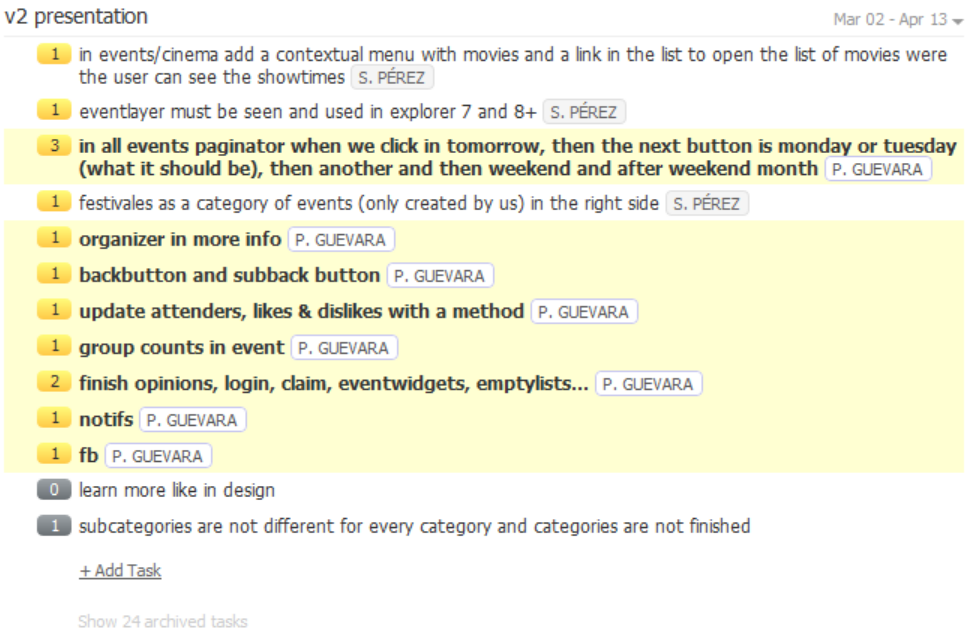


Ilustración 33 - Ejemplo de Sprint Backlog en Teambox

c) Código

Para compartir el código fuente utilizábamos SVN (Subversion). Se trata de un software de control de versiones que se integraba totalmente con nuestro IDE (Interface Development Environment), en nuestro caso Eclipse, y nos permitía trabajar con diferentes versiones del código.

Este tema ya está muy controlado hoy en día en la industria de desarrollo de software por lo que básicamente escogimos el estándar de facto.

d) Otros

Por último, citar el uso de otras herramientas interesantes en nuestro entorno colaborativo:

- Dropbox: Es un disco duro virtual que además permite compartir directorios. Toda la documentación, incluida esta memoria la archivábamos allí de tal manera que todos los documentos estaban disponibles para todos los miembros del equipo.

- TestSwarm, PhpUnderControl: Estos dos programas sirven para desarrollar en un entorno que trabaje con TDD (Test Driven Development). Nosotros perdimos mucho tiempo instalándolos y configurándolos. Al final dejamos de utilizarlos porque aunque sus ventajas a largo plazo son claras necesitan demasiada implicación para un equipo tan pequeño como era el nuestro.

2. Análisis de requisitos y especificación

Como hemos visto en el capítulo *1.4.Metodología utilizada* la fase de análisis de requisitos y especificación se va realizando y cambiando a lo largo de todo el proyecto, a cada iteración, concretamente en la reunión de "Scrum Planning Meeting". Por ese motivo este capítulo se ha redactado a posteriori, es decir una vez acabado el proyecto, y es la suma o mejor dicho evolución de la especificación y de los requisitos una vez finalizada la aplicación. Por último aclarar que los requisitos que trataremos a continuación son la suma de los requisitos de los 3 PFCs con el fin de dar al lector una visión global de los requisitos del sistema Eventlayer.

2.1. Actores

En una web como Eventlayer destinada a ser usada para fines lúdicos y no de trabajo, el conocer al usuario de tu sistema es crítico. En el mejor caso existirán 100 sustitutivos en la red que el usuario puede utilizar en lugar de tu aplicación para "resolver su necesidad". Digo en el mejor de los casos porque la mayoría de veces los productos de los que hablamos buscan "crear una necesidad" en el usuario.

Esto nos obliga a cuidar mucho la experiencia del usuario (User Experience) porque de otro modo no va a utilizar nuestra aplicación ya que no existe una clara necesidad para hacerlo. A lo que me refiero es que el usuario debe disfrutar con el uso de la aplicación, su manejo debe ser "engaging[15]". Y para esto hay que tener muy en cuenta el "actor" con el que nuestra aplicación está tratando.

Estos dos párrafos están relacionados con la identificación de los actores por un lado y con el requisito no funcional llamado "usabilidad" (ver capítulo *2.3.Requisitos no funcionales* apartado Usabilidad).

Un ejemplo práctico con el que nos hemos encontrado son los siguientes diseños de la aplicación. Los dos representan el perfil de un usuario de Eventlayer. El problema es que el primer diseño contiene demasiada información. Seguramente no sería ningún problema si esta pantalla, léase historia, fuese dirigida a un actor tipo "trabajador de Eventlayer" al fin y al cabo es la herramienta que utiliza para trabajar y es comprensible que requiera una pequeña curva de aprendizaje. No obstante el actor tipo "usuario que viene a buscar algo para hacer esta noche" no tiene porqué considerar ningún tipo de proceso de aprendizaje, en ese caso cierra la aplicación y simplemente se va al bar de todos los fines de semana. Es por ese motivo que nos vimos obligados a desarrollar un diseño número 2, véase la figura en la siguiente página.

[15] Término anglosajón utilizado para describir, en este caso una aplicación web, que atraiga al usuario hasta el punto de tener la necesidad de volver a usarla.

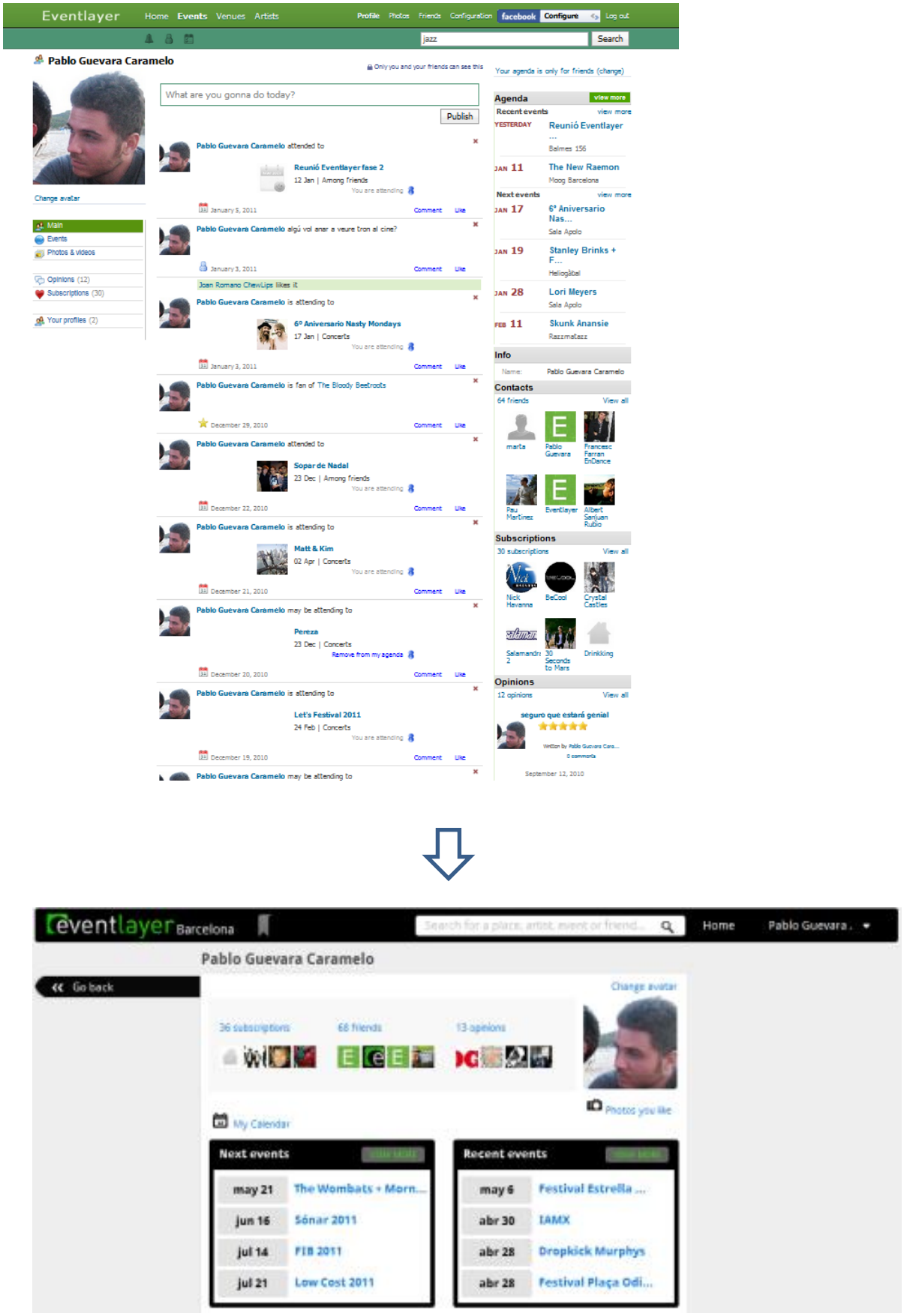


Ilustración 34 - Diseño de Perfil de usuario en la versión 1 (arriba) y en la versión 2 (abajo)

Los actores y por lo tanto los hábitos de uso (historias[16]) en la aplicación son los siguientes:

- **Usuario estándar.** Identificador "ANONYMOUS" si no está autentificado o "USER" si lo está. Aquél usuario que entra en Eventlayer.com buscando alguna propuesta de ocio para hacer, ya sea un evento, un descuento o un lugar de interés.

  Las historias relacionadas deben tener en cuenta que el requisito no funcional de usabilidad es muy importante.

  Este conjunto de historias representan la gran mayoría en el sistema y podemos identificar dos patrones:

  - Historias relacionadas con la guía de ocio: en su mayoría son de lectura de información en el sistema. Un par de ejemplo:
    - USER quiere buscar un evento por nombre para ver su información.
    - USER buscar conciertos de jazz para salir con sus amigos la semana que viene.

  - Historias relacionadas con la red social: aquí encontramos tanto de lectura como de escritura. Un par de ejemplos:
    - USER quiere consulta el perfil de un contacto para ver a qué eventos va a asistir.
    - USER quiere editar los datos de su fiesta de cumpleaños para qué sus amigos se enteren.

- **Promotor de eventos.** Identificador "PROMOTER". Aquél usuario que utilice Eventlayer.com para promocionar sus eventos, sus locales o los artistas a los que representa.

  El factor usabilidad no es tan importante y el actor valora en mayor medida la cantidad de funcionalidades diferentes que pueda utilizar (cantidad de historias específicas).

  En un futuro será necesario establecer diferenciación entre historias de un usuario de pago e historias de un usuario común pero no entra dentro del alcance de este PFC.

  Las historias relacionadas con este actor son del grupo "Historias relacionadas con la guía de ocio" y serán en su mayoría de actualización, es decir, aportar nuevo contenido a Eventlayer como guía de ocio. Un par de ejemplos:
    - PROMOTER quiere crear un perfil de un local para que la gente encuentre la información de su local en la guía de ocio y deje opiniones en él.
    - PROMOTER quiere crear un perfil de un artista que representa para que la gente pueda seguir su agenda de conciertos.

[16] Durante los siguientes capítulos al referirnos a historia estaremos hablando de las "historias de usuario" utilizadas en Scrum, ver capítulo *2.4.1. Agile development & Scrum.*

- **Trabajador de Eventlayer.** Identificador “STAFF”. Usuario responsable de la administración del sistema. Sus historias estarán relacionadas con tareas de administrador.

  La usabilidad en este caso no es un factor crítico, ayuda en el sentido que eleva la productividad en el STAFF que se dedique a tareas de gestión. No obstante, si lo comparamos con el resto de actores este requisito no tiene tanto peso aquí.

  Las historias relacionadas con este actor las clasificaremos como “historias de administración”. Un par de ejemplos de historias de STAFF:
    - STAFF quiere consultar los errores que se han producido en la aplicación para corregir su código.
    - STAFF quiere importar nuevos eventos de otras webs para que aparezcan en el listado de eventos de la aplicación.

2.2. Historias principales

Una buena manera de describir las historias en Scrum es siguiendo el siguiente patrón:

[identificador de actor] **quiere** [objetivo] **para** [razón]

Ilustración 35 - Patrón para describir las historias de usuario

Un ejemplo de esto sería: USER **quiere** encontrar un restaurante **para** ir con sus amigos.

Por otra parte, en Scrum podemos encontrar 2 tipos principales de historias:

- **Historias de usuario**. Son historias que el usuario lleva a cabo en la aplicación. Un par de ejemplos:
    - ANONYMOUS quiere registrarse en la aplicación para poder ejecutar historias de USER.
    - PROMOTER quiere añadir un evento para promocionarlo.

- **Historias técnicas**. Las historias técnicas son todas las historias que no son de tipo “historia de usuario”. Por lo tanto no contienen en su descripción el elemento [identificador de actor] que es el inicio de las historias en el patrón de la *Ilustración 35 - Patrón para describir las historias de usuario,* el resto del patrón se sigue igual. Un par de ejemplos:
    - Quiero que el sistema se inicie más rápidamente para que el usuario no tenga que esperar tanto entre acciones en la aplicación.
    - Quiero que el sistema permita loggear todos los errores ocurridos en la aplicación.

En cuanto a los product backlogs que hemos utilizado durante el proyecto, los atributos que incluíamos en estos artefactos eran los siguientes:

Identificador	Actores	Descripción	Coste	Prioridad	Módulo de implementación	Responsable
Una cadena que identifica a la historia. Empiezan por U- las historias de usuario y por T- las historias técnicas.	Los identificadores de los actores que participan	Descripción de la historia	Coste estimado de la historia	Cuanto mayor es el número más importancia tiene la historia	Identificador del módulo de implementación al que pertenece la historia	PABLO, SERGI O ADRIÀ según quién sea el responsable de la implementación de la historia

Ilustración 36 - Tabla descriptiva de nuestros Product Backlogs

Además de los campos de la tabla cada una de las historias normalmente debería de constar de 3 partes:

a. **Tarjeta**. Una descripción concreta, siguiendo el patrón de la *Ilustración 35 - Patrón para describir las historias de usuario.* Se refiere al campo Descripción de la tabla.
b. **Conversación**. Una sección donde anotar información extra de la historia.
c. **Confirmación**. Una sección donde se establecen los test que debe pasar la historia para que sea aceptada. Normalmente se refiere a test unitarios en TDD (Test Driven Development) ver *1.4 Metodología utilizada* para más información.

No obstante en los backlogs que detallaremos a continuación sólo hemos dejado los campos Identificador, Actores y Descripción; además, tampoco hemos incluido las partes b. Conversación y c. Confirmación ya que nuestra idea en este apartado de la memoria es simplemente dar una idea general de los requisitos que cumple Eventlayer como aplicación.

Por último aclarar que las historias que incluimos a continuación son representativas. En el transcurso de los sprints las historias eran más específicas, eso quiere decir que cada una de las historias de las siguientes tablas en realidad han sido 2 o 3 historias más concretas y más cortas con tal de no tener unos sprints demasiado largos.

2.2.1. Web Backlog

Identificador	Actores	Descripción	Responsable
		Historias relacionadas con la guía de ocio	
U-0001	USER & ANONYMOUS	actores quieren encontrar un evento para salir esta noche, mañana, esta semana o este mes	PABLO
U-0002	USER & ANONYMOUS	actores quieren ver cuáles son los próximos conciertos, obras de teatro, etc. para ver su información	PABLO
U-0003	USER & ANONYMOUS	actores quieren ver un listado (Top10) de los mejores lugares de interés para poder ver su información, su ubicación y opiniones de otros usuarios	PABLO
U-0004	USER & ANONYMOUS	actores quieren encontrar descuentos para poder comprar ofertas de ocio con descuentos a partir del 50%	PABLO
U-0005	USER & ANONYMOUS	actores quieren buscar un evento, lugar, artista o contacto para ver su información	SERGI
U-0006	USER	USER quiere ver su calendario para poder ver cuáles son sus próximos eventos	PABLO
U-0007	USER	USER quiere añadir los eventos que le interesan a su calendario para no olvidarse de cuando se celebran	PABLO
U-0008	USER	USER quiere suscribirse a un artista o lugar para recibir alertas de sus nuevos conciertos u otra clase de eventos que se celebren en el lugar o con el artista al que se van a suscribir	SERGI
U-0009	USER	USER quiere dejar una opinión en un artista o en un lugar para que otros usuarios la lean y la comenten	PABLO
U-0010	USER	USER quiere dejar un comentario en un evento para que otros usuarios lo lean y le contesten	PABLO
U-0011	USER	USER quiere poder ocultar del listado los eventos, lugares o descuentos que no le interesan y destacar los que sí le interesan para tener un listado personalizado	PABLO
U-0012	PROMOTER	PROMOTER quiere crear/editar/eliminar un evento público para promocionarlo	PABLO
U-0013	PROMOTER	PROMOTER quiere crear/editar/eliminar un perfil de su local, artista u organización para poder crear eventos relacionados y que la gente pueda suscribirse y dejar opiniones	PABLO

U-0014	PROMOTER	PROMOTER quiere añadir tags a sus eventos para que la gente pueda encontrarlos más fácilmente	SERGI
U-0015	USER	USER quiere ver un listado de los eventos que más le interesan para no tener que buscarlos él mismo	SERGI
U-0016	USER & ANONYMOUS	actores quieren poder ver la cartelera de los cines de su ciudad para consultar información de las películas y los horarios	SERGI
		Historias relacionadas con la red social	
U-0101	ANONYMOUS	ANONYMOUS quiere registrarse en el sistema para ejecutar historias de USER	PABLO
U-0102	ANONYMOUS	ANONYMOUS quiere logearse en el sistema para ejecutar historias de USER	PABLO
U-0103	USER	USER quiere deslogearse del sistema para que nadie pueda utilizar su cuenta	PABLO
U-0104	USER	USER quiere añadir/eliminar a otro usuario como contacto para ver su perfil y asistir con él a eventos	PABLO
U-0105	USER	USER quiere importar sus amigos de Facebook para tenerlos como contactos en la aplicación	PABLO
U-0106	USER	USER quiere crear/editar/borrar un evento entre amigos para organizar su cumpleaños	PABLO
U-0107	USER	USER quiere decidir el día, el sitio donde cenar y quien lleva el coche para ir al concierto xxx con sus amigos	PABLO
U-0108	USER	USER quiere ver a qué eventos próximos, opiniones o lugares de interés van a ir sus amigos para ver si alguno de esos eventos le interesa (muro)	SERGI
U-0109	USER	USER quiere ver el perfil de uno de sus contactos para saber a qué eventos va a ir, qué lugares ha comentado, que descuentos ha comprado… (perfil)	SERGI
U-0110	USER	USER quiere recibir notificaciones cuando ciertas acciones relacionadas con él sucedan para poder responder a esos sucesos	PABLO
U-0111	USER	USER quiere cambiar la configuración de la aplicación para adaptarla a sus gustos	SERGI
U-0112	USER	USER quiere cambiar el idioma de la aplicación para poder entender los menús y explicaciones	SERGI

U-0113	USER	USER quiere que cuando se produzca un error en el sistema se le notifique lo mejor posible y pueda volver a utilizar la aplicación inmediatamente para saber cuándo sus acciones no se han llevado a cabo satisfactoriamente	PABLO
U-0114	USER	USER quiere poder cambiar su asistencia en un evento para informar a sus amigos que va a asistir seguro, tal vez o que no va a asistir	PABLO
		Historias relacionadas con administración	
U-0201	STAFF	STAFF quiere ver un listado de eventos, lugares y descuentos de otras páginas de ocio para poder importarlos al sistema	SERGI
U-0202	STAFF	STAFF quiere ver un listado de los errores ocurridos en el sistema para poder corregirlos	PABLO
U-0203	STAFF	STAFF quiere poder borrar los comentarios y opiniones de cualquier usuario que considere no oportunas para mantener un buen contenido en la web	SERGI
U-0204	STAFF	STAFF quiere poder tener estadísticas de las acciones de los usuarios	SERGI
		Historias técnicas	
T-0001	---	Quiero un script de código ejecutable en Linux para poder actualizar el código del sistema de producción en un solo comando	SERGI

Ilustración 37 - Web Backlog

2.2.2. Móvil Backlog

En el caso del móvil lo que hicimos fue seleccionar un rango pequeño de funcionalidades de la aplicación web ya que su implementación es más laboriosa y además se ha de llevar a cabo por duplicado ya que para cada móvil es diferente.

Identificador	Actores	Descripción	Responsable
		Historias relacionadas con la guía de ocio	
U-1001	USER & ANONYMOUS	actores quieren ver cuáles son los próximos eventos para ver su información	**ADRIÀ**
U-1002	USER & ANONYMOUS	actores quieren ver cuáles son los próximos conciertos, obras de teatro, espectáculos deportivos, eventos culturales, etc. para ver su información	**ADRIÀ**
U-1003	USER & ANONYMOUS	actores quieren buscar un evento para salir esta noche, mañana, esta semana o este mes	**ADRIÀ**
U-1004	USER	USER quiere ver información detallada de un evento para saber puntos específicos como la descripción, el horario, los artistas y lugares que participan, etc., así como también que dicen otros USERs acerca de este	**ADRIÀ**
U-1005	USER	USER quiere dejar un comentario en un evento para que otros usuarios lo lean y le contesten	**ADRIÀ**
U-1006	USER	USER quiere valorar positiva o negativamente un evento para que otros actores sepan en general si gusta o no gusta el evento	**ADRIÀ**
U-1007	USER	USER quiere ver qué personas asisten al evento para identificar amigos	**ADRIÀ**
U-1008	USER	USER quiere ver a un artista o lugar para ver de sus nuevos conciertos u otra clase de eventos que se celebren en el lugar o con el artista	**ADRIÀ**
U-1009	USER	USER quiere valorar positiva o negativamente el artista o lugar para que otros actores sepan en general si gusta o no el artista o lugar	**ADRIÀ**
U-1010	USER	USER quiere añadir los eventos que le interesan a su calendario para no olvidarse de cuando se celebran	**ADRIÀ**
		Historias relacionadas con la red social	
U-1101	USER	USER se loguea en el sistema utilizando su email y contraseña para ejecutar historias de USER	**ADRIÀ**
U-1102	USER	USER se loguea en el sistema utilizando sus credenciales de facebook para ejecutar historias de USER	**ADRIÀ**

U-1103	USER	USER se desloguea del sistema y actúa como anónimo para que nadie pueda utilizar su cuenta	**ADRIÀ**
U-1104	USER	USER quiere recibir notificaciones cuando ciertas acciones relacionadas con él sucedan para poder responder a esos sucesos	**ADRIÀ**

Ilustración 38 - Móvil Backlog

2.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que, a diferencia de los requisitos funcionales, no especifican las funcionalidades del sistema sino que determinan cómo estas funcionalidades se llevan a cabo. Los requisitos no funcionales son difíciles de testear y por ello muchas veces se evalúan subjetivamente aunque no por eso son menos importantes que los requisitos funcionales. Dos proyectos con los mismos requisitos funcionales y el mismo alcance pueden dar lugar a dos productos completamente diferentes con diferentes requisitos no funcionales.

La intención de este proyecto es que una vez acabado el PFC podamos continuar con Eventlayer como proyecto empresarial. Por eso los requisitos no funcionales se centrarán en cómo debe ser el sistema para usarlo públicamente por muchos usuarios concurrentemente. Además, uno de los aspectos diferenciadores de Eventlayer debe ser la simplicidad y facilidad en su manejo como hemos comentado en el apartado de Producto.

A continuación citaremos los requisitos no funcionales del proyecto así como la manera como los hemos abordado.

I. Accesibilidad

Se refiere a la capacidad de acceso y uso de la aplicación web por todas las personas independientemente de sus discapacidades físicas, intelectuales o técnicas.

Para ello se tendrá especial cuidado en seguir los estándares webs y usar tecnologías en las que se basan. Concretamente deberá funcionar en navegadores sin JavaScript activado como son los navegadores para invidentes o como puede ser el propio motor de búsqueda de Google.

La aplicación deberá ser compatible con los principales navegadores web. Más específicamente deberá funcionar correctamente en Internet Explorer 8.0 o superior, Firefox 3.0 o superior, Google Chrome 2.0 o superior y Opera 9 o superior.

Cabe destacar que no es un requisito soportar versiones anteriores de Internet Explorer a la 8.0 ya que a pesar de ser muy usadas, éstas no siguen los estándares web y es muy difícil (sino imposible) emular toda la riqueza de la interfaz web de nuestra aplicación.

II. Fiabilidad

Relacionado con la disponibilidad, el sistema deberá funcionar el mayor tiempo posible sin errores. Para ello el sistema deberá seguir los siguientes cuatro puntos:

a) **Madurez:** es la capacidad del producto para evitar fallar como resultado de fallos de software. Para ello usaremos, siempre que podamos, tecnologías y componentes externos contrastados y usados en multitud de proyectos diferentes.

b) **Tolerancia a fallos:** es la capacidad del producto para mantener las máximas funcionalidades posibles en caso de fallos. Crearemos dos componentes software de gestión de errores que los interceptarán y minimizarán sus efectos. Además, estos componentes guardarán los errores en un registro (l*ogs*) para su posterior recuperación

y corrección tanto en el lado del cliente como en el servidor. Este punto se detallará en la memoria de Pablo Guevara en el apartado 5.*1 Gestión de errores*.

c) **Capacidad de recuperación:** capacidad del producto software para restablecer el nivel de funcionalidad y de recuperar los datos en caso de fallo. Se ejecutarán copias de seguridad regularmente que se guardarán en un servidor de apoyo. También contaremos de scripts para el servidor que facilitarán la puesta en marcha de todo el sistema. Estos pueden verse en el apartado *1 Mantenimiento del sistema*.

d) **Cumplimiento de la fiabilidad:** capacidad del producto software para adherirse a normas, convenciones o regulaciones relacionadas con la fiabilidad. En nuestro caso y al ser un desarrollo propio no nos planteamos solicitar ninguna norma relacionada con la fiabilidad.

III. Eficiencia

Para cumplir cada uno de los siguientes puntos contaremos con un servidor Intel Core2Duo con CPU 2x2.66+ GHz, 4 Gigabytes de memoria RAM y 6 Terabytes de disco duro. No nos planteamos ampliarlo.

a) **Comportamiento temporal:** es la capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas. Como norma general, tendremos en cuenta que durante una interrupción un usuario puede mantener la atención en una misma página web un tiempo de 10 segundos. El sistema deberá ser capaz de llevar a cabo sus funciones principales en un tiempo menor aunque habrá algunas excepciones como el envío de fotos.

   Para mejorar la velocidad de respuesta de la página web hemos hecho uso de Google Page Speed[17], una herramienta que Google ofrece de manera gratuita y que permite evaluar el rendimiento de una página web así como obtener sugerencias sobre cómo mejorarla. Además muestra un número según el rendimiento, siendo éste más alto cuanto más rápida sea la web.

   En nuestro caso, la puntuación obtenida es de 89/100 con solo una sugerencia importante: “servir imágenes escaladas”. Para mejorar este punto tendríamos que servir las imágenes desde el servidor en el mismo tamaño en el que se muestran en la interfaz de usuario para no malgastar ancho de banda, pero implementarlo implicaría reconvertir todos los avatares de eventos y usuarios cada vez que hubiese un cambio en el diseño de la interfaz.

   Las demás sugerencias sólo mejorarían la velocidad de la interfície en unos pocos milisegundos así que las ignoramos. A continuación vemos los resultados obtenidos en Google Page Speed:

[17] http://code.google.com/speed/page-speed/

Ilustración 39 - Resultados obtenidos con Google Page Speed sobre la velocidad de la web

b) **Utilización de recursos:** es la capacidad del software para usar las cantidades y tipos de recursos adecuados cuando lleva a cabo su función bajo condiciones determinadas.

c) **Cumplimiento de la eficiencia:** capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia. En nuestro caso, no nos adheriremos a ninguna norma.

IV. Usabilidad

La usabilidad es un aspecto relacionado con la interfaz del usuario y que hace referencia a la rapidez y facilidad con la que los usuarios llevan a cabo sus acciones. Se deben cumplir los siguientes cuatro puntos:

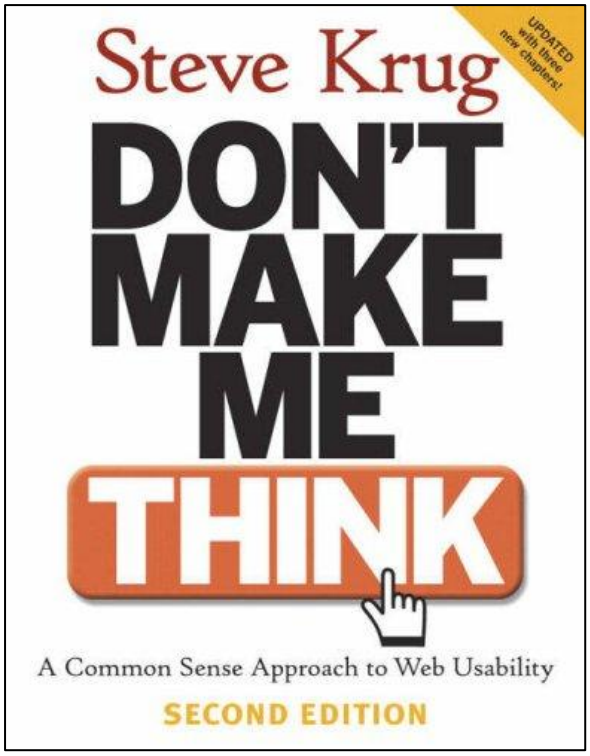


Ilustración 40 - El libro de Steve Krug "Don't make me think" ha servido como guía para tomar la mayoría de decisiones sobre usabilidad. Fuente: Amazon.com

a) **Una aproximación al usuario**: para desarrollar un producto usable se tiene que pensar en el usuario que lo va a utilizar, conocer sus necesidades y trabajar junto a ellos. Hemos pedido a voluntarios que lleven a cabo ciertas tareas básicas en nuestra aplicación web, concretamente ahora tenemos 500 usuarios registrados, y hemos analizado y corregido las dificultades encontradas para mejorar la aplicación.

b) **Amplio conocimiento del contexto de uso**: se debe minimizar el tiempo en que el usuario lleva a cabo sus tareas con los distintos elementos de la interfaz y maximizar el éxito que éste tiene en predecir la acción. Para ello hemos medido el tiempo en que los usuarios llevan a cabo las tareas con la aplicación web dando importancia a que no la hayan usado previamente para no condicionar el resultado de las pruebas.

c) **El producto ha de cumplir las necesidades del usuario:** los usuarios son gente diversa intentando llevar a cabo una tarea, se relacionará la usabilidad con la productividad y la calidad de su resultado.

d) **Fácil de usar por los usuarios**: son ellos y no los desarrolladores o diseñadores los que determinarán si el producto es fácil de usar.

V. Mantenibilidad

En este punto se agrupan los elementos que facilitan la labor de corrección y ampliación del producto. En nuestro caso le daremos una gran importancia, ya que a largo plazo puede ser necesario ampliar el equipo de desarrolladores y éstos deberán acoplarse con la mayor comodidad posible. Se deben cumplir las características siguientes:

a) **Capacidad para ser analizado:** es la capacidad del producto software para diagnosticarle los fallos que puedan surgir. Nuestra estrategia se basará en el uso de *logs* con los accesos web de los usuarios y errores web para detectar cualquier anomalía. También hemos hecho uso de herramientas como Zend Debugger para analizar el comportamiento del código PHP y de FireBug para analizar el código JavaScript. Los comentarios que vamos escribiendo en el código están en inglés.

b) **Capacidad para ser cambiado:** capacidad del producto software que permite que una determinada modificación sea implementada. En este caso, el desarrollo del producto lo haremos comentando aquellos puntos más conflictivos pensando que el código será revisado y modificado en un futuro. También nos aprovecharemos de herramientas como Zend Studio que facilitan las labores de desarrollo y mantenimiento.

c) **Estabilidad:** capacidad del producto software para evitar efectos inesperados debidos a modificaciones del software. La estabilidad del sistema se ha probado en la fase de desarrollo y será facilitada por el uso de componentes externos cuya implementación ya ha sido testada por multitud de proyectos.

d) **Capacidad para ser probado:** capacidad del producto software que permite que el software modificado sea validado. En nuestro sistema, se podrá probar el sistema de manera rápida gracias al uso de un entorno de desarrollo y de lenguajes de programación interpretados (PHP, JavaScript) que no requieren una compilación previa.

e) **Cumplimiento de la mantenibilidad:** capacidad del producto software para adherirse a normas o convenciones relacionadas con la mantenibilidad. En nuestro caso, no adheriremos nuestro sistema a ninguna norma.

VI. Seguridad

Debido a la naturaleza pública de las aplicaciones web, deberemos tener un especial cuidado con la seguridad de nuestro sistema para evitar ataques externos. Nos centraremos en tres puntos:

a) **Confidencialidad:** es la propiedad de prevenir la divulgación de datos confidenciales a personas o sistemas sin autorización. Llevaremos a cabo diversas acciones:

   - **Contraseñas cifradas:** las contraseñas de los usuarios se guardarán cifradas en la base de datos. De hecho ni siquiera los administradores de Eventlayer, aún con acceso a la base de datos pueden tener acceso a las contraseñas ya que están cifradas usando MD5 y un algoritmo de refuerzo para más seguridad.

   - **Medidas de seguridad en el servidor:** se guardará un registro de todos los accesos al sistema, ya sean válidos o fallidos y periódicamente se buscarán accesos de carácter maligno.

   - **Información anónima:** la información confidencial de un usuario determinado sólo podrá recuperarse usando la sesión del mismo usuario y siempre de manera anónima, es decir, ningún usuario podrá obtener datos para los que no ha sido autorizado expresamente.

   Por último se intentará minimizar el uso de contraseñas de usuarios genéricos en favor de contraseñas de usuarios específicos. Esta es una norma típica en cualquier auditoría de seguridad.

b) **Integridad:** Es la propiedad que busca mantener los datos libres de modificaciones no autorizadas. Se realizarán las siguientes acciones:

   - **Permisos:** El sistema comprobará los permisos del usuario antes de llevar a cabo cualquier acción. Para ello usaremos *ACLs* (listas de control de acceso) que indicarán qué puede modificar y ver un usuario determinado. Estos controles se llevarán a cabo en el servidor para evitar que un usuario malintencionado pueda modificarlos.

   - **Filtros sobre datos introducidos por el usuario:** Todos los datos que puede introducir el usuario son filtrados y validados para evitar información errónea o que pueda modificar el sistema de información de una manera maliciosa. Un ejemplo serían las sentencias SQL que se filtran para evitar ataques de tipo SQL Injection, o los campos de los formularios donde se evitan ataques de tipo CrossSite Scripting.

   - **Restricción de consultas en Base de Datos:** No se permitirán consultas que modifiquen la estructura de la misma desde la aplicación web, aunque se podrán ejecutar desde la interfaz de administración.

c) **Disponibilidad:** El sistema deberá estar disponible el mayor tiempo posible, salvando períodos de pocos segundos para llevar a cabo operaciones de mantenimiento de código y servidor y que son inevitables dados los recursos de que disponemos.

   Que el sistema tenga la máxima disponibilidad posible es importante porque los usuarios interactúan con el sistema y no dependen de que éste esté disponible o no para guardar

datos y realizar consultas. A pesar de las medidas tomadas, este punto estará sujeto a la disponibilidad del servicio de hosting donde alojamos la web, cuando este no esté disponible nuestra aplicación web tampoco lo estará.

VII. Escalabilidad

La escalabilidad es la propiedad de un sistema que indica su habilidad para extender el margen de operaciones sin perder calidad, es decir que el sistema funcione con una eficiencia aceptable independientemente del número de usuarios que estén utilizando la aplicación. En una aplicación web esta propiedad es importante porque no podemos saber, a priori, cuanta gente va a usar nuestro servicio. Además, debido a que nuestro sistema funciona como una red social, su crecimiento se estima que puede ser exponencial gracias a las recomendaciones entre amigos.

En nuestro caso prepararemos el sistema para un número de usuarios suficientemente grande como para que, en caso de llegar al límite, el sistema ya sea rentable y podamos financiar una mejora de las infraestructuras. Para ello haremos uso de bases de datos NoSql que facilitan la escalabilidad como se verá más adelante en *4.2.3 Elección de bases de datos*.

VIII. Idioma

El sistema deberá soportar multilenguaje. En el momento de su apertura al público deberá estar traducida al español, catalán e inglés. Para ello contamos con 3 subdominios (es.eventlayer.com, en.eventlayer.com, cat.eventlayer.com) que modifican el idioma de la interfaz web.

Las traducciones se han realizado mediante un software externo para facilitar las labores de traducción a gente sin conocimientos técnicos. Se usará software estándar, concretamente usaremos aquellos basados en la biblioteca de código abierto Gettext, como el software Poedit[18]:

[18] Se puede descargar gratuitamente en http://www.poedit.net/.

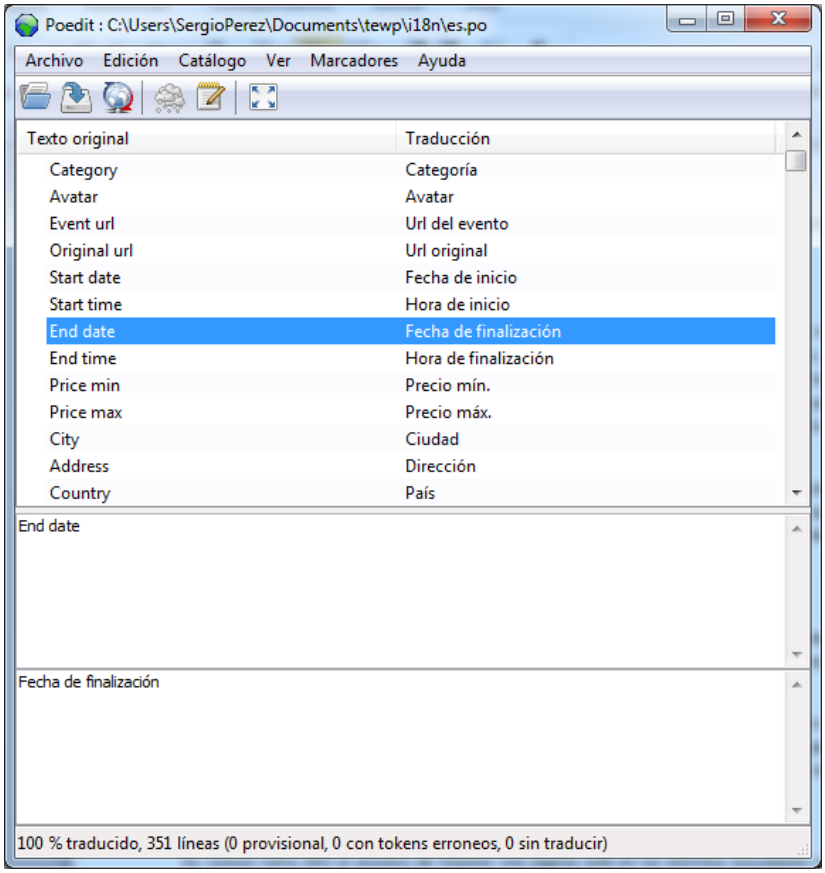


Ilustración 41 - Interfaz del software Poedit mediante el cual hemos traducido la aplicación web

IX. Requisitos SEO

SEO significa Search Engine Optimization, u optimización para motores de búsqueda. Con este críptico nombre se conoce el proceso de mejorar la posición de una página web en los distintos buscadores de manera orgánica, es decir, sin pagar dinero al buscador para acceder a una posición destacada. Así se consigue que el buscador muestre nuestra página en los primeros lugares cuando se hace una búsqueda que guarda relación con nuestros contenidos.

Podemos separar las propiedades que mejoran la posición de una web en los buscadores según si necesitan una planificación a largo plazo o no, de manera que nos limitaremos a implementar y mejorar aquellos puntos en que sea importante implementarlo cuanto antes. En un futuro, si el proyecto Eventlayer sale adelante como iniciativa empresarial ya se mejoraría este requisito.

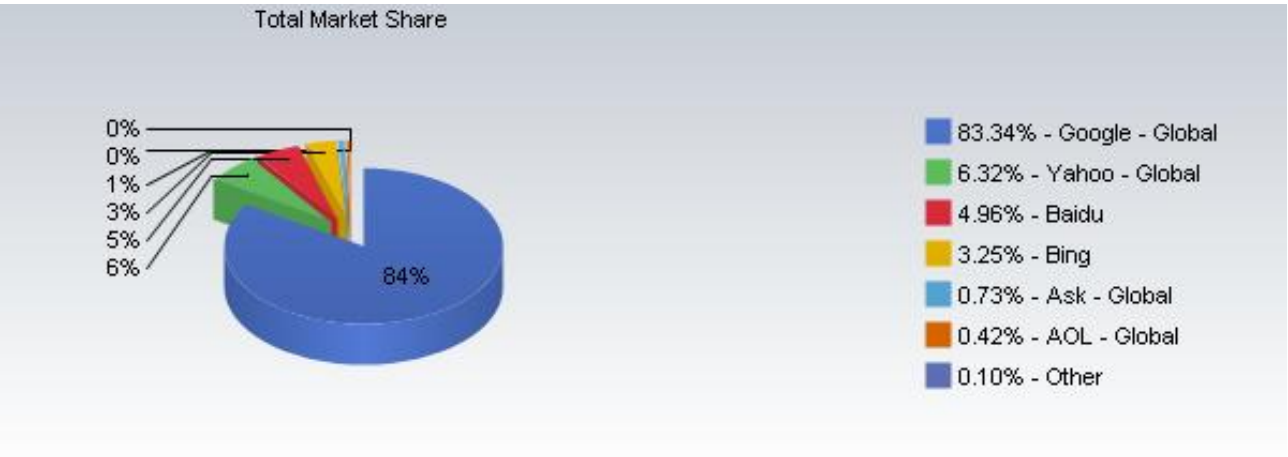


Ilustración 42 - Porcentaje de market share de buscadores a nivel mundial en 2010. Fuente: http://albertval.com/datos-de-interes/

En nuestro país el mercado de los buscadores está claramente dominado por Google[19] con un 98% de Market Share y nos centraremos en las directrices que mejorarán la posición en este buscador. Nos preocuparemos por los siguientes puntos:

- **Que toda la aplicación sea navegable sin ayuda de tecnología JavaScript** de manera que los robots que indexan las páginas web puedan añadir el contenido al buscador Google. Esto es debido a que el robot que usa Google para rastrear sitios de internet no es capaz de ejecutar el código JavaScript y si nuestros links dependen exclusivamente de esta tecnología no será capaz de interpretarlos. Nosotros usaremos esta tecnología pero en caso de que esté desactivada los enlaces continuarán funcionando gracias al código HTML por defecto.

- **Uso de títulos y meta correctos,** con información sobre el contenido de cada página y diferentes entre sí que facilitarán tanto su indexación como su identificación por parte de los usuarios que encuentren las páginas en Google.

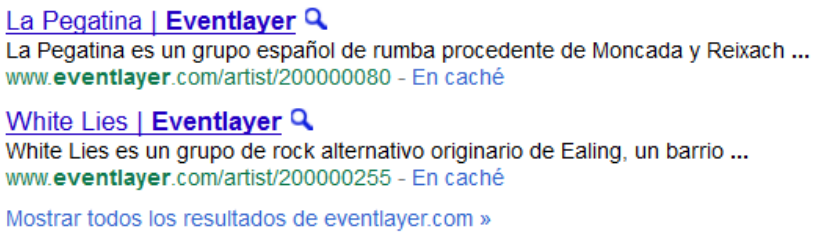


Ilustración 43 - Ejemplo de dos páginas de Eventlayer indexadas en Google con diferente título e información

- **Uso de un fichero robots.txt** para evitar que los buscadores indexen contenido no deseado que perjudicaría tanto la imagen de Eventlayer como la relevancia del contenido general de nuestra aplicación. Dejaremos de indexar páginas privadas, páginas de administración y páginas que sirven para mostrar errores al usuario.

[19] Fuente: http://albertval.com/datos-de-interes/

```
User-agent: *
Disallow: /funny
Disallow: /sampling
Disallow: /admin
Disallow: /home
Disallow: /media
Disallow: /auth
Disallow: /errors
Disallow: /*noindex*
```

Ilustración 44 - Contenido del fichero robots.txt que evita la indexación de contenido no relevante, se puede ver en http://www.eventlayer.com/robots.txt

Como hemos comentado dejaremos para una etapa posterior la implementación de técnicas más avanzadas y que hubiesen consumido mucho tiempo al proyecto como la obtención de enlaces hacia nuestra página o la elaboración de contenido propio con palabras clave.

3. Planificación y análisis económico

Tal como hemos comentado en el apartado de metodología hemos realizado el proyecto utilizando la metodología ágil Scrum. Por esta razón, las diferentes etapas del proyecto (Especificación, Aprendizaje, Diseño, Implementación, Documentación y Pruebas) se solapan e hicieron imposible valorar el tiempo dedicado a cada una de ellas por separado. Por eso la valoración la hemos hecho sobre las historias: una historia es una descripción en lenguaje natural de los requisitos y suele agrupar a varios de ellos.

A pesar de que no obtuvimos todas las historias al inicio del proyecto, es importante resaltar que en todo momento hemos tenido una lista de historias. Cada dos semanas aproximadamente (en cada sprint) decidíamos cuales de estas eran las más prioritarias y nos repartíamos la responsabilidad de llevarlas a cabo. Tener esta lista de historias nos ha servido para:

- Tener claro en qué estábamos trabajando en cada momento.
- Detectar problemas de desarrollo a tiempo y poder reorientar la fuerza de trabajo.
- Llevar un seguimiento global del proyecto y ver si era factible terminar en los plazos previstos.

Además, en la metodología Scrum es clave calcular el coste de llevar a cabo las diferentes historias ya que se priorizan segun su coste e importancia: aquellas que puedan llevarse a cabo con menos recursos tendrán más posibilidades de elegirse para el próximo Sprint.

3.1. Planificación

En un proyecto tan grande y complejo como el nuestro, es necesario llevar a cabo una planificación y gestión del tiempo. En nuestro caso gracias a reuniones semanales y reuniones diarias siguiendo la metodología Scrum, y que están detalladas en *1.4.4.Colaboración y coordinación* y gracias a un Product Backlog y Sprint Backlog que se detallan en el mismo punto.

3.2. Identificación de las etapas del proyecto

Como hemos dicho, no podemos separar las etapas del proyecto en las etapas que corresponderían a un desarrollo usando la metodología Scrum pero podemos agrupar 3 fases principales:

1. En una primera fase, se desarrolló el framework PHP y el framework Javascript y paralelamente se hizo un analisis de tecnologías para el posterior desarrollo de la aplicación web en sí. Se puede ver el desarrollo de estas partes en el PFC de Pablo Guevara, arquitectura y en el apartado *4.2.1 Elección de las tecnologías* de Sergi Pérez.

2. En una segunda fase, se desarrolló la aplicación Eventlayer.

3. En una tercera fase, que empezó a mediados de la segunda y paralela a esta, se desarrollaron las aplicaciones móviles. Estas se pueden encontrar en el PFC de Adrià Heredia.

3.3. Tareas y dedicación estimada

Scrum no tiene en cuenta diagramas de Gantt ni diagramas Pert, de manera que indentificamos los principales grupos de historias individualmente y para cada una comparamos la estimación a priori con el tiempo dedicado. Cada una de estas historias incluye la especificacion, aprendizaje, diseño, implementación, documentación de código y pruebas. Aunque estas historias se llevaron a cabo en tareas más de duración más corta, su número es demasiado elevado para valorarlas individualmente. Finalmente, hay una etapa individual que corresponde a la realización de este documento.

A continuación están las tareas principales. Consideramos necesario mostrar las tareas de los 3 PFCs ya que forman parte de un mismo proyecto global, Eventlayer; muchas tareas no se entienden individualmente sino como una parte de un proyecto más amplio que el propio PFC. Estas tareas no tenían un responsable fijo sino que su responsable se decidía en el Daily Scrum, la reunión diaria al inicio de la jornada laboral (ver apartado *1.4.4.Colaboración y coordinación)*. De todos modos, y para facilitar la comprensión del cálculo de los costes de personal de los diferentes PFC, las agruparemos según su responsable.

Tabla 1 - Tareas y dedicación de Sergi Pérez

Tarea	Duración estimada	Duración real	Responsable
Realización del modelo de datos	40	60	Sergi
Análisis de bases de datos	90	180	Sergi
Análisis de frameworks	90	90	Sergi
Análisis de motor de búsqueda	40	60	Sergi
Muro	120	120	Sergi
Motor de búsqueda	120	140	Sergi
Recomendador de eventos	80	80	Sergi
Mashup de webs y APIs	110	180	Sergi
Cartelera	60	60	Sergi
Vistas web	120	260	Sergi
Instalación del servidor	10	20	Sergi
Instalación y configuración de software del servidor	20	60	Sergi
Planificación	100	140	Sergi
Realización de diagramas de datos	20	20	Sergi
Realización de memoria	130	130	Sergi
Total	**1150**	**1600**	

Las diferentes variaciones de tiempo corresponden a:

- **Realización del modelo de datos:** esta tarea se refiere a modelar los datos del modelo de la aplicación y adaptarlos a las diferentes bases de datos del proyecto. Se estimó en 40horas, acabaron siendo algunas horas más debido al uso de bases de datos no relacionales.

- **Análisis de bases de datos**: esta tarea consistió en analizar las diferentes bases de datos del mercado y su elección. Se alargó debido al elevado número de opciones NoSql y que no habíamos contemplado por desconocimiento.

- **Análisis de frameworks:** esta tarea consistió en analizar los diferentes del mercado y su elección. El tiempo dedicado correspondió al estimado inicialmente.

- **Análisis del motor de búsqueda:** esta tarea consistió en analizar los diferentes motores de búsqueda. Se estimó en unas 40 horas y se alargó debido al análisis de los diferentes plugins de Solr, el motor utilizado.

- **Muro:** esta tarea se refiere a la implementación del componente muro y su duración real fue muy similar a la estimada inicialmente.

- **Motor de búsqueda:** esta tarea se refiere a la implementación del motor de búsqueda y su duración real fue muy similar a la estimada inicialmente.

- **Recomendador de eventos:** esta tarea se refiere a la implementación del recomendador de eventos y su duración real fue muy similar a la estimada inicialmente.

- **Mashup de webs y APIs**: esta tarea consistía en la implementación del Mashup. Su desarrollo se alargó debido a situaciones imprevistas, como webs que necesitaban el uso de cookies para ser parseadas o webs que baneaban nuestra ip cuando se hacían demasiadas peticiones en un lapso de tiempo determinado.

- **Cartelera:** se trataba de adaptar el componente mashup para la creación de una cartelera de cine, duró el tiempo previsto.

- **Vistas web**: esta tarea consiste en la implementación del interfaz de la web mediante HTML y CSS. Debido a un rediseño del interfaz, tuvimos que dedicar más horas de las previstas para implementarlo.

- **Instalación de servidor y instalación y configuración de software**: estas tareas consisten en la instalación del servidor de la web y el software necesario para llevar a cabo el proyecto. A mitad de proyecto tuvimos que cambiar el servidor, lo que implicó reinstalar y configurar el software del proyecto.

- **Planificación**: debido al rediseño de la interfaz, tuvimos que dedicar horas extras a planificar nuevos sprints y tareas para el proyecto. Este tiempo corresponde a los 10 minutos diarios de Daily Scrum y a la reunión bisemanal para decidir nuevas tareas que llevar a cabo.

- **Realización de diagramas de datos**: consiste en la realización de diagramas de soporte para la memoria del proyecto.

- **Realización de memoria:** consiste en la redacción de este documento, gracias a la documentación previa del código del proyecto se ha podido realizar en un tiempo parecido al previsto.

Aunque la desviación de tiempo es importante, aproximadamente el 45% de ellas son debidas al rediseño que llevamos a cabo una vez abierta la beta pública y muchas otras son debidas a mantenimiento del mashup y servidor. Además, al ser un proyecto con ambición empresarial, tampoco no consideramos que la desviación fuese a modificar el curso del proyecto.

Tabla 2 - Tareas y dedicación de Pablo Guevara

Tarea	Duración estimada	Duración real	Responsable
Gestión del proyecto y metodología	120	200	Pablo
Arquitectura del servidor	290	350	Pablo
Arquitectura del cliente	260	440	Pablo
Gestión de errores	120	140	Pablo
CRUD	90	110	Pablo
Gestión de usuarios	80	90	Pablo
Notificaciones	40	60	Pablo
Gestión de la asistencia a eventos	90	110	Pablo
Integración con Facebook	130	130	Pablo
Redacción de la memoria	130	160	Pablo
Total	**1350**	**1790**	

Las diferentes variaciones de tiempo corresponden a:

- **Gestión del proyecto y metodología**. Esta tarea se refiere a las horas empleadas en la gestión del proyecto como tal. El gestor de proyectos tiene una responsabilidad integradora entre muchas "fuerzas" como son la coordinación de las prioridades de otras tareas, comunicación entre los miembros del equipo, elección y control de la metodología… La desviación ha sido principalmente porque el resto de tareas han tardado más en realizarse y por lo tanto el proyecto se ha alargado requiriendo más horas para su gestión.

- **Arquitectura del servidor**. Esta tarea consistió en diseñar e implementar la arquitectura de nuestro sistema en la banda del servidor por medio de la integración de un framework externo (Zend Framework). Sabía que era un punto muy laborioso y requirió mucho tiempo de documentación y de pruebas. Es por eso que se extendió más de lo que esperaba.

- **Arquitectura del cliente**. Esta tarea consistió en diseñar e implementar una arquitectura para acceder a Eventlayer desde navegadores que siguieran los estándares web. Un principio se calculó un coste similar al de la arquitectura del servidor pero el hecho de que el framework que teníamos que integrar en cliente (Jquery) fuese mucho menos completo que el del servidor hizo que esta tarea se alargase considerablemente.

- **Gestión de errores**. La tarea consistió en la implementación de un sistema de errores global para la web de Eventlayer. Fue una tarea sin una desviación demasiado grande ya que no era un apartado excesivamente complicado.

- **CRUD**. Consistía en crear un sistema genérico para crear, leer, modificar y eliminar información del sistema. En inglés el acrónimo significa Create Read Update y Delete. Fue una tarea que gracias a tener una buena arquitectura desarrollada no supuso un gran problema.

- **Gestión de usuarios.** Se trataba del típico login, registro y gestión de contactos de cualquier red social. No hubo problemas en su diseño ni en la implementación.

- **Notificaciones**. Se trataba de crear un sistema para informar al usuario de diferentes sucesos que estaban relacionados con él (invitación a un evento, contestación de un comentario…). Fue una tarea que duró bastante más de lo que se pensaba porque se ha modificado varias veces la manera en la que se notificaba al usuario, no llegábamos a un consenso claro.

- **Gestión de la asistencia a eventos.** Se trataba de crear pequeñas aplicaciones que ayudaran al usuario a quedar con sus amigos más fácilmente para ir a un evento. Por ejemplo con el uso de votaciones para decidir donde se iba a ir a cenar. Se tardó más tiempo porque se fueron añadiendo nuevas funcionalidades a medida que se nos ocurrieron más ideas.

- **Integración con Facebook**. Esta tarea consistía en conseguir la mejor integración posible de Eventlayer con Facebook. La verdad que con la aparición de la nueva versión de la API de Facebook se aceleró bastante el proceso y se acabó en el tiempo estimado.

- **Redacción de la memoria**. La redacción de la memoria ha tenido una desviación considerable ya que intenté llegar a un nivel de corrección más alto de lo que me había propuesto en primera instancia. Gracias a esa implicación extra los capítulos de la introducción de los que me responsabilizaba han quedado mucho más claros y los de implementación bastante más completos de lo que me había propuesto en la estimación.

Tabla 3 - Tareas y dedicación de Adrià Heredia

Tarea	Duración estimada	Duración real
Estudio de mercado	30	35
Elección de las tecnologías	5	5
Modelo de la aplicación	2	2
Arquitectura	10	20
Android		
Base de datos	20	25
Configuration	2	2
Llamadas al servidor	15	15
Parser	25	35
Registry	1	1
Lista de eventos	50	85
Login / Logout	40	65
Perfil evento	100	150
Perfil artista / lugar	30	25
Perfil usuario	35	20
Alertas	65	70
iPhone		
NSUserDefaults	2	2
Configuration	2	2
Llamadas al servidor	30	30
Lista de eventos	70	95
Login / Logout	60	70
Perfil de un evento	130	140
Perfil de un artista / lugar	35	30
Perfil del usuario	40	30
Alertas	75	75
Realización de memoria	130	130
Total	**1004**	**1159**

- **Estudio de mercado**: como su nombre indica esta tarea se basa en realizar un exhaustivo análisis del mercado actual acerca de los smarthones y su previsión de cara al futuro. La desviación de tiempo se debe al constante bombardeo de datos sobre cómo evoluciona el mercado.

- **Elección de las tecnologías**: gracias al análisis del mercado (tarea anterior) pudimos realizar elección de las plataformas móviles sobre las que implementar aplicaciones nativas de forma cuidadosa.

- **Modelo de la aplicación**: esta tarea consistió en diseñar el modelo de las aplicaciones. El modelo de un sistema software consiste en representar los elementos del mundo real que intervendrán en nuestro problema (eventos, artistas, lugares, asistentes, etc.). Como los elementos que intervienen son los mismos para las dos aplicaciones tendremos un solo modelo.

- **Arquitectura**: la tarea consistió en diseñar e implementar la arquitectura de las dos aplicaciones. La arquitectura en los dos sistemas se basa en el patrón de 3 capas pero al ser plataformas con diferentes lenguajes de programación obviamente se implementaron de manera distinta en cada plataforma. La desviación de tiempo fue causada porque inicialmente optamos por una arquitectura mucho más complicada, consistente en el patrón de 3 capas con un controlador Front controller (ver PFC de Pablo, capítulo 5.1.1.1 MVC). Este tipo de arquitectura ralentizaba el tiempo de respuesta de las aplicaciones.

- **Implementación**: está formada por un conjunto de tareas que se dividen en tareas de Android y de iPhone. Cada tarea es un módulo de implementación. Las desviaciones de cada tarea son debidas a diferentes motivos:

    - **Android**: el emulador integrado en Eclipse aún no siempre funciona correctamente, aunque sí es cierto que en las últimas versiones del SDK ha mejorado. Por eso y porque se han realizado muchos cambios de requisitos en cada módulo a lo largo del tiempo se han perdido muchas horas de más.

    - **iPhone**: el principal problema es que personalmente desconocía el lenguaje de programación Objective-C, por lo que inicialmente se requirió muchas horas de documentación y de pruebas. Además, el compilador de iOS no ayuda mucho a detectar dónde está exactamente el problema en el código por lo que se han pasado más tiempo del debido en detectar y corregir errores. También como en Android se realizaron muchos cambios de requisitos desde la idea inicial. Por todos estos motivos se han dedicado más horas de las debidas.

3.4. Análisis económico

Una vez sabemos las horas dedicadas a los diferentes PFCs, podemos llevar a cabo su estudio económico. Este se divide en coste de personal y coste de material utilizado como hardware y software. No se tendrá en cuenta el coste de ocupación, debido a que hemos realizado toda la carga de trabajo desde casa.

3.4.1. Coste de personal

Aunque el proyecto ha sido realizado por una persona, consideramos los diferentes roles que son necesarios para realizar este PFC. Para la siguiente tabla consideramos una media de 240 días laborables por año, 8horas por día laborable y un coste de seguridad social a cargo de la empresa de un 33%:

Rol	Sueldo anual bruto	Coste/año	Coste/día	Coste/hora
Jefe de proyecto	42.000 €	55.860 €	232.75 €	29.09 €
Analista	30.000 €	39.900 €	166.25 €	20.78 €
Diseñador BD	27.000 €	35.910 €	149,62 €	18.70 €
Administrador de sistemas	24.000 €	31.920 €	133.00 €	16.63 €
Programador	21.000 €	27.930 €	116.37 €	14.55 €

Para el jefe de proyecto se estimará un 5% de horas del total.

Para el PFC de Sergi Pérez, el tiempo total de analista corresponde a la suma de las tareas Planificación, Análisis de bases de datos, Análisis de Frameworks y Análisis de motor de búsqueda así como el 20% de Muro, Motor de búsqueda, Recomendador de eventos, Vistas, Mashup y cartelera y el 40% de realización de la memoria. El diseñador de BD es el encargado de la realización del modelo de datos y sus diagramas. El administrador de sistemas se encarga de la instalación del servidor y del software del servidor. Por último, el programador se encarga del 80% de Muro, Motor de búsqueda, Recomendador de eventos, Vistas, Mashup y cartelera y el 60% de realización de la memoria.

Los costes de personal están calculados en la siguiente tabla:

Tabla 4 - Tabla de coste de personal para el PFC de Sergi Pérez

Rol	Horas	Coste/hora	Total
Jefe de proyecto	57.5	29.09 €	1672.68
Analista	494	20.78 €	10265.32
Diseñador BD	60	18.70 €	1122
Administrador de sistemas	30	16.63 €	498.9
Programador	566	14.55 €	8235.3
Total	**1207.5**		**21794.2**

Por tanto, el coste de personal para realizar el PFC de Sergi Pérez es de 21794.2 €.

Para el PFC de Pablo Guevara, el tiempo total de analista corresponde a las tareas de Gestión del proyecto y metodología, el 40% de la realización de la memoria y al 30% del resto de tareas. El programador se encarga del 60% de la realización de la memoria y el 70% del resto de tareas.

Los costes de personal están calculados en la siguiente tabla:

Tabla 5 - Tabla de coste de personal para el PFC de Pablo Guevara

Rol	Horas	Coste/hora	Total
Jefe de proyecto	67.5	29.09 €	1963.58
Analista	418	20.78 €	8686.04
Programador	932	14.55 €	13560.6
Total	**1417.5**		**24210.22**

Por tanto, el coste de personal para realizar el PFC de Pablo Guevara es de 24210.22 €.

Para el PFC de Adrià Heredia, el tiempo total del analista corresponde a las tareas de Estudio de mercado y elección de tecnologías, así como un 40% de la realización de la memoria y al 30% del resto de tareas. El programador se encarga del 60% de la realización de la memoria y el 70% del resto de tareas.

Los costes de personal están calculados en la siguiente tabla:

Tabla 6 - Tabla de coste de personal para el PFC de Adrià heredia

Rol	Horas	Coste/hora	Total
Jefe de proyecto	50.2	29.09 €	1460.32
Analista	338.7	20.78 €	7038.19
Programador	665.3	14.55 €	9680.12
Total	**1054.2**		**18178.63**

Por tanto, el coste de personal para realizar el PFC de Adrià Heredia es de 18178.63 €.

Una vez calculados los tres costes de personal por separado, obtenemos **un coste de personal total de 64183 €.**

3.4.2. Coste de hardware

Durante el proyecto hemos contratado un servidor dedicado[20] que nos ha permitido instalar software con total libertad, concretamente lo hemos alquilado a partir de septiembre 2010. Su coste[21] es de:

- Instalación Intel Core i5-2400 4x3.1+ GHz: 49.99 €
- Alquiler: 106.19 €/mes, por tanto 1062 €.

Además, se ha contado con dos ordenadores portátiles y uno de sobremesa. Como la vida media del equipo informático es de 3 años, se amortizará sólo el porcentaje correspondiente a los 10meses de desarrollo:

- Sony Vaio Vgn-fz31M: 1072.26 € *10/36= 297.85 €
- Apple MacBook Pro[22]: 2689.99 € *10/36= 747.22 €
- HP Pavilion Elite HPE-540es PC Sobremesa[23]: 1099€ *10/36 = 305.28 €.

[20] Se entiende por servidor dedicado cuando se contrata un servidor propio que no se comparte con otros usuarios.
[21] http://www.ovh.es/productos/superplan_storage.xml
[22] http://www.fnac.es/Apple-MacBook-Pro-15-2-8-GHz-8-GB-RAM-Ordenador-portatil-Mac-Portatil/a419846?PID=1384&Mn=-1&Ra=-5000&To=0&Nu=5&Fr=2

El coste total del hardware es de 2462.34 €, o 820.78€ por cada PFC.

3.4.3. Coste de software

Para realizar el proyecto cada integrante del proyecto ha usado el siguiente software:

- Microsoft Windows 7 Ultimate: 299.99 €[24]
- Microsoft office 2010[25]: 139.00 €
- Zend Studio 8.0[26]: 300€

Igual que sucede con el coste del hardware, este coste tiene un periodo de amortización de 36 meses. El coste total es 299.99+300+139 * 10/36 = 638.60 € por cada PFC, o 1915.8 € en total.

3.4.4. Coste total

Finalmente, calculamos el coste total del proyecto en la siguiente tabla:

Concepto	Coste
Coste de personal	**64183.0** €
Coste de hardware	**2462.3** €
Coste de software	**1915.8** €
	68561.1 €

Si queremos valorarlo individualmente, debemos tener en cuenta que los costes de personal eran diferentes. Por tanto los costes de cada PFC serían los siguientes:

PFC	Coste total
Coste del PFC de Sergi Pérez	**23253.5** €
Coste del PFC de Pablo Guevara	**25669.6** €
Coste del PFC de Adrià Heredia	**19638.0** €

3.5. Proyecciones financieras del Business Plan

A continuación se incluye una proyección financiera de Eventlayer.com a partir de su apertura como empresa en Septiembre del 2011. Para realizar la proyección financiera tendremos en cuenta los siguientes indicadores:

[23] http://www.escapistmagazine.com/videos/view/zero-punctuation/3581-Duke-Nukem-Forever-for-real-this-time?2
[24] http://emea.microsoftstore.com/es/es-ES/Microsoft/Windows-7-Ultimate-Actualizacion?WT.mc_id=MicrosoftComES_WINDOWS_SHOP_WIN7
[25] http://emea.microsoftstore.com/es/es-ES/Microsoft/Office
[26] http://shop.zend.com/eu/zend-studio-for-eclipse.html?src=greybox

- **Patrocinio web**: El patrocinio de la web completa se llevará a cabo en campañas específicas. Teniendo en cuenta que los visitantes de nuestra web serán todos de la misma región (Barcelona), se espera que los patrocinadores tengan alguna relación con la ciudad y prácticamente el 100% de los usuarios estén interesados o se fijen en la publicidad. Esperamos obtener unos 80 euros por cada mil visitas (no por 1000 impresiones sino por 1000 usuarios únicos).

- **Anuncios por usuario**: Contaremos que cada usuario que visite la web verá un total de 2 anuncios por sesión. Este es un número muy conservador, ya que si el usuario navega a través de las diferentes secciones de la web irá recibiendo varios anuncios.

- **Coste de anuncio:** Los anuncios tendrán diferente coste según si se muestran a usuarios registrados o no registrados. Para los usuarios no registrados, seleccionaremos los anuncios dentro de la categoría por donde navega el usuario (teatro, deportes, música, etc.) y por su localización, y esperamos obtener unos 10 euros por cada mil visitas. Para los usuarios registrados, también segmentaremos por la edad y preferencias y esperamos obtener un retorno de hasta 40 euros por cada mil impresiones (*CPM*).

- **Newsletter:** Los usuarios registrados recibirán un email semanal con eventos que se ajusten a sus gustos, así como algunos eventos patrocinados. Se calcula que cada newsletter contendrá una media de 3 anuncios, con un coste de 40 euros por cada mil usuarios que lo reciban.

	Septiembre 2011	Octubre	Noviembre	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio	Agosto	Septiembre
Ingresos												
Usuarios web	4000	10000	15000	20000	35000	65000	85000	105000	130000	150000	220000	315000
Usuarios web registrados	1000	2500	3750	5000	8750	16250	21250	26250	32500	37500	55000	78750
Usuarios web NO registrados	3000	7500	11250	15000	26250	48750	63750	78750	97500	112500	165000	236250
Usuarios newsletter	1000	2500	3750	5000	8750	16250	21250	26250	32500	37500	55000	78750
Patrocinio web	0	0	1200	0	0	5200	0	0	0	0	17600	0
Publicidad	140	350	525	700	1225	2275	2975	3675	4550	5250	7700	11025
Publicidad usuarios registrados	80	200	300	400	700	1300	1700	2100	2600	3000	4400	6300
Publicidad usuarios NO registrados	60	150	225	300	525	975	1275	1575	1950	2250	3300	4725
Publicidad newsletter	480	1200	1800	2400	4200	7800	10200	12600	15600	18000	26400	37800
Ingresos Totales	620	1550	3525	3100	5425	15275	13175	16275	20150	23250	51700	48825
Gastos												
Servidor	120	120	120	120	120	120	120	120	120	120	240	240
Dietas	800	800	800	1000	1000	1000	1000	1000	1000	1000	1200	1200
Alquiler oficinas	0	1200	1200	1200	1200	1200	1200	1200	1200	2000	2000	2000
Material oficinas	0	2000	200	200	200	200	200	200	200	200	200	200
Sueldos equipo programación (4pers.)	2000	2000	2000	2000	2000	2000	2000	2000	6000	6000	6000	6000
Sueldo equipo marketing	2000	2000	2000	2000	4000	4000	4000	4000	4000	4000	4000	4000
Publicidad Facebook	150	150	150	150	500	500	500	500	2000	2000	2000	2000
Publicidad Adsense	400	400	400	400	800	800	800	800	800	800	800	800
Publicidad Spotify	0	0	0	0	2200	0	0	2200	0	0	2200	0
Registro marca, dominios	500	0	0	0	0	0	0	0	0	0	0	0
Flyers, posters	2000	2000	2000	2000	2000	1000	1000	1000	1000	1000	1000	1000
Creació empresa	4000	0	0	0	0	0	0	0	0	0	0	0
Gestor	400	400	400	400	400	400	400	400	400	400	400	400
Gastos Totales	12370	11070	9270	9470	14420	11220	11220	13420	16720	17520	20040	17840
Balance	**-11750**	**-9520**	**-5745**	**-6370**	**-8995**	**4055**	**1955**	**2855**	**3430**	**5730**	**31660**	**30985**

4. Diseño del sistema

La fase de diseño es una de las más importantes en el proceso de construcción de software. Consiste en a partir de la especificación realizar una serie de decisiones que nos den las pautas para la construcción de un sistema que cumpla los requisitos de la especificación de la manera más óptima.

Esto se consigue básicamente actuando en 2 frentes:

1. Escoger las tecnologías existentes que mejor se adecúen a los requisitos de la especificación.

2. Tomar las mejores decisiones en el diseño del código de software.

El punto 1 se desarrolla en el capítulo de elección de tecnologías y el punto 2 se ha llevado a cabo utilizando el uso de los patrones de software, en el capítulo *4.1 Patrones de software* se ha añadido un capítulo donde se habla de su uso.

Hemos dividido el diseño en diseño del sistema web y diseño de las aplicaciones móviles. Por lo tanto el *capítulo 4.2.* variará según la memoria, en la de Pablo y Sergi será *4.2 Diseño del sistema web* y en la de Adrià será *4.2. Diseño del sistema móvil*.

Por último advertir que debido al uso de Scrum (ver capítulo *2.4. Metodología utilizada*) y su modelo iterativo el diseño del sistema no se ha llevado a cabo todo en a la vez sino por partes durante el proceso de implementación. Por lo tanto la fase de diseño queda repartida también en el capítulo *6. Implementación del sistema*.

4.1. Patrones de software

Los patrones de software son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software.

Un patrón de software es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Los patrones de software pueden ser de 2 tipos:

- **Patrones de diseño.** Ofrecen soluciones para problemas puntuales en el diseño del código. Ejemplos de este tipo de patrones serían el patrón Singleton, el patrón Prototype, el patrón Amistad…

- **Patrones arquitectónicos.** Ofrecen soluciones para problemas que afectan al conjunto global del código, lo que se denomina como arquitectura. Ejemplos de este tipo de patrones serían el patrón MVC, el patrón de las 3 capas…

Los patrones que utilizaremos nosotros son aquellos que atienden al paradigma de orientación a objetos[27]. Esto hace que la principal tarea de los patrones sea definir qué responsabilidad tiene cada clase[28], es decir, que parte de código debe ir en cada clase. Esto para alguien que no está familiarizado con el desarrollo de software no tiene demasiada importancia, pero podría decirse que es el factor fundamental, la división de responsabilidades entre clases, para un software que cumpla con unos buenos niveles de "modularidad" y "cambiabilidad " que son dos características básicas para que un proyecto de construcción y mantenimiento de software sea óptimo. Todo esto está relacionado con "bajo acoplamiento" y "alta cohesión" que son dos conceptos que se pueden buscar en cualquier tipo de literatura de desarrollo de software por si el lector deseara ampliar conocimientos al respecto.

Durante los capítulos 4 *Diseño del sistema* y 5 *Implementación del sistema* se hará referencia mucho al concepto de patrón, eso es así ya que es considerada una muy buena práctica diseñar nuestro sistema en base a estos patrones.

Por último un ejemplo de la documentación, suelen seguir este estándar, de un patrón de diseño tal y como la consultamos nosotros cada vez que tomamos una decisión de diseño en nuestro sistema, este en concreto, patrón Decorator, lo usamos en nuestros formularios:

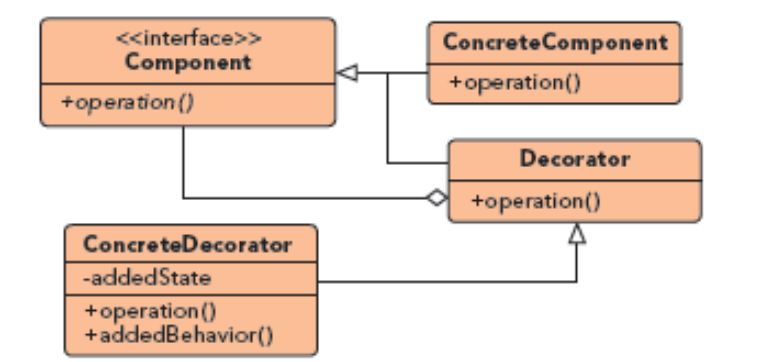


Ilustración 45 - Modelo UML del patrón Decorator

- **Name**: DECORATOR Class And Object Structural
- **Purpose:** Allows for the dynamic wrapping of objects in order to modify their existing responsibilities and behaviors.
- **Use When:**
  - Object responsibilities and behaviors should be dynamically modifiable.

[27] Es una forma de programar en la que se agrupa el código fuente en unidades llamadas clases. Estas clases emulan a los objetos reales que representan, en nuestro caso serían eventos, artistas o descuentos…
[28] Cada una de las unidades en las que se divide el código en el paradigma de orientación a objetos.

- Concrete implementations should be decoupled from responsibilities and behaviors.
- Subclassing to achieve modification is impractical or impossible.
- Specific functionality should not reside high in the object hierarchy.
- A lot of little objects surrounding a concrete implementation is acceptable.

4.2. Diseño del sistema web

4.2.1. Elección de las tecnologías

La realización de un proyecto como Eventlayer hubiese sido imposible sin el uso de tecnologías externas. El hecho de contar con recursos limitados y un equipo reducido nos obligó a usar tecnologías ya implementadas y contrastadas para acelerar la implementación del proyecto.

El uso de éstas siempre ha sido precedido por fases de pruebas y largas lecturas de sus respectivas documentaciones pero que a la larga ha sido compensado, no sólo ahorrándonos su implementación sino ahorrándonos las fases de diseño, pruebas y la propia fase de documentación del código. Además, contamos con el hecho de saber que el código usado ya había sido probado en multitud de otros proyectos.

Las tecnologías que usamos se pueden dividir en dos grupos. Por un lado, tecnologías que forman la base del proyecto o frameworks. Un **framework** consiste en un conjunto de librerías ya implementadas a partir del que se construye otro software. En este caso, hemos usado un framework para cada uno de los lenguajes de programación principales que usamos en el proyecto web: uno para el lenguaje PHP y otro para el lenguaje JavaScript. Respecto a la documentación, un aspecto interesante es que la mayoría de frameworks tienen un especial cuidado en este apartado lo que facilitó enormemente su uso. Por otro lado, también hemos usado varias bases de datos con varias funciones diferenciadas que nos permitirán escalar todo el sistema sin esfuerzo.

En el siguiente apartado *4.2.2 Elección de frameworks* citaremos, para cada software externo usado, qué criterios de selección evaluaremos para su elección y evaluaremos las diferentes alternativas que tuvimos en cuenta. En cualquier caso, hay una serie de requisitos comunes que citaremos a continuación:

- **Documentación**: Las tecnologías elegidas deberán tener la documentación necesaria para favorecer la comprensión y uso del mismo, de manera que el desarrollo de código basado en él sea sencillo y amigable.

- **Comunidad**: Se dará prioridad a aquellas tecnologías con una comunidad mayor, es decir con el mayor número de proyectos haciendo uso de él. Eso garantizará, en primer lugar, que la mayoría de funciones ya han sido probadas y la mayoría de errores que pueda haber han sido corregidos. En segundo lugar, el hecho de que su uso sea extensivo facilitará la búsqueda de documentación y ejemplos extra oficiales en blogs y foros de internet.

Hay que recalcar que el estudio de las tecnologías empezó junto al proyecto y a medida que necesitábamos un nuevo software de apoyo se hacía una valoración de todas las alternativas existentes y la última información disponible. En cualquier caso, y debido a la duración del proyecto, si hoy hiciésemos esta misma comparación se podría considerar incompleta ya que algunas de las alternativas que hoy tendríamos en cuenta ni siquiera existían.

4.2.2. Elección de frameworks

Un framework no es más que un conjunto estructurado de software y librerías de software que tiene la función de servir de base para otro proyecto. En nuestro caso hemos usado uno para cada uno de los lenguajes principales en los que está implementado el proyecto.

4.2.2.1. Elección del framework PHP

En este caso, el framework elegido servirá de base para desarrollar toda la parte del servidor que implementamos en lenguaje PHP. Hemos elegido este lenguaje porque, a pesar de haber alternativas como el lenguaje Ruby o java, es un lenguaje de programación rápido y cómodo que el equipo ya conocía.

Nos basaremos en los siguientes criterios de selección:

- **Patrón Modelo-Vista-Controlador:** Nos interesa encontrar un framework que soporte este patrón. La ventaja principal es poder separar las vistas del controlador, es decir, separar la parte lógica de la aplicación de la parte que ve el usuario con la mejora en mantenibilidad de código que esto conlleva.

- **Object Relational Mapping:** Más conocido por sus siglas ORM, es una técnica que convierte datos del sistema en PHP en datos relacionales para poder guardarlos en una base de datos.

- **Entorno de desarrollo integrado (IDE):** Se conoce como entorno de desarrollo integrado un conjunto de herramientas que permiten trabajar con el código de manera fácil y automatizar las tareas más comunes. Se priorizará aquél framework que cuente con un entorno de desarrollo adaptado a él.

- **Modularidad:** Es la capacidad de un sistema de ser descompuesto en varias partes llamadas módulos, realizando cada una de ellas una tarea específica para conseguir un objetivo común. Esto nos permitirá construir un sistema mantenible y extensible.

- **Roadmap:** Un roadmap no es más que una hoja de ruta donde se indica mediante fechas y objetivos el desarrollo que seguirá el software. Buscaremos un framework con una estrategia de desarrollo clara a largo plazo que nos permita confiar nuestro sistema en su desarrollo.

Consideraremos las siguientes alternativas:

- **Cake PHP:** Framework PHP basado en los principios del framework Ruby On Rails escrito en lenguaje Ruby. Cuenta con una gran comunidad de desarrolladores y colaboradores.

- **Code Igniter:** Está desarrollado por la empresa EllisLab, su característica principal es su rapidez de ejecución que consigue con una estructura de ficheros muy ligera.

- **Zend Framework:** Escrito por los desarrolladores del mismo lenguaje PHP, Zend Framework cuenta con una estructura de ficheros poco acoplada que permite hacer uso de partes independientes de su código.

- **Symfony:** es un framework PHP patrocinado por Sensio Labs, una compañía francesa que provee consultoría y otros servicios sobre tecnologías Open Source. Se puede extender fácilmente gracias al uso de plugins existentes o propios.

	Documentación	Comunidad	MVC	ORM	IDE	Modularidad	Roadmap
Cake PHP	Sí, algo pobre	Sí	Sí	Sí	No	No	No
Code Igniter	Sí, muy extensa y detallada	Sí	No	No	No	No	No
Zend Framework	Sí, muy extensa y detallada	Sí	Sí	No	Sí	Sí	Sí
Symfony	Sí, algo pobre e incompleta	Sí	Sí	Sí	No	Sí	Sí

Code Igniter es el framework que sale más mal parado de la comparación. Aunque no era una alternativa a tener en cuenta cuando realizamos la comparación, con el tiempo se creó otro framework llamado Kohana a partir de éste y que intentaría solucionar la mayoría de problemas. Cake PHP, a pesar de contar con características interesantes, también fue descartado porque consideramos que no ofrecía la flexibilidad necesaria para adaptarlo a nuestras necesidades.

Finalmente, entre Symfony y Zend Framework nos decidimos por el último. Además de contar con un entorno de desarrollo integrado muy potente, nos ofrece una gran colección de extensiones por parte de empresas más o menos conocidas que permiten añadir multitud de funcionalidades al framework en

sí. Además, cuenta con un roadmap muy detallado que podíamos seguir en todo momento para saber qué versión iba a salir, cuando y con qué mejoras y nuevas características.

Con el tiempo hemos podido comprobar que fue una gran elección y seguramente de todas las alternativas que existían hoy es la que cuenta con un futuro más estable y prometedor, aunque también han surgido otras alternativas últimamente muy válidas como el framework Yii.

Ilustración 46 - Logo de Zend Framework, el framework elegido para el lenguaje PHP

4.2.2.2. Elección del framework JavaScript

Nos basaremos en los siguientes criterios de selección:

- **Eventos:** Una característica del lenguaje JavaScript es que permite asociar eventos que tienen lugar en los diferentes elementos de la interfaz y llevar a cabo acciones cuando estos se ejecutan. Nuestro framework JavaScript debe proveer una interfaz sencilla para definir nuestros propios eventos y acciones.

- **Velocidad**: Buscaremos un framework rápido que no ralentice la interacción del usuario con la interfaz de la aplicación.

- **Modularidad:** Igual que en el caso de del framework para PHP, buscaremos un framework modulable para mejorar la mantenibilidad y extensibilidad.

- **Ajax:** Una característica clave en nuestro sistema será el uso de llamadas AJAX para evitar recargar la aplicación web completa. Esto consiste en, gracias a la tecnología JavaScript, hacer una llamada al servidor que no nos devolverá la página completa sino un contenido parcial con el que actualizaremos la página actual. De esta manera mantenemos la información y estado con el que trabaja el usuario (al no recargar la página completa, seguimos trabajando con el misma interfaz web) y mejoramos la eficiencia de las interacciones con el sistema al no tener que recibir ni procesar los datos completos. El framework JavaScript debe proveer un interfaz simple que permita llevar a cabo este tipo de llamadas.

- **Componentes visuales:** Se valorará positivamente la existencia de widgets o componentes visuales que se usarán para mejorar la experiencia del usuario.

Consideraremos las siguientes alternativas, entre las que hemos dejado fuera al conocido framework ExtJs por tener una licencia de pago:

- **Jquery:** Usada por gran variedad de empresas como Microsoft y Nokia, se promocionan como una de las más fáciles de usar.

- **MooTools:** Un framework javascript que destaca por estar orientado a objetos, con sistema de creación de clases y herencia compleja.

- **Scriptaculous:** Este framework está basado en otro framework javascript, Prototype, y añade multitud de efectos visuales que no proporciona el framework original.

- **YUI:** Es la abreviación de Yahoo User Interface, no es un simple framework JavaScript sino que contiene ficheros CSS y otras utilidades para la interfaz web. Está desarrollado y avalado por la empresa Yahoo.

	Documentación	Comunidad	Eventos	Velocidad	Modularidad	Ajax	Comp. Visuales
JQuery	Sí, muy completa y con ejemplos	Muy alta	Sí	Alta	Sí	Sí	Sí, con plugins
MooTools	Sí	Alta	Sí	Alta	Sí	Sí	Insuficientes
Scriptaculous	Sí	Alta	Sí	Lenta	Si	Sí	Sí
YUI	Sí, muy detallada	Alta	Sí	Lenta	Sí	Sí	Sí, muchos y muy detallados

Como podemos ver, todos los frameworks para el lenguaje JavaScript ofrecen unas características y funcionales parecidas. Lo más importante es que las funcionalidades básicas están cubiertas en cada uno de ellos y esto nos da la seguridad que, aunque hagamos una mala elección, las consecuencias no serán tan nefastas como en las elecciones de las otras tecnologías.

En este caso valoramos que tanto MooTools y Scriptaculous parecían tener un desarrollo lento y un futuro incierto. El framework YUI es el que cuenta con un mayor número de componentes visuales oficiales, mientras que la comunidad de usuarios de JQuery hace posible encontrar centenares de plugins con cualquier funcionalidad imaginable.

Finalmente nos decidimos por JQuery porque tiene una sintaxis mucho más simple y limpia. Por ejemplo, la siguiente línea de código que hace uso del framework YUI:

```
var a = YAHOO.util.Selector.query('div p');
```

Puede ser escrita haciendo uso de JQuery con:

```
var a = $('div p');
```

Ilustración 47 – Logo de jQuery, el framework elegido para el lenguaje JavaScript

4.2.3. Elección de bases de datos

Para guardar los datos de la aplicación usaremos una base de datos estándar MySql. En este punto no ha habido mucha duda, ya que es una base de datos libre y gratuita y ofrece las funcionalidades básicas necesarias. El problema surge a la hora de elegir las bases de datos que complementarán a ésta, ya que hay multitud de elecciones. Uno de ellos es que el sistema que estamos construyendo debe ser fácilmente escalable y soportar un gran número de visitas, por ello usaremos bases de datos del tipo NoSql.

NoSql es un término usado para agrupar una serie de bases de datos que no proporcionan garantías ACID (atomicidad, consistencia, aislamineto y durabilidad). Sus funcionalidades son, en general, más reducidas que las bases de datos relacionales pero ofrecen ventajas difíciles de conseguir con ellas: eficiencia, capacidad para tratar con facilidad grandes volúmenes de datos y una escalabilidad horizontal sencilla. En los últimos años, con la llegada de la web 2.0 las grandes aplicaciones web como Facebook.com y Twitter.com han apostado por su desarrollo de manera que hoy en día podemos adquirir y usar estas herramientas a coste cero.

4.2.3.1. Base de datos para guardar estadísticas

Una característica importante de nuestra aplicación web reside en que analizando el registro de las acciones del usuario podemos darle un servicio personalizado y mejorar su experiencia. El sistema recogerá datos de las interacciones del usuario con los eventos, artistas y locales para ofrecerle una guía

de ocio personalizada. La información guardada también se usará para definir rankings y obtener todo tipo de información que puedan necesitar los diferentes módulos del sistema.

Para implementar este sistema de datos tendremos en cuenta los siguientes requisitos:

- **Cambiabilidad**: A priori no podemos saber todas las estadísticas futuras que necesitaremos guardar, así que priorizaremos aquellas bases de datos que necesiten pocas modificaciones para añadir un nuevo tipo de datos.

- **Map-Reduce**: Necesitamos una base de datos capaz de ejecutar operaciones estadísticas sobre todos sus datos de manera eficiente ya que la cantidad de datos que guardaremos será elevada, esto es lo que nos ofrece Map-Reduce. Map-Reduce es un modelo de programación introducido por google que permite trabajar con grandes volúmenes de datos.

- **Facilidad de uso:** Como hemos visto el proyecto Eventlayer es muy amplio y abarca una gran cantidad de tecnologías. Por eso priorizaremos aquél software con una curva de aprendizaje menos elevada y con el que veamos que podemos conseguir lo que nos proponemos.

MySql no cumple ni con el requisito de la cambiabilidad ni con el de soportar el algoritmo Map Reduce. Una solución para implementar este sistema en MySql pasaría por agrupar todos los valores de cada acción registrada en un atributo y guárdalo en la base de datos pero de esta manera no podríamos trabajar con estos valores en el mismo motor de almacenamiento. En cuanto al segundo punto, no hay manera posible de salvarlo.

Por eso consideraremos las siguientes alternativas:

- **MongoDb:** Base de datos de código abierto que guarda los datos en formato JSON y puede ejecutar funciones MapReduce escritas en Javascript y consultas complejas comparables a las sentencias SQL relacionales. JSON es un formato para intercambiar datos en lenguaje javascript y que poco a poco está sustituyendo a XML[29].

- **Hadoop:** Desarrollado por la fundación Apache, es un proyecto de código libre y que permite trabajar con grandes volúmenes de datos. Está inspirado en los proyectos Google File System y Google Map Reduce, aunque estos no son públicos (sí existen algunos papers con sus características y especificaciones públicas).

- **CouchDB:** Es una base de datos de código abierto que pertenece a las bases de datos NoSql y que ejecuta MapReduce mediante unos filtros llamados “views”. Se accede y modifican sus datos mediante llamadas HTTP como si fuese un servidor web.

[29] http://es.wikipedia.org/wiki/JSON

En este caso, las tres alternativas valoradas cumplen el requisito de poder trabajar con el algoritmo Map Reduce así como tener capacidad para trabajar con grandes volúmenes de datos. Además, las 3 bases de datos permiten una alta cambiabilidad de los datos guardando objetos o documentos semi-estructurados, es decir, con una estructura flexible para cada documento.

La gran diferencia entre las 3 es la facilidad de uso. Mientras que MongoDB proporciona los componentes necesarios para trabajar con ella usando el lenguaje PHP y CouchDB lo hace mediante un protocolo estándar, por último Hadoop requiere implementar algún conector propio. También hay diferencias a la hora de ejecutar las consultas: CouchDB requiere que se especifique antes de ejecutarlas cómo van a ser, es decir, no permite consultas dinámicas.

Este último punto fue decisivo, ya que nuestro sistema necesita ejecutar consultas diferentes para cada usuario del sistema y así poder ofrecer recomendaciones personalizadas. Por eso elegimos MongoDB. Elección que, por otra parte, no hemos lamentado hasta la fecha.

Ilustración 48 – Logo de MongoDB, la base de datos elegida para guardar las estadísticas del sistema

4.2.3.2. Base de datos para la implementación del muro

Unos de los objetivos de este PFC es la implementación de un sistema de mensajes tipo muro parecido a los que podemos encontrar en webs tan conocidas como Facebook.com o Tuenti.com. La implementación de este módulo se hará usando la base de datos que elijamos en este apartado y que servirá para guardar y obtener los mensajes con las acciones de tus amigos en el sistema.

El problema para implementar este módulo es que un mismo mensaje debe aparecer en varios muros, que corresponden a los muros de los amigos del usuario que realiza la acción. Así, si un usuario confirma su asistencia en un evento, todos sus amigos podrán ver este mensaje e interactuar con él. Hay dos maneras de conseguir esta duplicidad:

- *On read*: sólo se guardará el mensaje una vez en el sistema. A la hora de crear un muro leeremos los mensajes de cada uno de los amigos y los organizaremos para obtener el muro.

- *On write*: cada vez que se cree un nuevo mensaje se guardará varias veces, una para cada muro que debe aparecer. A la hora de crear el muro, solo tendremos que leer una lista de mensajes.

La primera opción es la ideal cuando el número de amigos es pequeño, puesto que es la más simple y no introduce redundancia de datos, pero para sistemas grandes aparece un problema de eficiencia. En una base de datos MySql se tendrían que hacer varias consultas que podrían llevar a colapsarla y por eso usaremos la segunda opción, usando una base de datos NoSql. Los mensajes se guardarán varias veces en la base de datos y cuando un usuario acceda a su muro, este se podrá implementar con una sola consulta a su lista de mensajes.

Para elegir la base de datos utilizada tendremos en cuenta dos factores:

- **Cambiabilidad**: Igual que la base de datos para el sistema estadístico, a priori no podemos saber todos los tipos de mensajes que vamos a tener que mostrar al usuario. Priorizaremos aquella base de datos que con pocos cambios acepte un nuevo tipo de mensajes.

- **Escalabilidad**: Es indispensable encontrar una base de datos de alta y fácil escalabilidad debido al gran número de mensajes que tenemos previsto guardar.

Por eso consideraremos las siguientes alternativas:

- **MongoDb**: MongoDb tiene una gran ventaja y es que, al usarlo como base de datos para el sistema de estadísticas, nos ahorraríamos la instalación y aprendizaje de uso del mismo.

- **Cassandra:** Cassandra es una base de datos creada por Facebook.com y que poco a poco han ido usando otras grandes empresas de internet como Digg.com. Sus funcionalidades son limitadas: actúa como una base de datos con 4 o 5 índices (se puede configurar de una manera u otra), el esquema de los datos se debe especificar al crear las tablas y sus consultas contienen límites como obtener el orden de los datos ordenando uno solo de los campos. Gracias a estas limitaciones puede conseguir una velocidad de inserción de datos altísima y la escalabilidad de los datos es automática.

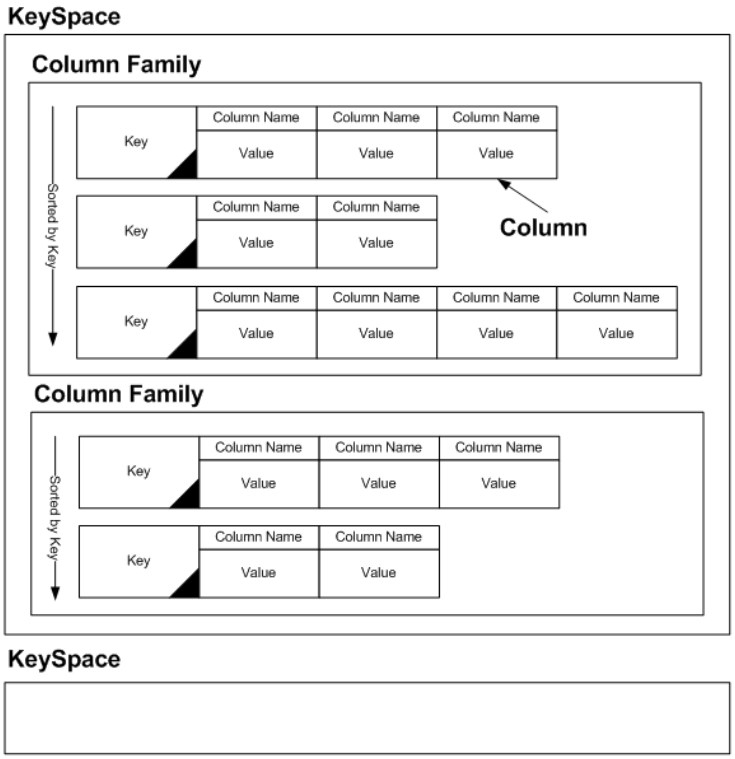


Ilustración 49 - Estructura interna de Cassandra, siempre se ordenan los resultados por el campo Key y cada fila puede tener diferentes columnas. Fuente: http://javamaster.wordpress.com/2010/03/22/apache-cassandra-quick-tour/

- **Tokyo Cabinet:** A priori parece la mejor de las bases de datos de las llamadas por "clave valor". Reciben este nombre porque funcionan guardando un valor en una clave como si se tratase de un índice. Después, dada una clave, puede recuperar el valor guardado. Esta base de datos es capaz de guardar hasta 1.000.000 de valores en menos de 2 segundos, cumpliendo sobradamente con nuestros requisitos de escalabilidad.

Para tomar la decisión final tuvimos en cuenta que Tokyo Cabinet no cumplía con nuestros requisitos en cuanto a funcionalidades, ya que para modelar las listas de mensajes de los muros debíamos inventarnos alguna manera para añadir información y no solo guardarla. Es decir, dada una lista guardada mediante una clave poder modificar su valor guardado.

Tras un largo análisis de la estructura de datos de Cassandra, vimos que nuestros mensajes podían ajustarse a ella: los mensajes siempre se ordenan por fecha, mediante unos índices (usuario > fecha > mensajes) y el esquema de la tabla siempre sería la misma. Los datos, que representan cada uno de los mensajes, pueden soportar atributos. Así que nos decidimos por usar Cassandra.

Ilustración 50 - Logo de Cassandra, la base de datos elegida para guardar los mensajes de muro

4.2.4. Elección del motor de búsqueda

El motor de búsqueda es la base del sistema de búsqueda que vamos a implementar. Su función será indexar la información disponible en el sistema y dar respuesta a las diferentes consultas que realice el usuario. Es necesario buscar un software especial que sea capaz de trabajar con grandes volúmenes de datos de manera eficiente ya que es importante dar una respuesta rápida al usuario en cada una de las acciones que éste lleve a cabo.

Valoraremos las siguientes características para elegir el software elegido:

- **Conectividad con MySql**: El motor deberá poderse conectar con nuestra base de datos principal para obtener los datos ya que allí guardamos la información de eventos, artistas y locales.

- **Filtros espaciales**: Deberá ser posible restringir los resultados de la búsqueda según una localización que introduzca el usuario y una distancia dada.

- **Recursos:** El software deberá ocupar pocos recursos (CPU y memoria RAM) ya que correrá en el mismo servidor que el servidor web y las demás bases de datos y no debe ralentizar el sistema.

- **Búsqueda "full text"**: El motor deberá indexar las palabras de los campos de texto de los diferentes elementos y buscar aquellos en los que aparezcan las palabras introducidas por el usuario.

Consideraremos las siguientes alternativas:

- **Sphinx:** Este motor de búsqueda presenta la ventaja que ya lo habíamos usado en algún proyecto anterior. Ofrece tiempos de indexación de datos muy rápidos, conectividad con MySql y utiliza pocos recursos en el servidor. Es ligeramente más lento que Solr a la hora de llevar a cabo las búsquedas.

- **Solr:** La alternativa de Sphinx, es un motor ligeramente más lento pero cuenta con la característica que permite el uso de plugins. Mediante plugins se pueden añadir modificaciones y filtros a prácticamente cualquier etapa del proceso, desde la indexación de datos a la ejecución de las consultas.

Esta decisión fue bastante complicada pero acabamos decidiéndonos por Solr. Éste ofrece una extensibilidad que permitirá ampliar el sistema de búsquedas con filtros por localización, consultas complejas, agrupación de datos (por ejemplo contar los eventos de cada categoría) o importar datos que están estructurados de manera compleja en la base de datos.

Ilustración 51 - Logo de Solr, el motor de búsqueda elegido para implementar el sistema de búsqueda

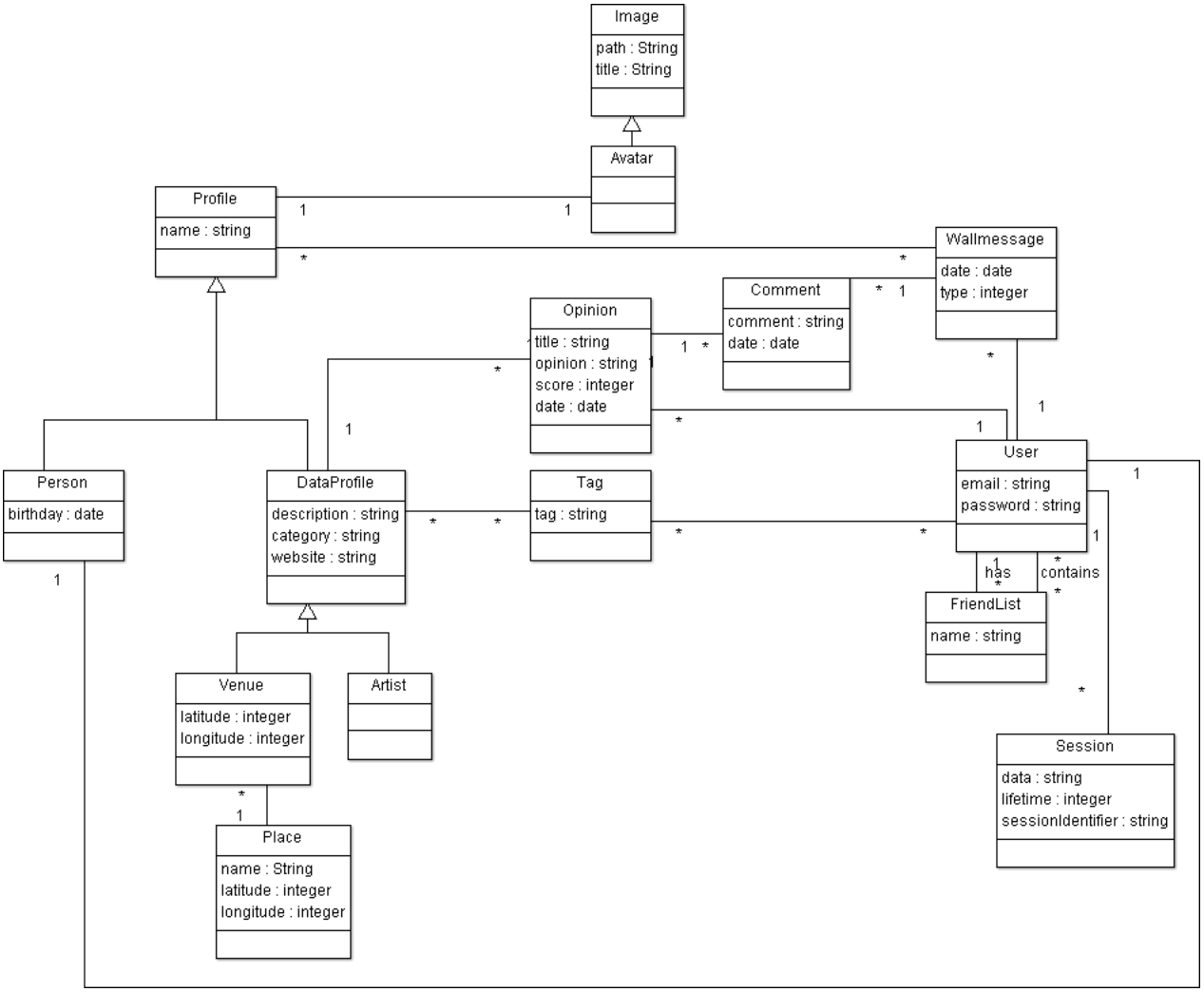
4.2.2. Modelo de datos

Un modelo de datos permite describir los elementos que intervienen en una realidad o en un problema dado y la forma en que se relacionan dichos elementos entre sí. A continuación veremos el modelo conceptual, físico y lógico así como los atributos de las diferentes clases en el modelo físico.

4.2.2.1. Modelo conceptual

Representa la vista lógica y es independiente del sistema gestor de bases de datos. Por comodidad, separaremos nuestro modelo conceptual en tres grupos de clases: en el primero incluiremos las clases que hacen referencia a los perfiles, en segundo lugar las clases que hacen referencia a un evento y por último, adjuntamos el diagrama que muestra cómo se guardará la información del usuario.

4.2.2.2. Modelo conceptual: perfiles



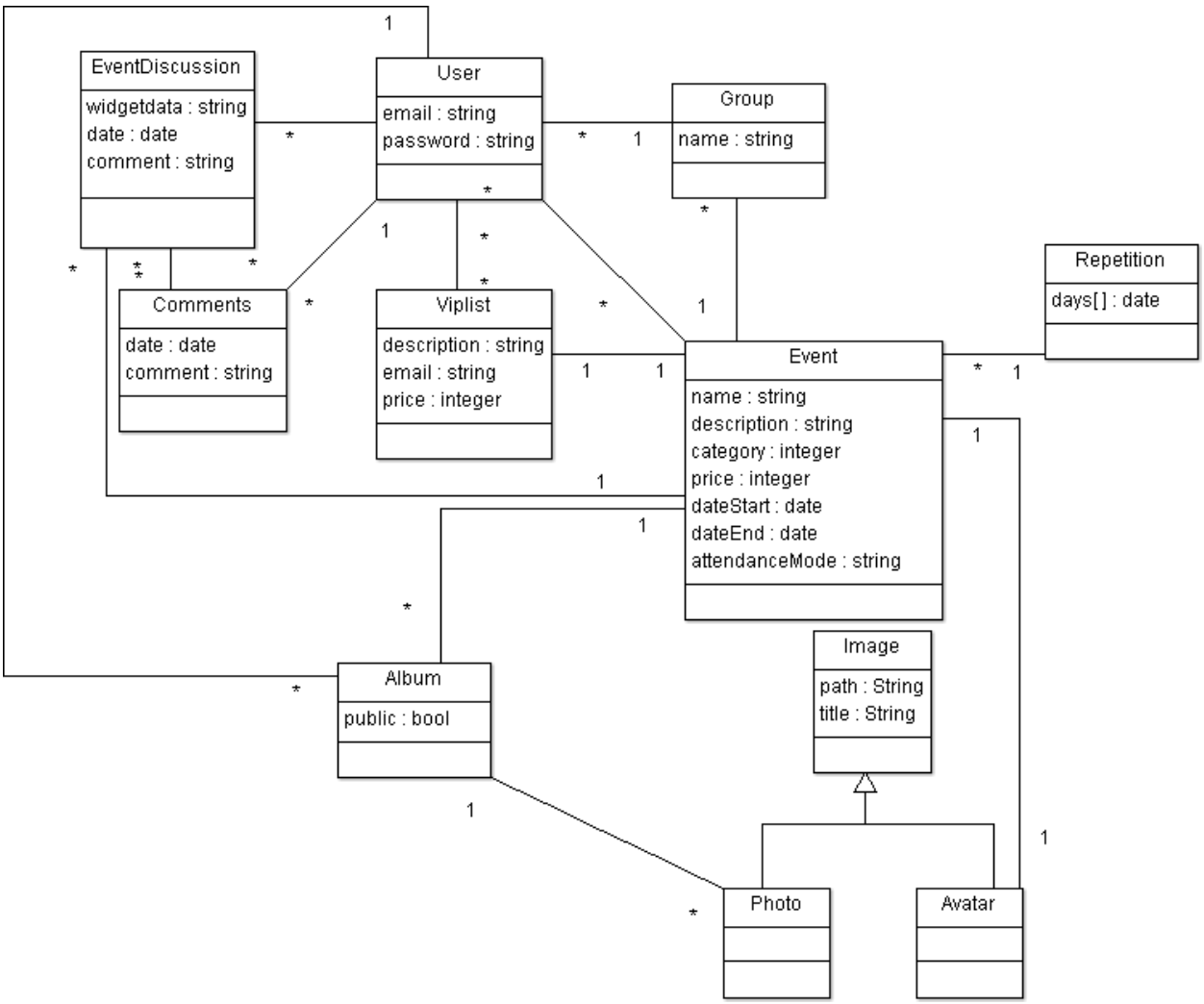
Esquema 1 - Modelo conceptual de clases relacionadas con Perfil

Los perfiles los estructuramos teniendo en cuenta que tendremos tres tipos de perfil: los dos primeros son los perfiles de los artistas y de los locales, y los agruparemos bajo la clase DataProfile para aprovechar las funcionalidades que ambos tienen en común. Así, añadiremos la clase Tag y Opinión que corresponden a las opiniones y etiquetas que los usuarios añaden a estos perfiles.

El tercer tipo de perfil es Person y corresponde a las páginas con información de un usuario determinado. Los tres tipos (Artist, Venue y Person) se agrupan bajo la clase Profile. Ésta se relaciona con los diferentes Wallmessages o mensajes de muro y contienen un avatar o imagen que mostrar en su perfil.

Por último, un usuario contiene varias FriendList o listas de amigos en las que organiza todos sus contactos en el sistema y una Session que identifica a un usuario en el sistema.

4.2.2.3. Modelo conceptual: eventos



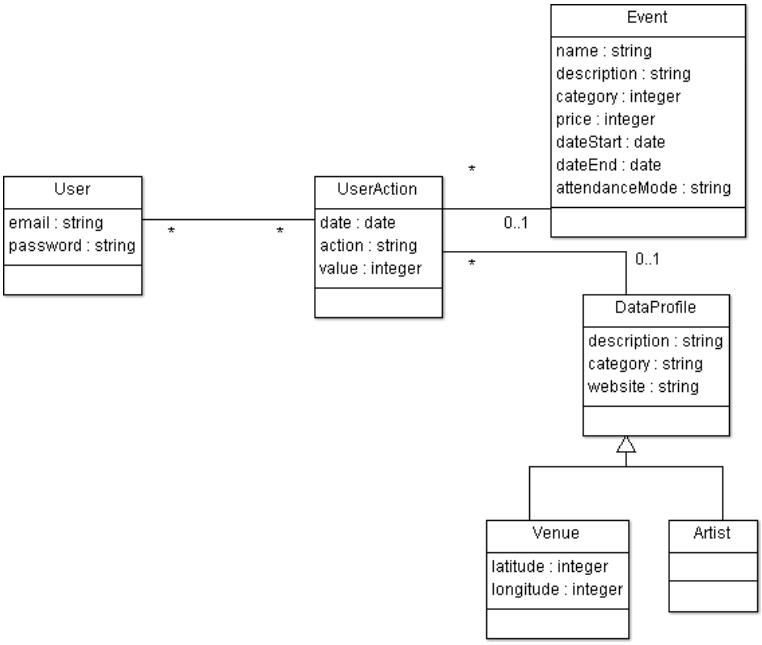
Esquema 2 - Modelo conceptual de clases relacionadas con Evento

Este diagrama se estructura en torno a la clase Event que representa cada uno de los eventos del sistema. Un evento puede contener:

- **Groups**, que son los diferentes grupos de asistentes al evento.
- **Viplist**, son ofertas especiales para asistentes al evento.
- **Repetition**, contiene la información sobre si el evento se repite a lo largo del tiempo y cuando tienen lugar estas repeticiones.
- **Album**, un grupo de fotos subido por el mismo usuario a un mismo evento.

- **Avatar**, la imagen que se muestra en el perfil del evento.

4.2.2.4. Modelo conceptual: acciones del usuario



Esquema 3 - Modelo conceptual de clases relacionadas con Usuario

Finalmente, las acciones de los usuarios se estructurarán en torno a la clase UserAction. Ésta se asocia o bien con un perfil de tipo Artist o Venue o con un Evento y con el usuario que realiza la acción.

4.2.3. Modelo lógico

A partir del modelo conceptual se crea el modelo lógico. Para crearlo se tiene en cuenta el tipo de bases de datos que vamos a utilizar. En nuestro caso tenemos 3 tipos de bases de datos y por tanto 3 modelos lógicos con una representación parcial de los datos del modelo conceptual. Tendremos una base de datos jerárquica para guardar los mensajes de muro, una base de datos jerárquica para guardar las UserActions, y una base de datos relacional donde guardaremos el resto de información. No tendremos en cuenta el indexador del motor de búsqueda porque este funciona autónomamente con datos de la base de datos Mysql ya que no tenemos un control sobre cómo se estructura de manera interna.

Debido a la similitud entre el modelo lógico y físico (se mostrará en el próximo capítulo) y a la gran cantidad de clases y atributos que contiene nuestro proyecto, nos ahorraremos adjuntar el modelo lógico.

4.2.4. Modelo físico

Es la aplicación del modelo lógico a un sistema gestor de base de datos determinado, en nuestro caso MySql como base de datos relacional, Cassandra para guardar los mensajes de muro y MongoDB para guardar la información sobre las interacciones entre el usuario y el sistema (UserActions).

4.2.4.1. Modelo físico: mensajes de muro

Los mensajes de muro se guardan en la base de datos no relacional Cassandra. Esta permite atributos dinámicos, o lo que es lo mismo, guardar diferentes atributos para cada uno de los mensajes de muro. El modelo es el siguiente:

```
Wall-id {
      date {
            wallmessage-id
            wallmessage-data
      }
}
```

4.2.4.2. Modelo físico: acciones del usuario

Este modelo corresponde al modelo conceptual “acciones del usuario” y se materializa gracias a la base de datos no relacional MongoDB. El modelo es el siguiente:

```
id {
      user
      type
      action
      value
      target
      date
}
```

Donde “id” es un atributo formado por varios de sus campos.

4.2.4.3. Modelo físico: base de datos relacional

En este caso guardamos el modelo en la base de datos Mysql. A continuación mostraremos las consultas SQL que permiten crear las tablas y campos (no están todas, simplemente es un resumen):

En este caso guardamos el modelo en la base de datos Mysql. A continuación mostraremos las consultas SQL que permiten crear las tablas y campos (no están todas, simplemente es un resumen):

```
CREATE TABLE `album` (
```

```
  `id` int(11) NOT NULL auto_increment,
  `user_id` int(11) NOT NULL,
  `event_id` int(11) NOT NULL,
  `visibility` tinyint(4) NOT NULL,
  `count` int(11) NOT NULL,
  `date_created` datetime NOT NULL,
  `last_modified` datetime NOT NULL,
  PRIMARY KEY  (`id`)
);

CREATE TABLE `artist` (
  `id` int(11) NOT NULL auto_increment,
  `name` tinytext character set utf8,
  `category` tinyint(4) NOT NULL,
  `description` text character set utf8 NOT NULL,
  `avatar_id` int(11) NOT NULL default '0',
  `user_id` int(11) default NULL,
  `website_url` tinytext character set utf8,
  `date_created` datetime NOT NULL,
  `last_modified` datetime NOT NULL,
  `tags_date` datetime NOT NULL default '2010-01-01 00:00:00',
  `sponsored` tinyint(4) NOT NULL,
  `likes` int(6) NOT NULL default '0',
  `dislikes` int(6) NOT NULL default '0',
  PRIMARY KEY  (`id`)
) ;
```

```
CREATE TABLE `avatar` (
  `id` int(11) NOT NULL auto_increment,
  `date_created` datetime default NULL,
  `last_modified` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `user_id` int(11) NOT NULL,
  `event_id` int(11) NOT NULL,
  `profile_id` int(11) NOT NULL,
  `image_id` int(11) NOT NULL,
  PRIMARY KEY  (`id`)
);


CREATE TABLE `event` (
  `id` int(11) NOT NULL auto_increment,
  `price` tinyint(4) default NULL,
  `price_max` tinyint(4) default NULL,
  `name` tinytext collate utf8_spanish_ci,
  `description` text collate utf8_spanish_ci,
  `category` int(6) default NULL,
  `type` tinyint(4) NOT NULL,
  `date_created` datetime default NULL,
  `date_start` datetime NOT NULL,
  `time_start` time default NULL,
  `date_end` datetime NOT NULL,
  `time_end` time default NULL,
(…)
```

5. Implementación del sistema

La fase de implementación nos la hemos dividido de una manera muy simple. Agrupar las historias similares en lo que hemos llamado **módulos de implementación** y cada uno de nosotros era responsable de un número finito de esos módulos. Gracias al uso de Scrum repartirnos estos módulos de manera equitativa ha sido muy fácil ya que al ser incremental podíamos corregir desviaciones en la carga de trabajo.

En cuanto a la manera de documentar estos módulos**,** se hará de tal manera que se explique de manera resumida las problemáticas que nos hemos encontrado en cada punto y la descripción de las soluciones que hemos decidido utilizar. Además para los módulos que tenga sentido se incluirá un punto en el que se expondrán las **historias de usuario** relacionadas. Estas historias se cogerán del capítulo 2 *Análisis de requisitos y especificación* poniendo sus identificadores de historia y sus descripciones en el módulo que se está documentando.

Finalmente una aclaración referente a la etapa de implementación del proyecto. En algunos módulos mostraremos dos versiones de las vistas v1 (versión 1) y v2 (versión 2). Esto es así porque durante el proyecto sometimos la primera versión de las vistas a unos tests de usabilidad (ver capítulo *2.3 Requisitos no funcionales* apartado Usabilidad) hechos por un grupo reducido de usuarios y con las conclusiones que obtuvimos pudimos diseñar la segunda versión de las mismas.

A continuación la documentación relativa a los módulos de los que me he responsabilizado, que forman parte del sistema Web.

5.1. Arquitectura

El diseño e implementación de la arquitectura es un punto clave en la construcción de un sistema de software. Es la etapa en la que se diseñan los componentes básicos que serán utilizados por los demás módulos de la aplicación. Un viejo dicho corrobora esta afirmación y dice así:

“the three most important things in software development are: architecture, architecture and architecture - in that order” - Por algún Gurú de la ingeniería del software.

El objetivo de la implementación de la arquitectura de un sistema de software podríamos decir que se trata de diseñar y construir el Framework que se va a utilizar en la construcción de la aplicación.

Ya hemos hablado de la definición de framework pero vuelvo a incluir la definición: “Un framework es un conjunto estructurado de software, librerías de software que tiene la función de servir de base para otro proyecto”. Además incluiría en esta definición que el framework también propone una serie de mejores prácticas y convenciones a seguir en la construcción del futuro proyecto.

La estrategia que hemos seguido nosotros en cuanto a la arquitectura ha sido la de escoger dos frameworks genéricos ya construidos y de uso público y adaptarlos a nuestras necesidades específicas.

Si se consulta el capítulo *4.2.1.1 Elección de frameworks* se verá que hemos escogido los siguientes frameworks para nuestro sistema

- **Zend Framework.** Es un framework en PHP que utilizaremos en el servidor.
- **Jquery.** Es un framework en JavaScript que utilizaremos en el navegador.

En los capítulos que siguen se explicará de qué manera se han adaptado estos frameworks al sistema Eventlayer.

5.1.1. Servidor (Zend Framework)

Zend Framework es un framework muy completo. Sus 2 principales características son:

1. Sigue completamente el paradigma de programación de orientación a objetos.
2. Sigue el patrón arquitectónico MVC (ver capítulo 5.2.1.1. MVC)
3. Se divide una serie de clases principales que pueden utilizarse, o no, unas independientes de las otras. A partir de ahora llamaremos a estas clases principales **componentes de Zend**

El punto 3 es muy interesante y favorece a una característica muy valorada en ingeniería del software como es la "modularidad". Básicamente permite que utilicemos sólo los componentes de Zend que vayamos a necesitar y de esta manera nos permite personalizar y construir nuestro propio framework a partir de Zend.

Eso es lo que decidimos hacer nosotros, crear nuestro propio framework PHP ampliando las funcionalidades de los componentes Zend. Los componentes de Zend se denominan de la forma Zen_<nombre del componente> e.g.: Zend_Date (componente dedicado al manejo de las fechas), Zend_Db (componente dedicado al manejo de las bases de datos)… De la misma manera, nosotros, para cada uno de los componentes creábamos mediante herencia nuestro propio componente que llamaríamos **componente de Tewp** (Tewp viene de The Events Web Project y lo pusimos antes de saber que nombre recibiría el proyecto). De esta manera tendremos Tewp_Date, Tewp_Db…

5.1.1.1. MVC

El componente principal de nuestro framework es el Tewp_Controller, este componente es el encargado de tratar la acción del usuario, por ejemplo el click en el botón crear evento, que llega por internet hasta el servidor y es el encargado de distribuir esa llamada por el resto de componentes hasta generar una respuesta que se devuelve vía internet al usuario.

Es por lo tanto el encargado de coordinar al resto de clases y por lo tanto define que responsabilidades tienen la mayoría de las otras clases principales ya que es el que controla lo que llamaríamos el flujo de trabajo de la aplicación.

Este componente se basa en el patrón arquitectónico MVC (Model View Controller). Este patrón es el estándar de facto en la construcción de aplicaciones web. Consiste en dividir todo el código relativo a la lógica de negocio en 3 tipos de clases.

Aclarar que cuando decimos "lógica de negocio" hablamos de todas las clases que relativas a tareas propias del producto que se está modelando. Es decir, si una clase contiene código que tiene relación con por ejemplo como se gestionan las peticiones de amistad en tu producto, hablaremos de lógica de negocio. Si por el contrario es código relativo a las tareas de detección de errores en el sistema esto no se considera código de lógica de negocio.

Los 3 tipos de clases son:

- **Modelos.** Son la representación específica de la información con la cual el sistema opera.
- **Vistas.** Se encarga de presentar el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario. En nuestro caso HTML+CSS[30].
- **Controladores.** Se encarga de responder a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

El modelo UML típico de este patrón es el siguiente:

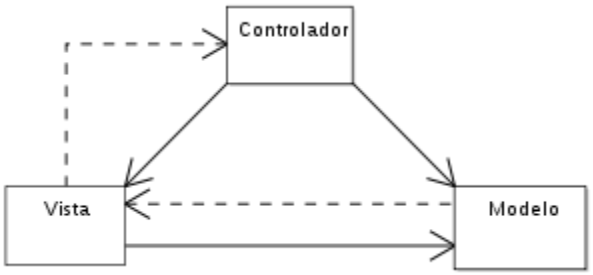


Ilustración 52 - Un diagrama sencillo que muestra la relación entre el modelo, la vista y el controlador. Nota: las líneas sólidas indican una asociación directa, y las punteadas una indirecta.

Además hemos complementado el patrón MVC con el patrón de clase **controlador** llamado Front Controller. Este patrón consiste en que todas las peticiones del usuario al servidor (requests) pasen por una clase común llamada Front Controller. Esto ofrece una gran ventaja de cara a la programación ya que disponemos de un punto común en el que realizar tareas comunes, por ejemplo, comprobar la acreditación del usuario, es decir la típica autentificación.

Gracias al uso de Zend Framework implementar este patrón y además integrarlo en el MVC es relativamente fácil usando el componente Zend_Controller. A continuación el diagrama de flujo propuesto por Zend.

[30] El código con el que se construyen las páginas web.

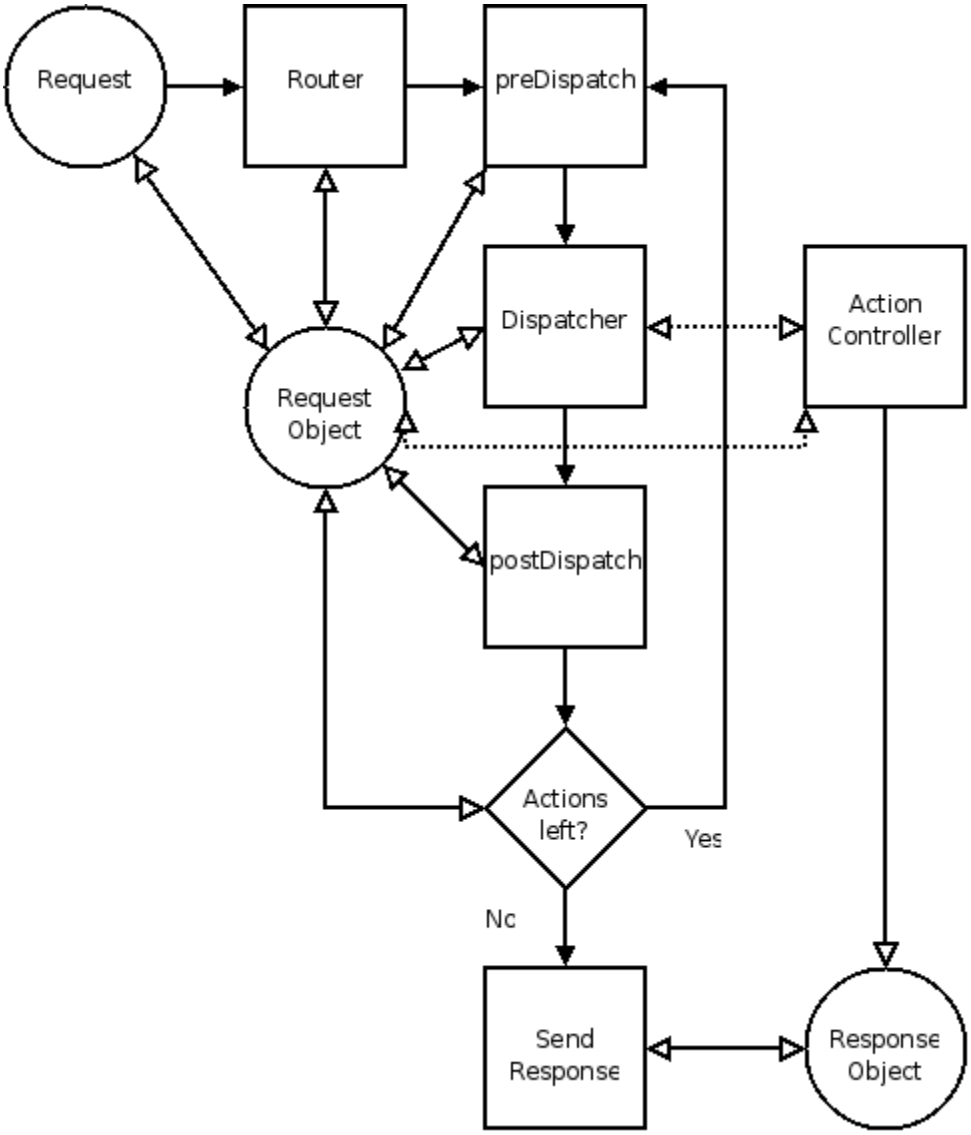


Ilustración 53 - Diagrama de flujo MVC del Zend_Controller

En el diagrama podemos ver como la requests del usuario son tratadas por la clase Router, esta clase sería la que implementaría el patrón Front Controller. La clase Router parsea la url que ha solicitado el usuario y escoge un controlador que como indica el patrón MVC se encargará de responder al evento de la request.

Cuando decimos que parsea la url nos referimos a que si el usuario clica a un botón de la web que a su vez llama al servidor con la url http://www.eventlayer.com/events/featured/page/3/ el Router hará lo siguiente:

1. Se ejecutarán las acciones comunes, entre ellas detectar que el usuario tenga permisos para acceder a esa página.
2. Se escogerá el controlador "eventsController" y se inicializará
3. Se llamará a la función featuredAction del controlador eventsController

4. Se pasarán como argumentos de la llamada a la función el parámetro page=3

El siguiente punto importante del flujo que se muestra en el gráfico es el postDispatch que es el momento en el que se ejecutan las vistas que generan la página web y que para hacerlo hacen uso de los modelos.

Otro aspecto a destacar en nuestra implementación del MVC es el uso del patrón Composite View **en las vistas**. Este patrón hace que la representación de vistas sea más manejable ya que gestiona los diferentes elementos de una página por medio de una plantilla. Frecuentemente, las páginas webs contienen una combinación de contenido dinámico y elementos estáticos, tales como cabeceras, pies, logos, imágenes de fondo, etc. Esto significa que una página web se construye a través de multitud de pequeñas vistas. Por ejemplo la siguiente pantalla se podría dividir en pequeñas vistas de la siguiente manera:

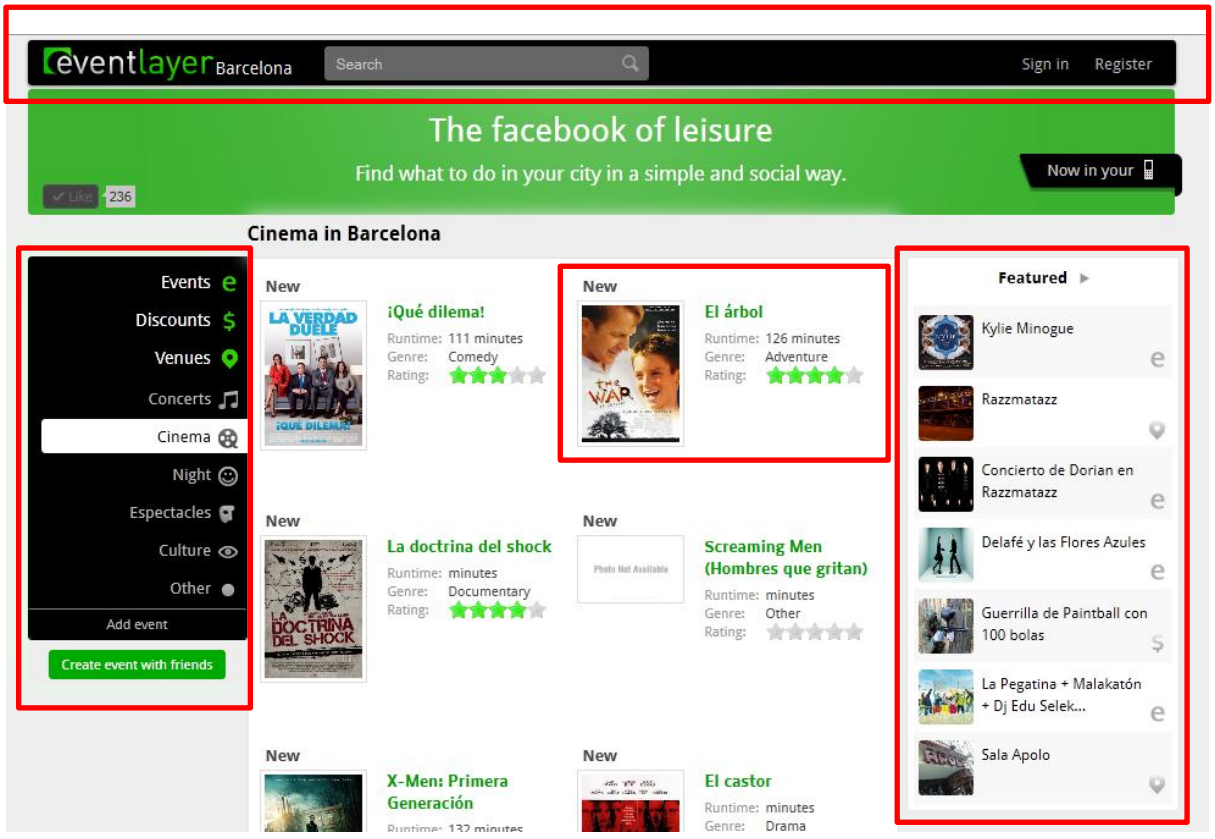


Ilustración 54 - Ejemplo de división en vistas usando Composite View. Cada recuadro representa una composite view que unidas forman la página de nuestra cartelera.

Esta división nos permite ahorrar código ya que muchas páginas comparten elementos similares, por ejemplo la barra superior. Si no usáramos este patrón deberíamos repetir el código de la barra en cada vista, con el composite, sin embargo, simplemente escogemos al final, en nuestro caso utilizando la clase Zend_Layout como unimos las diferentes vistas.

Hasta aquí hemos hablado de la utilización del MVC pero lo hemos mejorado utilizando en el **controlador** el Front Controller y en las vistas el **composite view.** En cuanto al **modelo** hemos optado por utilizar el patrón Service Layer.

Este patrón consiste en definir unas clases intermedias entre los modelos y las vistas que nos permiten reutilizar el modelo no sólo para las vistas sino como veremos en el capítulo *5.1.1.2 SOA* también para las vistas de los móviles.

5.1.1.2. SOA

A parte del patrón arquitectónico MVC en Eventlayer hemos seguido el patrón arquitectónico SOA (Service-oriented architecture).

Este patrón permite la creación de sistemas altamente escalables que exponen el negocio de la organización que modelan, en este caso nuestro producto Eventlayer. A su vez brinda una forma bien definida de exposición e invocación de servicios, normalmente vía web, lo cual facilita la interacción entre diferentes sistemas propios, en nuestro caso las aplicaciones móviles, o de terceros.

A continuación unos apuntes sobre la terminología SOA:

- **Servicio.** Una función sin estado, auto-contenida, que acepta una(s) llamada(s) y devuelve una(s) respuesta(s) mediante una interfaz bien definida. Los servicios pueden también ejecutar unidades discretas de trabajo como serían editar y procesar una transacción. Los servicios no dependen del estado de otras funciones o procesos.

- **Sin estado.** En una SOA los servicios no son dependientes de la condición de ningún otro servicio. Reciben en la llamada toda la información que necesitan para dar una respuesta. Debido a que los servicios son "sin estado", pueden ser secuenciados (orquestados) en numerosas secuencias (algunas veces llamadas tuberías o pipelines) para realizar la lógica del negocio.

- **Proveedor.** La función que brinda un servicio en respuesta a una llamada o petición desde un consumidor. En nuestro caso el servidor de Eventlayer.

- **Consumidor.** La función que consume el resultado del servicio provisto por un proveedor

- **Orquestación.** La forma en la que se comunicarán el proveedor y el consumidor.

De todos estos puntos el que hay que tener más en cuenta es el de “orquestación” ya que actualmente disponemos de varias alternativas para hacerlo como son SOAP, WSDL o REST… y por lo tanto hay que tomar una decisión importante.

Nosotros hemos optado por REST que es la más reciente y que trabaja directamente con llamadas HTTP que son las llamadas normales de cualquier web. Para ello utilizamos el componente de Zend, Zend_Rest_Server que ya implementa todo el protocolo REST del que al ser un estándar también existen librerías de código para utilizar en los móviles y de esta manera no tener que implementar el protocolo para cada dispositivo consumidor.

A parte de la utilización de este componente de Zend desde el primer momento, a sabiendas de que utilizaríamos SOA, organizamos todo nuestro Modelo en forma de servicios utilizando el patrón Service Layer (ya se ha comentado el uso de este patrón en el capítulo *5.1.1.1.MVC*).

Para utilizar la Service layer, para cada modelo principal como puede ser Evento, Artista, Lugar… se crea una clase de servicio que cumple con el "Sin estado" del que hablábamos en la terminología de SOA en la página anterior. Básicamente esta clase extra la llamábamos de la forma Service_Event, Service_Artist… todas estas clases son estáticas [31] y heredan de la clase Tewp_Service que contiene código estándar que permite estandarizar la respuesta del modelo a una forma entendible por Zend_Server_Rest.

Esta respuesta estándar de la que hablamos la hemos diseñado a partir del patrón Value Object que es un patrón de software que consiste básicamente en crear una copia de un modelo en un formato simple como es en nuestro caso un Array [32] en PHP. Para hacerlo de una manera automática en cada clase Service existe la función toVO que transforma los modelos de la respuesta en VOs o Value Objects.

A continuación un esquema UML del servicio Service_Event con las funciones principales:

[31] En la terminología de Programación Orientada a Objetos una clase es estática cuando no se instancia, es decir cuando no existen más copias de ella en el sistema y además no tiene estado.
[32] Un Array en PHP es un objeto estándar del lenguaje PHP que consiste en una estructura de datos en forma de matriz.

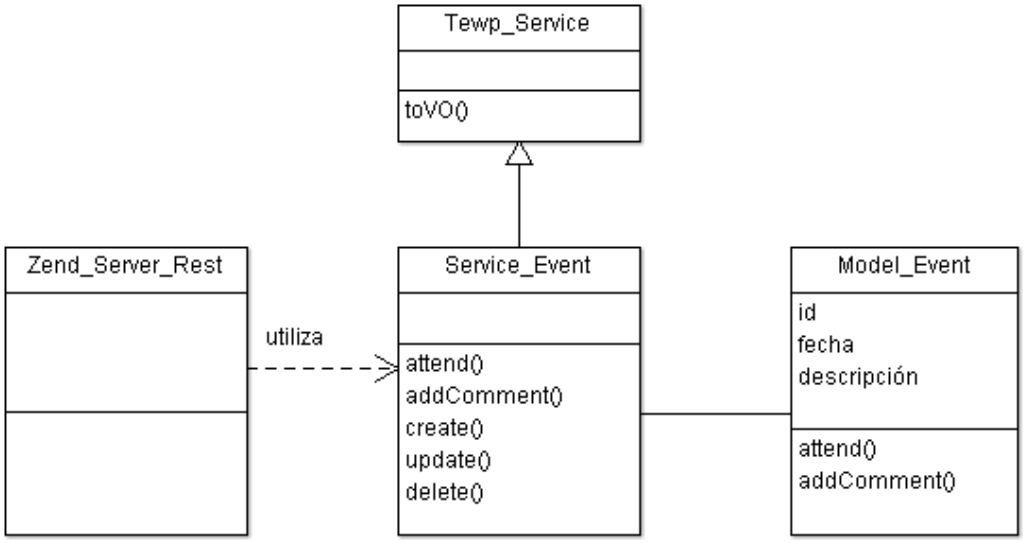


Ilustración 55 - Ejemplo de Servicio siguiendo Service Layer

En el diagrama se puede observar la función toVO de la que ya hemos hablado y se puede ver como el servicio no tiene los atributos del modelo ya que debe ser “sin estado”.

Y por último un ejemplo que pone toda esta teoría en práctica. El ejemplo consiste en una de las llamadas que utilizan las aplicaciones de Eventlayer en los móviles y la respuesta estándar de nuestro servidor.

La llamada es:

http://www.eventlayer.com/api/index/index/package/event?method=search&params[count]=10&params[offset]=0&params[filters][day]=MONTH&params[filters][category]=0&sk=729a6801691f1b9a42e848815a640ad9

La salida estándar es:

```
▼<Service_Event generator="zend" version="1.0">
  ▼<search>
    ▼<key_0>
       <id>5881</id>
       <name>'1395 días sin rojo', exposición en MACBA</name>
       <description/>
       <category>9</category>
       <type>1</type>
       <price/>
       <priceMax/>
       <attendanceMode>1</attendanceMode>
       <dateStart>1307606400</dateStart>
       <dateEnd>1315173599</dateEnd>
       <timeStart>10:00</timeStart>
       <timeEnd/>
       <user>2</user>
       <mainGroup/>
       <repetition>513</repetition>
      ▼<avatar>
         /content/photo/preview/f717dac2c809f92a639fc2a314729266.jpg
       </avatar>
       <dateCreated>2011-02-21 16:19:46</dateCreated>
       <lastModified>1307622997</lastModified>
       <websiteUrl/>
       <attendance>0</attendance>
       <group/>
       <tagsDate>1125498005</tagsDate>
       <city/>
       <likes>0</likes>
       <like>0</like>
       <dislikes>0</dislikes>
       <comments>0</comments>
       <attenders>0</attenders>
       <attenderPreviews/>
       <venuePreviews/>
    </key_0>
    ▼<key_1>
       <id>5882</id>
       <name>'1395 días sin rojo', exposición en MACBA</name>
       <description/>
       <category>9</category>
       <type>1</type>
```


Ilustración 56 - Ejemplo de salida estándar de nuestro servidor en modo REST

La gracia de este ejemplo es que utilizando exactamente el mismo modelo se pueden utilizar los datos tanto desde nuestras vistas en el servidor (para crear el HTML de la web) como para las vistas en las aplicaciones móviles.

5.1.1.3. Organización de los ficheros

La organización de los ficheros es muy importante en un proyecto de gran envergadura como Eventlayer, el proyecto a día de hoy, Junio 2011 tiene 9753 ficheros de código. La distribución está inspirada en la que propone Zend Framework en http://framework.zend.com/manual/en/intro.

Aclarar que los nombres tienen relación con la estructura de los ficheros, está todo explicado en el link anterior del Zend Framework, ya que Zend fue diseñado cuando PHP todavía no tenía disponibles los namespaces[33] (o packages como se denominan en lenguaje Java) y por lo tanto podemos encontrarnos con nombres de clases tan atípicos como:

Zend_Controller_Action_Helper_AutoComplete_Abstract

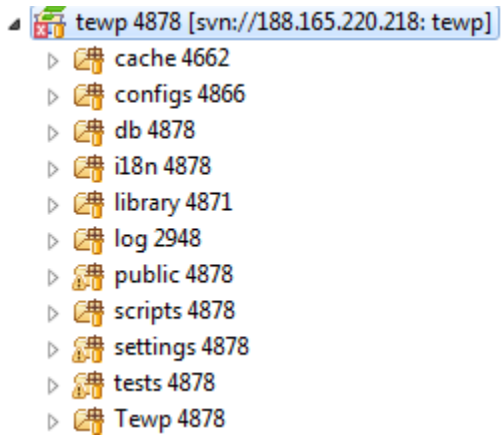


Ilustración 57 - Directorio raíz del sistema

De la ilustración del directorio raíz destacamos la carpeta "library" y la carpeta "Tewp" que contienen archivos .PHP. La primera es la carpeta donde se encuentra todo el código referente al Framework que utilizamos y en la segunda se encuentra todo el código referente a nuestra aplicación, a Eventlayer. Eso no quiere decir que no haya código nuestro en la carpeta de library ya que como hemos dicho hemos tenido que implementar nuestras propias clases de framework, en particular 640 ficheros de código.

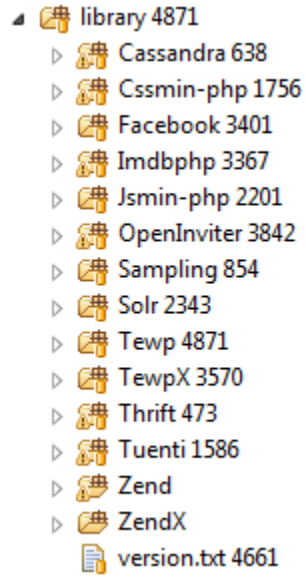


Ilustración 58 - Directorio de nuestro framework

La segunda ilustración corresponde al interior de la carpeta library. Podemos ver las carpetas con nuestro framework particular Tewp y TewpX, las carpetas con Zend Framework Zend y ZendX y a parte otras carpetas con otras librerías que hemos utilizado como son las librerías de Facebook (para comunicarnos con la famosa red social) , Tuenti (para interactuar con esta red social española), OpenInviter (para recoger contactos de los correos de nuestros usuarios)…

[33] Es un sistema que implementan algunos lenguajes de programación para poder diferenciar unas clases de otra aunque tengan el mismo nombre. En Java se denominan packages.

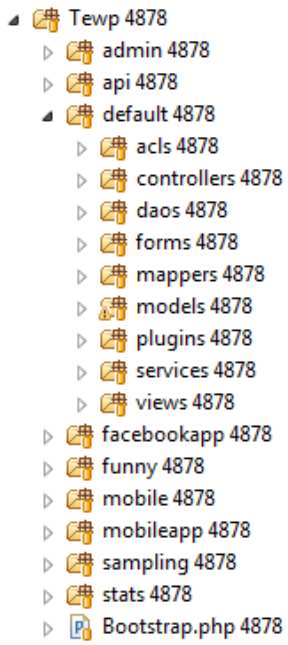


Ilustración 59 - Directorio con el código de Eventlayer

En la siguiente ilustración tenemos el directorio con los archivos relativos a la lógica de negocio de Eventlayer.

La primera división es referente a lo que Zend llama módulos. Tenemos el módulo de administración, el módulo de la aplicación de Facebook, el módulo de estadísticas… Se podría decir que son casi como webs separadas unas de otras.

Si nos centramos en el módulo "default", que es el que se usa por defecto, podemos identificar las carpetas "controllers", "views" y "models" que pertenecen a la división del MVC que hemos comentado en el capítulo *5.1.1.1 MVC.* A parte tenemos otras carpetas como forms, mappers, plugins… que son otros tipos de clases que recomienda tener Zend.

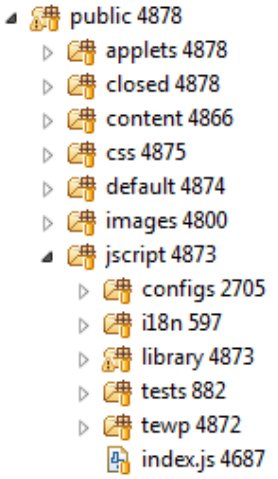


Ilustración 60 - Directorio de nuestra aplicación en JavaScript

La siguiente captura pertenece al directorio de carpetas del código JavaScript.

Se ha intentado respetar la misma organización que en las carpetas con código .PHP, no obstante un requisito impuesto por el servidor, utilizamos Apache, es que estos archivos deben estar bajo el directorio public. Esto es así ya que esta es la única carpeta accesible por el navegador del usuario y el JavaScript, a diferencia del PHP, al ser ejecutado en el navegador debe estar disponible de manera pública.

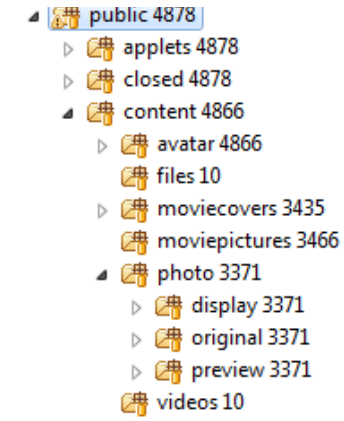


Ilustración 61 - Directorio Content

Por último una captura de la carpeta content para mostrar la manera en la que se organiza el contenido que va generando el usuario con su participación en la aplicación.

5.1.2. Cliente (Jquery)

En la parte del cliente, lo que viene a ser el código JavaScript escogimos, como puede verse en el capítulo 5.1. Elección de las tecnologías, el framework Jquery.

A diferencia del framework Zend Framework que utilizamos en el servidor, Jquery es un framework mucho menos completo. Esto es así ya que normalmente las aplicaciones web no hacen un uso intensivo de la parte cliente. Si miramos, por ejemplo un blog, sólo se utiliza JavaScript para animar ciertos botones y ciertas partes de la página.

No obstante la interfaz de Eventlayer se asemeja más a lo que son las RIA (Rich Internet Applications). Este tipo de aplicaciones web, un claro ejemplo es el cliente de correo GMail, se diferencian en que ejecutan gran parte del código de la aplicación en el cliente y de esta manera se minimiza el número de llamadas al servidor lo que convierte a estas aplicaciones en mucho más usables y rápidas de utilizar.

Es por ese motivo que inspirados en la filosofía de módulos que propone Zend Framework (ver capítulo *5.1.1.Servidor (Zend Framework)*), nos dedicamos a construir nuestro propio framework JavaScript.

5.1.2.1. Herencia en JavaScript

El primer problema que nos encontramos a la hora de implementar nuestro framework en JavaScript fue la falta de un mecanismo de herencia completo.

En JavaScript existe la herencia por prototipado pero no una herencia mediante clases y objetos como en PHP. Por suerte John Resign un gurú del JavaScript y el creador del propio framework Jquery ofrece en su página web personal un plugin que se puede utilizar para emular la herencia de PHP en JavaScript (http://ejohn.org/blog/simple-javascript-inheritance/).

El plugin en cuestión es una clase JavaScript, en nuestro framework la hemos implementado con el nombre de Tewp.Core_Class de la que heredan el resto de clases. El truco está en que en esta clase maestra se define la función “extend” que contiene el código necesario para emular la herencia por clases. Un ejemplo real del uso de este plugin en nuestra aplicación sería el siguiente:

A continuación el código de inicialización del controlador de eventos en JavaScript:

```
Application.Page_EventsController = Tewp.Controller.extend( {
    init : function(sandbox, params) {
        this._super(sandbox);
        this.createFilter('eventsList');

        this.createFilter('moviesList');
        this.createFilter('cinemasList');
        this.createFilter('cinemaMoviesList');

    } …
```

Como vemos el controlador extiende de la clase Tewp.Controller y lo hace utilizando la función extend de la que hemos estado hablando.

A su vez la clase Tewp.Controller que vendría a ser una implementación inspirada en la clase Tewp_Controller que utilizamos en el Servidor que a su vez hereda de Zend_Controller (esta clase se explica en profundidad en el capítulo *5.1.1.1 MVC.*

```
Tewp.Controller = Tewp.Core_Class.extend( {
            init : function(sandbox, params) {
                this._sandbox = sandbox;
                this._params = params;

                this.createFilter('categoriesList');

            }…
```

Y aquí vemos como la clase Tewp.Controller hereda también utilizando extend de la clase Tewp.Core_Class que es el plugin del que hemos hablado y que nos permite este tipo de herencia en JavaScript.

Gracias a este mecanismo (la función .extend) hemos podido replicar en el cliente la mayoría de clases del framework del servidor y de esta manera tenemos nuestro propio framework en cliente y en servidor y ambos siguen las mismas convenciones y utilizan las mismas clases principales.

5.1.2.2. Nested MVC

La siguiente problemática a solucionar en el framework del cliente era el uso del patrón arquitectónico MVC de la misma manera que lo hemos utilizado en el servidor.

La cuestión es que mientras que este patrón arquitectónico es ya casi un estándar de facto en el lado del servidor, en el lado del cliente, debido al poco tiempo que llevan las aplicaciones RIA entre nosotros no hay un diseño oficial claro.

Realmente fue difícil, estamos hablando de Enero del 2010, de encontrar una solución clara a nuestro problema. Finalmente en el libro Ajax in Action de David Crane encontramos una solución a nuestro problema con lo que él llamaba “nested MVC” o MVC anidado.

Este patrón arquitectónico se basa, como indica su nombre, en anidar un MVC en el cliente que ejecute llamadas contra el MVC del servidor. De esta manera tendremos en el cliente vistas, controladores y modelos que se comunicarán con el servidor mediante llamadas AJAX[34].

A continuación la ilustración contenida en el capítulo de Nested MVC del libro Ajax in Action que fue la que nos inspiró a nosotros en el diseño de nuestras clases MVC en JavaScript.

[34] Asynchronous JavaScript and XML, es un tipo de llamadas utilizadas en web que no obligan a que el navegador recargue la pantalla completamente, de esta manera re reducen los tiempos de respuesta y la usabilidad de la aplicación web.

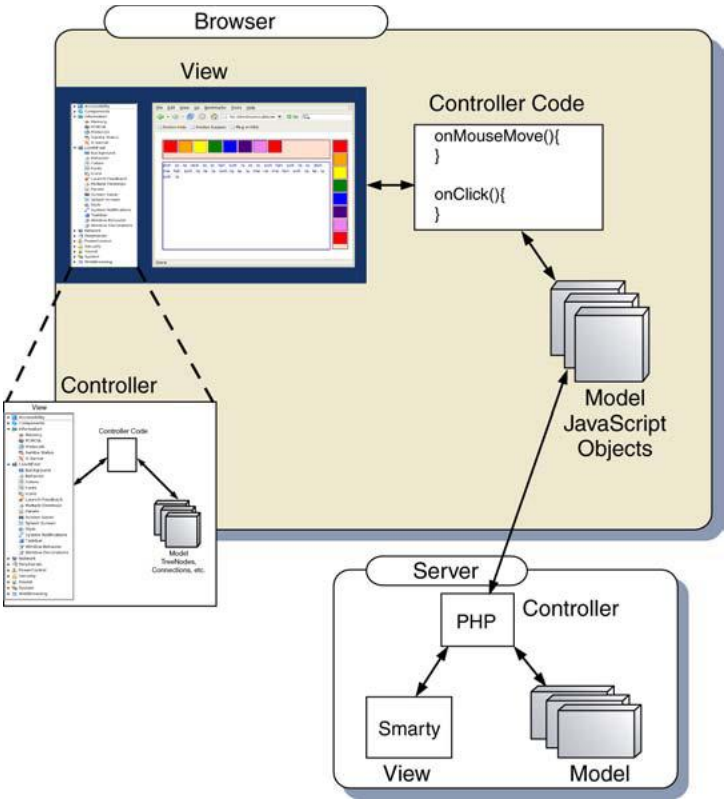


Ilustración 62 - Esquema del patrón Nested MVC

En la ilustración vemos como el modelo JavaScript se conecta con el controlador PHP. Nosotros no lo hemos implementado así en todos los casos sino que a veces también hay una conexión entre el controlador de JavaScript y el Controlador de PHP sin pasar por el modelo.

El principal motivo de esto es que hay acciones como por ejemplo cuando el usuario pide abrir el listado de descuentos en los que no se está actualizando ningún modelo en cliente sino que simplemente se quiere pedir al servidor una nueva vista que mostrar.

La implementación de este Nested MVC es bastante larga de explicar, básicamente la idea principal es que la primera vez que se carga la página de Eventlayer se inicializa la aplicación RIA en el navegador del usuario. Esto significa que se crean los modelos y se inicializa el controlador principal. A partir de ese momento ya no se producen más recargas totales de la página, todas las siguientes llamadas se hacen mediante Ajax y nuestro nested MVC.

Este hecho es fácil de comprobar ya que la barra superior de nuestra aplicación nunca se recarga en ninguna llamada. Otras páginas que trabajan de la misma manera son Facebook y Twitter. A la primera les es últil porque de otra manera no podrían mantener abierto el chat y la segunda lo hace igual que Eventlayer, por cuestiones de velocidad y usabilidad. La ventaja es clara, una llamada Ajax siempre es más eficiente que una recarga total de la página.

Esto se hace aún más evidente en casos como Eventlayer en el que durante la inicialización de la aplicación se cargan todos los amigos del usuario de tal manera que se puedan ejecutar búsquedas más rápidas con técnicas de Pre-fetching (precarga de datos).

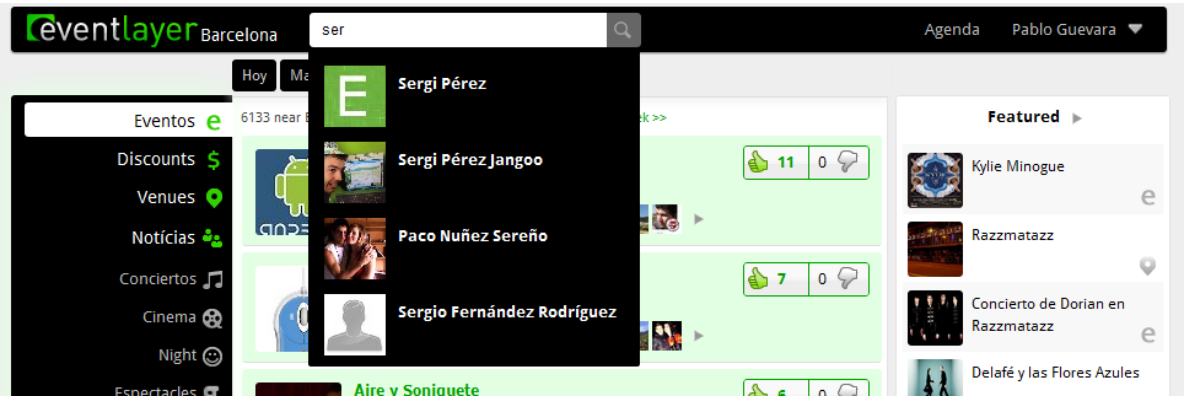


Ilustración 63 - Barra superior de Eventlayer en la que se puede ver como se precargan los amigos del usuario para que las búsquedas sean inmediatas.

Finalmente un esquema UML en el que se puede ver como el controlador de JavaScript (en este caso el de eventos) utilizando la clase Tewp.Server (una clase implementada en nuestro framework de JavaScript) se comunica con el FrontController del servidor.

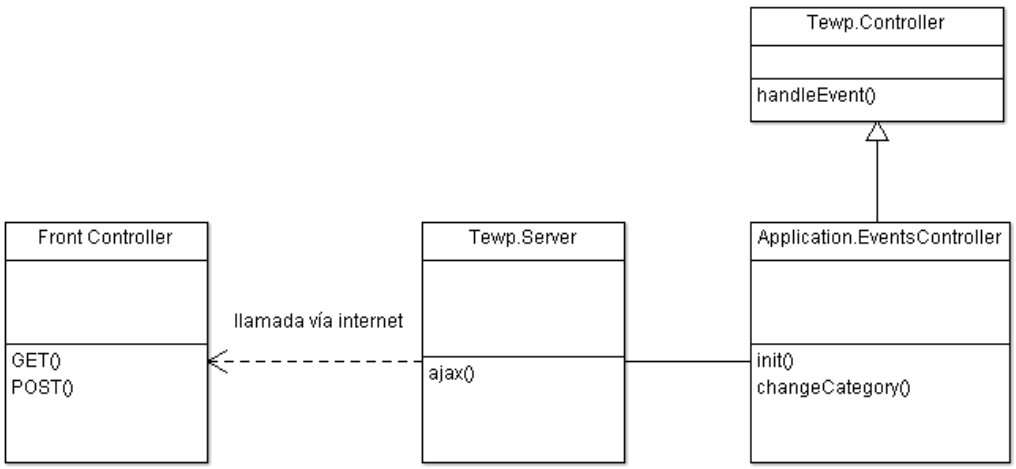


Ilustración 64 - Esquema de comunicación vía Nested MVC en Eventlayer

5.2. Gestión de errores

Historias de usuario relacionadas:

- **U-0113** – USER & ANONYMOUS quiere que cuando se produzca un error en el sistema se le notifique lo mejor posible y pueda volver a utilizar la aplicación inmediatamente para saber cuándo sus acciones no se han llevado a cabo satisfactoriamente.
- **U-0202** – STAFF quiere ver un listado de los errores ocurridos en el sistema para poder corregirlos.

Una buena gestión de errores en un sistema es fundamental para mantener un buen grado de "fiabilidad" que es uno de los requisitos no funcionales principales en cualquier aplicación (ver capítulo 3.3. Requisitos no funcionales). Es lo que se conoce como tolerancia a fallos.

Los requisitos de la gestión de errores que debía cumplir nuestro sistema eran:

1. Que el sistema continúe siendo utilizable después del error.
2. Que el sistema nos informe del error de tal manera que podamos encontrarlo fácilmente para así poder solucionarlo.

La estrategia que vamos a seguir nosotros en nuestro gestor pasa por la utilización del paradigma de gestión de errores basado en excepciones. Este paradigma se caracteriza por el uso de unas clases especiales que heredan de una clase padre llamada Exception. Estas clases son controladas (la terminología utilizada es "capturadas") con sentencias de código try y catch. Este tipo de paradigma está en la mayoría de lenguajes modernos y es la opción más recomendable. En PHP y JavaScript que son los lenguajes que hemos utilizado nosotros existe soporte para este su utilización.

5.2.1. Tipos de errores

Lo primer que se debe hacer antes de diseñar un sistema gestor de errores es enumerar y conocer los diferentes tipos de errores que nos podemos encontrar en nuestro sistema. Esto es importante ya que no es algo trivial y define el alcance de nuestro gestor. A continuación el análisis que nosotros llevamos a cabo:

- **ERR-TIPO-1 - Errores provocados por un código mal programado.** Estos errores son impredecibles, son debidos a un código imperfecto que da lugar a situaciones no contempladas. A su vez estos errores pueden ser de 2 tipos:

    - ERR-TIPO-1.1. Errores de código recuperables.

Son los errores típicos de acceso a una posición errónea en un Array, división entre 0 … En la mayoría de lenguajes, incluido los nuestros (PHP y JavaScript) existen excepciones propias del lenguaje que se crean en estos casos, nosotros sólo deberemos capturarlas y tratarlas. Son excepciones del tipo Tewp_Exception aunque nosotros también.

- ERR-TIPO-1.2. Errores de código irrecuperables.

Este tipo de errores sólo se dan en lenguajes de código totalmente interpretado. Esto es así ya que en los lenguajes compilados o semi-compilados (caso de Java) existe un proceso previo de detección de estos errores durante la compilación y por lo tanto no se dan durante la ejecución de la aplicación. En PHP se conocen como "Fatal error" y son errores tan graves como un código con una sintaxis errónea, un código haciendo referencia a una variable inexistente… Este tipo de errores no crean una excepción si no un Fatal error y además detienen la aplicación. Para tratar este tipo de errores nosotros utilizaremos una función llamada de "handling" que es ejecutada cuando se producen estos errores y la aprovecharemos para crear nosotros nuestra propia excepción que llamaremos Tewp_Exception_Type_FatalError.

Además nosotros añadiremos un último tipo de excepciones de la clase ERR-TIPO-1 - Errores provocados por un código mal programado que son necesarios para tratar los errores de JavaScript en el servidor

- **ERR-TIPO-1.3. Errores de código en JavaScript.**

Este tipo de errores son los errores que se producen en el cliente y son enviados, una vez tratados, al servidor para que sea el servidor quien los salve y los añada al fichero de logs que haga falta. Más adelante se hablará más en profundidad de los logs de errores. Las excepciones que utilizaremos serán del tipo Tewp_Exception_Type_Javascript.

- **ERR-TIPO-2-Errores de retorno.**

Estos errores los crea el propio programador para comunicar en una función o en una clase que se ha ejecutado esa función con unos parámetros que no son correctos. En PHP y en JavaScript no se suelen utilizar pero en Java se denominan excepciones Checked. En nuestro caso estas excepciones serán del tipo Tewp_Exception_Checked

- **ERR-TIPO-3-Errores de permisos.**

Estos errores se dan cuando el usuario intenta acceder a lugares de la aplicación en los que no tiene suficientes permisos. No son errores inesperados como los del tipo **ERR-TIPO-1** si no que los creamos nosotros mismos cuando detectamos una falta de permisos. El tipo de excepciones que utilizaremos para estos casos son de tipo Tewp_Exception_Forbidden.

- **ERR-TIPO-4-Errores de usuario.**

Estos errores son creados por el programador para notificar al usuario que ha cometido un error en la utilización de la aplicación. Nosotros los utilizamos básicamente en JavaScript y lo que hacemos es que cuando un usuario inserta un campo de una manera inadecuada por ejemplo nosotros creamos una excepción del tipo Tewp_Exception_ToUser y esta excepción se convierte en un mensaje de error que se le muestra al usuario para informarle.

5.2.2. Jerarquía de excepciones

Como hemos dicho nuestra estrategia en la gestión de errores pasa por una política de creación y control de excepciones, por lo tanto una vez analizados los posibles errores que pueden producirse en nuestra aplicación hace falta definir el modelo de clases de las excepciones que se va a utilizar.

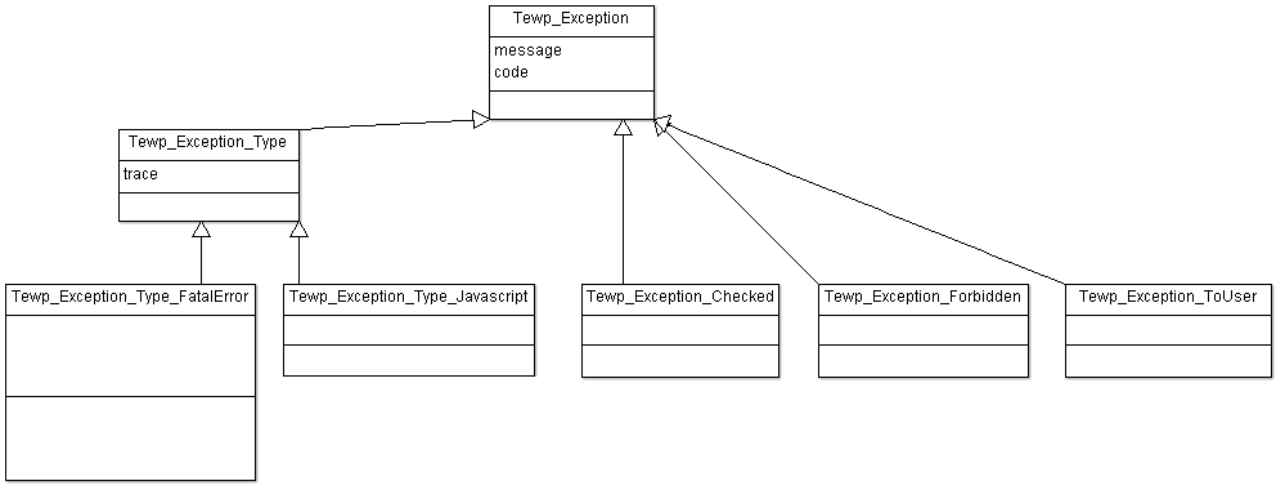


Ilustración 65 - Diagrama de excepciones utilizadas en Eventlayer

5.2.3. Método de reacción ante errores

Una vez definidas las excepciones a utilizar es hora de exponer el método de reacción frente a un error. En nuestro caso lo explicaremos de manera secuencial dividiendo las etapas por las que pasa en la aplicación tras un error y lo haremos centrándonos en las clases principales que intervienen en estas etapas.

a. Error_Handler.

La primera etapa consiste en capturar el error, esto significa que una vez el error se produce debemos ser capaces de detectar su aparición lo antes posible. Para hacerlo se utilizan las sentencias try catch y el uso de excepciones, esto es un procedimiento estándar y viene incluido en el lenguaje de programación que utilicemos. El problema real es dónde ponemos esta sentencia para que sea efectiva.

Lo que hemos hecho nosotros es crear dos clases llamadas Error_Handler, una para PHP y otra para JavaScript. Su utilización es la siguiente:

En PHP:

```
Tewp_Exception_Error_Handler::execute($errorHandlersEnabled,     $errorHandlersEnabled,
$errorHandlersEnabled, $fn, APPLICATION_PATH . '/../log/tewp-fatalerrors.log', array(
    'webservice' => 'webservice'

), $includeTrace);
```

En JavaScript:

```
var callback = new Tewp.Core_Callback(scope, handlerFn,extraArgs);
handlerFn = callback.toFn();
var wrappered = Tewp.Exception_Error_Handler.wrapper(handlerFn);

return wrappered;
```

El propósito de estas clases es el de que cualquier error producido en el código que se ejecute en su ámbito sea capturado (lo hace mediante try-catch) en la clase, se ejecuten algunas tareas comunes y se mueva el flujo a la siguiente etapa (2. ErrorController).

En PHP es fácil resolver la pregunta de dónde debe colocarse este handler, puesto que todas las llamadas del usuario pasan por el FrontController (ver capítulo *5.1.1.1.MVC*) pondremos allí el handler.

En JavaScript es más complicado ya que el usuario puede iniciar la acción en cualquier parte de las vistas ya sea clickando a un botón, a un link, rellenando un formulario… Además los errores también pueden llegar del servidor. Es por eso que hemos introducido el handler en todas las vistas con eventos iniciados por el usuario y además en la clase Tewp.Server (ver capítulo *5.1.2.2.Nested MVC*) que es la encargada de la comunicación con el servidor.

Por lo tanto, en resumen, el handler tiene la responsabilidad de capturar la excepción lo antes posible y si el tipo de la excepción o el error no es el correcto, corregirlo. A partir de ese momento se llama al ErrorController, concretamente a su función onError() pasando como parámetro la excepción capturada.

b. ErrorController.

La responsabilidad del ErrorController es básicamente la de decidir qué acciones llevar a cabo en base al tipo de error que haya ocurrido.

Por ejemplo si el tipo de la excepción es Tewp_Exception_Type_FatalError (error en la programación) el mensaje que contenga se loggeará para tener constancia del error, sin embargo si es del tipo Tewp_Exception_ToUser (error del usuario) simplemente se mostrará un mensaje de error mostrando al usuario las indicaciones para corregir su última acción realizada.

Es por lo tanto el ErrorController, uno en PHP y otro JavaScript, una clase en la que existe el código que contiene la casuística de cómo actuar en base al tipo de error encontrado.

Las 2 acciones más comunes que ejecutarán corresponden a las siguientes fases en el tratamiento del error 3. Tewp_Logger para dejar constancia del error y 4. Error views para mostrar al usuario un mensaje de error.

c. Tewp_Logger.

En esta fase de lo que se trata es de que el error quede grabado en alguna parte (loggear) del sistema de tal manera que los administradores puedan consultar los errores que se han producido durante un determinado periodo de tiempo para poder corregirlos y que no vuelvan a ocurrir en un futuro.

El proceso de encontrar un problema en el código a partir de un log de error se conoce como “pruebas de regresión” esto es así porque dependiendo del error no está claro donde exactamente está el fallo ene l código y por lo tanto habrá que utilizar un poco de ingeniería inversa.

Es por eso que cuando más específica y completa sea la información que guardamos de cada error más fácil va a resultar encontrar el origen del problema. Para ello se utilizan 3 parámetros principales:

i. **Tipo de la Excepción.** En el capítulo *5.2.1.*Tipos *de errores* se describe el mapeo entre los tipos de excepciones y el tipo de error que representan.
ii. **Mensaje del error.** El mensaje de error nos da el motivo del error.
iii. **Traza.** La traza nos dice en qué punto se produjo el error.

Estos 3 parámetros se guardan en una línea de log en ficheros de texto en una carpeta de nuestro sistema. De esta manera y por medio de nuestra sección de administradores en la propia web de Eventlayer se pueden consultar los errores producidos en la aplicación por orden cronológico inverso. A continuación un ejemplo del visor de errores:

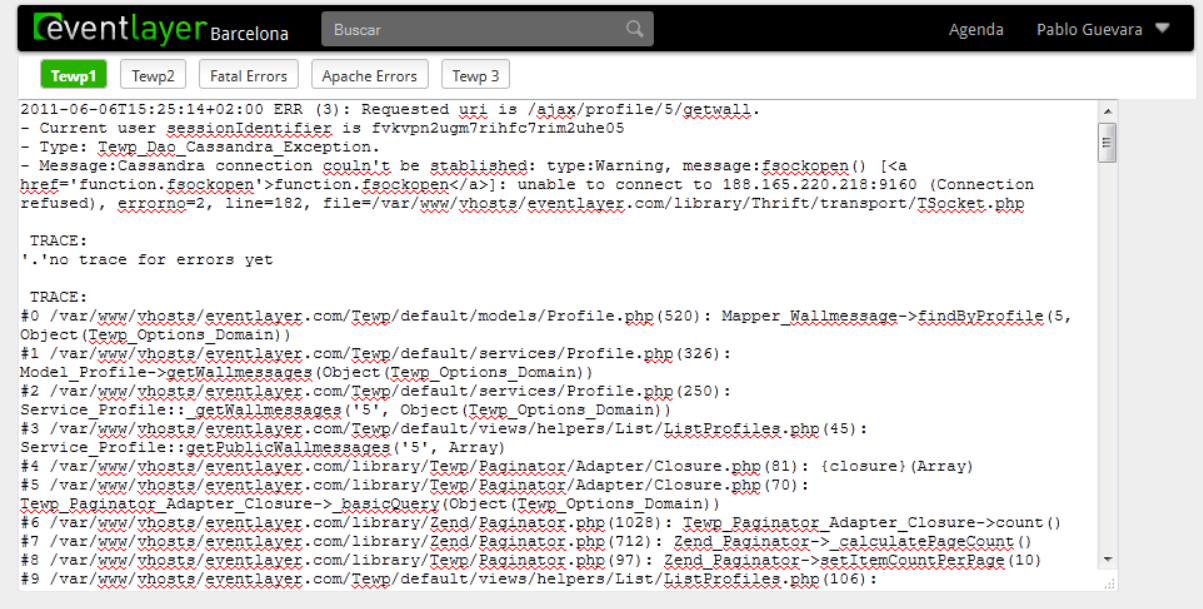


Ilustración 66 - Visor de errores producidos en Eventlayer.com

En el ejemplo se puede ver que:

a) El tipo de la excepción es Tewp_Dao_Cassandra_Exception que es una excepción del tipo ERR-TIPO-1.1. Errores de código recuperables.
b) El mensaje de error dice “Cassandra connection couldn’t be stablished …”. En resumen que no se pudo conectar con la base de datos Cassandra.
c) La traza nos indica que sucedió en la clase Profile.php en la línea 520.
d) Error views.

Una vez se ha interceptado el error y se ha loggeado es importante mostrarle un aviso al usuario para advertirle de que ha ocurrido algo con lo que él no contaba. Es por eso que utilizaremos diferentes vistas de error para informarle dependiendo de un caso u otro.

Estas vistas pueden variar bastante, por ejemplo una pantalla de error típica es la que avisa al usuario de que la acción que intenta llevar a cabo no puede hacerse sin estar registrado:

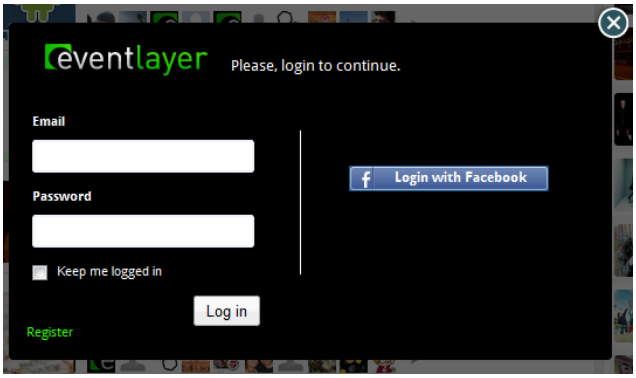


Ilustración 67 - Pantalla de error por no estar registrado

Otro tipo de vistas de error son para indicar al usuario que debe corregir la manera en la que llevó a cabo su última acción y para ello hay que darle instrucciones para que la próxima vez lo haga bien:

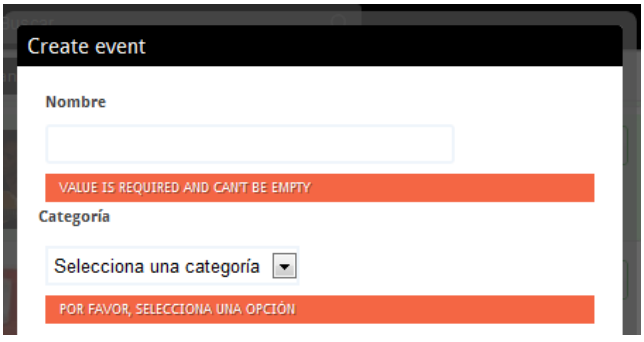


Ilustración 68 - Errores que se muestran en un formulario cuando los campos son obligatorios

Por último tenemos el caso de que el error que se ha producido no sea culpa del usuario sino un problema con nuestro propio código, en ese caso simplemente pediremos disculpas al usuario y le invitaremos a que pruebe más tarde:

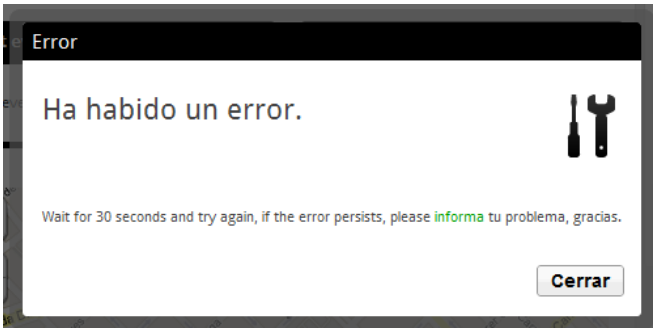


Ilustración 69 - Pantalla de error, cuando el error es por culpa del código de Eventlayer

5.3. CRUD

Historias de usuario relacionadas:

- **U-0001, U-0002, U-0003, U-0004**. Historias relacionadas con los listados de eventos, descuentos, lugares…
- **U-0008, U-0012, U-0013, U-0106**. Historias relacionadas con la creación, edición o eliminación de eventos, descuentos, lugares…

La manera en la que generalmente un sistema de información, Eventlayer lo es, crece en funcionalidades es mediante la incorporación de nuevos "entidades". Entendemos por entidad un objeto que queremos representar en nuestra aplicación.

Por ejemplo, la primera entidad que incorporamos al sistema fue "event". Pero después añadimos la entidad "venue", luego "artist" (lugar), luego "user" y así sucesivamente. A cada nuevo concepto de la realidad que se quiere modelar en un sistema de información se debe incorporar una entidad en el sistema.

Para manejar una entidad existen 4 acciones básicas

- Creation o creación en español. Hay que dar de alta un event, un venue o un artista y añadir su información en el sistema para poder trabajar con él virtualmente.
- **R**ead o lectura en español. Esto se refiere tanto a poder leer la información de un event, venue, artista en concreto y también la de poder listarlos y filtrarlos según convenga.
- **U**pdate o actualizar en español. Los datos de un event, venue, artista… deben de poder modificarse para mantenerlos actualizados.
- **D**elete o eliminar en español. Los datos deben poder borrarse cuando sean redundantes o incorrectos.

De estas 4 acciones recibe el nombre este capítulo CRUD (Create, Read, Update and Delete). Un buen sistema de información no debe sólo permitir realizar el CRUD para cada entidad sino que debe

facilitarle al programador el poder añadir nuevas entidades y ejercer sobre ellas el CRUD lo más automáticamente posible.

Ese fue el objetivo y el alcance en el diseño de este módulo de implementación, conseguir una manera lo más automatizada posible de poder añadir nuevas entidades al sistema. Hay que decir que crear un CRUD genérico es mucho más complicado que hacer el código individualizado, no obstante, el ROI (Return of Investment) es claro si tenemos muchas entidades en nuestro sistema.

En concreto, en Eventlayer tenemos a día de hoy, Junio del 2011, 61 entidades diferentes por lo que tener un CRUD genérico representa un gran ahorro de tiempo.

5.3.1. Tipos de entidades

El primer paso para crear el sistema CRUD genérico fue analizar qué tipos de entidades podíamos encontrarnos. Al hablar de entidades hablamos de la parte del modelo según el MVC (ver capítulo *5.1.1.1.MVC*) o Dominio según el patrón arquitectónico Domain Model. Estos dos patrones son complementarios y nosotros decidimos utilizarlos conjuntamente.

El patrón Domain Model nos ayuda a diseñar cómo será la M de nuestro MVC, es decir, como será el interior de nuestro Modelo.

Domain Model define 3 conceptos a la hora de modelar una lógica de negocio, entendiendo como lógica de negocio aquellos objetos de la realidad que utiliza nuestro producto o negocio (como ya hemos dicho: event, venue, artista….). Los 3 conceptos principales son:

- **Entity** o entidad**.** Ya nos hemos visto obligados a definir entidad al inicio del capítulo *5.3.CRUD* y lo volvemos a hacer diciendo que es cada uno de los objetos que va a modelar nuestro sistema de información. Si mapeamos entidad en terminología de Programación Orientada a Objetos (ver capítulo *4.1.Patrones de software*), una entity sería una clase.
- **Attributes** o atributos**.** Son los campos de información que tiene cada una de nuestras entidades. Event por ejemplo tendrá una fecha, una descripción…
- **Relations** o relaciones**.** Son las relaciones que se establecen entre las diferentes entidades. Por ejemplo un event tendrá una relación con la entidad artist ya que un evento se realiza con un artista determinado.

Ahora bien, según el tipo de relation entre 2 entitys podemos clasificarlas en 2 tipos:

- **Root Entity.** Es una entidad autónoma por sí misma. No depende de otras relations para tener sentido. Un ejemplo claro sería la entity event. En el sistema podemos tener un listado de event y éste nos será útil para su CRUD.
- **Leaf Entity.** Es una entidad que necesita de otra entidad, normalmente una root entity, para tener sentido en el sistema de información. Un ejemplo de esto sería la entity comment. Un

comentario siempre se realiza sobre otra entidad, por ejemplo, tenemos comentarios en un evento, comentarios en una opinión…

Esta clasificación de 2 tipos de entidades da lugar a que debamos plantearnos 2 variaciones en nuestro CRUD genérico. Una que nos permita Create, Read, Update y Delete de una entidad root y otra que nos permita hacer lo mismo pero para una entidad leaf que dependerá de una root y por lo tanto no tendremos la misma casuística a controlar.

5.3.2. Clases de nuestro CRUD genérico

Teniendo en cuenta los tipos de acciones que deben poder ejecutarse sobre cada entidad: creación, lectura, actualización y eliminación; y los tipos de entidades que tendremos que gestionar Root y leaf entities a continuación nuestra propuesta de gestor CRUD genérico.

Esta propuesta consiste en una serie de clases que se deben crear para cada nueva entidad y que una vez configuradas permiten realizar el CRUD de la entidad que representan. Para ello se ha creado en nuestro framework para cada una de estas clases una **clase genérica** de la que heredaran el resto de clases de ese tipo y que contiene ya implementadas parte de las funcionalidades.

5.3.2.1. Controlador y vistas

Los controladores son sólo necesarios para root entities. Las leaf entities utilizarán los controladores de la root con la que se relacionan. No obstante existirán vistas tanto para las root como para las leaf entities.

Para cada entidad hemos dicho que será necesario su lectura y su listado (con la posibilidad de utilizar filtros). Por lo tanto para cada entidad tendremos 2 tipos de vistas que mostrar

- Una vista en la que se muestre un listado de todas las entidades del sistema o un subconjunto filtrado de ellas (listar).
- Otra vista en la que se muestre toda la información de una entidad en concreto (leer).

Es por eso que a la R (read) del CRUD se le suele añadir la L de listar.

Por lo tanto tenemos 5 acciones CLRUD que ofrecer para cada entidad y tenemos que hacerlo mediante un mecanismo válido en una aplicación web como es Eventlayer.

Para conseguir esto en existe un estándar llamado REST (ya hemos hablado de él en el capítulo *5.1.1.2. SOA*). Este estándar, también denominado protocolo de comunicación, hace un mapeo entre las acciones CLRUD y las acciones HTTP que es el protocolo que utiliza internet en sus comunicaciones.

Las acciones de CLRUD ya hemos concretado cuales son y las acciones de HTTP son GET, POST, PUT y DELETE.

Aquí la tabla de mapeo que propone REST (fuente: http://en.wikipedia.org/wiki/REST)

Recurso	GET	PUT	POST	DELETE
Collection URI, such as `http://eventlayer.com/events/`	**List** the URIs and perhaps other details of the collection's members.	**Update** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URL is assigned automatically and is usually returned by the operation.	**Delete** the entire collection.
Element URI, such as `http://eventlayer.com/event/56`	**Read** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Update** the addressed member of the collection, or if it doesn't exist, create it.	Treat the addressed member as a collection in its own right and **create** a new entry in it.	**Delete** the addressed member of the collection.

Lo que viene a decir esta table es la manera en la que una url hace referencia a una de las acciones de CLRUD.

Por lo tanto nuestro controlador que recordemos es el encargado de decidir qué acciones debe llevar a cabo el sistema en base a la request (url) que el usuario pide debe contener toda esta casuística. Para poner algunos ejemplos para que esto se entienda mejor, si el usuario pide http://eventlayer.com/event/9218 con http GET el sistema mostrará la pantalla relativa al READ del evento número 5:

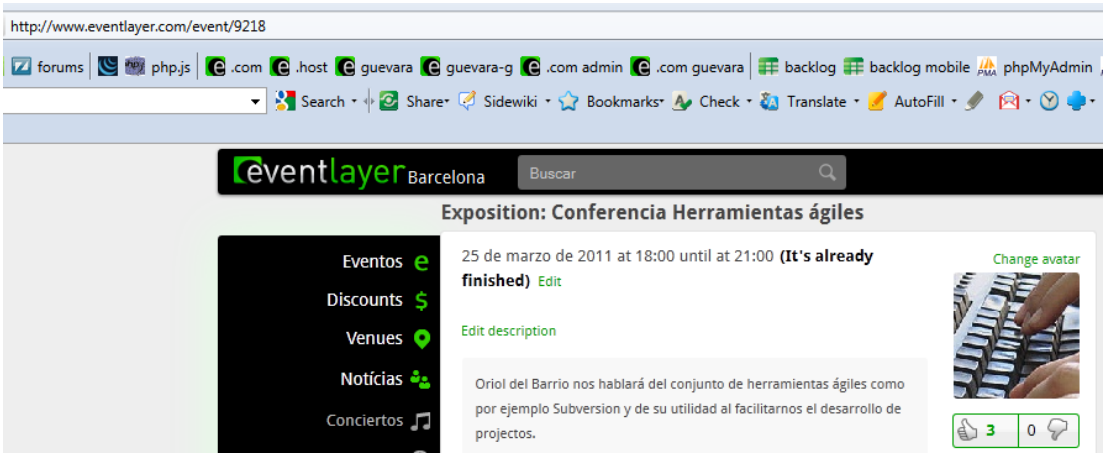


Ilustración 70 - Resultado siguiendo REST de eventlayer.com/event/9218 con http GET

Si por el contrario el usuario pide http://eventlayer.com/event/9218 con http DELETE el sistema mostrará la pantalla relativa al DELETE del evento número 5:

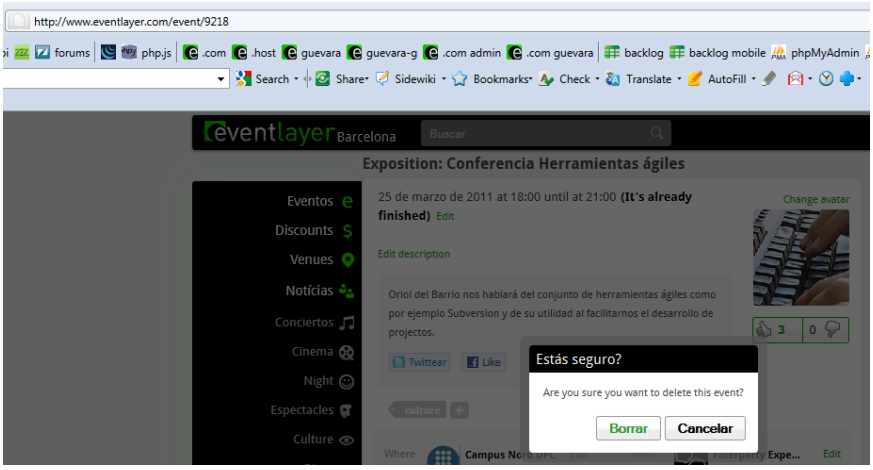


Ilustración 71 - Resultado siguiendo REST de eventlayer.com/event/9218 con http DELETE

Además en REST se hace la distinción entre element y collection, como se ve en la tabla de REST un element se hace referencia con el nombre del recurso en singular eventlayer.com/**event/**… mientras que para hacer referencia a la colección (listado de eventos) se utiliza el nombre del recurso en plural Eventlayer.com/events/…

Por lo tanto para adaptarlo a nuestro sistema basado en Zend deberemos crear 2 controladores para cada entidad, uno para la colección y otro para el elemento. A continuación el diagrama de clases que utilizamos en los controladores y las vistas para el caso concreto de la entidad event.

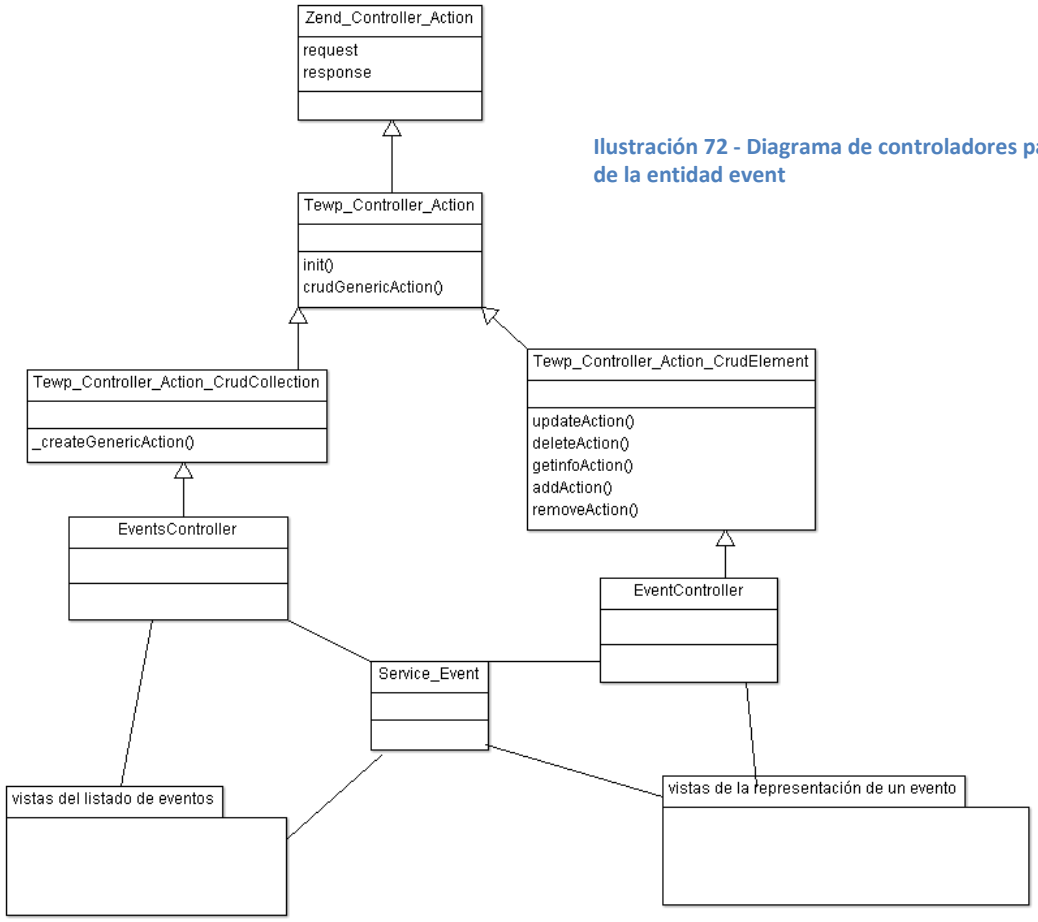


Ilustración 72 - Diagrama de controladores para el caso de la entidad event

En el diagrama se puede ver que la clase de la que todas heredan es Zend_Controller_Action que es una clase de Zend que contiene la mayoría del código que utilizamos y automatiza los procesos del MVC (ver capítulo *5.1.1.1.MVC*).

Más abajo tenemos Tewp_Controller_Action que implementa una serie de funciones genéricas para el CRUD. A continuación dos clases CrudCollection y CrudElement que definen el código genérico para implementar todas las acciones CRUD según sea Collection o Element.

Por último ya tenemos las clases que se crean específicamente para cada entidad como son el EventsController y el EventController que junto a las vistas generan las pantallas que ve el usuario durante la ejecución de las acciones.

Por último explicar que la clase que sirve de enlace entre estos controladores y vistas y los datos que representan es la clase Service_Event que es de tipo Service y de la que hablaremos en los siguientes capítulos.

5.3.2.2. Model

Clase necesaria tanto para root como para leaf entities.

Estas clases recibirán el nombre de Model_Event, Model_Venue, Model_Artist y así sucesivamente. Todas heredarán de la clase genérica Tewp_Model y su responsabilidad es la de contener los atributos de la entidad en el sistema.

Por ejemplo una parte del código de Model_Event sería:

```
/**
   * @var string
   * @attribute
   */
  protected $_name;

  /**
   * @var string
   * @attribute
   */
  protected $_description;

  /**
   * TODO Add where to find the correspondence with the string category
   * @var integer
   * @attribute
   */
  protected $_category;
```

Aquí se puede ver como para cada atributo de un evento se define una variable protected y se añade un comentario de cabecera donde se indica que es un @attribute y también se indica el tipo e.g.: @var string.

Estos comentarios son importantes ya que la clase genérica Model_Event los utilizará para realizar sus acciones genéricas.

5.3.2.3. Service

Clase sólo necesaria para root entities. Las leaf entities utilizarán el service de la root con la que se relacionen.

Estas clases recibirán el nombre de Service_Event, Service_Venue, Service_Artist y así sucesivamente. Todas heredarán de la clase genérica Tewp_Service y su responsabilidad es la de manipular las clases modelo.

Estas clases están relacionadas con el patrón Service Layer del que se ha hablado en el capítulo (*5.1.1.2. SOA*) y son clases estáticas que tienen entre otros métodos las funciones

- **Create.** Crea un modelo del tipo $modelClass con los datos $data.
    - `public static function create($data, $modelClass = null)`
- **GetInfo.** Devuelve el modelo o los modelos con el id(s) $ids teniendo en cuenta los parámetros $params.
    - `public static function getInfo($ids, $params = array())`

- **Search.** Devuelve los modelos que cumplen con las propiedades $properties (por ejemplo que la fecha sea anterior que hoy) y que siguen las condiciones de $params (por ejemplo sólo retornar los 5 primeros encontrados).
    - `public static function search(array $properties =null, $params = null)`
- **Update.** Actualiza el modelo con la nueva información $data.
    - `public static function update($data)`
- **Delete.** Elimina el modelo(s) con id(s) $ids.
    - `public static function delete($ids)`
- **Add.** Esta función se utiliza para las leaf entities que pertenecen a la root entity del servicio. Se encarga de añadir la leaf de tipo $modelName con la información de $arguments.
    - `public static function _add($modelName, $arguments)`
- **Get.** También relative a leaf entities. Devuelve la información de la leaf de tipo $modelName y que cumple con las condiciones de $arguments.
    - `public static function _get($modelName, $arguments)`
- **Remove.** Esta función también es relativa a leaf entities. Se encarga de eliminar la leaf de tipo $modelName que cumple las condiciones de $arguments.
    - `public static function _remove($modelName, $arguments)`

5.3.2.4. Mapper

Esta clase es necesaria tanto para root como para leaf entities.

Estas clases recibirán el nombre de Mapper_Event, Mapper_Venue, Mapper_Artist y así sucesivamente. Todas heredarán de la clase genérica Tewp_Mapper y su responsabilidad es la de guardar en base de datos la información de los modelos.

El tema del almacenamiento de modelos en bases de datos en muy controvertido. Hay libros enteros dedicados a este tema. Por supuesto es un tema de gran importancia ya que, qué sería un sistema de información si no pudiese almacenar y recuperar de una manera sencilla los datos de un medio persistente como es una base de datos.

El problema básicamente viene cuando se utiliza el patrón Domain Model, en Eventlayer lo utilizamos, junto con el uso de bases de datos relacionales, la mayoría de bases de datos lo son (MySQL, Oracle…).

Lo que ocurre es que en el Domain Model se trabaja con objetos (clases) y las bases de datos relacionales lo hacen con relaciones (filas y columnas en una base de datos). Estos dos conceptos no son, a priori, fácilmente interoperables y por lo tanto debemos buscar una buena estrategia para conseguirlo,

Hay varios patrones de diseño que nos ayudan a esta tarea, el patrón Row Data Gateway, el patrón ActiveRecord… Nosotros escogimos el patrón **Data Mapper**.

Este patrón consiste en añadir una clase por cada entidad que será la responsable de realizar el mapeo entre el modelo y su representación en base de datos. Son por lo tanto las responsables de conocer la estructura de la base de datos y cómo se relaciona con el modelo que representa.

El framework de Zend es de los pocos frameworks que no incluye un componente para implementar este patrón. Esto no representa en principio ningún problema ya que hay muchos proyectos gratuitos con este fin, se conocen como ORM (Object Relational Mapper). Uno de los más famosos es Hibernate que es para el lenguaje Java, pero también existen en PHP como son Propel o Doctrine.

El problema de estos ORM es que las consultas que hacen a la base de datos son genéricas, es decir, no están preparadas para aprovechar los índices [35] de una base de datos optimizada y además realizan muchos Joins [36] entre tablas.

Lo que ocurre con esto es que dificulta la escalabilidad el sistema (ver capítulo *2.3.Requisitos no funcionales*). Esto quiere decir que en el momento en el que empezáramos a recibir muchas visitas el primer cuello de botella del rendimiento sería la base de datos.

Esto es así ya que a mayor número de usuarios mayor número de consultas por segundo son ejecutadas a una misma base de datos. La solución típica es dividir la base de datos en n servidores diferentes (sharding[37]) y para ello las consultas a las bases de datos deben ser personalizadas.

Por este motivo decidimos implementar nuestro propio ORM que nos permitiera personalizar las consultas para poder optimizarlas. Realmente no pensábamos que fuera a suponer una carga de trabajo tan grande pero tenemos que reconocer que fue un trabajo muy duro. La complicación es que hay que tener en cuenta los diferentes casos de las relaciones entre entidades. Estas relaciones pueden ser One to Many, Many to One y Many to Many [38]y para cada una de ellas hay que programar un código que trate su casuística particular.

[35] Es un sistema de mejora del rendimiento en bases de datos
[36] Los joins son unas operaciones sobre bases de datos en las que se une la información de varias tablas en una sola
[37] Es la técnica consistente en dividir una base de datos en varias. Puede ser vertical (división por columnas) u horizontal (división por filas).
[38] Son los tipos de relaciones entre 2 entidades. Por ejemplo 1 evento puede tener muchos artistas por lo tanto esa relación sería One to Many entre event y artista. Sin embargo sólo tiene 1 fecha por lo que sería una One to One entre event y date.

Para el diseño nos inspiramos en el libro Patterns of Enterprise Application Architecture de Martin Fowler. Tiene un capítulo dedicado íntegramente al patrón Data Mapper.

Otros patrones complementarios al Data Mapper que utilizamos fueron:

- **Lazy loading**: este patrón de diseño permite utilizar el modelo y sólo pedir su información a base de datos en caso de necesitarla.
- **Entity Map**: este patrón de diseño permite guardar una referencia a cada uno de los modelos que se cargan de base de datos y así establecer una cache [39] de modelos.

En cuanto a la implementación utilizamos todas las ventajas que ofrece la versión 5.3 de PHP, como es el uso de closures [40]. También fue clave el uso Zend_Reflection que es un componente de Zend Framework que permite manipular el código de las clases en tiempo de ejecución de esa manera podíamos leer todos los atributos de un modelo y compararlos con los atributos de la tabla con la que estaba relacionada para así determinar si la estructura coincidía.

Tampoco me voy a extender mucho más sobre este tema ya que como he dicho hay mucha literatura al respecto y es todo bastante técnico.

5.3.2.5. Zend_Form

Por último tenemos la clase formulario que es necesaria tanto para root entities como para leaf entities.

Nuestra implementación de formulario sigue el concepto del formulario Zend_Form que es un componente de Zend Framework. Esto hace que un formulario tenga 2 responsabilidades:

a) Filtrar todos los datos que provengan del usuario para así evitar problemas de integridad en la base de datos (un número en un campo de tipo texto por ejemplo) o problemas de seguridad en el sistema por contenido maligno.

b) Mostrarle al usuario un formulario HTML que pueda rellenar de una manera intuitiva.

A priori para un lector familiarizado con temas de ingeniería de software le puede resultar raro que un mismo componente tenga responsabilidades de la capa de dominio [41] a) y también de la capa de vista b). Realmente parece que estamos acoplando demasiado estas 2 partes pero en varios artículos del framework Zend se justifica la idea al decir que de hacerlo de otra forma la información sobre la estructura de los datos que ingresa el usuario (el formulario) se estaría duplicando.

[39] Un elemento que permite guardar información que será utilizada en un futuro y así evita el coste de volver a generarla

[40] Es una técnica de programación de las más modernas que permite guardar funciones en variables y variar su scope (alcance de la variable)

[41] Según un patrón arquitectónico muy famoso en ingeniería del software (patrón de capas) siempre se ha de intentar programar teniendo por separado las vistas de la capa de dominio (los datos)

La verdad que en mi punto de vista el Zend_From es una gran idea. Nosotros lo hemos acoplado a nuestros modelos de la siguiente forma:

```
class Model_Event extends Tewp_Model {
    protected $_formClasses = array(
        'create' => 'Form_Model_Event_Create',
        'update' => 'Form_Model_Event_Update',
        'updatePartialDate' => 'Form_Model_Event_UpdatePartialDate' );
```

En este código lo que se puede ver es que el modelo event tiene 3 formularios, es decir, el usuario podrá modificar su información sobre el evento con 3 formularios HTML diferentes, uno en su creación, otro en su edición y en este caso un tercero para editar únicamente su fecha.

Por ejemplo esta sería la definición del formulario de modificación de la fecha:

```
class Form_Model_Event_UpdatePartialDate extends Tewp_Form {

  $label = $translator->translate('Start date');
  $startDate = new Form_Model_Event_Element_Date('dateStart', $label, $id);
  $this->addElement($startDate);

  $label = $translator->translate('Start time');
  $timeStart = new Tewp_Form_Element_Time('timeStart', $label, $id);
  $this->addElement($timeStart);

  $label = $translator->translate('End date');
        $endDate = new Tewp_Form_Element_DateBiggerThan('dateEnd', $label, $id,
      array(), array(
            'dateStart' => $startDate,
            'timeStart' => $timeStart
       ));
  $this->addElement($endDate);

        // user is added later but created here since end date needs it
  $label = $translator->translate('End time');
  $timeEnd = new Tewp_Form_Element_TimeBiggerThan('timeEnd', $label, $id, array(),
array(
            'dateStart' => $startDate,
            'timeStart' => $timeStart,
            'dateEnd' => $endDate
       ));
  $this->addElement($timeEnd);
```

Como se puede observar se crean dos elementos de fecha que sólo permitirán al usuario añadir fechas y otro 2 de tiempo que sólo permitirán al usuario introducir una hora determinada.

A continuación una captura de pantalla de cómo queda este formulario una vez se muestra al usuario.

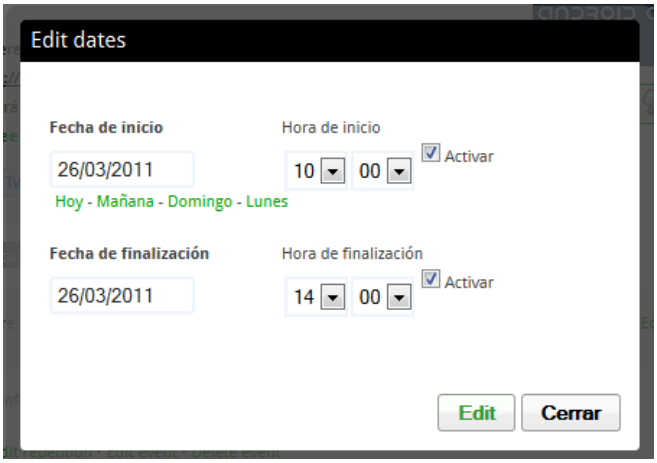


Ilustración 73 - Formulario de editar las fechas de un evento

La gracia de esta forma de trabajar es que el mismo código nos permite filtrar los datos del usuario en el modelo a) y además mostrarle al usuario un formulario que pueda rellenar en una vista b).

5.3.3. Diagrama del proceso de CRUD

Una vez definidas las clases que intervienen en nuestro CRUD genérico vamos a ver mediante un diagrama de colaboración UML como se relacionan poniendo el ejemplo de la creación de un evento.

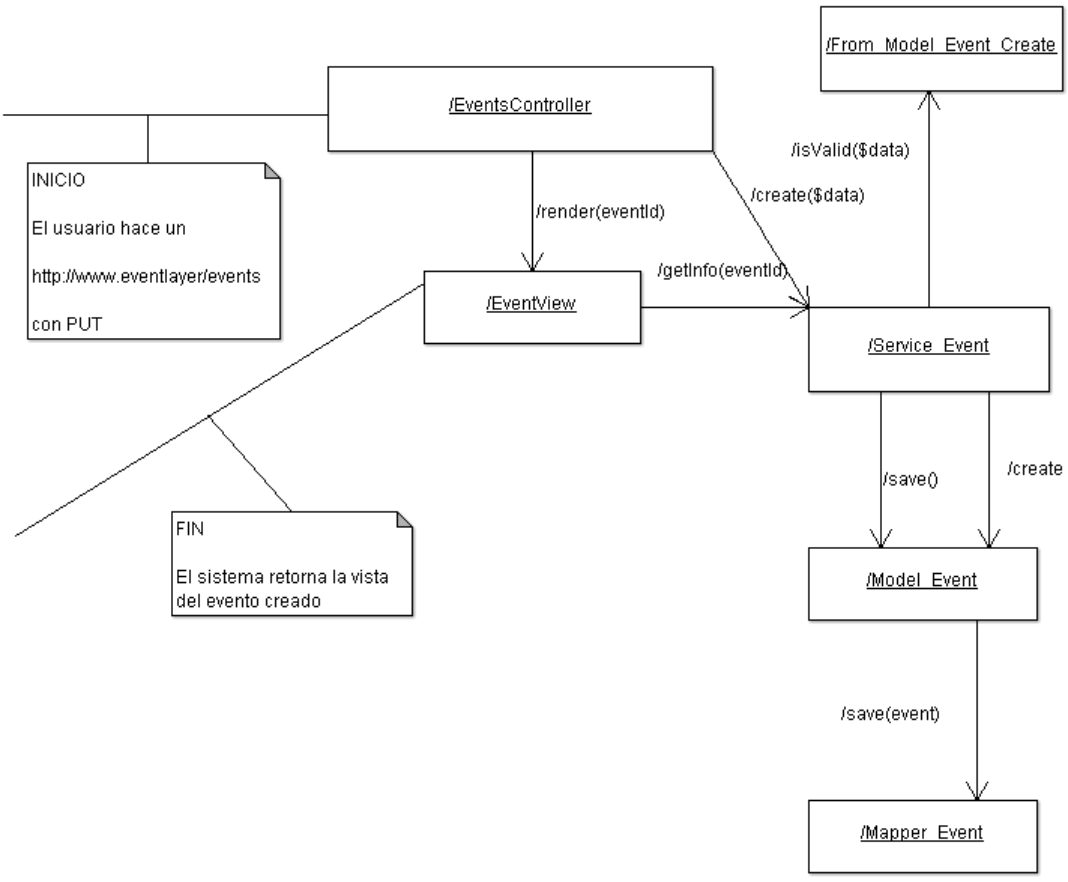


Ilustración 74 - Diagrama de comportamiento UML de la creación de un evento utilizando nuestro sistema de CRUD genérico

5.4. Gestión de usuarios

Historias relacionadas:

- U-0101 – ANONYMOUS quiere registrarse en el sistema para ejecutar historias de USER
- U-0102 – ANONYMOUS quiere logearse en el sistema para ejecutar historias de USER
- U-0103 – ANONYMOUS quiere deslogearse del sistema para que nadie pueda utilizar su cuenta.
- U-0104 – USER quiere añadir/eliminar a otro usuario como contacto para ver su perfil y asistir con él a eventos.

La gestión de usuarios al igual que cualquier gestión de entidades (user es una entidad) es sinónimo de CRUD (Create, Read, Update y Delete, ver capítulo *6.1.3. CRUD*). No obstante hay entidades complejas que requieren de un código adicional y específico para resolver sus historias de usuario que se escapa al comportamiento genérico.

Este es el caso de la gestión de usuarios. La gestión de usuarios tiene 2 historias concretas (A323, A342) que podríamos simplificar con los conceptos de "registro" y "autentificación" que requieren de un diseño e implementación extra que es el que trataremos a continuación.

Este módulo de implementación se llevó a cabo al más puro estilo SCRUM (ver capítulo *1.4 Metodología utilizada*) por lo tanto primero se hizo una investigación mirando otras web similares a Eventlayer (la mayoría fueron redes sociales) y se diseñaron las vistas que deberían implementarse.

5.4.1. Registro

El registro de un usuario consiste en dar de alta un usuario con sus datos y validar su email. Es este segundo punto el de validar su email lo que supone un comportamiento extra fuera de lo que sería un CRUD genérico. Las pantallas que debíamos implementar eran las siguientes:

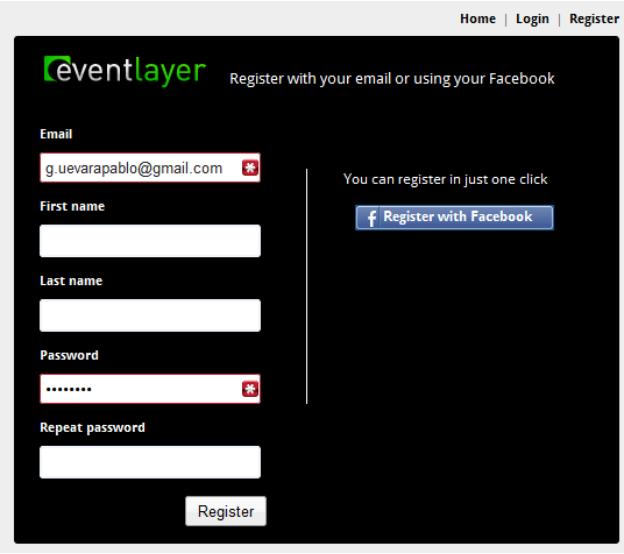


Ilustración 75 - Pantalla de registro en Eventlayer

Ilustración 76Ilustración 63 - Pantalla de registro en Eventlayer

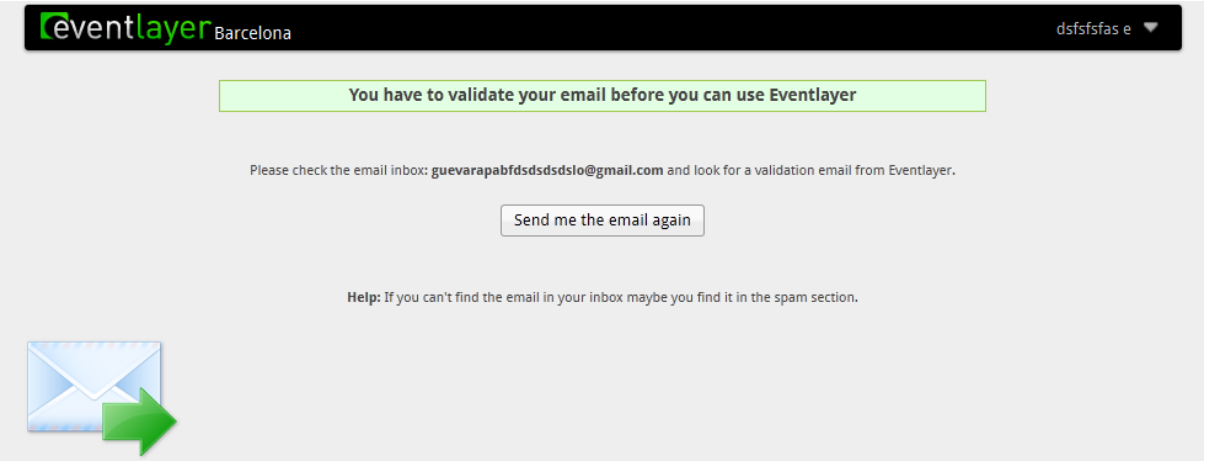


Ilustración 77 - Pantalla de validación del email

La primera de las pantallas es simplemente un formulario que el usuario debe rellenar y la segunda pantalla es una vez el usuario se ha registrado el sistema todavía no es utilizable hasta que el usuario valide su email.

Validar el email es necesario para **asegurarnos que no se crean usuarios falsos en el sistema. La validación consiste en que se envía un correo** a la dirección del usuario registrado con un link que el usuario debe clickar.

Para implementar este sistema de validación se utilizan dos campos en el modelo del usuario:

- **State.** Se trata de un campo de tipo integer que puede adoptar los valores 0 no registrado, 1 registrado pero no validado y 2 registrado y validado.
- **EmailToken.** Se trata de un campo de tipo string que consiste en una cadena de texto generada aleatoriamente de 27 caracteres que es única para cada usuario.

De esta manera el link que recibe el usuario en su correo es de la forma:

www.eventlayer.com/auth/validateemail/token/6dsfs5ew206rtfs5ew2089d7w3w

Así podemos asumir que el usuario no ha podido predecir esta cadena aleatoria de 27 caracteres a no ser que haya tenido acceso al email que le hemos enviado lo que significa que es un email real.

Esta técnica también la utilizamos cuando el usuario quiere recuperar su contraseña porque se le ha olvidado. El usuario en este caso recibe un email con un token y al clicarlo se le conduce a una pantalla donde puede introducir de nuevo su contraseña.

5.4.2. Login

Se conoce como login al proceso de autentificación de un usuario. No confundir autentificación con autorización que son 2 conceptos utilizados en seguridad de sistemas de información.

- **Autentificación** (login). La autentificación es el proceso mediante el cual el sistema confirma que el usuario es quien dice ser.
- **Autorización.** La autorización es el proceso mediante el cual el sistema confirma que el usuario autentificado tiene acceso a la parte del sistema a la que intenta acceder.

La autentificación (login) lo efectuaremos a través de la pantalla de login (introducción de email y contraseña) y utilizaremos el sistema de cookies y sesión para mantener la autentificación del usuario por un tiempo de tal manera que el usuario no deba introducir su contraseña a cada momento.

Este sistema denominado **cookies y sesión** se basa en un estándar que implementan todos los navegadores modernos llamado cookies. Una cookie es un pequeño archivo de información que se guarda en el navegador del usuario. Esta información la puede consultar el servidor a cada llamada del usuario por lo que si guardamos en la cookie un número aleatorio suficientemente largo que llamamos "cookie id" podremos considerar que el portador de la cookie es realmente quien dice ser su "cookie id".

Lo que hacemos nosotros es guardar en base de datos una id de usuario asociada a esa cookie id. Por lo tanto supondremos que el portador de una cookie id es el usuario con el id asociado a esa cookie id en nuestra base de datos.

Todo este diseño es bastante estándar, hay que tener en cuenta que la mayoría de páginas de internet cuentan con un sistema de autentificación por lo que tenía que ser ya un proceso bastante maduro.

Para la implementación utilizamos los componentes de Zend Framework: Zend_Auth y Zend_Session.

En cuanto a la autorización, hemos utilizado lo que se conoce como **ACL** (Access Control Lists) que es un concepto que podemos implementar utilizando Zend_Acl y que consiste en definir unas listas en las que se definen los permisos de cada tipo de usuario.

Nuestros tipos de usuarios son 4:

- ANONYMOUS. Un usuario sin autentificar.
- USER. Un usuario autentificado.
- WORKER. Un usuario autentificado que trabaja para Eventlayer.
- BOSS. Un administrador con todos los permisos en el sistema.

Un ejemplo de las ACLs que utilizamos sería el siguiente:

```
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'addvenuespopup');
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'setgroup');
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'like');
```

```
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'addartistspopup');
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'addtagspopup');
$this->deny(Enum_UserGroup::ANONYMOUS, 'default/event', 'adddiscussion');
```

En este ejemplo lo que se hace es que se deniega (deny) el acceso para el tipo de usuarios ANONYMOUS al módulo "default" es la parte de la web que ven los usuarios y para las acciones de addvenuepopup, stgroup, like… Es decir que un usuario no autentificado no podrá ni añadir un lugar al sistema, ni crear un grupo, ni hacer like… Realmente una manera muy fácil y visual de como denegar y otorgar privilegios a los usuarios de un sistema de información.

5.4.3. Gestión de contactos

En cualquier red social, Eventlayer lo es, existe lo que se conoce como la gestión de contactos. Con esto nos referimos a que el usuario puede tener una relación de "amistad" con otros usuarios de la red social que les permite realizar una serie de funcionalidades extra entre ellos que no pueden llevar a cabo con otros usuarios no "amigos" de la página.

Aquí hay 2 paradigmas que se pueden seguir:

1. **Followers y Following.** Es el ejemplo que sigue Twitter con todos sus usuarios y el que sigue Facebook con las páginas de marcas, artistas o personalidades públicas. Un usuario puede establecer una relación de contacto de manera unilateral, simplemente va al usuario y escoge hacer follow (o like en Facebook). A partir de ese momento se establece la relación de "amistad" en el sistema.

2. **Peticiones de amistad y contactos.** Es el ejemplo que sigue Facebook con sus usuarios. Consiste en que un usuario envía una petición de amistad a otro usuario y en este caso no se formaliza la relación de "amistad" hasta que el otro usuario acepta esa petición.

En Eventlayer utilizamos el segundo tipo y por ejemplo cuando se intenta acceder a la agenda (perfil) de un usuario con el que no existe una relación de amistad el sistema no nos lo permite:

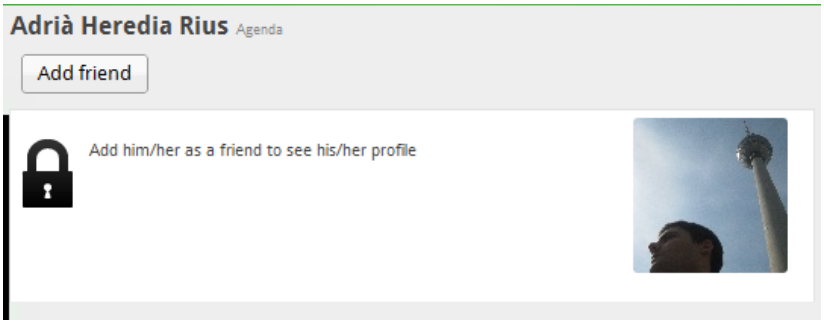


Ilustración 78 - Perfil de un usuario que no es nuestro contacto

Lo que debe hacer el usuario en este caso es clickar al botón de Add Friend y el propietario de ese perfil recibirá una petición de amistad que puede aceptar o eliminar:

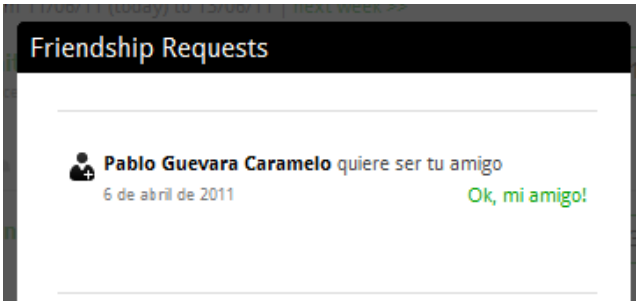


Ilustración 79 - Petición de amistad en Eventlayer

Una vez le enviamos una solicitud de amistad y nos la acepta ya podremos ver su agenda o perfil completo. Además podremos enviarle eventos para que asista con nosotros y recibir alertas sobre su actividad en nuestro muro.

Ilustración 80 - Perfil de un usuario que es nuestro amigo en Eventlayer

5.5. Notificaciones

Historias relacionadas

- **U-0110 –** USER quiere recibir notificaciones cuando ciertas acciones relacionadas con él sucedan para poder responder a esos sucesos.
- **U-0111 –** USER quiere cambiar la configuración de la aplicación para adaptarla a sus gustos (está parcialmente relacionada con este módulo ya que el usuario debe poder configurar la aplicación para no recibir más notificaciones vía email).

Las notificaciones en Eventlayer son mensajes que se muestran en el sistema para informar al usuario de diferentes acciones que se han producido en el sistema, que están relacionadas con él, y ofrecen ciertas opciones para que el usuario responda a esos sucesos ocurridos.

En el capítulo *2.2. Producto* se habla del muro. El muro es una sección de Eventlayer donde se muestran las acciones que han ido realizando nuestros amigos en la red social de Eventlayer ordenadas de manera cronológica con el propósito de que el usuario pueda seguir la actividad de su círculo de amigos en Eventlayer. No obstante, cuando hablamos de notificaciones nos referimos no a acciones que tienen que ver con nuestros amigos si no a sucesos que tienen que ver directamente con nosotros, por lo tanto, aclaramos, son cosas diferentes.

De cara al usuario las notificaciones reciben 3 nombres:

a) **Peticiones de amistad.** Cuando otro usuario quiere formar parte de tu círculo de amigos recibes este tipo de notificación. Las acciones que se le ofrecen al usuario son: aceptar su amistad en Eventlayer o borrar la notificación.

b) **Peticiones de asistencia.** Cuando otro usuario quiere asistir contigo a un evento recibes este tipo de notificación. Las opciones que se le ofrecen al usuario son: informar a tu amigo si vas a asistir seguro al evento, informar que a lo mejor asistirás al evento o informar que no tienes intención de asistir al evento.

c) **Notificaciones.** Este último tipo de notificaciones son variadas y se diferencian de los tipos a) y b) porque no necesitan acción de respuesta por parte del usuario. Por ejemplo se envían cuando alguien comenta en un evento en el que asistes o en un comentario que previamente hiciste.

En cuanto a las vistas de las notificaciones podemos ver un cambio bastante drástico entre lo que fue la versión 1 (imagen superior) y la versión 2 (imágenes inferiores):

Ilustración 81 - Vista de las notificaciones en la versión 1 (los tres iconos de esquina inferior izquierda)

Ilustración 82 - Vista de las notificaciones en la versión 2 (cuando no hay nuevas notificaciones)

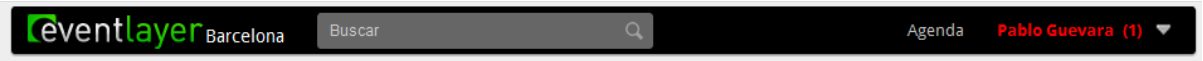


Ilustración 83 - Vista de las notificaciones en la versión 2 (cuando hay 1 nueva notificación)

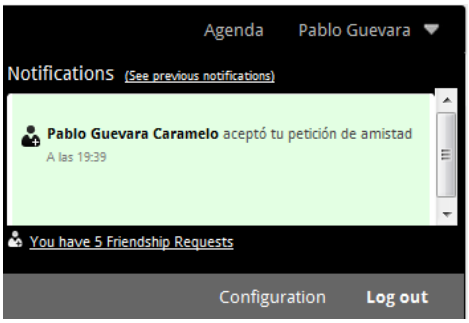


Ilustración 84 - Vista de las notificaciones en la versión 2 cuando se despliega el panel de notificaciones

Básicamente de lo que nos dimos cuenta al probar la aplicación con los usuarios era que no les quedaba claro que significaban los iconos de las notificaciones a primera vista. Además el hecho de tenerlos visibles incluso cuando no existían notificaciones que revisar provocaba ruido al resto de la aplicación.

Lo que decidimos hacer en la versión 2 fue integrar las notificaciones en un mismo panel y sólo destacarlas en el caso de que hubiera nuevas notificaciones (destacadas en color rojo llamativo).

5.5.1. Envío de notificaciones por email

Además de notificar al usuario por medio de nuestra aplicación también ofrecemos la posibilidad de notificarlo vía el envío de emails a su correo electrónico. Por defecto este servicio está activo pero el usuario puede desactivarlo desde el panel de configuraciones.

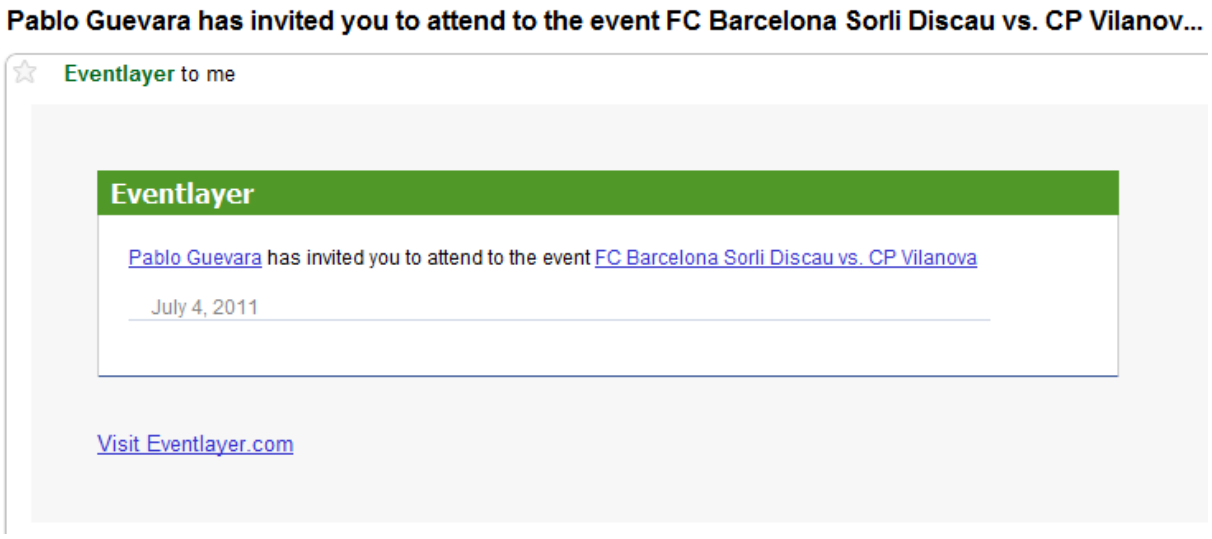


Ilustración 85 - Ejemplo de notificación por email en Eventlayer

Ilustración 86 - Opción para activar/desactivar las notificaciones por email en Eventlayer

5.5.2. Diseño del sistema de notificaciones (patrón observer)

De los 3 tipos de notificaciones, los tipos a) y b) se crean en determinadas circunstancias que se rigen por la lógica de negocio de nuestro producto. Esto significa que nosotros (los programadores de Eventlayer) decidimos en su momento que cuando un usuario pida ser amigo de otro se cree una notificación del tipo a) Peticiones de amistad. Esta norma queda implementada en el código de una manera directa. De la misma manera pasa con las peticiones b) Peticiones de asistencia.

No obstante, las peticiones del tipo c) son más complicadas. El usuario recibirá notificaciones sólo si está "vinculado" al suceso que acaba de ocurrir. Un usuario puede vincularse a un suceso de diferentes maneras. Por ejemplo si un usuario está asistiendo a un evento este usuario está "vinculado" a este evento, si por ejemplo un usuario hace un comentario en un evento éste quedará suscrito a las respuestas que hagan otros usuarios, etc.

Esto provocó que nos viéramos obligados a diseñar un sistema que notificase sólo a los usuarios vinculados y no a todos. Para hacerlo nos inspiramos en el patrón Observer.

Este patrón de diseño consiste en tener en nuestro sistema las siguientes 2 clases genéricas:

a) **Subject.** Esta es la clase que monitorizaremos, nos interesará detectar cualquier cambio en su estado o acción que se lleve a cabo sobre ella.

b) **Observer.** Esta clase es la clase que se "vincula" o utilizando la terminología de este patrón se "suscribe" a la clase de tipo subject para recibir notificaciones para algunos de sus cambios.

Aplicando este patrón a nuestro sistema de notificaciones, tendremos clases a) subject en eventos, opiniones, fotografías… Básicamente cualquier cosa que se pueda comentar. Por otro lado los usuarios de Eventlayer serán de tipo b) observer.

De esta manera, utilizando el patrón observer, cuando un a) subject (evento, opinión, fotografía…) sea comentado por un usuario, se notificará a todos los b) observers (usuarios de Eventlayer) que estén vinculados a ese subject.

A continuación un diagrama de clases del patrón observer que hemos utilizado:

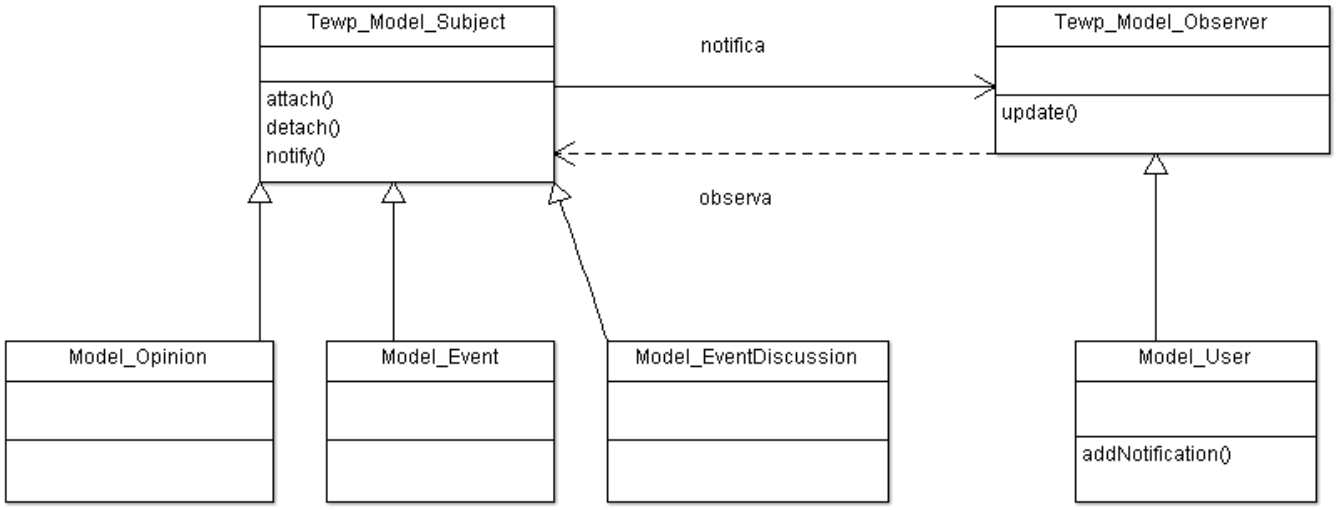


Ilustración 87 - Diagrama de clases UML del patrón Observer para notificaciones de Eventlayer

5.6. Gestión de la asistencia a eventos

Historias relacionadas

- **U-0107** – USER quiere decidir el día, el sitio donde cenar y quien lleva el coche para ir al concierto xxx con sus amigos.

Un aspecto diferenciador de Eventlayer son las funcionalidades que ofrece al usuario para organizar su asistencia a un evento. El propósito de estas funcionalidades es la de poner más fácil a nuestros usuarios la manera de quedar para asistir con sus amigos a este evento. Hemos bautizado a estas funcionalidades como "event widgets" (en el capítulo *1.2.Producto* se pueden ver las vistas de los widgets de los que hablamos a continuación). A continuación un breve resumen de los event widgets actuales:

- **Eventwidget-Decideday.** Este widget nos permite proponer una serie de fechas para el evento a nuestros amigos y cada uno de los asistentes indica qué fechas son las que le van bien y que fechas no.
- **Eventwidget-Poll.** Este widget nos permite proponer varias opciones, por ejemplo varios sitios donde cenar, y los asistentes pueden votar su opción preferida.
- **Eventwidget-Random.** Este widget nos permite proponer una tarea entre un rango de asistentes que el usuario puede seleccionar, a continuación el sistema escoge a alguien

aleatoriamente que será el encargado de ejecutar esa tarea. Por ejemplo para decidir quién conduce o quién compra las entradas al evento.

- **Eventwidget-Photos.** Este widget nos permite subir fotos al evento de manera pública (las verán todos los usuarios de Eventlayer) o privada (sólo quienes nosotros escojamos).

Cada uno de estos widgets contiene una lógica de negocio específica, es decir la programación de un código fuente que define una serie de acciones especiales que ejecuta ese widget. Por ejemplo, en el eventwidget-poll se tiene que llevar un control de la información relativa a los votos que efectúa cada usuario.

Por lo tanto debíamos diseñar un sistema que nos permitiera implementar estos widgets de una manera que los unos no dependieran de los otros pero a la vez aprovechasen todo el código posible para no tener que implementarlo para cada uno de los widgets. Es decir**, diseñar un sistema de plugins** que utilizasen un mismo sistema para almacenar los datos generados, en nuestro caso ese sistema sería el modelo Model_Eventdiscussion.

5.6.1. Diseño de la interfaz de los event widgets

Para empezar necesitábamos una interfaz común y un método de interactuar con los plugins que fuese igual para todos, de esta manera la curva de aprendizaje del usuario sería muy suave. El enfoque que decidimos utilizar fue que cada uno de estos event widgets se ejecutaría en 2 fases:

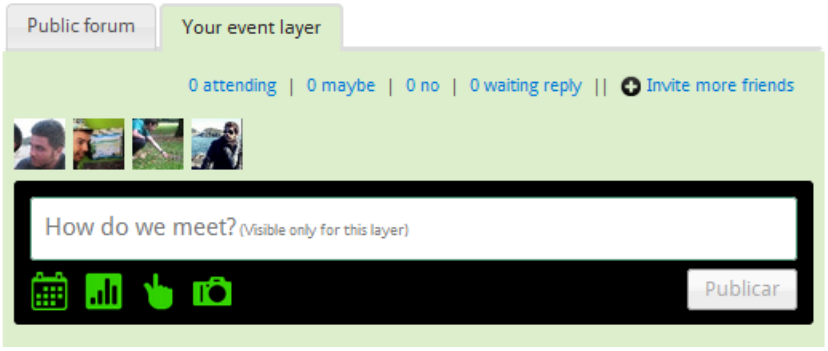


Ilustración 88 - Barra de event widgets de Eventlayer

a) **Fase A. Creación.** Uno de los usuarios se ocupa de crear el event widget, para hacerlo selecciona uno de los iconos de la barra de event widgets. A continuación se abrirá un popup que el usuario tendrá que rellenar con las opciones correspondientes y específicas para cada tipo de widget. A partir de ese momento se crea un event widget y se añade a la lista de comentarios del evento:

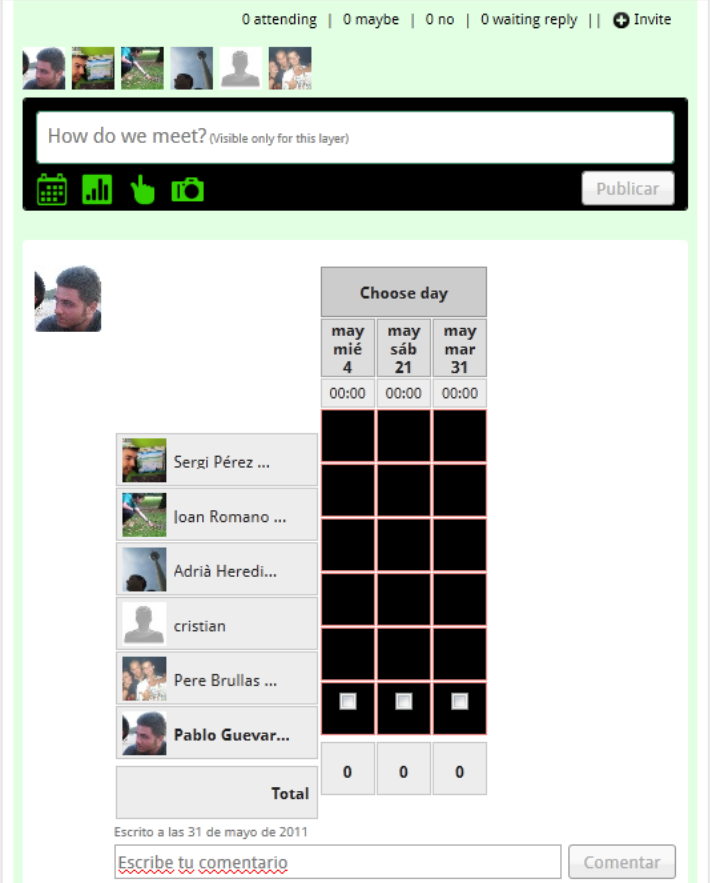


Ilustración 89 - Ejemplo de un Eventwidget-Decideday recién creado

b) Interacción. Cada uno de los usuarios que quieran interactuar con el widget creado pueden hacerlo desde el mismo listado de comentarios:

Ilustración 90 - Ejemplo de un fragmento de un Eventwidget-Decideday donde el usuario ha interactuado

5.6.2. Diseño del sistema de event widgets

Una vez definidas las vistas y las etapas que iban a tener los event widgets era el momento de establecer las partes principales del sistema que soportaría los event widgets que como hemos dicho debería tener las características de un sistema de plugins y utilizar el Model_Eventdiscussion como soporte para guardar la información específica de cada event widget.

El modelo Model_Eventdiscussion consta de los siguientes atributos:

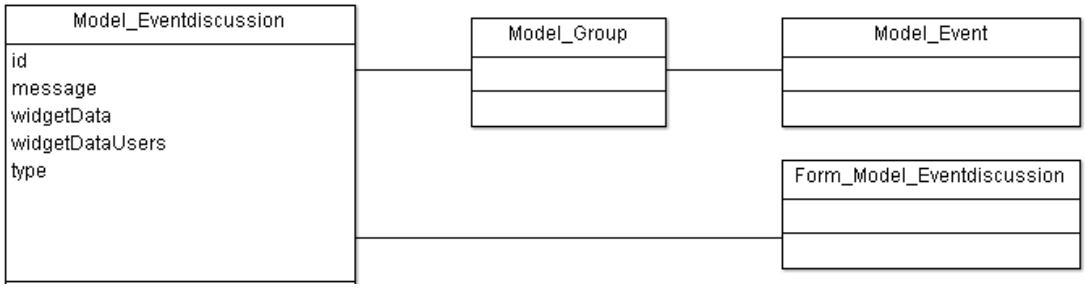


Ilustración 91 - Modelo donde se guarda toda la información de los event widgets

Del modelo representado en el diagrama de clases UML podemos destacar varios aspectos:

- La relación con "Model_Group" se debe a que dentro de cada evento podemos tener nuestro propio layer. Un layer como se puede ver en el capítulo *1.2.Producto* es un espacio privado, para ti y quien tu decidas, dentro de un evento (un concierto, obra de teatro…). El mismo usuario es quien decide que amigos estarán en ese layer y así poder chatear o usar los event widgets sólo con ellos. Es por eso que es necesario que los event widgets tengan una relación con Model_Group que es el modelo que implementa los layers.
- El atributo "type" nos indica de que tipo es el event widget. Es un valor numérico normal que nosotros internamente relacionamos con uno y sólo uno de los tipos de event widgets.
- El atributo "widgetData" es un campo donde se guarda la información personalizada de ese event widget en cuestión. Por ejemplo, si es un Eventwidget-Poll se guardarán aquí las opciones de la encuesta que se ha publicado.
- El atributo "widgetDataUsers" es un campo donde se guarda la información personalizada de ese widget fruto de la interacción con los usuarios. Por ejemplo, si es un Eventwidget-Poll se guardarán aquí las votaciones de cada uno de los usuarios que han votado en este widget.

Una vez definidas la interfaz y el modelo de almacenamiento era hora de utilizar el **patrón Adapter** para completar nuestro sistema de plugins. El patrón Adapter es uno de los patrones de diseño más utilizados, también se le conoce como polimorfismo que es una propiedad básica en cualquier lenguaje de programación orientado a objetos.

Este patrón consiste en definir una interfaz común para todos los plugins, en nuestro caso para todos los widgets, y de esta manera poder ejecutar los mismos métodos para todos con resultados diferentes dependiendo el event widget seleccionado.

Para el lector no especializado un ejemplo muy visual es imaginarnos que tenemos N bombillas que emiten una luz diferente. Si todas las bombillas utilizasen el mismo voltaje podríamos utilizar una sola batería común para hacerlas funcionar a todas.

En nuestro caso hemos creado una serie de clases llamadas Comp_Eventwidget_XXX (en lenguaje PHP) donde XXX se corresponde con el tipo de widget. Cada una de estas clases sería lo que en el patrón Adapter se denomina **Adaptee**. Estas clases definen 2 métodos generales: render() y interact(). El primer método se llama para crear el event widget (fase A) y el segundo para interactuar con él (fase B).

Además de estas clases existen sus homólogas en cliente (en JavaScript) que se denominan Jquery.Tewp_Component_Eventwidget_XXX donde XXX se corresponde con el tipo de widget. Estas clases tienen los mismos métodos y son necesarias ya que los event widgets tienen acciones que se realizan desde el navegador. Por ejemplo en un Eventwidget-Poll cuando el usuario selecciona una de las opciones para botarla el sistema JavaScript debe actualizar el gráfico de barras que se muestra al usuario para que refleje la acción que se acaba de llevar a cabo.

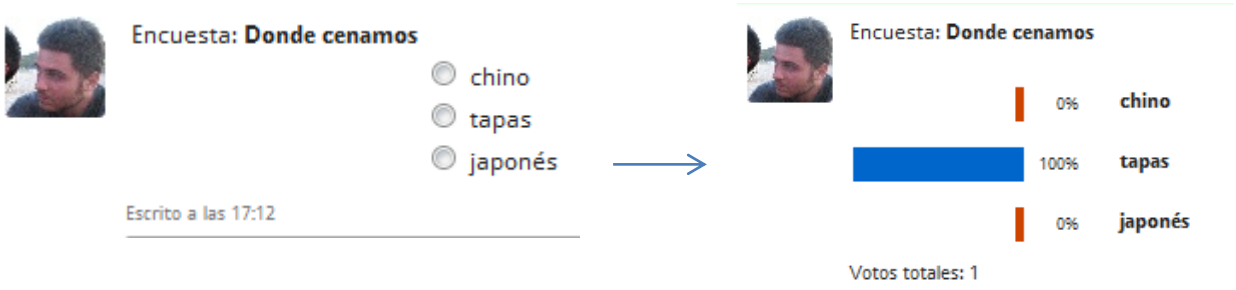


Ilustración 92 - Ejemplo de interacción del usuario con un event widget que hace necesario el uso de JavaScript en los event widgets

Las clases Adaptee de las que hemos hablado heredan de una clase común llamada Tewp_Eventwidget. Esta clase tiene la responsabilidad, además de ser una interfaz común como se indica en el patrón Adapter, se encarga de realizar tareas comunes a todos los tipos de event widgets.

Estas tareas comunes son tales como la interacción con el modelo Model_Eventdiscussion para el almacenamiento de la información y también la creación de las notificaciones oportunas para informar al resto de usuarios interesados de que se ha añadido un nuevo event widget (ver capítulo 5.5.Notificaciones para ver ampliado el asunto de las notificaciones).

5.7. Integración con Facebook

Historias relacionadas

- **U-0104** – USER quiere importar sus amigos de Facebook para tenerlos como contactos en la aplicación.

- **U-0101, U-0102** – Historias relacionadas con el login y el registro que también se pueden llevar a cabo utilizando la integración de Facebook.

- **U-0110 –** USER quiere recibir notificaciones cuando ciertas acciones relacionadas con él sucedan para poder responder a esos sucesos (está parcialmente relacionada ya que trataremos de sincronizar las notificaciones de Eventlayer con las de Facebook)

La integración con Facebook es uno de los aspectos más interesantes de Eventlayer. Esto es así por la forma tan característica que tienen los datos del sistema de Facebook, estos datos tienen forma de grafo [42] . De hecho, su API (Application Public Interface) recibe el nombre de Open Graph (Grafo Abierto). Para aclarar, una API, Open Graph lo es, vendría a ser la librería pública con la que un sistema permite que otros se integren con él.

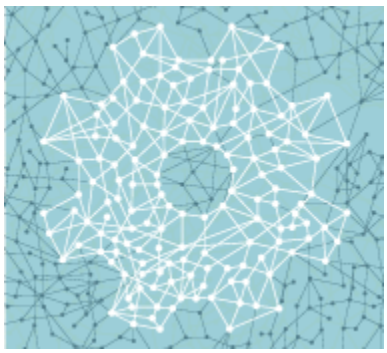


Ilustración 93 - Logo del Open Graph the Facebook

El motivo por el que decimos que los datos tienen forma de grafo es porque toda la información de Facebook está fuertemente conectada entre sí misma y entre varios usuarios a la vez.

Los puntos representan información diversa del usuario, lo que ocurre es que toda esta información está ligada a muchos otros puntos: se relaciona entre ella misma, entre la información y el usuario propietario e incluso entre la información y otros usuarios "amigos" del usuario creador.

Una fuente de información tan conectada nos permite sacar muchas conclusiones de cada usuario de tal manera que un sistema puede llegar a "conocer" a un usuario por medio de toda esa cantidad de información y además, gracias a la hiper-conexión de esta información, también podemos "conocer" a sus amigos. Esto ligado a que Facebook cuenta con información personal de casi 600 millones de usuarios que es el número oficial de usuarios registrados con los que cuenta Facebook en 2011 hacen que un sistema como Eventlayer pueda sacar buen provecho de todo esto.

Por supuesto cada uno de estos usuarios nos tiene que dar su consentimiento para tener acceso a su información, no obstante, el usuario normalmente no pone ningún impedimento ya que le supone a él mismo un gran ahorro de tiempo el no tener que introducir su información nuevamente en otros sistemas que no sean Facebook.

Por ejemplo, cada usuario de Facebook tiene un número N de amigos oficiales en esta red social donde N converge a +150 de media. La integración con Facebook nos permite tener acceso a sus nombres, fotos e identificadores y nos permite informar al usuario qué amigos de Facebook también están

[42] Nos referimos a un grafo matemático. Informalmente, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto.

registrados en Eventlayer y de esta manera aconsejarles que formalicen su relación de amistad en Eventlayer con un solo click.

5.7.1. Selección de los puntos de integración

El ecosistema que ha creado Facebook para que otros programadores creen aplicaciones que se integren con él es muy amplio y desde el inicio de la construcción de Eventlayer la API que ofrece Facebook ha cambiado por completo pasando de la versión 1 a la renovada versión 2, llamada Open Graph.

Es por eso que lo primero que tuvimos que plantearnos era, de todas las posibilidades de integración que ofrece Open Graph escoger las que más fueran a aportar a Eventlayer. A continuación enumeramos todas las posibilidades que ofrece Facebook (en negrita las que hemos utilizado):

- **Social Plugins.** Son los típicos componentes que podemos ver en la mayoría de webs como son el botón de Like, el componente de comentarios… Son componentes completamente diseñados por Facebook, muy poco personalizables pero muy fáciles de instalar. Ejecutan acciones muy simples que efectúan acciones sobre el perfil de usuario de Facebook pero no permiten importar su información.
- **Dialogs.** Son pantallas estándar de Facebook para realizar algunas acciones interesantes. Son más personalizables que los Social Plugins y se distinguen porque son pantallas completas y no simplemente componentes. Ejemplos de Dialogs son la pantalla para autorizar permisos de Facebook a nuestra aplicación o la pantalla para enviar invitaciones a tus amigos de Facebook.
- **Open Graph API.** Es la nueva API de Facebook. Se basa en 2 protocolos estándar: REST y OAuth[43] y puede ser accedida tanto por medio de JavaScript (Facebook JavaScript SDK), por medio de PHP (Facebook PHP SDK) o por medio de IOS (IPhone) y Android SDK. Se da el caso que nosotros utilizamos todos los SDKs, tanto JavaScript, PHP, IOS y Android. Un SDK (Software Developer Kit) es simplemente una implementación de una API para un lenguaje o plataforma determinada. El Open Graph API permite interactuar con Facebook casi como si se estuviera haciendo desde la propia web, por ejemplo se pueden subir fotos, vídeos, o comentar sin tener que utilizar la web oficial de Facebook. Por lo tanto nos permite tanto ejecutar acciones sobre la cuenta de Facebook del usuario como de importar su información para ser utilizada en Eventlayer.
- FQL**.** Facebook Query Language, permite una sintaxis al estilo SQL que es el lenguaje de búsqueda de información en las bases de datos estándar, para conseguir cualquier información de un usuario del que se tenga permisos.
- Chat API, Credits API. Son APIs específicas para acceder al chat de Facebook y a su nuevo método de pago (Facebook Credits)

[43] REST es el mismo protocolo que utilizamos nosotros en nuestra API de Eventlayer (ver capítulo 5.1.1.2. SOA) y OAuth es un protocolo de autentificación muy utilizado en internet.

- Legacy API. Es la API antigua de Facebook. La nueva API, Open Graph no tiene implementados todos los métodos que tenía la antigua todavía, estamos hablando de que cuando se redactó este PFC era Junio del 2011, por lo tanto hay algunos métodos que sólo están en la Legacy API.

Una vez nos documentamos exhaustivamente de todas las posibilidades que ofrecía el ecosistema de Facebook fue el momento de plantearnos qué era lo que le podía aportar a Eventlayer esa información que nos facilitaban. Decidimos trabajar en 3 puntos:

1. **Login.** Dar la posibilidad al usuario que pudiese autentificarse en Eventlayer utilizando su mismo email y contraseña que en Facebook.
2. **Importación de amigos.** Dar la posibilidad al usuario de que pudiese importar sus amigos de Facebook a Eventlayer.
3. **Sincronización de notificaciones.** Dar la posibilidad al usuario de que pudiese consultar las notificaciones de Eventlayer desde su propio Facebook.

5.7.2. Implementación del Login con Facebook

El objetivo que nos propusimos es que el usuario pudiese tanto logearse como registrarse en Eventlayer en un simple click. Además, un aspecto interesante era el hecho de importar algunos datos de su cuenta de Facebook como la foto de perfil, su nombre y su email.

En teoría no era un aspecto complicado, ya que es una historia de usuario por defecto en cualquier aplicación de Facebook y por lo tanto hay ya bastante documentación al respecto.

No obstante fue un proceso largo ya que tuvimos que implementar el login para diferentes casos (cada caso requiere utilizar un SDK diferente): el caso estándar de la web (utilizamos el SDK de JavaScript), para el caso en que el usuario accede a Eventlayer desde la aplicación de Eventlayer en Facebook (utilizamos el SDK de PHP) y también para el caso en el que se accede desde las aplicaciones de Eventlayer para los móviles (utilizamos el SDK de IPhone y de Android).

Los objetivos de cualquiera de estos logins son 3:

a) Conseguir el auth_token del usuario y su identificador de Facebook.

Como hemos dicho antes, Open Graph se basa en OAuth. Este protocolo consiste en que para cada usuario se genera un auth_token y todas las llamadas que el usuario efectúe deben llevar ese auth_token que será el que las autentifique como pertenecientes al usuario en cuestión. El token suele ser de 32 dígitos y las llamadas son del estilo: "https://graph.facebook.com/me?access_token=...".

Por lo tanto una de las tareas durante el login es conseguir ese token y guardarlo para poder continuar haciendo llamadas a Facebook en nombre de ese usuario, por ejemplo para importar sus amigos de Facebook.

En cuanto al identificador de Facebook es un identificador numérico que nos permite hacer referencia a un usuario en Facebook. Cada elemento del Open Graph tiene un identificador, desde los usuarios, hasta los vídeos o las fotos tienen un identificador para referirnos a ellos. Por lo tanto también deberemos guardarnos el identificador del usuario para hacer referencia a él.

b) Importar la información que nos interesa desde su cuenta de Facebook.

En el proceso de login importamos de la cuenta de Facebook del usuario su email, su nombre y apellidos y su foto de perfil. Para hacerlo utilizamos la SDK de PHP con la siguiente llamada al servidor de Facebook (se puede observar que utilizamos el access_token en la llamada):

```
$api_call = array(
    'method' => 'users.getinfo',
    'fields'            =>            'name,uid,first_name,middle_name,last_name,email,
pic_big,birthday,sex,pic_square' );

$api_call['access_token'] = $accessToken;

$users_getinfo = $this->getClient()->api($api_call);
```

c) Conseguir los permisos necesarios.

Con el auth_token podemos realizar llamadas en nombre del usuario, no obstante, el usuario nos tiene que aceptar una serie de permisos, nosotros elegimos cuáles le vamos a pedir en función de las llamadas que tengamos previsto hacer en su nombre.

Concretamente pedimos la autorización de poder ver la información común del usuario, de poder obtener su dirección de email, de poder publicar en su muro y sobretodo el acceso offline que nos permitirá realizar estas llamadas sin que el usuario esté conectado a Eventlayer en ese mismo momento.

Utilizamos la siguiente llamada por medio del SDK de PHP para que el propio Facebook le pida al usuario la autorización de los permisos que necesita Eventlayer:

```
$perms = 'email,publish_stream,offline_access'; // aquí escogemos que permisos pedir

$call[perms]=$perms;

return $this->getClient()->getLoginUrl($call);
```

Este código genera una pantalla como la siguiente en la que se le pregunta al usuario si está de acuerdo en otorgar los siguientes privilegios a la aplicación Eventlayer app:

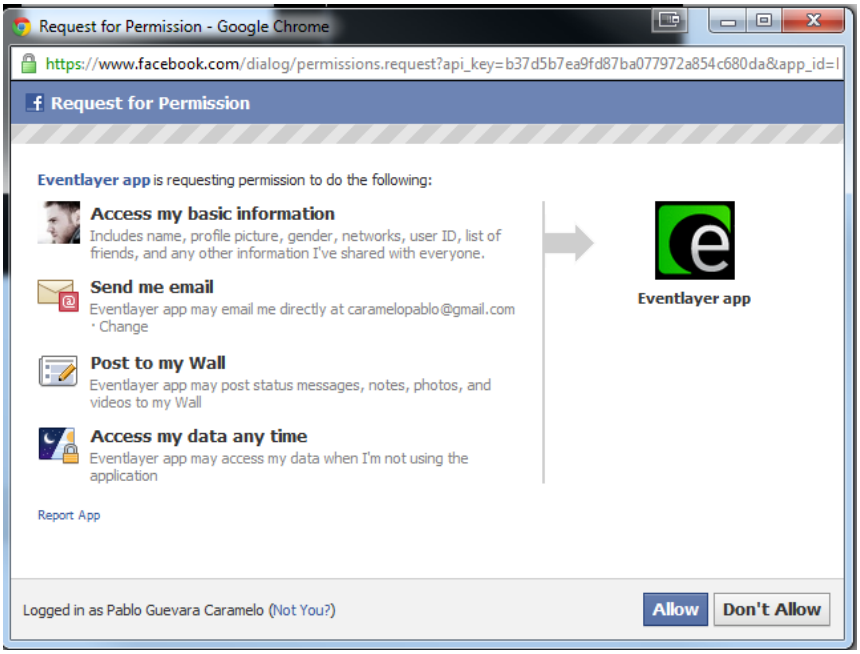


Ilustración 94 - Diálogo de petición de permisos en Facebook

5.7.3. Implementación de la importación de amigos

El objetivo de este punto es conseguir que los nuevos usuarios de Eventlayer puedan importar sus contactos desde Facebook de la manera más fácil posible.

Suelen utilizarse 2 técnicas diferentes, a continuación a) y b) que pueden ser complementarias, nosotros hemos creído conveniente implementar las 2.

a) Permitir al usuario enviar invitaciones a sus amigos de Facebook.

Esta historia la hemos implementado utilizando uno de los Dialogs de Facebook. Los dialogs son muy fáciles de utilizar. Simplemente utilizando esta línea de código en el SDK de JavaScript obtenemos el dialog (la pantalla) que el usuario puede utilizar para enviar invitaciones de Eventlayer a sus amigos de Facebook

```
FB.ui({method: 'apprequests', to: '<id de la aplicación de Eventlayer>',
message: <mensaje que recibirá el destinatario>, data: <datos que nos
puedan ser útiles a nosotros en la programación de la invitación>});
```

El diálogo resultante es algo como lo siguiente. El usuario simplemente escoge a qué amigos quiere enviarles una invitación y cada uno de ellos recibirá una notificación de Facebook informándole de esta invitación:

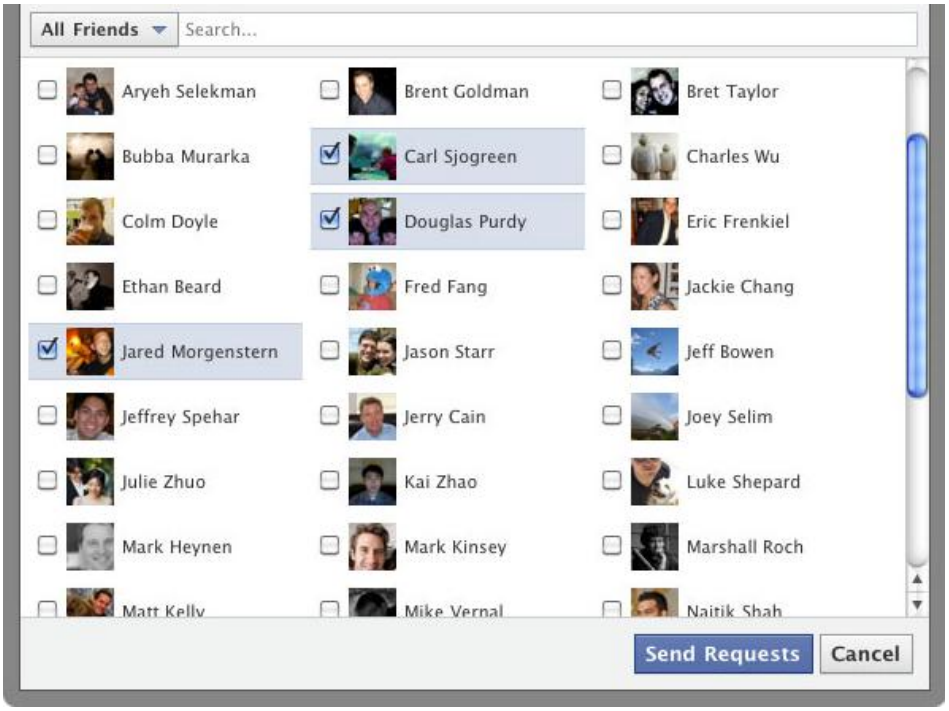


Ilustración 95 - Pantalla que puede utilizar el usuario para mandar invitaciones a sus amigos de Facebook para utilizar Eventlayer

b) Detectar qué amigos del usuario están ya registrados en Eventlayer e informarle de que puede hacerse amigo de ellos.

Esta historia es bastante más complicada que a) ya que no existe todavía ningún estándar para hacerlo.

La idea es utilizar la SDK de PHP con la API de Open Graph para mediante una llamada conseguir los ids de Facebook de los amigos del usuario actual:

```
$facebookIds = function () use ($that) {
                return $that->getClient()->api('/me/friends');

            };
```

A continuación se hace una búsqueda entre los usuarios registrados de Eventlayer con el fin de encontrar algún usuario con el id de Facebook de alguno de los amigos del usuario.

```
foreach ($facebookIds as $id) {
            $select->orWhere('t.facebook_identifier =' . $id);
        }

$resultSet = $this->_getDao()->fetchAll($select);
```

Los usuarios encontrados se agrupan entre los que ya son amigos en Eventlayer del usuario y los que no. Por último se muestran en una pantalla propia de Eventlayer donde el usuario puede decidir de qué usuarios quiere ser amigo a partir de ese momento.

Un ejemplo de la pantalla resultante sería la siguiente:

Ilustración 96 - Pantalla en Eventlayer que el usuario puede utilizar para importar sus amigos de Facebook a Eventlayer

En color gris aparecen los usuarios de los que ya eres amigo y en verde los usuarios de los que eres amigo en Facebook pero no en Eventlayer.

A partir de este momento se sigue la ejecución normal para establecer relaciones de amistad en Eventlayer definida en el capítulo 5.4.3. Gestión de contactos

5.7.4. Implementación de la sincronización de notificaciones

En el capítulo de *5.5 Notificaciones* ya hemos hablado de que las notificaciones de Eventlayer además de mostrarse en la barra superior de la web de Eventlayer también se envían por email.

Lo que nos pareció importante es que además de estos 2 sistemas (web y email) se pudieran consultar las notificaciones desde la pantalla principal de Facebook. Decidimos que este aspecto era importante ya que cada vez más gente padece de tener un email tan saturado que prácticamente no lo miran.

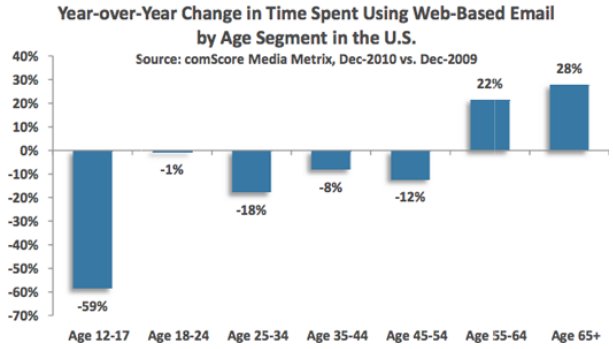
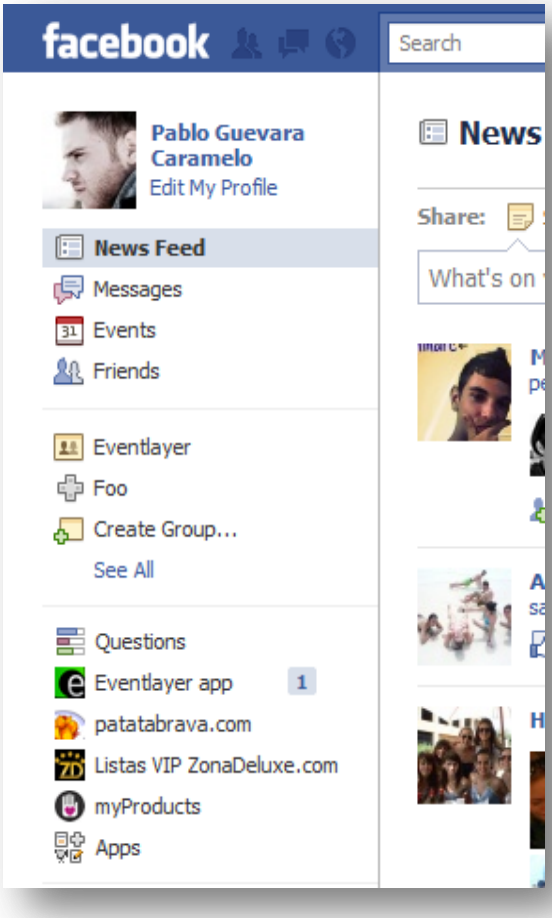


Ilustración 97 - Gráfico de Comscore de un estudio del uso del email en 2010

En el gráfico anterior se puede ver como precisamente en el segmento entre 25 y 34 años que es básicamente el segmento en el que se encuentran los usuarios potenciales de Eventlayer (ver capítuloAnexo I - Plan de Marketing - Segmentos) es el segundo que ha visto como se ha reducido más el uso del email.



Es por eso que muchos de nuestros usuarios apreciarán el poder tener sus notificaciones de Eventlayer centralizadas en Facebook.

A continuación se puede ver una captura de la pantalla principal de un usuario de Facebook en la que se muestra como Eventlayer le está informando de que existe una notificación pendiente de lectura.

El usuario sólo deberá pulsar en ese icono e inmediatamente podrá ver las notificaciones pendientes que tiene en Eventlayer.

Para implementar algo así lo que hicimos fue integrar nuestro sistema de notificaciones que ya hemos explicado en el capítulo de 5.6.4.3 Notificaciones con el SDK de PHP. Había 2 acciones a integrar:

a) Cuando se crea una nueva notificación en Eventlayer.

En este caso hay que añadir una nueva notificación en Facebook. Para hacerlo se utiliza el siguiente código en el SDK de PHP (donde $user_id y $access_token son el id de Facebook y el access token que hemos guardado del usuario en cuestión durante el registro):

```
$apprequest_url ="https://graph.facebook.com/" .
    $user_id .
    "/apprequests?message='INSERT_UT8_STRING_MSG'" .
    "&data='INSERT_STRING_DATA'&"  .
    $access_token . "&method=post";
```

```
$result = file_get_contents($apprequest_url);
```

Como podemos ver es simplemente una llamada estándar al servidor de Facebook por lo que no tuvo más complicación que mirarnos la documentación que se ofrece en la web Facebook for Developers[44].

b) Cuando el usuario lee las nuevas notificaciones.

Cuando el usuario lee las notificaciones de Eventlayer se debe actualizar su Facebook para que le deje de informar que tiene nuevas notificaciones pendientes.

Para ello utilizaremos la llamada por defecto del SDK de PHP de Facebook para borrar las notificaciones y así tener sincronizadas las notificaciones de las dos páginas.

```
$data = json_decode($requests);
foreach($data->data as $item) {
  $id = $item->id;
  $delete_url = "https://graph.facebook.com/" .
    $id . "?" . $access_token . "&method=delete";

  $result = file_get_contents($delete_url);
```

5.7.5. Comentarios finales sobre la integración con Facebook

El ecosistema que ha creado Facebook para que los programadores creen aplicaciones que se integran con su web es realmente impresionante. Algunos de los puntos que me gustaría destacar de la experiencia de haber trabajado con su API son:

- Se basa en estándares públicos y muy innovadores como OAuth y REST, algo que ya hemos comentado en la introducción de este capítulo.
- La API es inmensa y permite la creación de aplicaciones casi tan integradas como las suyas oficiales.
- El sistema de permisos es realmente sencillo tanto para el programador y como para el usuario.
- Existen SDKs para casi cualquier plataforma (PHP, JavaScript, IPhone, Android…)
- La documentación está bastante conseguida. No obstante lo que más me ha gustado es su Roadmap público, un documento en el que detallan las fechas en las que estarán las nuevas mejoras y las nuevas ampliaciones de su API. En Eventlayer empezamos a utilizar la API de Facebook justo en el momento en el que empezaron a migrar de su versión inicial a la Open Graph y gracias al Roadmap sabíamos cuando íbamos a poder implementar cada nueva funcionalidad por adelantado

[44] http://developers.facebook.com/

6. Pruebas

Las pruebas de software es un elemento que debe garantizar la calidad del software ya que gracias a ellas comprobaremos que el diseño y la implementación cumplen con la especificación (los requisitos iniciales).

Hay muchas maneras de realizar las pruebas para un sistema de software. Suelen diferenciarse por dos factores:

- **El actor que las realice.** Llamamos actor al ente que se ocupa de validar los requisitos contra el sistema software. Este puede ser bien un autómata (otro sistema de software) o una persona.

- **La etapa en la que se realizan.** Las pruebas pueden realizarse en varias etapas de elaboración de un software. Por ejemplo pueden realizarse durante la implementación de cada componente (pruebas unitarias) o una vez que el sistema, o una de sus partes globales, está acabada (pruebas de integración).

En nuestro caso intentamos utilizar en primera instancia las pruebas unitarias ya que son un sistema más moderno y que a largo plazo promete una mayor fiabilidad. Instalamos varios aplicativos para gestionar este método como PHPUndercontrol (para código PHP) y TestSwarm (para código Javascript). No obstante nos dimos por vencidos ya que requieren de una configuración que no es trivial y de un compromiso de tiempo bastante alto.

Es por eso que decidimos basar nuestra fase de pruebas en pruebas de integración ejecutadas por actores humanos, nosotros mismos.

De esta manera hemos creado una serie de checklists a partir de los requisitos (historias de usuario) principales que consisten en una serie de acciones a ejecutar sobre el sistema y que verifican cada uno de estos requisitos.

La clave está en nuestra perspicacia a la hora de imaginar todas las pruebas posibles y relevantes para asegurarnos que nuestro sistema cumple con sus objetivos.

Siguiendo el estilo del resto de capítulos comunes en las memorias a continuación presentamos una agrupación conjunta de todas las pruebas de integración que hemos visto oportunas tanto para los requisitos de la parte web: Pablo y Sergi, como para los requisitos de la parte móvil: Adrià.

6.1. Pruebas de la aplicación web

En este apartado se incluyen las pruebas de todas las historias de usuario de la aplicación web, tanto las de Pablo como las de Sergi.

Identificador	Requisito	Prueba	Comprobar que	Resultado	Responsable
U-0001-1	U-0001	Elegir una fecha de inicio y de finalización	Los eventos que aparecen corresponden a estas fechas		Pablo
U-0001-2	U-0001	Introducir la fecha en formato español dd/mm/aaaa o inglés mm/dd/aaaa	El sistema diferencia las fechas introducidas según el lenguaje elegido en el interfaz		Pablo
U-0002-1	U-0002	Seleccionar una categoría	Los eventos que aparecen corresponden a la categoría		Pablo
U-0003-1	U-0003	Abrir el Top10 de lugares	Los lugares son los más relevantes del sistema		Pablo
U-0004-1	U-0004	Abrir el apartado “descuentos”	Los descuentos se muestran correctamente		Pablo
U-0005-1	U-0005	Buscar un evento determinado	El evento aparece en el buscador	Funcionamiento correcto.	Sergi
U-0005-2	U-0005	Buscar un artista determinado	El artista aparece en el buscador	Funcionamiento correcto.	Sergi
U-0005-3	U-0005	Buscar un lugar determinado	El lugar aparece en el buscador	Funcionamiento correcto.	Sergi
U-0005-4	U-0005	Buscar un contacto determinado	El contacto aparece en el buscador	Funcionamiento correcto.	Sergi
U-0006-1	U-0006	Abrir el calendario	El calendario se muestra		Pablo

			correctamente		
U-0007-1	U-0007	Añadir y quitar un evento del calendario	El evento se añade y quita correctamente		Pablo
U-0008-1	U-0008	Subscribirse a un lugar o artista	El lugar o artista se añade a las subscripciones correctamente	Funcionamiento correcto.	Sergi
U-0008-2	U-0008	Añadir un nuevo evento al artista o lugar subscrito	El usuario recibe una alerta del nuevo evento		Pablo
U-0009-1	U-0009	Añadir una opinión en un perfil	La opinión se añade y muestra correctamente		Pablo
U-0010-1	U-0010	Añadir un comentario en un evento	El comentario se añade y muestra correctamente		Pablo
U-0011-1	U-0011	Destacar un evento, lugar o descuento	El ítem destaca respecto al resto de ítems corrientes		Pablo
U-0011-2	U-0011	Ocultar un evento, lugar o descuento	El ítem desaparece de la lista		Pablo
U-0012-1	U-0012	Creamos un nuevo evento en el sistema	El evento se crea con la información proporcionada		Pablo
U-0012-2	U-0012	Modificamos los distintos campos de un evento del sistema	El evento se modifica con la información proporcionada		Pablo
U-0012-3	U-0012	Eliminamos un evento existente	El evento deja de ser accesible y se elimina toda la información asociada		Pablo
U-0013-1	U-0013	Creamos un nuevo perfil en el sistema	El perfil se crea con la información proporcionada		Pablo

U-0013-2	U-0013	Modificamos los distintos campos de un perfil del sistema	El perfil se modifica con la información proporcionada		Pablo
U-0013-3	U-0013	Eliminamos un perfil existente	El perfil deja de ser accesible y se elimina toda la información asociada		Pablo
U-0014-1	U-0014	Añadimos y eliminamos un tag de un evento	El tag aparece y desaparece correctamente		Sergi
U-0015-1	U-0015	Abrir la sección de eventos recomendados	Los eventos son relevantes según el historial del usuario	Funcionamiento correcto.	Sergi
U-0016-1	U-0016	Abrir la cartelera y las vistas de una película	La cartelera está actualizada y la información es correcta.	Funcionamiento correcto.	Sergi
U-0101-1	U-0101	Registrar-se en el sistema	El usuario y su perfil se crean correctamente		Pablo
U-0102-1	U-0102	Loguear-se en el sistema	El usuario entra correctamente si utiliza su email y contraseña correcta y de otro modo se le prohíbe el acceso		Pablo
U-0103-1	U-0103	Desloguear-se en el sistema	El usuario sale de su cuenta y actúa como anónimo		Pablo
U-0104-1	U-0104	Añadir y eliminar un contacto en el sistema	El usuario elegido pasa a ser contacto correctamente y luego deja de serlo		Pablo
U-0105-1	U-0105	Importar contactos de Facebook	Se detectan y añaden		Pablo

			los contactos correctamente		
U-0106-1	U-0106	Creamos un evento entre amigos	El evento se crea correctamente y está disponible sólo para los amigos del usuario		Pablo
U-0107-1	U-0107	Se crearán y probarán los diversos widgets de un evento	Los widgets se añaden y comportan correctamente		Pablo
U-0108-1	U-0108	Realización de acciones diversas en la aplicación (añadir opinión, confirmar asistencia a un evento, añadir subscripción a perfil)	Aparece un nuevo mensaje en el muro de los contactos del usuario	Al eliminar la subscripción de un perfil (artista o lugar) el mensaje de muro no desaparece. Se ha solucionado.	Sergi
U-0109-1	U-0109	Realización de acciones diversas en la aplicación (añadir opinión, confirmar asistencia a un evento, añadir subscripción a perfil)	Aparece un nuevo mensaje en el perfil del contacto		Pablo
U-0110-1	U-0110	Realización de acciones que deben crear una nueva notificación	Aparece una nueva notificación en la cuenta de los usuarios		Pablo
U-0111-1	U-0111	Modificación de la configuración de la cuenta de un usuario	Los cambios se guardan correctamente	Funcionamiento correcto.	Sergi
U-0112-1	U-0112	Cambiar el idioma del interfaz	El interfaz se muestra en el idioma elegido	Funcionamiento correcto.	Sergi
U-0113-1	U-0113	Forzar un error en el servidor	El usuario recibe una notificación y puede seguir interactuando con la aplicación		Pablo

U-0113-2	U-0113	Forzar un error en el cliente	El usuario recibe una notificación y puede seguir interactuando con la aplicación		Pablo
U-0114-1	U-0114	Modificación de la asistencia en un evento	Se guardan los cambios correctamente		Pablo
U-0201-1	U-0201	Abrir el mashup e importar datos de diversas fuentes	Los datos se importan correctamente	Falla una fuente del mashup que ha modificado su interfaz. Se ha solucionado.	Sergi
U-0202-1	U-0202	Forzar un error en el servidor	Se guardan los datos del error para su revisión		Pablo
U-0202-2	U-0202	Forzar un error en el cliente	Se guardan los datos del error para su revisión		Pablo
U-0203-1	U-0203	Borrar un comentario y opinión usando un usuario con permisos de administración	Se pueden borrar correctamente	Funcionamiento correcto.	Sergi
U-0204-1	U-0204	Abrir las sección de estadísticas de un evento y perfil	Las estadísticas se muestran y actualizan correctamente	Funcionamiento correcto.	Sergi
T-0001	T-0001	Ejecución del script de actualización de código	El código se actualiza y el servidor se reinicia sin problemas	Funcionamiento correcto.	Sergi

7.

6.2. Pruebas de la aplicación móvil

Estas pruebas se llevarán a cabo tanto en la plataforma Android como en la plataforma iPhone:

Identificador	Requisito	Prueba	Comprobar que	Resultado	Responsable
U-1001-1	U-1001	Cargar lista de eventos al abrir aplicación	Aparece una lista con los eventos del día	Los eventos aparecen correctamente. Son del día actual.	Adrià
U-1002-1	U-1002	Elegir una fecha	Los eventos que aparecen corresponden a estas fechas	Los eventos mostrados pertenecen a la fecha escogida	Adrià
U-1003-1	U-1003	Seleccionar una categoría	Los eventos que aparecen corresponden a esta categoría	Los eventos mostrados pertenecen a la categoría	Adrià
U-1101-1	U-1101	Loguearse en el sistema	El usuario entra correctamente si utiliza su email y contraseña correcta y de otro modo se le prohíbe el acceso	El usuario puede loguearse correctamente	Adrià
U-1102-1	U-1102	Loguearse en el sistema vía facebook	El usuario entra correctamente utilizando sus credenciales de facebook	El usuario puede loguearse correctamente	Adrià
U-1103-1	U-1103	Desloguearse en el sistema	El usuario sale de su cuenta y actúa como anónimo	El usuario se desloguea y actúa como anónimo satisfactoriamente	Adrià
U-1004-1	U-1004	Acceder al perfil de un evento desde la lista inicial	Aparece correctamente el perfil del evento seleccionado	Se abre correctamente. El sistema reacciona un poco lento al abrir el 1er perfil	
U-1004-2	U-1004	Seleccionar muro público o privado	Los comentarios que aparecen corresponden a su muro	Los comentarios que aparecen pertencen al muro seleccionado. Los caracteres raros no se procesan bien.	Adrià
U-1005-1	U-1005	Añadir un comentario en un evento	El comentario se añade y muestra correctamente en el muro seleccionado	El comentario se añade al muro seleccionado. Los caracteres raros no se procesan	Adrià

				bien.	
U-1006-1	U-1006	Me gusta / No me gusta un evento	Se suma 1 al número likes o dislikes del evento y se destaca visualmente	Se suma 1 correctamente al likes o dislikes según hayamos clicado. No se destaca la suficiente para el usuario.	Adrià
U-1009-1	U-1009	Me gusta / No me gusta un perfil	Se suma 1 al número likes o dislikes del perfil y se destaca visualmente	Se suma 1 correctamente al likes o dislikes según hayamos clicado. No se destaca la suficiente para el usuario.	Adrià
U-1008-1	U-1008	Ver agenda de un perfil	Se ven correctamente los eventos pasados y próximos en las agendas correspondientes	Se ven los eventos satisfactoriamente en su agenda correspondiente, pasados o próximos	Adrià
U-1010-1	U-1010	Ver agenda del usuario	Se ven correctamente los eventos pasados y próximos en las agendas correspondientes	Se ven los eventos satisfactoriamente en su agenda correspondiente, pasados o próximos	Adrià
U-1008/10-1	U-1008, U-1010	Ver todos los eventos de una agenda, no solo los 4 máximos	Vemos en una lista todos los eventos de la agenda	Vemos correctamente la lista con todos los eventos	Adrià
U1008/10-2	U-1008, U-1010	Ver perfil del evento de las agendas	Accedemos al perfil del evento correctamente	Accedemos correctamente	Adrià
U-1014-1	U-1104	Recibir notificación	Recibimos una alerta cuando alguien	Notificación recibida correctamente. Caracteres raros no se procesan bien	Adrià
U-1104-2	U-1104	Ver notificación	Accedemos correctamente al perfil del evento al que se refiere	Accedemos correctamente	Adrià
U-1104-3	U-1104	Recibir sugerencia	Recibimos una alerta cuando un	Sugerencia recibida	Adrià

			amigo nos sugiere un evento o un perfil	correctamente. Caracteres raros no se procesan bien. No la podemos aceptar	
U-1104-4	U-1104	Ver sugerencia	Accedemos correctamente al perfil del evento o perfil al que se refiere	Accedemos correctamente	Adrià
U-1104-5	U-1104	Recibir petición de amistad	Recibimos una alerta cuando alguien quiere ser nuestro amigo en el sistema	Petición recibida correctamente. Caracteres raros no se procesan bien. No la podemos aceptar	Adrià

Anexo I - Plan de Marketing

A continuación el Plan de Marketing que se ejecutará para www.eventlayer.com a partir de su apertura como beta pública.

1. Sector

El sector en el que desarrolla la actividad Eventlayer es el ocio local en vivo. Decimos ocio en vivo para diferenciarlo del ocio digital (videojuegos, música...). Además concretamos más y decimos que segmentamos a sólo ocio local, es decir el que se desarrolla dentro de una misma ciudad, descartando así viajes, vuelos, alquiler de coches… Concretamente nos referimos a las siguientes categorías:

- Conciertos
- Cine
- Nocturno (discotecas, bares de copas…)
- Cultura (exposiciones, eventos culturales)
- Espectáculos (teatro, magia, monólogos…)
- Deportes
- Profesionales (ferias, congresos, ponencias…)
- Otros (quedadas de fans, fiestas locales…)

Ilustración 98 - Sectores de actuación de Eventlayer

Dentro del sector, nosotros como actor somos lo que se conoce como una guía de ocio. Es decir, el usuario confía en nosotros para encontrar ofertas de ocio que le vayan a interesar.

Además nos situamos en el sector de las redes sociales verticales. Son redes sociales especializadas en un ámbito en concreto, en nuestro caso somos una red social de ocio.

Por otro lado ya que nuestra actividad se desarrolla en internet y también ahora en dispositivos móviles debemos tener en cuenta el sector de búsqueda de ocio por internet y de uso de internet en dispositivos móviles.

2. Segmentos

Antes de continuar hace falta una puntualización. Según el modelo de negocio de Eventlayer (se puede revisar en el capítulo *1.2.4.Modelo de negocio)* nuestros macro-segmentos, por llamarlo de alguna manera, son 2:

a) Usuarios que buscan planes de ocio en internet.
b) Promotores de eventos o locales de ocio que quieran promocionarse.

No obstante esta versión del Plan de Marketing se centrará en los clientes del tipo a). Esta decisión se justifica con el hecho de que los clientes del tipo b) dependen básicamente de 3 factores totalmente cuantitativos:

i. Tráfico de usuarios del tipo a)
ii. Coste de impresión del anuncio
iii. Coste por click del anuncio

Es por eso que nos centraremos en conseguir a los usuarios del tipo a) ya que los del tipo b) dependen directamente de ello y de una buena política de precios.

Eventlayer como producto, es una guía de ocio online, puede enfocarse a un segmento muy amplio de usuarios, la descripción del segmento general podría ser:

- **Segmento1**. Usuarios de entre 30 y 40 años, residentes en España que asistan a eventos o locales en cualquiera de los sectores de la *Ilustración 98 - Sectores de actuación de Eventlayer,* que utilicen internet para uso doméstico y que pertenezcan a una capital de provincia.

De la definición del segmento se deduce que el uso de internet es requisito indispensable. También acotamos la edad entre 20 y 30 años ya que a priori nos es más fácil generar contenido para ese rango (discotecas, conciertos, bares…). Finalmente decimos que pertenezcan a una capital de provincia porque la expansión de Eventlayer empezará por las grandes urbes donde el tráfico de usuarios al que podemos llegar es más grande.

No obstante, podemos acotar aún más el segmento de usuarios para los cuales el producto les puede ser aún más interesante respecto al de la competencia.

- **Segmento1'.** Usuarios del *Segmento1* que además utilicen Facebook de manera habitual. La integración con Facebook es una de las grandes ventajas de Eventlayer.
    - Dato: 1 de cada 4 españoles utilizan esta red social generalista.

- **Segmento1''.** Usuarios del Segmento1' que además utilicen un Smartphone. Las aplicaciones móviles de Eventlayer son muy superiores a las de la competencia.
    - Dato: durante el año 2010 se vendieron más Smartphones que PCs.

3. Posicionamiento

El posicionamiento de Eventlayer cómo marca es el siguiente:

*Eventlayer te ayuda a encontrar qué hacer **en tu ciudad** de una manera **simple** y **social**.*

Por lo tanto los 3 adjetivos clave del posicionamiento son:

- **Es local.** Hablamos de Eventlayer Barcelona o Eventlayer Madrid como un conjunto. El efecto local es tan importante que el nombre de la ciudad forma parte del logo y por lo tanto de la parte más visible de la web:

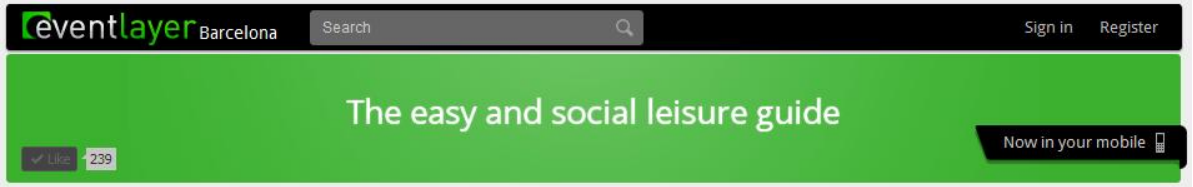


Ilustración 99 - Logo de Eventlayer Barcelona (esquina superior izquierda)

Al usuario le tiene que quedar claro que cuando quiera salir por su ciudad en Eventlayer va a encontrar todo lo que puede hacer en ella. Sin las distracciones de otras webs que te ofrecen escapadas o viajes a otros países (caso Atrapalo.com que más que una guía de ocio es una agencia de viajes).

- **Es simple.** Cuando alguien busca algo que hacer esta noche, o lo encuentra rápido y fácilmente o va a ir al bar que ya conoce. La navegación es minimalista y muy rápida, gracias al uso de nuestra tecnología apenas se hacen llamadas al servidor que es la parte que más incomoda al usuario. Un ejemplo a seguir es el cliente de correo GMail [45] no es que hayan reinventado el email sino que su uso es realmente simple y rápido.

- **Es social.** Es la única guía de ocio que te invita a traerte a tus amigos. Puedes ver dónde van a ir ellos y si te interesa puedes contactarles y organizar cómo vais a quedar, todo dentro de la misma web de Eventlayer o usando tu Facebook, tu correo o tu Smartphone. También puedes opinar, hacer preguntas a otros usuarios… Porque el ocio va de eso, de conocer y pasártelo bien con otra gente. De eso va también, Eventlayer.

Se ha diseñado todo el producto en base a estos 3 diferenciadores y se ha escogido un slogan que hace referencia a ese mismo posicionamiento:

Eventlayer la guía de ocio simple y social

Por último algunos de los slogans candidatos y que pueden utilizarse en alguna acción de Marketing específica han sido:

- Encuentra eventos y queda con tus amigos.
- No vuelvas a perderte un concierto, una obra de teatro, una exposición... Sé el primero en enterarte, díselo a tus amigos y organizad como quedar en un click.
- ¿Te perdiste el último concierto de tu banda preferida? ¿Se te olvidó comentarlo con tus amigos? ¿No los encuentras cuando necesitas comprar las entradas?
- La próxima vez que un amigo te pregunte ¿qué hacemos? o ¿Cómo quedamos? Ya sabes, Eventlayer.com
- Encuentra qué hacer. Organiza como quedar.
- La guía de ocio como si se hubiera inventado hoy.

Por último recordar que el insight en el usuario (qué problema o necesidad resuelve el producto) quedaría de la siguiente manera:

[45] Nos referimos al email de Google: gmail.com

a) Buscar algo para hacer en mi ciudad
b) Estar enterado de si hay algo por hacer interesante en mi ciudad
c) Ver que hacen mis amigos
d) Quedar con mis amigos

4. Plan de promoción

Antes de seguir con el Plan de Comunicación específico de Eventlayer incluimos unos pequeños apuntes sobre división de los medios de comunicación que hemos utilizado. La división escogida no es la tradicional Above the line [46]y Below the line[47]. Desde la aparición del Social Media y el tremendo impacto que internet tiene ahora en las campañas publicitarias se ha visto necesario un cambio de paradigma.

En la nueva clasificación tenemos los medios comprados, se definen como aquellos medios en los que debes pagar por aparecer (hasta ahora los únicos). Los medios propios, son aquellos medios que podemos utilizar sin más coste que nuestro tiempo. Finalmente los medios ganados son aquellos medios en los que nuestro producto es promocionado sin que nosotros tengamos que actuar (el famoso “boca-oreja”).

4.1. Estrategia

Básicamente utilizaremos una estrategia de penetración PULL[48], es decir, intentaremos dar a conocer la “personalidad” de Eventlayer y su diferenciación respecto a la competencia. Esta es la elección más sensata teniendo en cuenta que es un producto nuevo y que por lo tanto no tiene suficiente notoriedad, ni influencia para un PUSH y además, por ahora nuestro presupuesto en medios pagados tampoco nos permite los suficientes recursos para crearla a corto-medio plazo.

En cuanto a nuestra estrategia de expansión, el modelo a seguir es el de expansión por ciudades. Eventlayer es un producto del tipo conocido como glo-cal, que es una mezcla entre global y local y consiste en "pensar globalmente y actuar localmente". En nuestro caso el contenido lo pueden generar los usuarios y otras fuentes globales, no obstante nuestra idea es actuar localmente, especialmente en las primeras etapas cuidando mucho el contenido (conciertos, obras de teatro, descuentos…)

Por lo tanto empezaremos dedicando nuestros esfuerzos de promoción en Barcelona, después seguramente Madrid, Valencia, Bilbao… y así hasta completar todas las capitales de provincia españolas. Después de eso daremos el salto internacional ya que la web está totalmente preparada para la internacionalización tanto de fechas, moneda e idiomas.

[46] Above the line: son los medios más utilizados por la gente, tradicionalmente Televisión, Radio, Prensa

[47] Below the line: son todos los medios que no pertenecen a Above the line.

[48] Estrategias PULL vs PUSH: La estrategia “push” orienta sus esfuerzos de comunicación en el distribuidor. La estrategia “pull” orienta sus esfuerzos de comunicación en el comprador.

4.2. Recursos

A partir de la apertura de la beta pública en Barcelona, dedicaremos un recurso de uno de nosotros de 4h/día a la ejecución del plan de promoción de Eventlayer. El recurso comentado cubrirá la ejecución de los medios propios y ganados.

En cuanto al presupuesto que dedicaremos en medios comprados, se puede ver nuestra proyección de inversión en publicidad a lo largo del primer año en el capítulo de planificación y costes con el concepto de Gastos y a continuación las filas de publicidad. No obstante, adjuntaremos aquí un pequeño resumen:

	Septiembre 2011	Octubre	Noviembre	Diciembre	Enero	Febrero	Marzo	Abril	Mayo	Junio	Agosto	Septiembre
Gastos												
Publicidad Facebook	150	150	150	150	500	500	500	500	2000	2000	2000	2000
Publicidad Adsense	400	400	400	400	800	800	800	800	800	800	800	800
Publicidad Spotify	0	0	0	0	2200	0	0	2200	0	0	2200	0
Gastos Totales	550	550	550	550	3500	1300	1300	3500	2800	2800	3000	2800

Ilustración 100 – Resumen de la proyección de inversión en publicidad el primer año

4.3. Medios pagados

En este punto el medio escogido es claro, nuestra campaña será totalmente vía internet. El motivo principal es que de los anuncios web que realicemos a nuestro producto hay tan solo un click. Es decir el usuario simplemente clicará en el banner que hayamos contratado y ya habrá llegado a nuestro producto. En segundo lugar este medio es el más económico comparado con televisión, radio o prensa.

En el caso de que nuestro presupuesto en promoción aumentara sí nos contemplaríamos la posibilidad de utilizar algún otro medio a medio-largo plazo para afianzar la notoriedad de la marca.

Nuestro objetivo es mantener dos campañas de anuncios en buscadores (SEM) [49]indefinidas, una en Google Adwords y la otra en Facebook en la que revisaremos la estrategia periódicamente según los ratios de conversión que obtengamos.

- **Facebook**. Optaremos por las "Page Like Story" y las "Page Post Story" (en la siguiente ilustración se pueden ver) de Facebook vinculadas por un lado a nuestra web www.eventlayer.com pero también probaremos a hacerlo con nuestra fan page de

[49] SEM: Search Engine Marketing. Se trata de publicidad en buscadores mediante pago por impresión de anuncio.

Facebook. Un simple test A/B nos dirá cuál es el mejor método. Los objetivos de conversión[50] son bien un like en nuestra fan page o el registro del usuario en nuestra web.

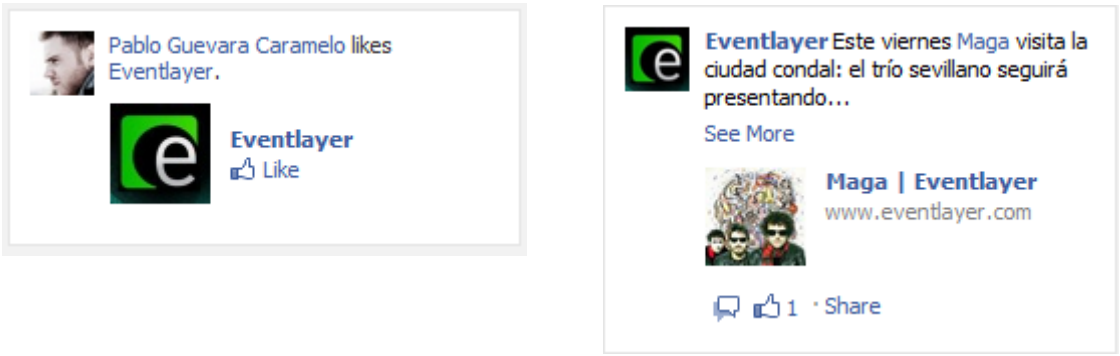


Ilustración 80 - Ejemplo de Page Like Story (izquierda) y Page Post Story (derecha) en la red de publicidad de Facebook

- **Google Adwords**. Pondremos anuncios tanto en la red de búsquedas, como en la red de contenido de este servicio. Utilizaremos las palabras clave más buscadas de la tabla *Ilustración 44*. La idea es probar varias landing pages[51] y nuestro objetivo de conversión es el registro del usuario.

Palabra clave	Búsquedas locales mensuales
"restaurantes barcelona"	40500
"conciertos barcelona"	18100
"teatro barcelona"	12100
"discotecas barcelona"	8100
"eventos barcelona"	6600
"fiesta barcelona"	6600
"restaurantes en barcelona"	6600
"conciertos en barcelona"	5400
"teatro en barcelona"	5400
"espectaculos en barcelona"	2900
"espectaculos barcelona"	2900
"fiesta en barcelona"	2900
"eventos en barcelona"	2400
"discotecas en barcelona"	1900

Ilustración 101 - Palabras clave más buscadas en las que puede interesar a Eventalyer aparecer

- **Spotify.** Creemos que es una buena plataforma para promocionar la sección de conciertos de Eventlayer por lo que si se cumplen las expectativas a partir de Enero empezaremos a

[50] Lo que queremos con seguir con cada anuncio.
[51] Página de nuestra web a la que llega el usuario desde, por ejemplo, un anuncio.

invertir en publicidad en esta plataforma (ver la proyección de gasto del capítulo *4.2.Recursos*).

- **Organización de concursos.** A medio/largo plazo, depende del presupuesto que tengamos para ejecutar el plan de promoción, la organización de concursos en los que se reparten entradas u otros premios a los usuarios tienen mucha aceptación, además si se organizan por Facebook su difusión es viral. La página de la aerolínea Vueling pasó en Mayo del 2011 de 40.000 fans en Facebook a 80.000 en un solo día por medio de su Vueling day en el que premiaban a sus "fans de Facebook" con vuelos a diferentes destinos.

4.4. Medios propios

Si en los medios pagados el factor del que dependía una mayor o menor difusión del producto era el dinero a invertir en este caso el limitador son las horas que invirtamos en la promoción vía los diferentes medios gratuitos que nos proporciona internet. Es decir la cantidad de recursos hombre/hora que decidamos invertir.

a) Canales de comunicación

Mantendremos 3 canales de comunicación principales con el usuario:

- **Fan page de Facebook.** Ya se empezó hace algún tiempo a utilizar nuestra página de Facebook para colgar los eventos más interesantes. Cada post recibía alrededor de 900 impresiones con un promedio de 250 suscriptos a nuestra fan page oficial, esto es así ya que mucha gente re-compartía los post entre sus amigos haciéndolos llegar aún a más gente.

- **Jack Eventlayer.** Inventaremos a un personaje ficticio en Facebook para dar una imagen más amigable a Eventlayer. Analizaremos el segmento que conforman los primeros usuarios que se interesen por nuestra página y crearemos una personalidad afín a este perfil de usuarios para que se sientan identificados. Jack Eventlayer se hará amigo de ellos en Facebook y les recomendará diferentes eventos de Eventlayer.

- **Cuenta de Twitter.** Aquí la estrategia es clara, twittear sin parar los eventos más importantes, normalmente la cuenta de Twitter se actualizará mucho más a menudo que la de Facebook ya que la filosofía de Twitter es más en "tiempo real" que la de Facebook.

Ilustración 103 - Captura de la página de Facebook de Eventlayer

b) Plan de seeding

Por otro lado el segundo aspecto más importante en nuestro plan de promoción, después de los canales de comunicación con el usuario de los que hemos hablado en el punto anterior, es lo que se llama el "Plan de Seeding" que consiste en detectar en que medios de internet se está generando diálogo sobre el tema del que trata tu producto y establecer una conversación con los usuarios de esos medios. Suelen ser blogs y foros sobre un tema en concreto. La figura del Community Manager es clave ya que se dedica a presentar nuestro producto y a escuchar las peticiones de los usuarios interesados en él. En nuestro caso los temas de diálogo que nos pueden interesar se dividen en:

- Blogs de música y conciertos
- Foros de universitarios y estudiantes de Erasmus
- Foros generales

Aquí una tabla con unos cuantos de los medios que hemos seleccionado:

Foros y páginas	Descripción
General y musica	
http://www.potenciadance.com/tema-10082-.html	Musica electronica
http://www.emoxion.com/forum/cataluna-vf100.html	Comunidad de musica electronica
http://hispamp3.yes.fm/foros/viewforum.php?f=10&sid=aa145c88aef7854796082824c107bf11	Foro general, con seccion de musica
http://www.forojovenes.com/musica/ayuda-conciertos-en-barcelona-y-alrededores-40501.html	Comunidad musica
Estudiantes y Erasmus	
http://www.erasmusworld.com/portal/modules/newbb/viewforum.forum=80	Portal de erasmus
http://www.patatabrava.com/forum/festes_i_concerts_a_barcelot49062.htm	Amplio portal de estudiantes
http://foros.universia.es/mvnforum/mvnforum/viewthread?thread=24205&lastpage=yes	Universia, gran comunidad estudiantil
http://foros.universia.es/mvnforum/mvnforum/viewthread?thread=24206&lastpage=yes	Universia, gran comunidad estudiantil
Otros	
http://foro.enfemenino.com/forum/f294/__f416_f294-Discotecas-listas-vip-en-barcelona.html#0r	Comunidad femenina
http://foro.enfemenino.com/forum/f635/__f427_f635-Conciertos-en-barcelona.html#0r	Comunidad femenina
http://foro.enfemenino.com/forum/contact1/__f26069_contact1-Amistad-por-barcelona.html#0r	Comunidad femenina
http://foro.enfemenino.com/forum/f118/__f891_f118-Planes-de-ocio-tiempo-libre-en-barcelona-y-alrededores.html#0r	Comunidad femenina
http://www.forodecine.com/showthread.php/7764-Carteleras?p=99333#post99333	Foro de cine en general
http://www.forocoches.com/	Foro general y de coches
http://www.tripadvisor.es/ShowTopic-g187497-i44-k4103960-Eventos_en_Barcelona-Barcelona_Catalonia.html	Viajeros (red social)
http://www.losviajeros.com/foros.php?t=159633	Viajeros (solo foro)
http://foros.todoenlaces.com/ocio-y-aficiones-vf7.html?sid=32219d565ed1f7a7311321515031fad9	Foro general, se puede hablar de todo

Ilustración 105 - Extracto de la tabla recopilatoria con los recursos para nuestro plan de seeding

c) SEO

Todavía no hemos explotado las técnicas SEO en nuestro sistema pero tenemos mucha experiencia en este ámbito, aún conservamos las palabras clave “vídeos para el Ipod” en primera posición de nuestro anterior proyecto videofindr.net en el que prácticamente todas las visitas (hasta 20.000 diarias) llegaban producto del SEO.

Actualmente la web está indexada por Google en español, catalán e inglés y a partir del 3er mes de explotación del producto cuando la mayoría de funcionalidades planeadas estén acabadas empezaremos a explotar este punto asignando un recurso a media jornada.

d) Repartir Flyers y Carteles

Finalmente, iniciar campañas de colgar carteles y repartir Flyers en puntos frecuentados por usuarios que formen parte de nuestro target principal como vendrían a ser universidades, discotecas o albergues juveniles. Un ejemplo que supo jugar muy bien con este punto fue patatabrava.com, una red social del ámbito universitario.

4.5. Medios ganados

Creemos en Eventlayer como producto para obtener un fluido boca-oreja. Es decir, si a la gente le gusta, lo va a compartir con sus amigos. No obstante contamos con varias acciones de marketing que nos pueden ayudar aún más a convertir Eventlayer en una marca viral y fresca para así conseguir un crecimiento rápido y basado en la opinión en primera mano de los usuarios.

a) Efecto viral en el registro

Cada vez que un usuario se registra en Eventlayer se le ofrece la posibilidad de invitar a sus amigos de sus cuentas de correo o desde su Facebook. A cada contacto le llegará una invitación a la web. Además cada vez que el usuario haga una acción como recomendar o invitar a sus amigos a un evento, aunque estos no estén registrados serán avisados al estilo de "Pablo Guevara te ha invitado a asistir con él al Concierto de U2". Además en el caso de Facebook esto será visible por otros usuarios que tenga a ambos en común. De esta forma conseguimos que nuestros propios usuarios inciten a otros a registrarse.

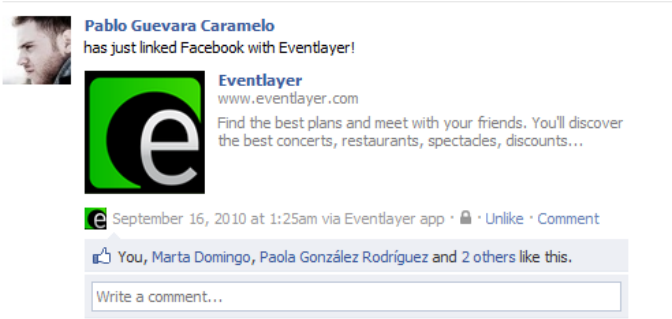


Ilustración 106 - Ejemplo de notificación en Facebook durante el registro de Eventlayer.com

b) Videos virales

Estamos planeando un vídeo viral en Barcelona al estilo de éste que se llevó a cabo en Nueva York: YouTube - Hey You! What Song are you Listening to?.

Básicamente el vídeo consistirá en entrevistar a gente aleatoriamente por la calle preguntándoles qué canción y grupo están escuchando. La sección de conciertos en Eventlayer ya es actualmente la más visitada de nuestra web, por lo tanto creemos que un vídeo así puede atraer a mucho público interesado en la música y en los conciertos en vivo que es de lo que se trata nuestro producto.

a) Modelo de embajadores

Copiaremos el modelo de embajadores de Yelp.com. Estos embajadores no reciben ninguna retribución monetaria por su labor, es más un valor emocional, y se encargarán de promocionar Eventlayer a modo de early adopter[52].

[52] Los usuarios que utilizan un producto antes de que llegue a las masas y se haga popular

b) Street marketing

La naturaleza local de Eventlayer hace que las acciones de promoción dentro de la ciudad de Barcelona que será por la que empezaremos y el resto de capitales donde nos vayamos expandiendo. A continuación unos ejemplos de estas acciones:

Ilustración 107 - Uno de los modelos de cartel que usaremos para la promoción de Eventlayer

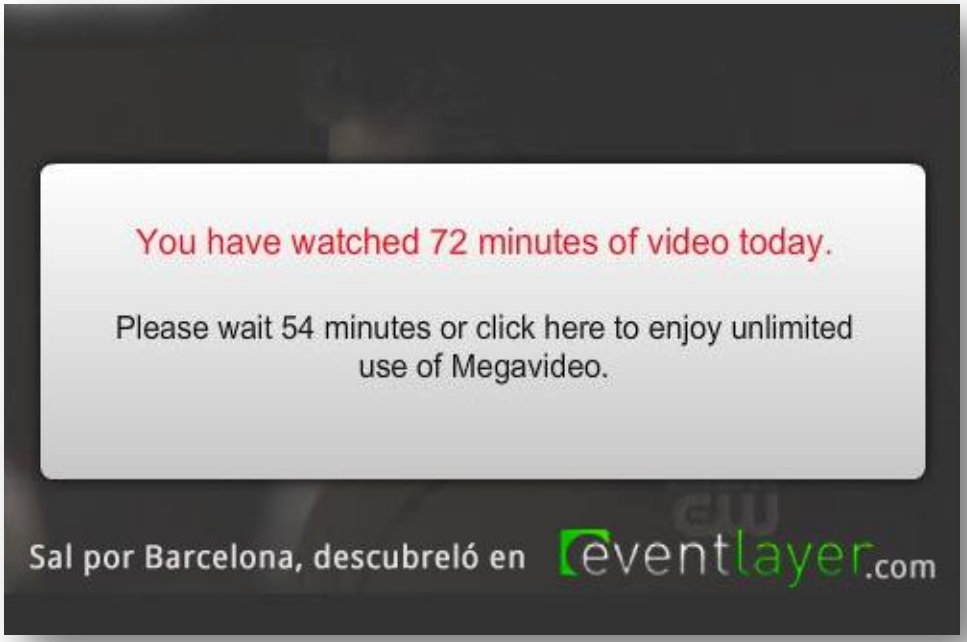


Ilustración 108 - Otro de los modelos de cartel que usaremos para la promoción de Eventlayer

Anexo II - Manual de usuario

La web de Eventlayer.com debe ser lo suficientemente intuitiva y fácil de utilizar para que ningún usuario necesite leerse un manual para utilizarla. Es lo que se denomina "usabilidad" que es un requisito funcional que hemos tenido muy en cuenta y está descrito en el capítulo 2.3.Requisitos no funcionales

Por lo tanto dedicaremos este apartado a hablar de las tareas que puede utilizar un administrador en el sistema ya que este tipo de acciones no están tan guiadas como las de un usuario común.

1. Mantenimiento del sistema

Las acciones que deben llevar a cabo los administradores en caso de reinicio del sistema son, todas ellas desde el directorio /var/opt del servidor:

- **Reiniciar Cassandra:**

  ./cassandra/bin/cassandra –f

- **Reiniciar Mongodb:**

  killall -9 mongod

  rm /data/mongodb/data/mongod.lock

  nohup /var/opt/mongodb/bin/mongod --dbpath /data/mongodb/data &

- **Reiniciar el repositorio de código SVN:**

  svnserve -d -r ./repository

- **Reiniciar Solr:**

  cd solr

  nohup java -Xms16M -Xmx128M -jar start.jar &

Las copias de seguridad se realizan automáticamente cada 6 horas gracias a cron. Las acciones que se llevan a cabo son:

```
cd /root/backup

mkdir data

mysqldump -uadmin -pPASSWORD tewp-production > ./data/productiondb.sql

mysqldump -uadmin - pPASSWORD wikidb > ./data/wiki.sql

cp -R  /var/opt ./data

cp -R  /var/www/vhosts/eventlayer.com ./data/com
```

```
cp -R  /var/www/vhosts/eventlayer.net ./data/net

rm -Rf ./data/net/httpdocs/tmp

cp -R  /data  ./data/data

tar -czf backup.tar.gz data

mv backup.tar.gz /var/www/vhosts/eventlayer.net/httpdocs/backup/
```

2. Añadir datos desde el mashup

Para añadir nuevos datos al sistema se utiliza lo que llamamos mashup que es un sistema que recopila eventos, lugares, artistas… de otras páginas web. Para hacerlo, el administrador debe dirigirse a la ruta */admin* de la aplicación web. Una vez allí, deberá hacer login con el email y contraseña de un usuario con permisos de administración. Para obtener permisos de administración, es preciso contactar previamente con el personal de la web: de esta manera evitamos que algún bug o despiste pueda llevar a que un usuario obtenga permisos mediante el interfaz web.

Una vez dentro, el usuario se encuentra con la pantalla de la *Ilustración 109 - Detalle de la interfície del mashup* donde puede realizar las siguientes acciones y que corresponden a la numeración de la imagen:

1. En la columna de la izquierda, el usuario puede seleccionar una fuente de dónde importar ítems o simplemente para ver los ítems importados anteriormente.
2. En el filtro superior, el usuario puede elegir entre eventos, lugares o artistas para importar.
3. El usuario puede comprobar la existencia de ítems parecidos a un ítem determinado clicando el cuadro de la esquina derecha superior. De esta manera, le aparecerá un listado secundario con ítems ya añadidos en Eventlayer. De esta manera queremos reducir la duplicación de datos.
4. El listado principal contiene un formulario con información por cada ítem del mashup. Desde aquí podemos modificar la información importada o llevar a cabo las acciones que veremos en el apartado siguiente.
5. Gracias a los dos paginadores (uno en la parte superior y uno en la parte inferior de la lista) podemos desplazarnos por los diferentes ítems del listado principal.
6. En la esquina derecha superior tenemos dos botones que sirven para añadir más ítems al listado principal, o para eliminar todos los ítems del listado principal.
7. En esta sección podemos identificar un ítem de la lista, en el párrafo siguiente se detallarán los diferentes elementos.

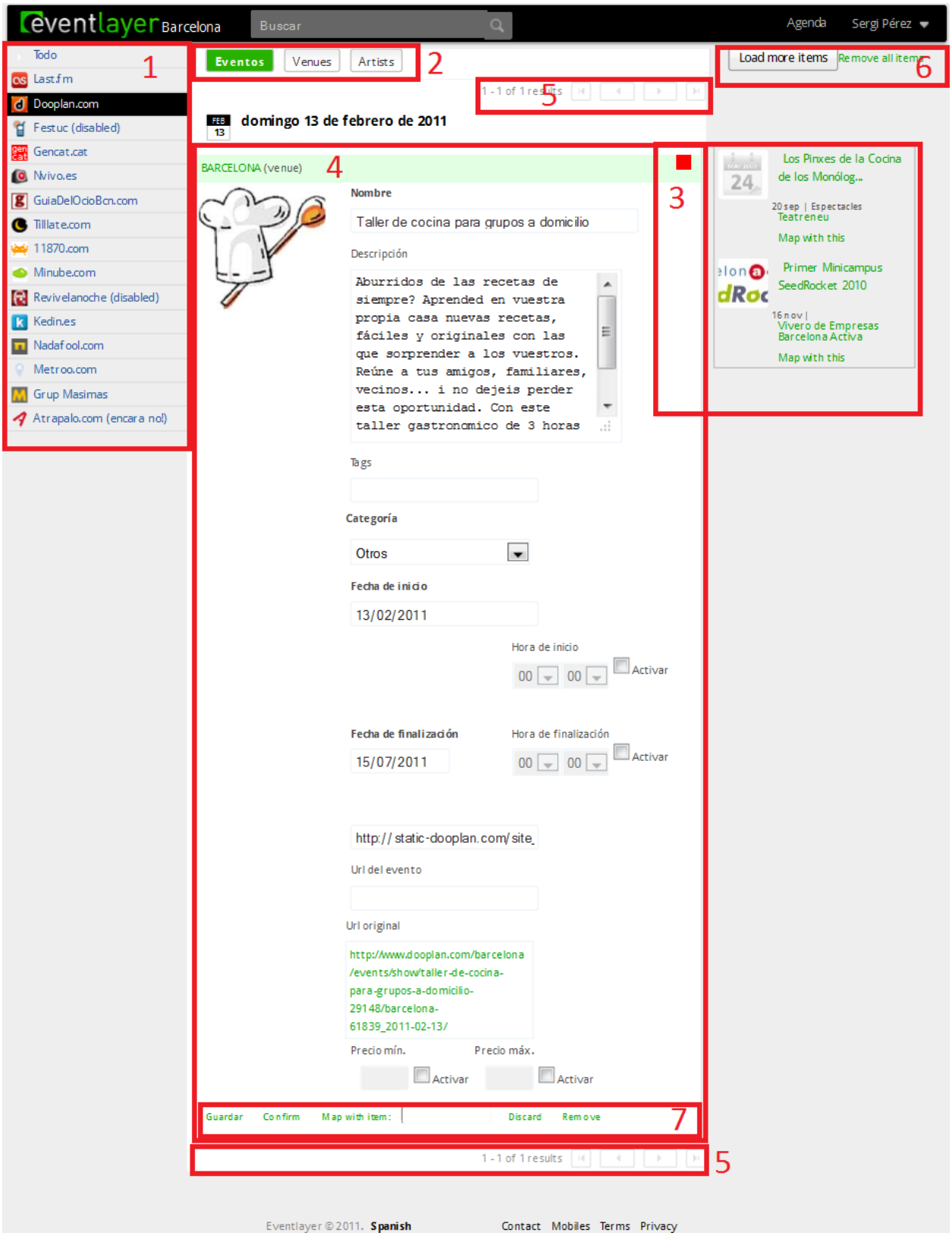


Ilustración 109 - Detalle de la interfície del mashup

Gracias a los botones que encontramos en cada ítem y que se muestra en *Ilustración 110 - Detalle de las acciones que pueden llevarse a cabo sobre un ítem* podemos llevar a cabo las siguientes acciones:

1. Guardar la información que hemos cambiado del ítem.

2. Confirmar este ítem, se creará un nuevo ítem en Eventlayer con los datos disponibles.
3. Si detectamos que este ítem ya existe en Eventlayer, escribiremos su identificador en el cuadro de texto y clicaremos en “map with ítem”: los dos usuarios se relacionarán y se actualizarán las dependencias disponibles.
4. Descartar el ítem, si consideramos que no nos interesa tenerlo en nuestra aplicación.
5. Borrar el ítem, por ejemplo si el parser ha proporcionado datos erróneos y queremos borrarlos definitivamente de la aplicación.

Ilustración 110 - Detalle de las acciones que pueden llevarse a cabo sobre un ítem

Anexo III – Bibliografía

1. Patrones de diseño

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlisside. *Design Patterns: Elements of Reusable Object-Oriented Software*.1994.

Martin Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Professional; 1 edition. 2002.

Refcardz. *Design Patterns Cheatsheet* [en línea]. 2008. <http://refcardz.dzone.com/assets/request/refcard/3091?oid=lan3091&uid=0> [2011, 3 de mayo].

2. Ingeniería de software

Martin Fowler.*UML MODE.* <http://martinfowler.com/bliki/UmlMode.html> [2011, 6 de Junio]

Programania. *¿Está UML muerto? ¿y RUP? Pequeña encuesta* [en línea] <http://www.programania.net/desarrollo-agil/%C2%BFesta-uml-muerto-%C2%BFy-rup-pequena-encuesta/> [2011,20 de Junio]

Wikipedia. *Design Driven Development* [en línea] <http://en.wikipedia.org/wiki/Design-driven_development > [2011,3 Abril]

3. Scrum y Agile Development

Kent Beck, James Grenning. *Manifesto for Agile Software Development* [en línea] <http://agilemanifesto.org/> [2011, 3 de Marzo]

Ken Schwaber and Jeff Sutherland. *The Scrum Guide* [en línea] <http://www.scrum.org/scrumguides/ > [2011,3 de Marzo]

Wikipedia. *Scrum* [en línea] <http://en.wikipedia.org/wiki/Scrum_%28development%29> [2011,3 Abril]

Wikipedia. *Agile Development* [en línea] < http://en.wikipedia.org/wiki/Agile_development> [2011,3 Abril]

4. Requisitos no funcionales

GRUPO ALARCOS. *Calidad del producto* [en línea]. 2006. <http://alarcos.inf-cr.uclm.es/doc/calidad/capitulo05.ppt> [2011, 16 de mayo]

CHICA De la Torre, Ernesto. *La rapidez importa, y mucho* [en línea]. 2009. <http://www.urbecom.com/blog/tag/tiempo-de-respuesta-pagina-web/> [2011, 16 de mayo]

5. Historia y evolución de los smartphones

Smartphone. *Wikipedia* [en línea].2011. <http://es.wikipedia.org/wiki/Smartphone> [2011, 2 de mayo]

Nuevos Gadgets. *Smartphones, historia y significado* [en línea].2010. <http://www.nuevosgadgets.info/2010/09/smartphones-historia-y-significado.html> [2011, 2 de mayo]

Punto Geek. *Breve historia de los smartphones* [en línea].2011. <http://www.puntogeek.com/2011/01/14/breve-historia-de-los-smartphones/> [2011, 2 de mayo]

XTRO. *El principio del fin de la era del PC* [en línea].2011. <http://www.xtro.es/2011/02/28/el-principio-del-fin-de-la-era-del-pc/> [2011, 2 de mayo]

Getafe Noticias. *¿Estamos ante el principio del fin de los PCs?* [en línea].2009. <http://www.getafenoticias.com/sociedad/%C2%BFestamos-ante-el-principio-del-fin-de-los-pc/> [2011, 10 de mayo]

6. Estudio de mercado smartphones

The Nielsen Company. *Android most popular system in US among recent smatphone buyers* [en línea].2010. <http://blog.nielsen.com/nielsenwire/online_mobile/android-most-popular-operating-system-in-u-s-among-recent-smartphone-buyers/> [2011, 5 de mayo]

Poder PDA. *Estudio del dominio en ventas de smartphones en EEUU* [en línea].2011. <http://www.poderpda.com/editorial/estudio-del-dominio-en-ventas-de-smartphones-en-estados-unidos/> [2011, 5 de mayo]

Cooperativa.cl. *Android gana terreno en el mercado de los smartphones de EEUU* [en línea].2011. <http://www.cooperativa.cl/android-gana-terreno-en-el-mercado-de-los-smartphone-de-estados-unidos/prontus_nots/2011-05-06/201448.html> [2011, 5 de mayo]

Entre Click. *Así está el mercado de smartphones en Europa* [en línea].2010. <http://www.entreclick.com/asi-esta-el-mercado-de-smartphone-en-europa-infografia/> [2011, 8 de mayo]

Marketing Directo. *Apple a la cabeza de los fabricantes de smartphones en Europa* [en línea].2011. <http://www.marketingdirecto.com/especiales/marketing-movil/apple-a-la-cabeza-de-los-fabricantes-de-smartphones-en-europa/> [2011, 8 de mayo]

Poder PDA. *Estudio sobre smartphones en Europa: 3 de cada 4 en España son Symbian* [en línea].2010. <http://www.poderpda.com/noticias/estudio-sobre-smartphones-en-europa-3-de-cada-4-en-espana-son-symbian/> [2011, 8 de mayo]

Marketing y Consumo. *Estudio sobre el uso de smartphones en España* [en línea].2010. <http://marketingyconsumo.com/estudio-sobre-el-uso-de-smartphones-en-espana.html> [2011, 8 de mayo]

Computing. *Mercado mundial desmartphones* [en línea].2011. <http://www.computing.es/Tendencias/201105050037/COMUNICACIONES-Canalys--Mercado-mundial-de-smartphones.aspx> [2011, 8 de mayo]

Androidsis. *Android ya es el sistema operativo para Smartphone más popular en Asia* [en línea].2010. <http://www.androidsis.com/android-ya-es-el-sistema-operativo-para-smartphone-mas-popular-en-asia/> [2011, 11 de mayo]

Xakata Móvil. *Android supera por primera vez a Symbian en el cuarto trimestre de 2010* [en línea].2011. <http://www.xatakamovil.com/mercado/android-supera-por-primera-vez-a-symbian-en-el-cuarto-trimestre-de-2010> [2011, 11 de mayo]

Xakata Móvil. *El Market de Android ya es la plataforma con más aplicaciones gratuitas por delente de la App Store de iOS* [en línea].2011. <http://www.xatakamovil.com/aplicaciones/el-market-de-android-ya-es-la-plataforma-con-mas-aplicaciones-gratuitas-por-delante-de-la-app-store-de-ios> [2011, 12 de mayo]

7. Elección de las tecnologías

Symfony. *Wikipedia* [en línea]. 2007. <http://es.wikipedia.org/wiki/Symfony> [2011, 18 de mayo]

CakePHP. *Wikipedia* [en línea]. 2006. <http://es.wikipedia.org/wiki/CakePHP> [2011, 18 de mayo]

EllisLab. *Wikipedia* [en línea]. 2009. <http://es.wikipedia.org/wiki/Code_Igniter> [2011, 18 de mayo]

Zend Framework. *Wikipedia* [en línea]. 2009. <http://es.wikipedia.org/wiki/Zend_Framework> [2011, 18 de mayo]

Comparison of Javascript Frameworks. *Wikipedia* [en línea]. 2008. <http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks> [2011, 18 de mayo]

DOMINGUEZ, J. *Javascript Frameworks comparison Table* [en línea]. 2008. <http://blogs.southworks.net/jdominguez/2008/01/javascript-frameworks-comparison-table/> [2011, 18 de mayo]

VELICHKOV, P. *Mootools vs Jquery vs Prototype vs YUI vs Dojo comparison revised* [en línea]. 2009. <http://blog.creonfx.com/javascript/mootools-vs-jquery-vs-prototype-vs-yui-vs-dojo-comparison-revised> [2011, 18 de mayo]

GRIGORIK, I. *Tokyo cabinet: beyond key value store* [en línea]. 2009. <http://www.igvita.com/2009/02/13/tokyo-cabinet-beyond-key-value-store/> [2011, 18 de mayo]

Lucene vs Solr. *Lucene Tutorial* [en línea]. 2009. <http://www.lucenetutorial.com/lucene-vs-solr.html> [2011, 18 de mayo]

ARNONE, A. RICHTER, N. *Search Engine Rodeo* [en línea]. 2007. <http://www.cs.montana.edu/~richter/Search_Engine_Rodeo.pdf> [2011, 18 de mayo]

RUSSAKOVSKI, A. *Comparison between Solr and Sphinx search servers* [en línea]. 2009. <http://beerpla.net/2009/09/03/comparison-between-solr-and-sphinx-search-servers-solr-vs-sphinx-fight/> [2011, 18 de mayo]

Mobile Application Development. *Wikipedia* [en línea].2011. <http://en.wikipedia.org/wiki/Mobile_application_development> [2011, 15 de mayo]

ALT1040. *De los grandes fallos a solucionar en Android y lo necesaria que es la crítica* [en línea].2010. <http://alt1040.com/2010/08/de-los-grandes-fallos-a-solucionar-en-android-y-lo-necesaria-que-es-la-critica> [2011, 18 de mayo]

Giz Móvil. *Porqué Google lo está haciendo muy mal con Android* [en línea].2010. <http://gizmovil.com/2010/08/porque-google-lo-esta-haciendo-muy-mal-con-android> [2011, 18 de mayo]

Xakata Móvil. *Razones por las que Android llegará a donde iPhone no* [en línea].2008. <http://www.xatakamovil.com/sistemas-operativos/razones-por-las-que-android-llegara-a-donde-iphone-no> [2011, 18 de mayo]

Gómez. S. *Entorno de desarrollo de Android* [en línea].2010. <http://www.sgoliver.net/blog/?p=1267> [2011, 20 de mayo]

Applesfera. *Apple lanza Xcode 4, disponible para todo el mundo en la App Store* [en línea].2011. <http://www.applesfera.com/apple/apple-lanza-xcode-4-disponible-para-todo-el-mundo-en-la-mac-app-store> [2011, 20 de mayo]

8. Arquitectura

The MVC pattern in theory and practice [en línea].--. <http://warp.povusers.org/programming/mvc.html> [2011, 22 de mayo]

Android Developers. *What is Android?* [en línea].2011. <http://developer.android.com/index.html> [2011, 23 de mayo]

Apple Developer. *The iOS Architecture* [en línea].2010. <http://developer.apple.com/library/ios/navigation/> [2011, 26 de mayo]

9. Recomendadores

SARWAR, B., KARYPIS, G., KONSTAN, J., RIEDL, J. *Item-based collaborative filtering recommendation algorithms* [en línea]. 2001. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.9927&rep=rep1&type=pdf> [2011, 3 de mayo].

CORNELIS, C., GUO, X., LU, J., ZHANG, G. *A fuzzy relational approach to event recommendation* [en línea]. 2005. <http://www.fuzzy.ugent.be/Chris/iicai.pdf> [2011, 3 de mayo].

MELVILLE, P., MOONEY, J., NAGARAJAN, R. *Content-Boosted Collaborative Filtering for Improved Recommendation* [en línea]. 2002. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.73.8546&rep=rep1&type=pdf> [2011, 3 de mayo]

CHEN, Y. *A personalized local event recommendation system for mobile user.*2005. <http://ethesys.lib.cyut.edu.tw/ETD-db/ETD-search/view_etd?URN=etd-0823106-163957> [2011, 3 de mayo]

ZOLLERS, A. *Emerging Motivations for Tagging: Expression, Performance, and Activism* [en línea]. 2007. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.7409&rep=rep1&type=pdf> [2011, 3 de mayo]

CURTU, B. *Recommenders* 2009. <http://www.slideshare.net/bcurtu/recommenders> [2011, 3 de mayo]

CommonQueryParameters - Solr Wiki [en línea]. 2006. <http://wiki.apache.org/solr/CommonQueryParameters> [2011, 3 de mayo]

SolrQuerySyntax - Solr Wiki [en línea]. 2007. <http://wiki.apache.org/solr/SolrQuerySyntax> [2011, 3 de mayo]

Distance sorting with spatial filtering [en línea]. 2010. <http://www.mail-archive.com/solr-user@lucene.apache.org/msg40316.html> [2011, 4 de mayo]

10. Implementación

David Crane. *Ajax in Action.*Manning Publications: 1 edition. 2005.

Weirophinney. *Zend Framework 2 Patterns* [en línea]. <http://www.slideshare.net/weierophinney/zend-framework-20-patterns-7482320 > [2011, 10 Mayo]

MOTTIER, C. *Introduction to GreenDroid library* [en línea].2010. <http://android.cyrilmottier.com/?p=240> [2011, 1 de junio]

Gallo Avello. D. *Lenguajes de consulta para XML* [en línea].--. <http://petra.euitio.uniovi.es/~labra/cursos/ext02/saxDom.PDF> [2011, 5 de junio]

Anexo IV - Glosario

ACID — Las características necesarias para que una serie de instrucciones puedan ser consideradas como una transacción.

ACL — Es una forma de determinar los privilegios de acceso a un objeto del sistema mediante listas de control.

Ajax — Acrónimo de **A**synchronous **J**avascript **A**nd **X**ml, es una técnica de desarrollo web que permite obtener y enviar datos desde el cliente del navegador web sin recargar la página, aumentando su interactividad, velocidad y usabilidad.

API — Significa interfaz de programación de aplicaciones del ingés "Application Programming Interface". Son un conjunto de reglas y especificaciones que se usa para comunicar dos aplicaciones, facilitando su interacción.

Bada — Océano en coreano, es un sistema operativo para teléfonos móviles desarrollado por Samsung.

Brew — *Binary Runtime Environment for Wireless* es una plataforma de desarrollo de aplicaciones móviles creada por Qualcomm. Actualmente es soportada por un gran número de modelos de teléfonos con tecnología CDMA.

Bug — Es el resultado de un fallo o deficiencia durante el proceso de creación de software. Dicho fallo puede presentarse en cualquiera de las etapas del ciclo de vida del software aunque los más evidentes se dan en la etapa de desarrollo y programación.

BSD — *Berkeley Software Distribution* es un sistema operativo derivado del sistema UNIX nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

DarwinBSD — Darwin es el sistema que subyace en Mac OS X. Ofrece servicios de sistema operativo de tipo UNIX basados en BSD que proporcionan una estabilidad y un rendimiento mayor que el de versiones anteriores de Mac OS.

Cassandra — Gestor de Bases de Datos de código abierto inicialmente desarrollado por Facebook y que permite trabajar con grandes volúmenes de datos.

CEO — Del inglés C*hief Executive oOficer* es el director ejecutivo, el encargado de máxima autoridad de la gestión y dirección administrativa en una organización.

COCOA	Cocoa es un conjunto de frameworks orientados a objetos que permiten el desarrollo de aplicaciones nativas para Mac OS X. En el caso de iOS la API se llama Cocoa Touch.
Cross Site Scripting	Es un tipo de vulnerabilidad web que permite a un atacante inyectar código en la parte del cliente que ven los otros usuarios. Su impacto va desde modificar la interfaz del usuario a saltarse controles de acceso o robo de cookies.
CSS	CSS es un lenguaje usado para definir la presentación de un documento escrito en lenguaje HTML o XML. Se basa en definir identificadores y clases a los elementos y dar formato visual a estos identificadores y clases.
Denormalización	Optimizar una base de datos por medio de guardar los datos varias veces, ya sea duplicándolos o guardando información pre construida.
Extender	En programación extender es una forma de aprovechar código fuente entre varios archivos o componentes.
Facebook.com	Aplicación web que permite estar conectado con tus amigos y compartir información con ellos, es uno de los máximos exponentes de la web social o web 2.0.
Firebug	Herramienta de desarrollo web inicialmente lanzada como plugin para Firefox pero actualmente disponible para otros navegadores. Permite modificar el código de la interfaz de una página web incluyendo CSS, HTML y Javascript así como monitorizar cambios en el código y peticiones web.
Firefox	Navegador web libre y de código abierto propiedad de la fundación Mozilla.
Framework	Es un conjunto estandarizado de conceptos, prácticas, criterios y código que permite crear un sistema de software de una manera más ágil y rápida.
Full text	Es una técnica de búsqueda para bases de datos donde se examinan todas las palabras del documento y se buscan las palabras dadas por el usuario.
Google Chrome	Navegador web desarrollado por Google.
Hosting	Servicio que permite a usuarios y organizaciones alojar sus propias páginas webs sin tener que mantener un servidor ni conexión a internet.

Html	Abreviación de Hypertext Markup Language, es el lenguaje de marcado en el que se codifican las páginas web.
HTML 5	Es la quinta revisión importante del lenguaje básico de la World Wide Web, HTML. HTML 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos.
IDE	Significa "entorno de desarrollo integrado" y se refiere a un conjunto de herramientas de desarrollo de software integradas entre sí.
Inteligencia artificial	Disciplina de la computación que se encarga de construir procesos que dada una entrada y usando el conocimiento almacenado en ellos, maximizan una medida determinada.
Internet Explorer	Navegador web desarrollado por Microsoft, principalmente para plataformas de Microsoft Windows.
JavaScript	Lenguaje de programación interpretado que se ejecuta del lado del cliente, usado principalmente para dotar a las aplicaciones web de la interacción que no proporciona el lenguaje Html.
Librería de software	Conjunto de código y datos que sirven de apoyo para desarrollar un software.
Linkedin.com	Sitio web orientado a negocios comparable a una red social.
Log	Término inglés que indica el registro de eventos que se suceden en el sistema.
Loggear	Significa guardar en un fichero o en otro soporte un suceso del sistema.
Mac OS X	Es un sistema operativo desarrollado y comercializado por Apple Inc. que ha sido incluido en su gama de computadoras Macintosh desde2002 hasta la actualidad.
Malware	Del inglés *malicious software*, es un tipo de software que tiene como objetivo infiltrarse o dañar una computadora sin el consentimiento de su propietario.
Mashup	Aplicación web que usa y combina datos de otras páginas y servicios web para ofrecer un servicio nuevo para el que no fueron concebidos originalmente.
Máquina Virtual Java	En inglés *Java Virtual Machine*, JVM, es un máquina virtual ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

Mineria de datos	Disciplina de la computación que se encarga de la extracción no trivial de información que reside en unos datos.
Mongodb	Base de datos de código abierto que forma parte de las bases de datos NoSql.
Motor de búsqueda	Sistema que busca archivos dadas unas palabras o filtros de entrada, de forma automática.
MySql	Sistema de gestión de bases de datos de código abierto.
Newsletter	Publicación distribuida de forma regular a una lista de usuarios que se han suscrito previamente.
Nosql	Término que agrupa las bases de datos no relacionales y que en general no proporcionan garantías de atomicidad, consistencia, aislamiento y durabilidad. En cambio, ofrecen una gran escalabilidad horizontal.
Open Source	También conocido como código abierto, se refiere al software desarrollado y distribuido libremente por una comunidad de programadores.
Parser	Un parser o analizador sintático es un componente de software que convierte el texto de entrada en otras estructuras.
Patrón de diseño	Utilizado en ingeniería del software es una solución óptima y probada a un problema común.
Patrón de 3 capas	Es un patrón de diseño que considera una buena práctica separar el código de un sistema en 3 capas. Una para las vistas, otra para las acciones de negocio propias del asunto que modela y por último otra para el tratamiento de los datos.
Popup	Un tipo de pantalla utilizada en sistemas informáticos que se caracteriza por aparecer sobre el contenido principal y poder cerrarse fácilmente sin perder el contexto anterior.
Postcondición	En ingeniería del software son una serie de reglas que debe cumplir un código al ser llamado.
Precondición	En ingeniería del software son una serie de reglas que debe cumplir un código al efectuar una llamada.
Opera	Navegador web creado por la empresa noruega Opera Software
PHP	Lenguaje de programación interpretado, se usa principalmente del lado del servidor para la creación de páginas web con contenido dinámico.

Red CDMA	Sistema para comunicaciones móviles que apareció antes que el sistema GSM. Esta tecnología fue la que impulso el uso de teléfonos móviles en todo el mundo.
Red GSM	Sistema global para las comunicaciones móviles (GSM, proviene del francés *Groupe Spécial Mobile)* es un sistema estándar, libre de pagos, de telefonía móvil digital.
SDK	Un kit de desarrollo de software o SDK (siglas en inglés de *software development kit*) es generalmente un conjunto de herramientas de desarrollo que le permite a un programador crear aplicaciones para un sistema concreto.
Sistema de información	Es un tipo de software que se caracteriza por tener como objetivo principal el dar acceso y posibilidades de manipulación a los usuarios sobre un conjunto variable de información.
Sql	Estándar para realizar consultas a bases de datos relacionales.
Sql Injection	Método de infiltración de código malicioso que se vale de vulnerabilidades a la hora de validar y filtrar datos introducidos por el usuario y que afecta a las consultas a bases de datos. Las sentencias SQL junto a estos datos inválidos dan lugar a nuevas consultas con fines diferentes de los que fueron programados y que se ejecutarán libremente.
Tablet	Una tablet es una computadora portátil con la que se puede interactuar a través de una pantalla táctil. El usuario puede utilizar una pluma o los dedos para trabajar con el ordenador sin necesidad de teclado físico, o mouse.
Thread	Un hilo de ejecución o subproceso es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente.
Tiempo real	Es aquel sistema digital que interactúa activamente con el entorno, en nuestro caso que interactúa activamente con los usuarios del sistema web.
Tuenti.com	Red social enfocada principalmente a la población española.
Twitter.com	Red social basada en el envío de mensajes cortos y uno de los grandes estandartes de la web 2.0.

Web 2.0	Es la nueva forma de aprovechar la red, permitiendo la participación activa de los usuarios, a través de opciones que le dan al usuario voz propia en la web, pudiendo administrar sus propios contenidos, opinar sobre otros, enviar y recibir información con otras personas de su mismo estatus o instituciones que así lo permitan. La estructura es más dinámica y utiliza formatos más modernos, que posibilitan más funciones.
UNIX	Es un sistema operativo portable, multitarea y multiusuario. Fue desarrollado en 1969 por un grupo de empleados de los laboratorio de AT&T Bell.