Memoria PFC

EVENTUAL

PFC- Programación aplicaciones móviles usando HTML5

Daniel Riego Retamero

2n ciclo Ingeniería Informática. PFC

Consultor: Roman Roset Mayals

Enero | 2014

Índice

1. Descripción del proyecto 3
2. Objetivos de aprendizaje 4
3. Calendario de trabajo: Planificación 5
4. Aplicaciones similares existentes 7
5. Entorno de trabajo 8
6. Tecnologías y librerías utilizadas 14
7. Análisis y diseño: Prototipado móvil 21
8. Análisis y diseño: Prototipado Web 30
9. Análisis y diseño: Flujo navegación Móvil 36
10. Análisis y diseño: Historias de Usuario: Aplicación Móvil 38
11. Análisis y diseño: Diagrama de clases 49
12. Arquitectura física de Eventual 51
13. Implementación. Aplicación cliente: Android 52
14. Manual de Usuario 66
15. Conclusiones 67
16. Referencias 68

1.Descripción del proyecto

La funcionalidad del proyecto implementado consiste principalmente en crear una aplicación que permita compartir comentarios y fotografías con otros usuarios que participen en un evento social. Para tener una idea más clara de la funcionalidad de la aplicación podemos pensar en un tipo concreto de evento social como puede ser una boda.

Últimamente en algunas bodas se ha puesto de moda dejar cámaras desechables en las mesas de los invitados, para que los propios asistentes capten fotografías del momento. Estas cámaras se recogen una vez finalizado el banquete y pueden pasar a ser parte del álbum "oficial" de la boda.

La ventaja de este tipo de fotografías es que las instantáneas capturadas son más informales y divertidas, ya que plasman situaciones y momentos de complicidad que seguramente pasarían desapercibidos a los ojos de un fotógrafo profesional.

Eventual

La idea es implementar una aplicación móvil, ***eventual***, que permita mediante el uso de la cámara del propio dispositivo, tomar esas fotografías y subirlas al evento en la nube que se habrá creado previamente para la ocasión, sin la necesidad del uso de cámaras desechables, ni que éstas sean recolectadas.

Mediante un identificador o token del evento, que podría estar la propia mesa de invitados, eventual permitirá a los asistentes subir, visualizar y comentar fotografías en tiempo real con el resto de invitados. Incluso éstas imágenes podrían ser proyectadas, según se van generando, en el mismo evento.

2. Objetivos de aprendizaje

Objetivo general

El objetivo general del proyecto es crear una aplicación social móvil usando tecnologías web estándar (CSS3, HMTL5 y JS), siguiendo los patrones de diseño de aplicaciones reales de éxito, en las que acceden millones de usuarios.

El objetivo del proyecto consistirá en crear una aplicación móvil hibrida basada en una interfaz ligera, que intercambiará mensajes JSON como una api REST, sobre la cual recaerá toda la lógica de la aplicación. Esta api REST se implementará usando tecnología J2EE y persistirá el modelo en una base de datos No SQL.

Objetivos principales:

Implementar una API *REST* para implementar los CRUD's de las principales entidades de la aplicación (usuario, evento, mensajes, fotografías). Se implementará usando Spring MVC y MongoDB como base de datos No SQL.
Desarrollar una aplicación web, usando *JqueryMobile* y BackBone.js para realizar el CRUD de una entidad de la aplicación, por ejemplo *Eventos* o *Usuarios*, accediendo a la capa REST previamente implementada.
Extender la aplicación web anterior, y convertirla en una aplicación Android que permita ejecutar en un emulador Android el CRUD anterior.
Realizar una aplicación que acceda a la cámara web del dispositivo, tome una fotografía y la suba al api REST y persistiéndola en la base de datos.
Familiarizarme con PhoneGap y PhoneGap Build como herramienta para realizar compilaciones para diferentes plataformas en la nube e instalarlas en dispositivos móviles reales.

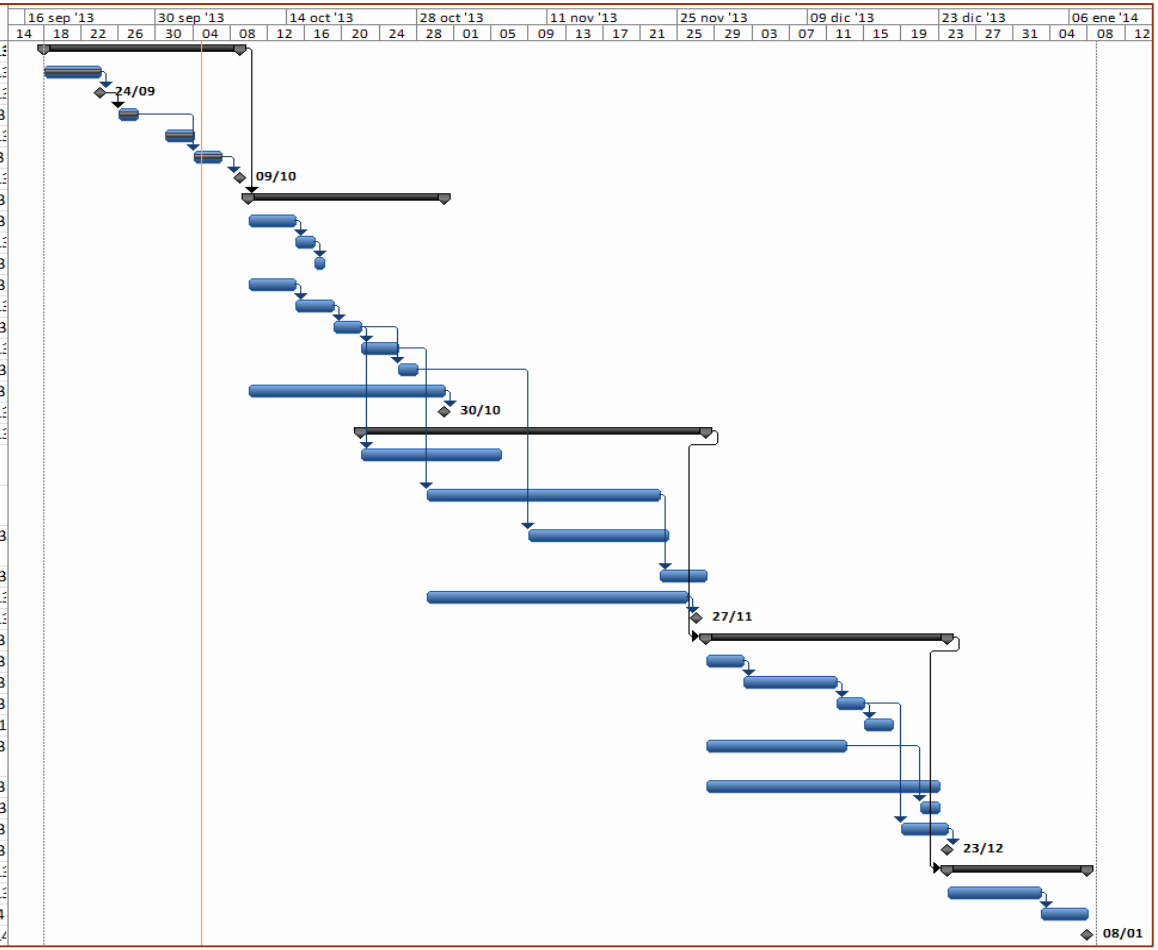
3.Calendario de trabajo: Planificación

Para realizar todo el trabajo necesario se ha dividido el calendario en 4 partes, coincidiendo con las entregas de las PAC's y entregas finales:

Definición del proyecto 18-09-13 / 09-10-13 • Concepción Idea • Propuesta Proyecto • Definición historias de usuario • Preparación entorno de trabajo • Elaboración plan de trabajo • **Entrega Pac1**	**Análisis y diseño** 10-10-13 / 30-10-13 • Selección y Estudio Framework HTML5 • Selección Tecnologías Servidor • Selección base de datos NO SQL • Detallar requerimientos funcionales • Definición api REST • Definición modelo de datos • Prototipado Interfaces Usuario Aplicación Móvil • Prototipado Interfaces Usuario Aplicación WEB • Actualización parcial de Memoria • **Entrega Pac2**	**Implementación** 22-10-13 / 27-11-13 • Desarrollo api RESTful J2EE (eventos, usuarios, imágenes) • Desarrollo interfaces graficas aplicación móvil (CRUD eventos, usuarios…) • Desarrollo Interfaces Web • Integración api REST / aplicación móvil • Actualización parcial de Memoria • **Entrega Pac3**
Pruebas y plan de instalación 28-11-13 / 23-12-13 • Corrección Errores • Despliegue aplicación web/REST en Cloud • Pruebas en dispositivos reales • Carga de datos para la evaluación • Actualización parcial de Memoria • Compartir aplicación repositorio público (Github) • Elaboración tutorial instalación y ejecución • **Entrega Final 1/2**	**Memoria y Entrega Final** 24-12-13 / 08-01-14 • Actualización final de la Memoria • Realizar Video Presentación • **Entrega Final 2/2**	

Calendario de trabajo: Diagrama de Gannt

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo
1		**Definición del proyecto**	**21 días**	**mié 18/09/13**
2		Concepción Idea	6 días	mié 18/09/13
3		Propuesta Proyecto	0 días	mar 24/09/13
4		Definición historias de usuario	2 días	jue 26/09/13
5		Preparación entorno de trabajo	3 días	mar 01/10/13
6		Elaboración plan de trabajo	3 días	vie 04/10/13
7		Entrega Pac1	0 días	mié 09/10/13
8		**Análisis y diseño**	**21 días**	jue 10/10/13
9		Selección y Estudio Framework HTML5	5 días	jue 10/10/13
10		Selección Tecnologías Servidor	2 días	mar 15/10/13
11		Selección base de datos NO SQL	1 día	jue 17/10/13
12		Detallar requerimientos funcionales	5 días	jue 10/10/13
13		Definición api REST	4 días	mar 15/10/13
14		Definición modelo de datos	3 días	sáb 19/10/13
15		Prototipado Interfaces Usuario Aplicación Mi	4 días	mar 22/10/13
16		Prototipado Interfaces Usuario Aplicación W	2 días	sáb 26/10/13
17		Actualización parcial de Memoria	21 días	jue 10/10/13
18		Entrega Pac2	0 días	mié 30/10/13
19		**Implementación**	**37 días**	mar 22/10/13
20		Desarrollo api restfull J2EE (eventos, usuarios, imágenes..)	15 días	mar 22/10/13
21		Desarrollo interfaces graficas aplicación móvil (CRUD eventos, usuarios...)	25 días	mar 29/10/13
22		Desarrollo Interfaces Web (life streaming lista de updates)	15 días	sáb 09/11/13
23		Integración api REST / aplicación móvil	5 días	sáb 23/11/13
24		Actualización parcial de Memoria	28 días	mar 29/10/13
25		Entrega Pac3	0 días	mié 27/11/13
26		**Pruebas y plan de instalación**	**26 días**	jue 28/11/13
27		Validación	4 días	jue 28/11/13
28		Corrección Errores	10 días	lun 02/12/13
29		Despliegue aplicación web/REST en Cloud	3 días	jue 12/12/13
30		Pruebas en dispositivos reales	3 días	dom 15/12/1
31		Carga de datos para la evaluación (usuarios, eventos, comentarios)	15 días	jue 28/11/13
32		Actualización parcial de Memoria	25 días	jue 28/11/13
33		Compartir aplicación repositorio público (Git	2 días	sáb 21/12/13
34		Elaboración tutorial instalación y ejecución	5 días	jue 19/12/13
35		Entrega Final 1/2	0 días	lun 23/12/13
36		**Memoria y Entrega Final**	**15 días**	mar 24/12/13
37		Actualización final de la Memoria	10 días	mar 24/12/13
38		Realizar Video Presentación	5 días	vie 03/01/14
39		Entrega Final 2/2	0 días	mié 08/01/14



4. Aplicaciones similares existentes

Desafortunadamente la idea en la que se basa ***eventual*** no es única ni original, tras realizar una breve búsqueda en internet se han encontrado varias aplicaciones, algunas de mejor calidad que otras, que ya han implementado este concepto.

Estas son algunas de las más importantes.

WedPics | https://www.wedpics.com/

WedPics se basa en el mismo concepto que Eventual, todo y que esta última es una aplicación enfocada a cualquier tipo de eventos, mientras que WedPics, tal como indica su nombre, se centra en bodas.

La idea es la misma, los invitados realizan fotos con sus teléfonos móviles inteligentes a una cuenta creada por los novios y van subiendo fotos a un álbum virtual, visible en tiempo real por todos los asistentes, incluso en el mismo viaje de novios.

Existen versiones para iOS y Android.

Video de la demostración de la aplicación. https://www.wedpics.com/#div_header_video

EverSnap | https://www.eversnapapp.com/

Eversnap es muy similar a la aplicación anterior y por tanto a ***eventual***. En este caso ellos también incluyen el concepto de mostrar en el mismo evento una presentación con las fotos que se han realizado durante el mismo.

La publicidad en la página principal así lo indica.

"Eversnap app collects all guests' photos & videos in one central online album, which you can also show live on a screen during your event"

Existen versiones para iOS y Android.

5. Entorno de trabajo

Para desarrollar, testear, ejecutar y publicar la aplicación se ha utilizado el siguiente software:

Ubuntu 13.04

Ubuntu es un sistema operativo basado en Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado **Unity**. Su nombre proviene de la ética **Ubuntu**, en la que se habla de la existencia de uno mismo como cooperación de los demás.

Este ha sido el sistema operativo elegido para desarrollar y testear la aplicación, ya que de las dos principales plataformas móviles existentes, Android e iOS, se ha optado por implementar la aplicación en la primera, y todas las herramientas necesarias tienen una versión para este sistema operativo. Otro punto a favor para elegir este sistema es que la aplicación final, la parte servidor, se instalará en un servicio de computación en la nube basado en este sistema operativo.

Eclipse Juno

Eclipse ha sido el entorno de desarrollo seleccionado para implementar parte de la aplicación móvil y la totalidad de la aplicación web.

Eclipse es un entorno de desarrollo integrado (IDE) desarrollado en Java. Está basado en un espacio de trabajo sobre el cual se puede ir añadiendo plugins ampliando su funcionalidad y productividad. Se utiliza principalmente para desarrollar aplicaciones Java, pero también soporta otros lenguajes como PHP, JavaScript, Python, etc.

Entorno de trabajo

Java SDK 1.6

La plataforma **Java** es un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el lenguaje de programación Java u otros lenguajes que compilen a *bytecode* y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino una máquina virtual encargada de la ejecución de las aplicaciones, y un conjunto de bibliotecas estándar que ofrecen una funcionalidad común.

Eventual está compuesto por dos aplicaciones, una versión cliente móvil y un aplicación web con una funcionalidad ligeramente similar, pero pensada para ser utilizada en dispositivos no móviles. Ambas aplicaciones consumen y persisten datos mediante una capa REST que se ha implementado íntegramente utilizando tecnologías Java.

Por otra parte aunque la versión móvil está escrita principalmente en JavaScript y HTML, hay una pequeña parte escrita en este lenguaje, ya que se instalará en un sistema Android, implementando usando está tecnología.

Android SDK 4.2 | Android Development Tools

El kit de desarrollo de Android, es un conjunto de librerías y herramientas de desarrollo necesario para construir, testear y depurar aplicaciones para ser ejecutadas en la plataforma Android.

Entre las principales herramientas incluidas en Android Tools está **AVD Manage**r que permite configurar diferentes versiones de dispositivos Android virtuales, con diferentes versiones del sistema operativo, para poder instalar y ejecutar la aplicación en un emulador.

Entorno de trabajo

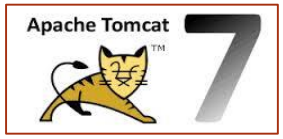
MongoDB 2.2

MongoDB (de la palabra en inglés "hu**mongo**us" que significa enorme) es un sistema de base de datos NoSQL orientado a documentos, desarrollado bajo el concepto de código abierto.

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos JSON con un esquema dinámico (MongoDB llama ese formato BSON), haciendo que la integración de los datos en aplicaciones sea más fácil y rápida.

Nuestra aplicación móvil no guarda información en el dispositivo, a excepción de las credenciales de usuarios y datos de caché, sino que intercambia objetos JSON con el servidor en todo momento. Utilizando **MongoDB** modelamos las entidades de la aplicación con la misma estructura tanto en el cliente como en la base de datos.

Apache Tomcat 7 | Jetty 7



Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de Servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los Servlets y de JavaServer Pages (JSP) de Sun Microsystems.

Jetty es un servidor HTTP 100% basado en Java y un contenedor de Servlets escrito en Java. Jetty se publica como un proyecto de software libre bajo la licencia Apache 2.0.

Se han utilizado ambos contenedores de Servlets indistintamente para instalar la parte web de la aplicación y la API REST. Se ha optado por ellos ya que son los que más fácilmente se integran, utilizando tecnología Java, con la plataforma **Heroku**.

Heroku

Heroku, es una de las primeras plataformas de computación en la nube y empezó a desarrollarse en 2007 con el objetivo de soportar solamente el lenguaje de programación Ruby, pero posteriormente se ha extendido el soporte a Java, Node.js, Scala, Clojure , Python y PHP. La base del sistema operativo es Ubuntu.

Se ha optado por esta plataforma de computación en la nube ya que reúne las características principales que se han estimado necesarias para el desarrollo de la aplicación:

- Soporte de Java íntegro. No usa una versión capada del framework como por ejemplo ***Google App Engine*** que no permite utilizar librerías de terceros, uso de hilos de ejecución, acceso a disco, etc.
- Servicio gratuito. Totalmente gratuito con limitaciones como memoria RAM reducida, sólo permitir una instancia de la aplicación activa al mismo tiempo, entrar en modo hibernación tras un periodo de inactividad, número limitado de horas mensuales. Todas estas limitaciones se han considerado asumibles para el propósito de este PFC.
- **Maven** como herramienta de construcción e instalación.
- No es necesario adaptar el código desarrollado y operativo en un entorno local para que funcione en la plataforma.
- Fácilmente configurable.

MongoHQ

MongoHQ es una plataforma de base de datos como servicio, para alojar, compartir y gestionar instancias de bases de datos basadas en MongoDB

MongoHQ es el servicio que la mayoría de usuarios de Heroku usa para alojar sus instancias de MongoDB, debido a la facilidad de integración entre ambos servicios. Ese es el principal motivo por el que se ha optado este servicio para la persistencia de datos en **Eventual**.

Assembla

Assembla es una empresa que provee herramientas de colaboración y de seguimiento de errores (*Bug Tracking System*) y tareas basadas en la nube para organizar y administrar proyectos de código abierto y comerciales para el desarrollo de software.

Se ha utilizado esta herramienta como repositorio de código para los dos proyectos implementados: la aplicación cliente y la aplicación web/ REST.

La tecnología usada en este repositorio ha sido SVN, debido a la fácil integración con eclipse gracias al plugin **Subclipse**.

Balsamiq Mockups

Balsamiq Mockups es una aplicación desarrollada con Flex y Adobe Air. Es una herramienta de prototipado rápido que permite diseñar interfaces de usuario de forma sencilla y rápida. Se ha utilizado para el prototipo tanto de la aplicación móvil, como de la aplicación web.

Google Chrome

Google Chrome es un navegador web desarrollado por Google basado en el motor de renderizado **Blink** (*fork de WebKit*).

Se ha utilizado Google Chrome para validar el funcionamiento de la aplicación y depurar el JavaScript a medida que se ha ido generando, gracias al uso de las herramientas de desarrollador que contiene.

Siempre que no se requiera de características propias del dispositivo como la cámara, se puede optar por el uso de un navegador para validar la aplicación de forma rápida, en lugar de un emulador, ya que esto agiliza enormemente el desarrollo.

GenyMotion

GenyMotion es un emulador de Android basado en virtualización. A medida que se ha ido avanzando en el desarrollo de la aplicación, y se ha requerido el uso cada vez más continúo del emulador Android, su pobre rendimiento ha sido más problemático, haciendo que el testear y depurar la aplicación fuera extremadamente lento y tedioso.

Por este motivo se ha recurrido finalmente al uso de esta herramienta, la cual consigue un rendimiento muchísimo superior.

6.Tecnologías y librerías utilizadas

De las múltiples librerías y frameworks existentes en el mercado para desarrollar la aplicación, finalmente nos hemos decantado por utilizar ***Jquery Mobile*** para desarrollar la interfaz gráfica emulando el aspecto de las aplicaciones móviles. Esta librería es un Framework muy completo y no sólo un conjunto de componentes gráficos como es ***Twitter Bootstrap*** usado en la parte WEB.

Antes de empezar a utilizar esta librería ha habido un proceso de recopilación de información sobre la misma. Durante este proceso se han encontrado varias opiniones contrarias al uso de Jquery Mobile, debido a que es una librería excesivamente pesada y que puede dar muchos problemas de rendimiento. Por ese motivo se ha optado por no utilizarla el stack completo, y sólo se ha utilizado los componentes gráficos y se ha recurrido a ***Backbone.js*** para la navegación entre páginas, la comunicación con la capa REST, la presentación de las vistas y la gestión de modelos.

Para acceder a las funcionalidades del móvil desde JavaScript se ha utilizado ***Phonegap***, concretamente para el acceso a la galería de fotografías del móvil y para el uso de la cámara. Por otro lado ***Backbone.js*** usa de forma implícita ***Jquery*** y también usa ***Underscore*** para la renderización de las vistas mediante plantillas.

Concretamente las principales tecnologías utilizadas para desarrollar el conjunto de la aplicación son las siguientes:

Jquery Mobile 1.3.2

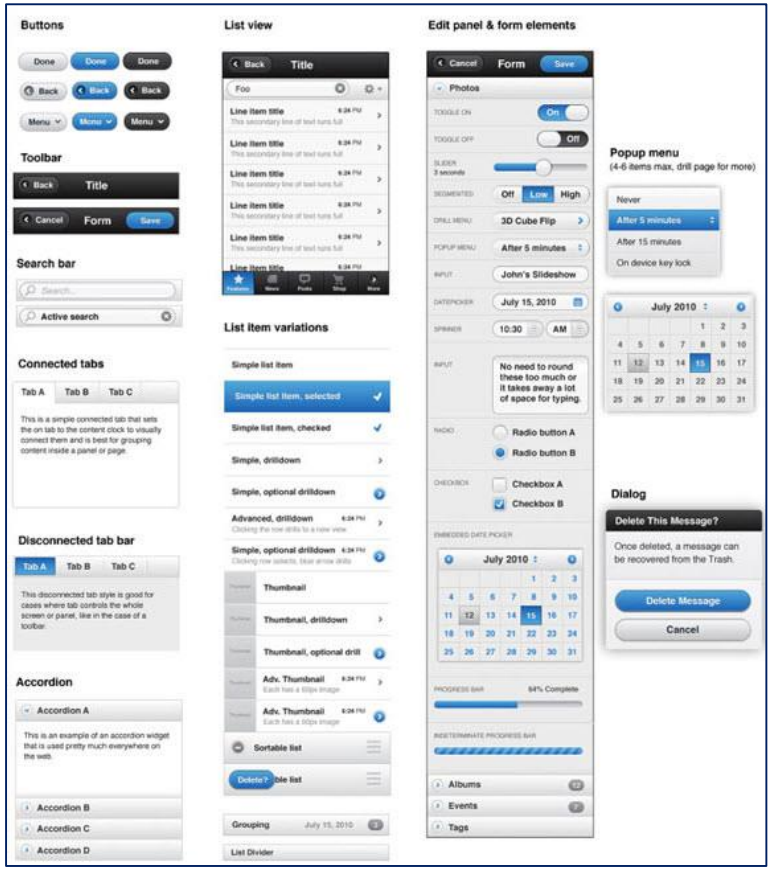
Jquery Mobile es una framework de interfaz gráfica de usuario basado en HTML5, diseñado para construir páginas webs y aplicaciones, que sean accesibles por teléfonos inteligentes, tabletas, y pc's de escritorio.

Está compuesto tanto por un conjunto de librerías JavaScript como por ficheros CSS e imágenes, permitiendo crear aplicaciones que se ejecuten en diferentes plataformas.

Su gran similitud con la plataforma iOS ha generado cierta controversia últimamente con la aparición de iOS 7, ya que el gran esfuerzo realizado por parte de los desarrolladores de esta librería para que las aplicaciones parecieran nativas ha quedado un tanto obsoleto, debido al cambio radical que ha experimentado la interfaz gráfica del nuevo sistema operativo.

Tecnologías y librerías utilizadas

Una de las principales ventajas de esta librería es la gran cantidad de componentes ya implementados que contiene, totalmente intuitivos y funcionales, dotados de apariencia nativa y de una gran facilidad de integración en una aplicación HTML existente. La curva de aprendizaje necesaria para utilizar JqueryMobile, conociendo previamente HTML y Jquery, es inexistente.



Componentes gráficos JQuery Mobile

Ejemplo de componentes gráficos Eventual

La parte negativa es que tanto las animaciones y transiciones entre páginas, como los diálogos emergentes y menús son bastante pesados. Dan la sensación de no son estar todo lo optimizados que deberían, lo que hace que la aplicación vaya más lenta de lo que cabría esperar, y por tanto, no se consiga la fluidez de una aplicación nativa.

Otro de los puntos negativos es que, al estar todos los componentes ya totalmente implementados, y haber poca variedad entre ellos, apenas unos cambios de color entre los diferentes temas, hace que casi todas las aplicaciones desarrolladas utilizando este framework sean iguales. Se requiere de un gran conocimiento de diseño para poder obtener resultados diferentes y personalizados, a pesar del esfuerzo que realizan los desarrolladores de Jquery Mobile como el editor de temas de CSS.

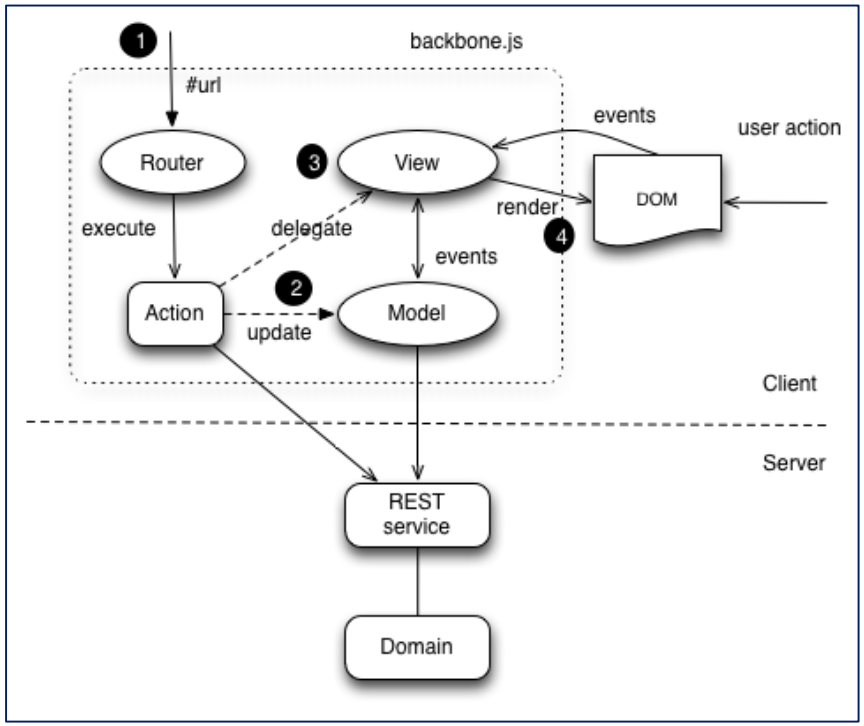
A pesar de los inconvenientes anteriores, se ha considerado que esta es la mejor opción para el propósito de la aplicación desarrollada, teniendo en cuenta que se quería integrar esta librería con Backbone.js.

BackBone.js 1.1.0

Backbone.js es una librería JavaScript pensada como interfaz JSON RESTful , basada en el paradigma de modelo-vista-controlador (MVC) en el cliente. Es extremadamente ligera, ya que solo depende de la librería Underscore.

Backbone.js se ha construido partiendo de la premisa en que toda la comunicación entre el cliente y el servidor se debe realizar mediante una API REST, pensando en que el navegador no ha de ser el único cliente que consuma esa API, ya que potencialmente pueden ser cualquier tipo de dispositivo, como teléfonos inteligentes, tabletas, relojes, otras aplicaciones, etc.

Backbone estructura el código de la aplicación cliente, partiendo de la anterior premisa de forma que las transiciones entre páginas, las vistas que se muestran, o la respuesta a acciones de usuario, recaen en el cliente, y simplemente se realizan peticiones al servidor para consumir o persistir datos.



Esquema de Backbone.js

El concepto principal que hay que seguir cuando se usa Backbone.js es conseguir que las vistas escuchen cambios en el modelo y reaccionen consecuentemente a esos cambios, de forma que actualicen la vista en función de los cambios recibidos.

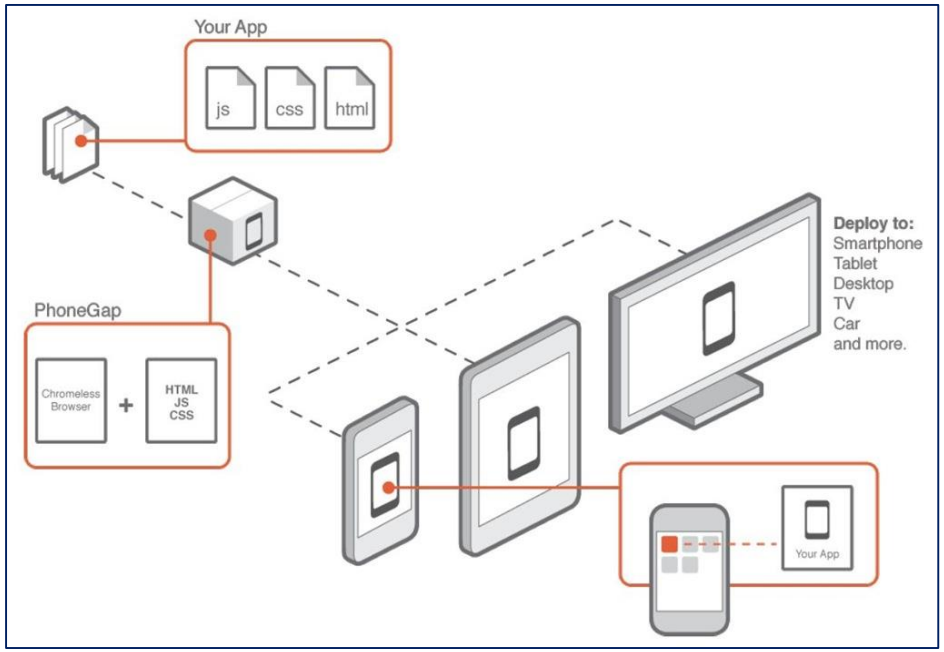
- Backbone.Model: Representación de un registro de dominio en la lógica de negocio. En nuestro caso serían entidades tales como Evento, Usuario, Imagen.
- Backbone.Collection: Un grupo o colección de modelos en la parte cliente que implementa lógica para ordenación, filtrado, adición o sustracción.
- Backbone.Router: Mapeo entre la URL introducida en el navegador / dispositivo y la función a ejecutar.
- Backbone.View: Vista o pieza de interfaz de usuario (UI) lógica y reutilizable. Puede estar asociada a un modelo o estar compuesta por varios.

Esta filosofía encaja perfectamente con la idea de aplicación a desarrollar, y por este motivo ha sido elegida.

PhoneGap 2.9.0

PhoneGap es un framework gratuito y código abierto que permite crear aplicaciones móviles multiplataforma a partir de tecnologías web como HTML5, CSS3 y JavaScript. PhoneGap está realizado sobre **Apache Cordova**, y actualmente es propiedad de Adobe.

Este framework permite crear aplicaciones híbridas, a mitad de camino entre las nativas y las puramente web, esto se consigue a partir de una serie de APIs que permiten controlar características de los dispositivos como el acelerómetro, la cámara, las notificaciones o la geolocalización



Esquema funcionamiento de PhoneGap

PhoneGap es básicamente una arquitectura de ejecución de aplicaciones Web bajo una capa que permite interactuar con funcionalidades nativas del OS en el que se ejecute.

En **eventual** se ha utilizado este framework únicamente para el acceso a la cámara del dispositivo y para navegar entre las imágenes almacenados en el dispositivo.

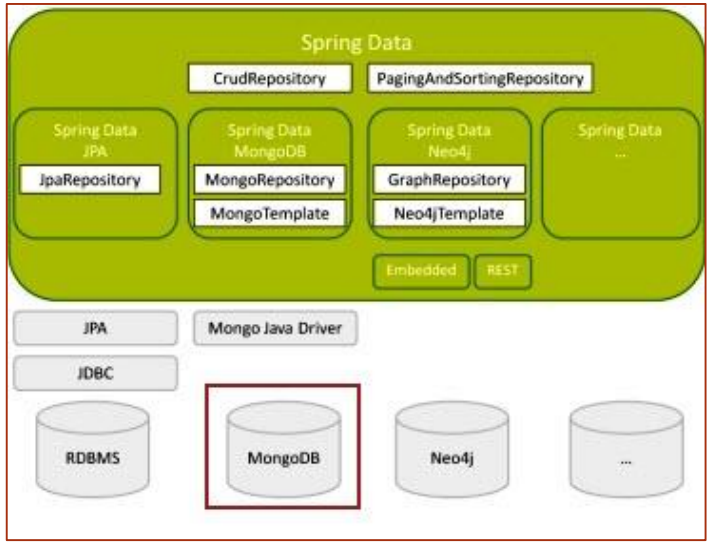
Spring MVC | Spring Data MongoDB

Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto desarrollado principalmente para la plataforma Java.

Spring MVC es parte del framework **Spring** e implementa el paradigma MVC en la parte servidor. Este framework se encarga de recibir una petición y resolverla mostrando la vista correspondiente, a partir de los datos del modelo.

En nuestra aplicación se ha utilizado esta tecnología para implementar totalmente la API REST. La aplicación cliente móvil realiza una petición al servidor, y Spring MVC se encarga de devolver una respuesta JSON a la petición solicitada. Los datos obtenidos están almacenados, a su vez, en la base de datas de documentos MongoDB.

Spring Data para **MongoDB** es parte del proyecto Spring Data, que persigue proporcionar un modelo único y consistente de programación basada en Spring para proporcionar acceso a las nuevas bases de datos, al mismo tiempo que mantiene las características propias de las mismas.

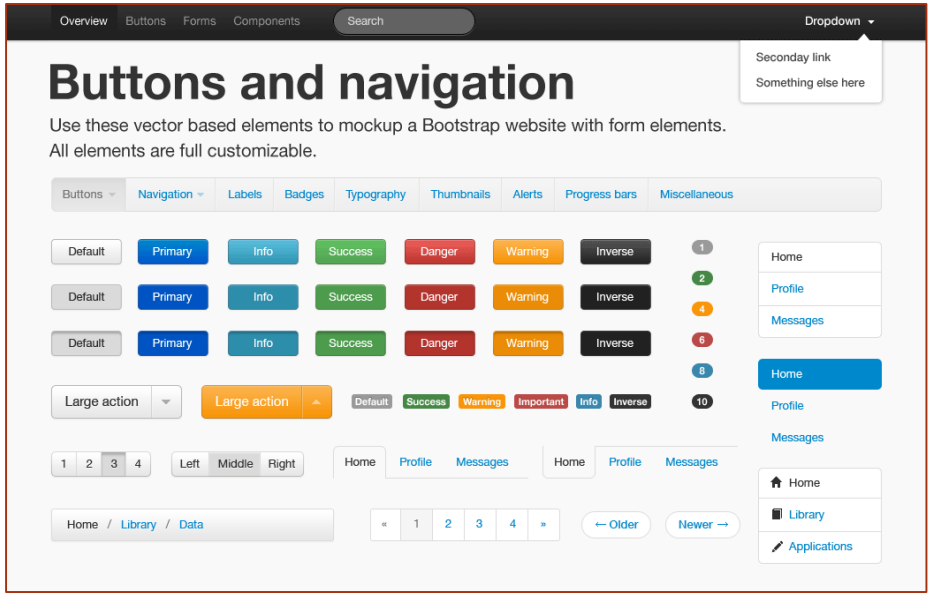


Spring Data para Mongo DB, proporciona una integración con la base de datos de documentos MongoDB, y esta es la librería utilizada para acceder a la base de datos.

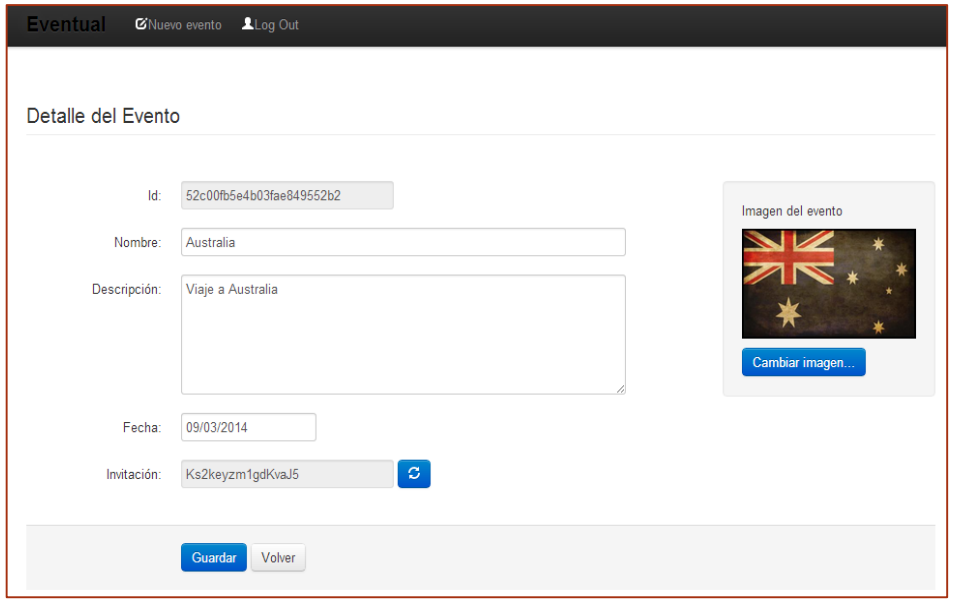
Twitter Bootstrap

Twitter Bootstrap es un framework que simplifica el proceso de creación de diseños web combinando CSS y JavaScript. Ha sido desarrollado por Twitter, y la mayor ventaja que aporta es que se pueden crear interfaces que se adapten a los distintos navegadores (diseño responsivo), de forma sencilla y rápida.

Se ha utilizado este framework en este proyecto para el desarrollo de los componentes de la versión web de la aplicación.

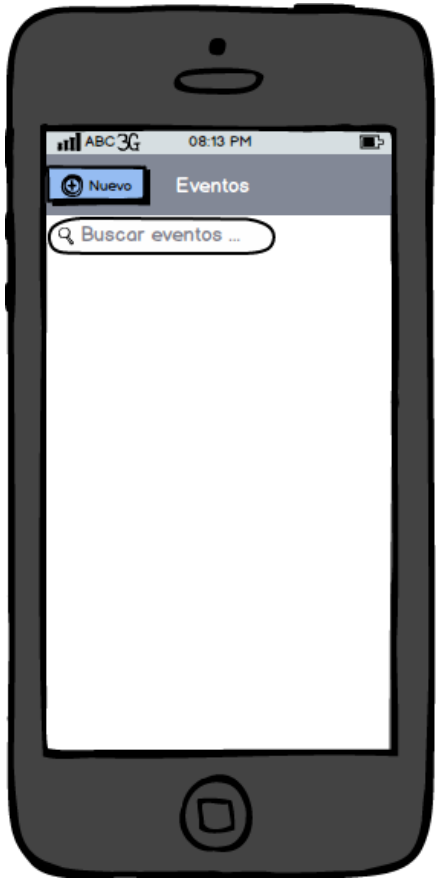


Ejemplo de componentes gráficos Twitter Bootstrap



Ejemplo de componentes gráficos Eventual

Interfaz Móvil: Listado de eventos sin usuario



Pantalla: Listado de eventos

Pantalla inicial de la aplicación en la que se mostrará un listado de eventos coincidentes con la consulta efectuada en el buscador.

Todos los eventos mostrados tendrán como participante o creador al usuario que está autenticado en el dispositivo.

Pantalla: Login

Usuario no creado

En el caso que el usuario no esté creado, ya sea porque es la primera vez que accede a la aplicación o porque se han eliminado los datos de las credenciales de usuario en el dispositivo, se mostrará una pantalla de login en la que se deberán introducir los datos correctos.

También se incluirá un botón de "Nuevo usuario" que enviará al usuario a la pantalla de creación de usuario.

Interfaz Móvil: Creación de usuario / Listado de eventos

Pantalla: Creación de usuario

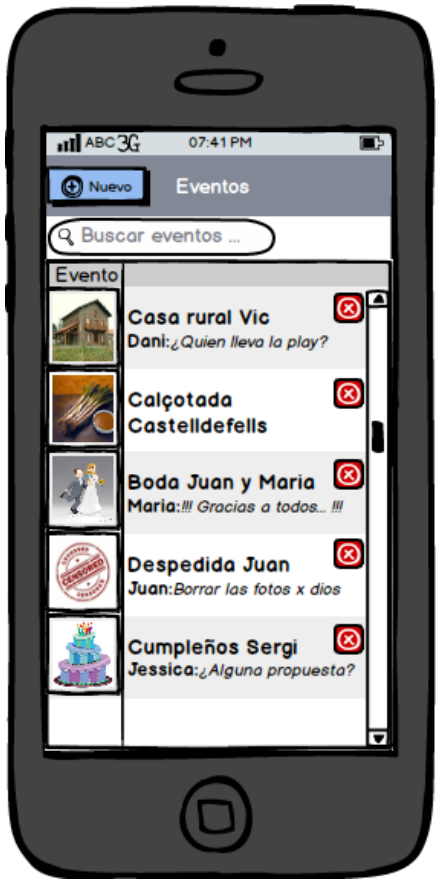
Pantalla para la creación del usuario.

Constará de tres campos de texto correspondientes a la dirección de email, nombre de usuario y password.

Opcionalmente el usuario podrá añadir una imagen a su usuario.

El botón "volver" situado en la esquina superior izquierda permitirá al usuario volver a la pantalla inicial sin guardar los cambios.

El botón guardar, volverá a la pantalla inicial una vez validados y guardados los cambios.



Pantalla: Listado de eventos

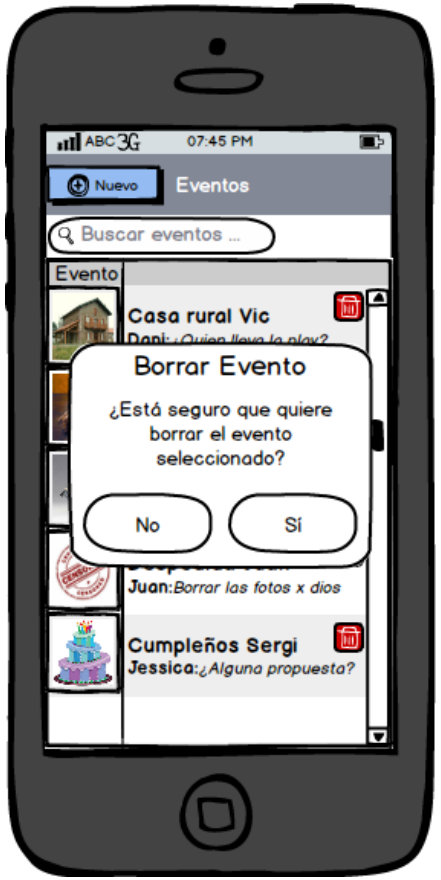
Usuario creado

Se mostrará el listado de eventos correspondientes a los criterios de la búsqueda.

Estos eventos contendrán al usuario logado como uno de los participantes.

El listado será parte de un control con scroll y cada uno de los elementos del listado consistirá en la imagen del evento, el nombre, el último comentario realizado en el mismo y un botón para eliminar el evento.

Interfaz Móvil: Borrado de eventos / Chat



Pantalla: Borrado de evento

El usuario sólo podrá borrar los eventos creados por él mismo.

En el caso que pulse el botón borrar de algún ítem del listado se le preguntará al usuario si desea borrar el evento.

Esto se realizará mediante un diálogo emergente.

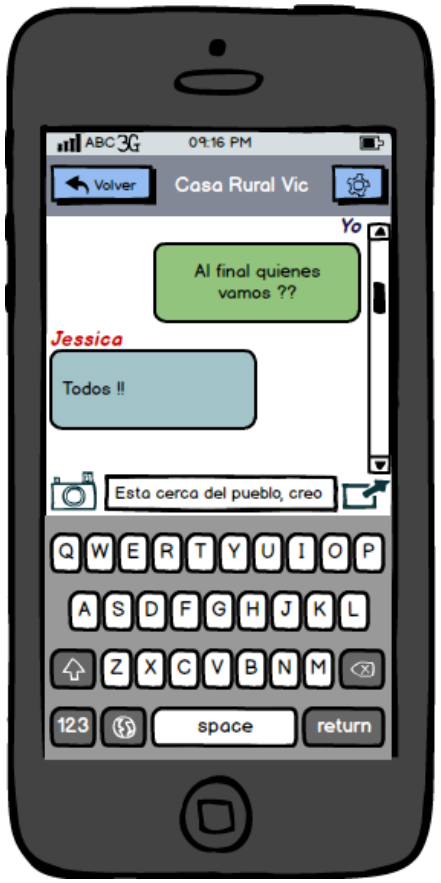
Pantalla: Chat

Desde la pantalla "Listado de eventos", al pulsar sobre cualquier ítem del listado, se accederá al chat del evento.

En esta pantalla se mostrará los últimos comentarios/imágenes realizados por los participantes del evento.

En la parte superior de la pantalla se permitirá volver al menú de eventos o visualizar la información del evento. Los mensajes aparecerán alineados a la derecha o izquierda en función del creador del comentario. En la parte inferior habrá habilitado un campo de texto para añadir comentarios.

Interfaz Móvil: Añadir comentario / Nueva fotografía



Pantalla: Añadir comentario

Al posicionarse sobre el campo de texto para incluir el comentario, se mostrará el teclado virtual del dispositivo móvil.

Al introducir el texto y pulsar sobre el botón "enviar", situado en la parte superior derecha del teclado, el comentario se añadirá al listado de comentarios del evento.

Pantalla: Nueva fotografía

Si en lugar de enviar el comentario, pulsamos en el botón de la cámara situado en la parte inferior izquierda de la pantalla "chats", se mostrará esta pantalla en la que se podrá añadir el comentario y seleccionar una fotografía para añadir al evento.

Interfaz Móvil: Origen nueva fotografía / Comentarios fotografía

Pantalla: Origen fotografía

Al pulsar en el botón editar, situado en la parte inferior derecha dela fotografía se podrá añadir o editar la fotografía.

Al realizar esta acción se preguntará al usuario por el origen de la imagen, si será una fotografía existente en el dispositivo móvil, o se usará la cámara en este momento para obtener la imagen.

Esto se realizará mediante un menú emergente.

Pantalla: Comentarios imagen

Desde la pantalla "Chat" al pulsar sobre cualquier imagen del evento, se mostrará una pantalla con los comentarios realizados a una imagen.

En la parte inferior de la pantalla, habrá un campo de texto, donde se podrá añadir un nuevo comentario a esta imagen.

Interfaz Móvil: Comentar Imagen/Detalle evento

Pantalla: Comentar Imagen

Desde la pantalla "Comentarios Imagen", al posicionarse sobre el campo de texto para incluir el comentario, se mostrará el teclado virtual del dispositivo móvil.

Al introducir el texto y pulsar sobre el botón "enviar", situado en la parte superior derecha del teclado, el comentario se añadirá al listado de comentarios de la imagen.

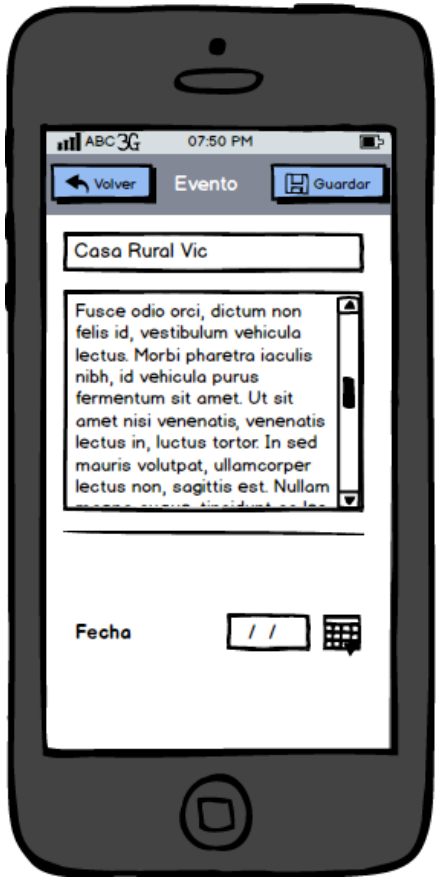
Pantalla: Detalle evento

Desde la pantalla "Chat" al pulsar el botón "Info" situado en la parte superior derecha de la pantalla se accederá a la pantalla "Detalle del evento" en el que se mostrará la siguiente información del mismo:

- Imagen editable del evento.
- Descripción del evento.
- Fecha del evento
- Imágenes existentes en el evento.

Conjunto de usuarios participantes en el evento. Desde este listado se podrán eliminar o añadir nuevos participantes al evento, siempre que el usuario logado sea el administrador.

Interfaz Móvil: Editar Evento/Buscar usuarios



Pantalla: Editar evento

Desde la pantalla "Detalle de evento" al pulsar sobre el botón editar, se podrá acceder a la pantalla de edición del evento.

En esta pantalla se podrá editar el nombre del evento y la descripción.

También se podrá modificar la fecha.

Pantalla: Buscar usuario

Desde la pantalla "Detalle de evento" al pulsar sobre el botón añadir usuario, se podrá acceder a la pantalla de búsqueda de usuarios.

En este listado se mostrarán todos los usuarios de la aplicación que cumplan con los criterios de búsqueda del buscador situado en la parte superior de la pantalla y que no estén ya incluidos en el evento actual.

Al seleccionar un usuario este se incluirá en la lista de participantes del evento.

Interfaz Móvil: Galería imágenes/ Detalle imagen

Pantalla: Galería imágenes

Desde la pantalla de detalle de un evento se puede acceder al listado de todas las imágenes de un evento.

En esta pantalla se mostrará un listado de imágenes pertenecientes a un evento.

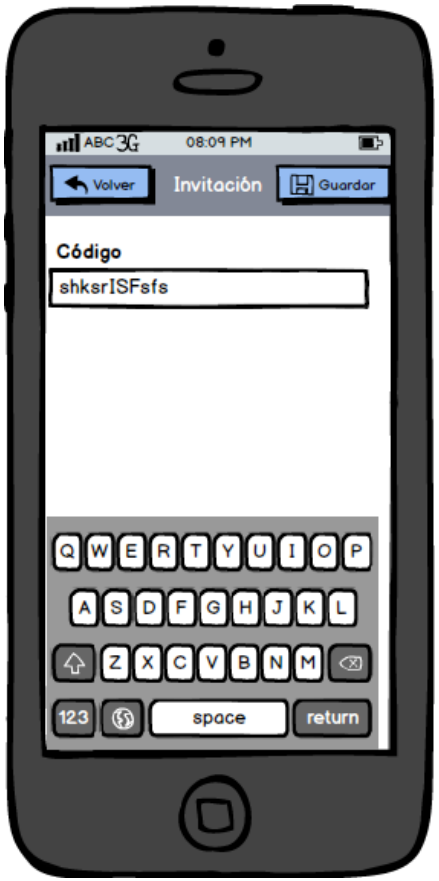
Si se pulsa en el icono "comentarios" de una imagen se podrá acceder a la pantalla de "Comentar fotografía".

Si se pulsa sobre una imagen, se mostrará el detalle de esta imagen.

Pantalla: Detalle imagen

Desde la galería de imágenes, pulsando sobre una imagen determinadas, podrá visualizar el detalle de esa imagen en una galería/ carrusel.

Interfaz Móvil: Invitación



Pantalla: Código invitación

Desde la pantalla de "Invitación" se podrá añadir un código generado previamente en la versión web de la aplicación.

Si este código es válido el usuario se auto asignará al evento al cual pertenece el token generado.

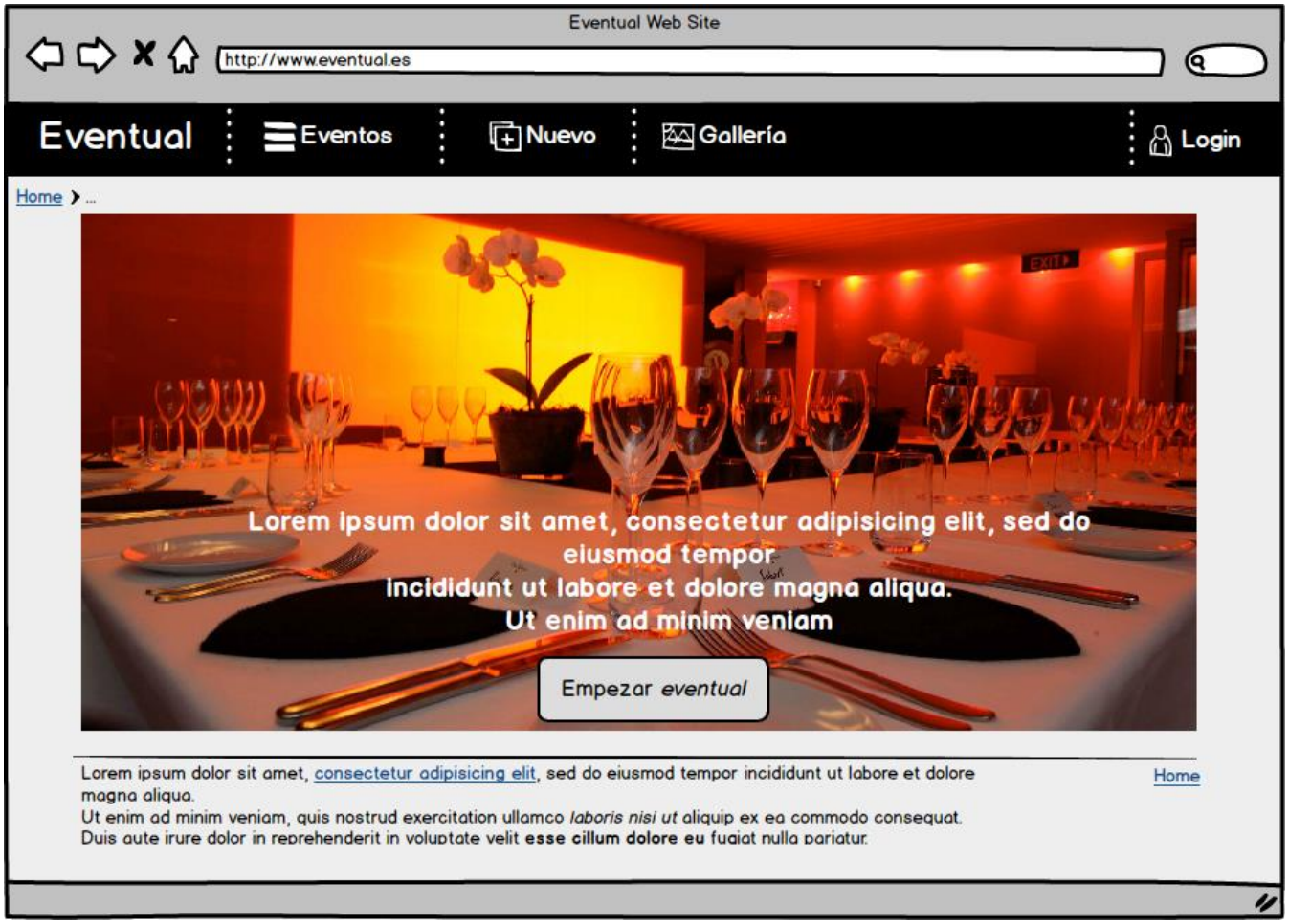
Interfaz Web: Home

Pantalla: Página principal

La aplicación dispondrá de una web en la que se podrá visualizar parte de la información que los usuarios de la aplicación móvil van generando en sus diferentes eventos.

En la pantalla principal, *Home*, se podrá visualizar la página inicial de la aplicación, en la que se mostrará una imagen de fondo y un gran botón instando al usuario a iniciar el recorrido por los eventos.

En la parte superior de la pantalla se podrá ver las principales áreas de la Web: Eventos, galería de imágenes y página de login.



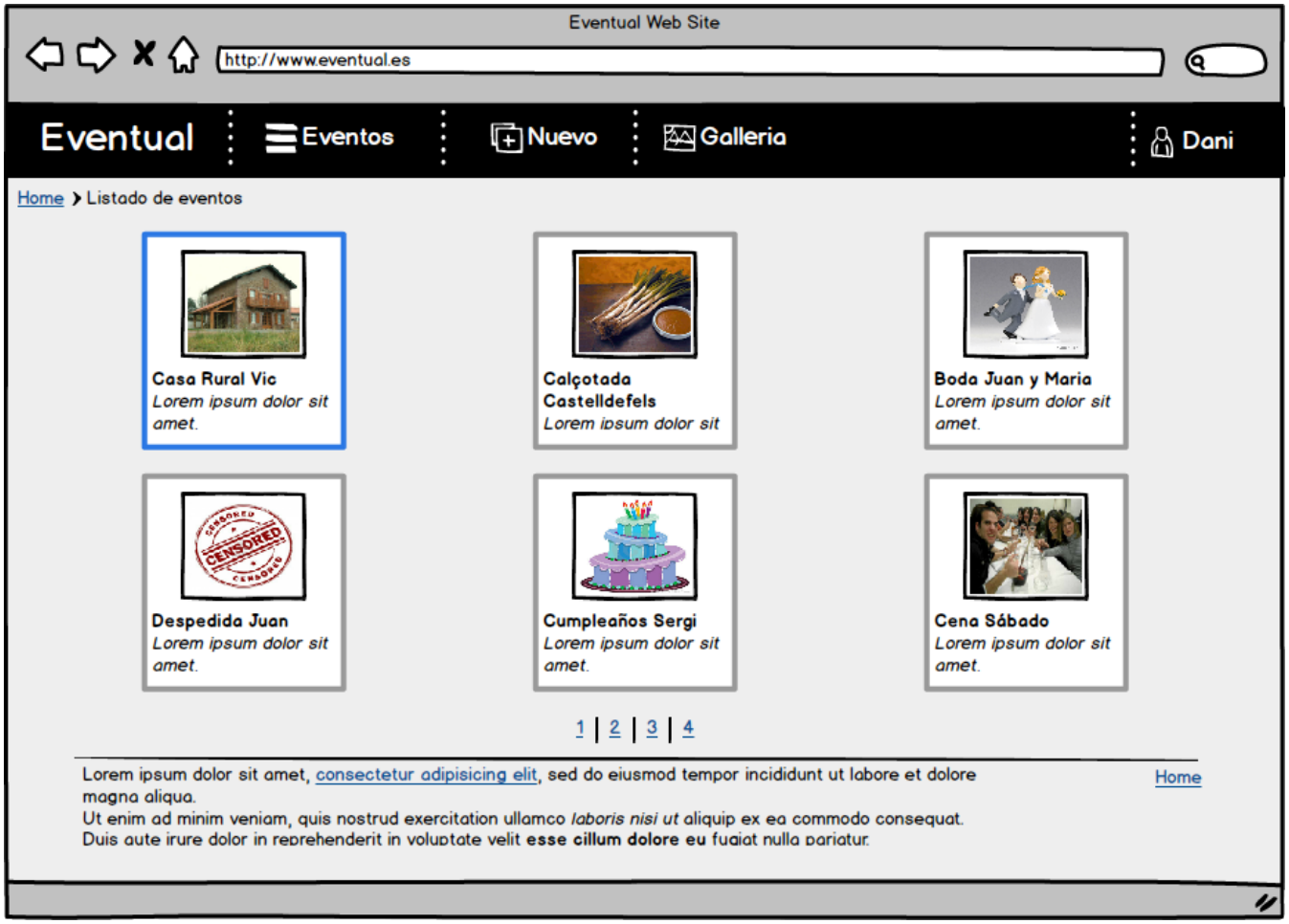
Interfaz Web: Listado de eventos

Pantalla: Listado de eventos

Desde la pantalla principal, pulsando en el menú "Eventos" situado en la parte superior de la pantalla se podrán visualizar la información más visual de los eventos asociados al usuario logado: Imagen, nombre y descripción.

En la parte inferior de la pantalla habrá una pequeña paginación para que el usuario pueda desplazarse fácilmente entre los diferentes eventos.

Cada imagen será linkable y enviará al detalle del evento.



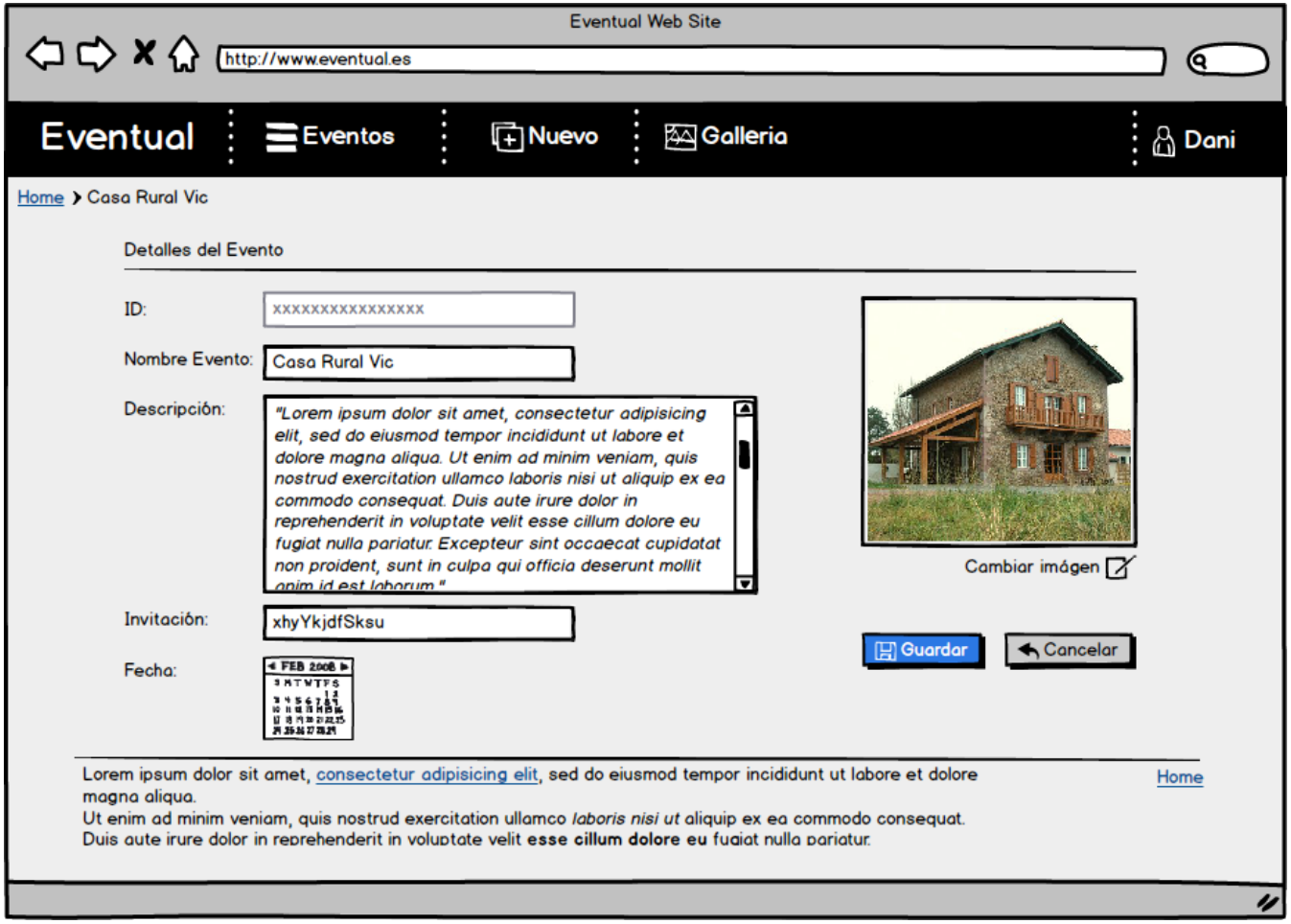
Interfaz Web: Detalle de evento

Pantalla: Detalle de evento

Desde el listado de eventos, al pulsar sobre cualquier evento se podrá acceder al detalle del mismo.

En función del rol del usuario en el evento podrá modificar información.

La información mostrada será muy similar a la pantalla de detalle de evento de la aplicación móvil, nombre, descripción, fecha, imagen principal y datos del evento.



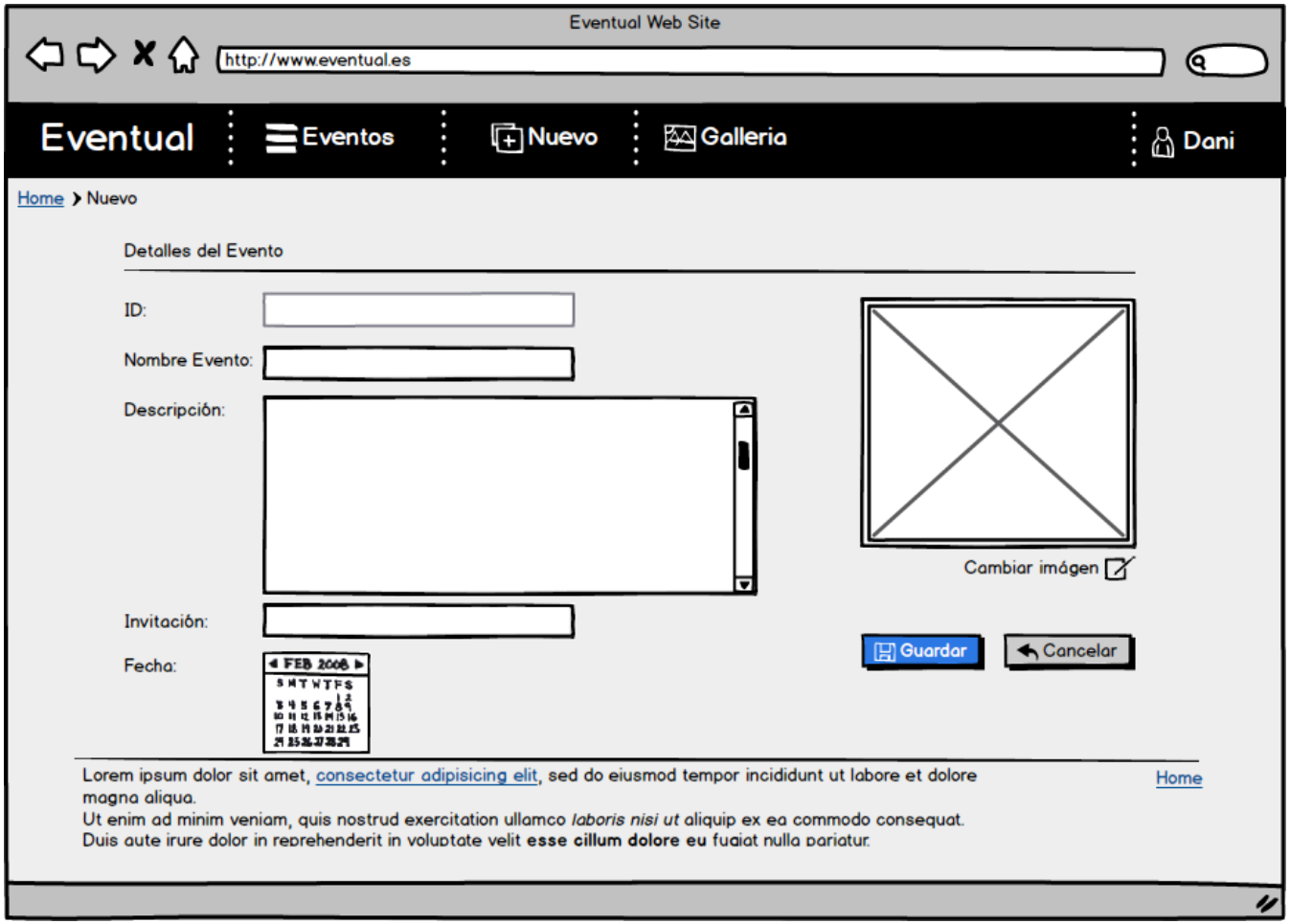
Interfaz Web: Nuevo evento

Pantalla: Nuevo evento

Desde el menú superior de la aplicación web, se habilitará un link que permitirá la creación de un nuevo evento.

En esta pantalla se podrá crear la estructura básica para que los usuarios desde el móvil puedan subir sus comentarios y fotografías.

La información a crear es la misma que la que se puede visualizar en el detalle del evento: nombre, descripción, fecha, imagen principal y datos del evento.



Interfaz Web: Imágenes del evento

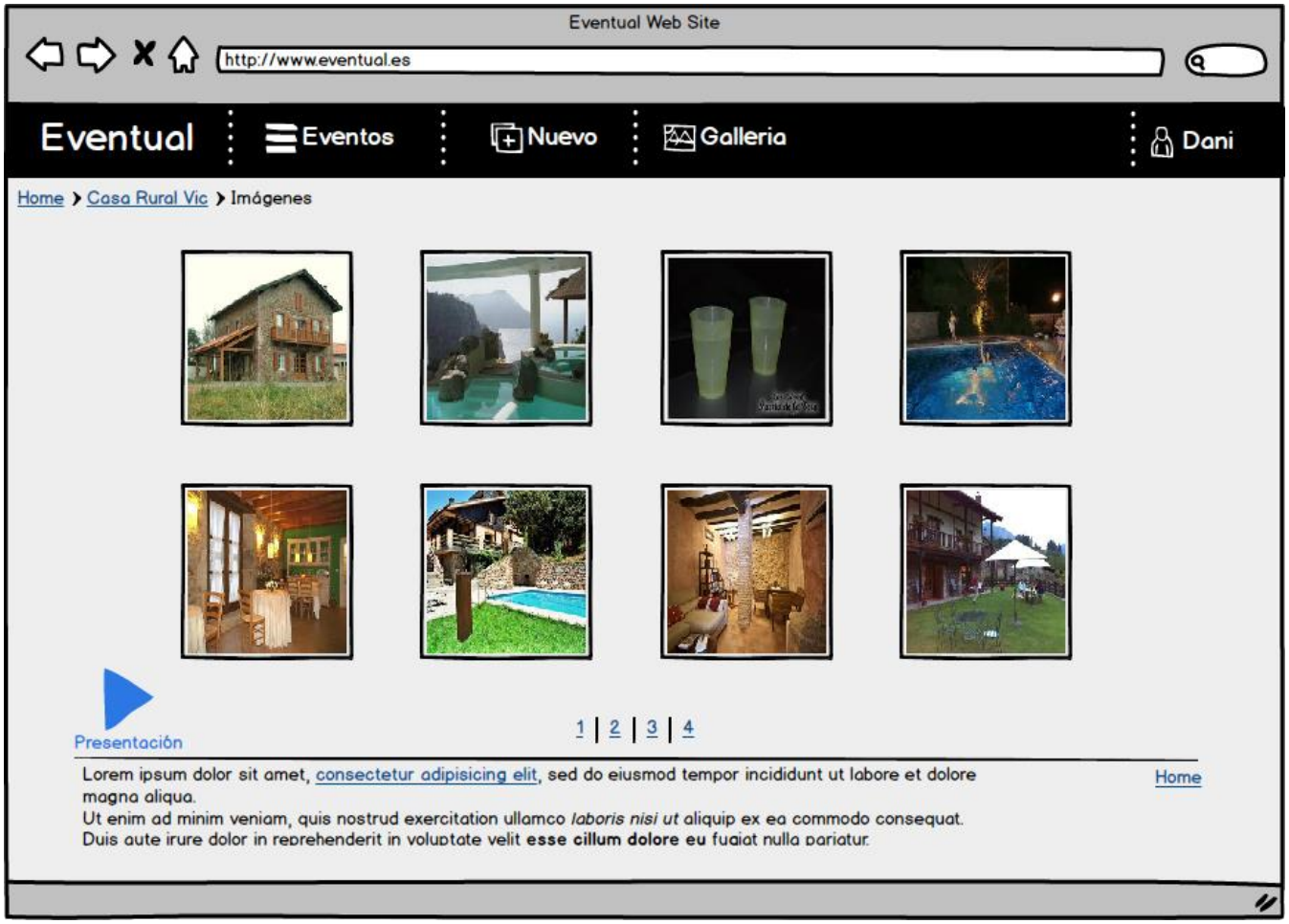
Pantalla: Imágenes del evento

Desde la pantalla de listado de eventos al pulsar sobre la imagen de cualquier evento se podrá visualizar el listado de imágenes que se han añadido al evento.

En la parte inferior de la pantalla habrá una pequeña paginación para que el usuario pueda desplazarse fácilmente entre las diferentes imágenes.

Cada imagen será linkable y enviará al detalle de la imagen.

En la parte inferior izquierda de la pantalla se habilitará un botón, play, para poder visualizar la presentación de las imágenes en pantalla completa.

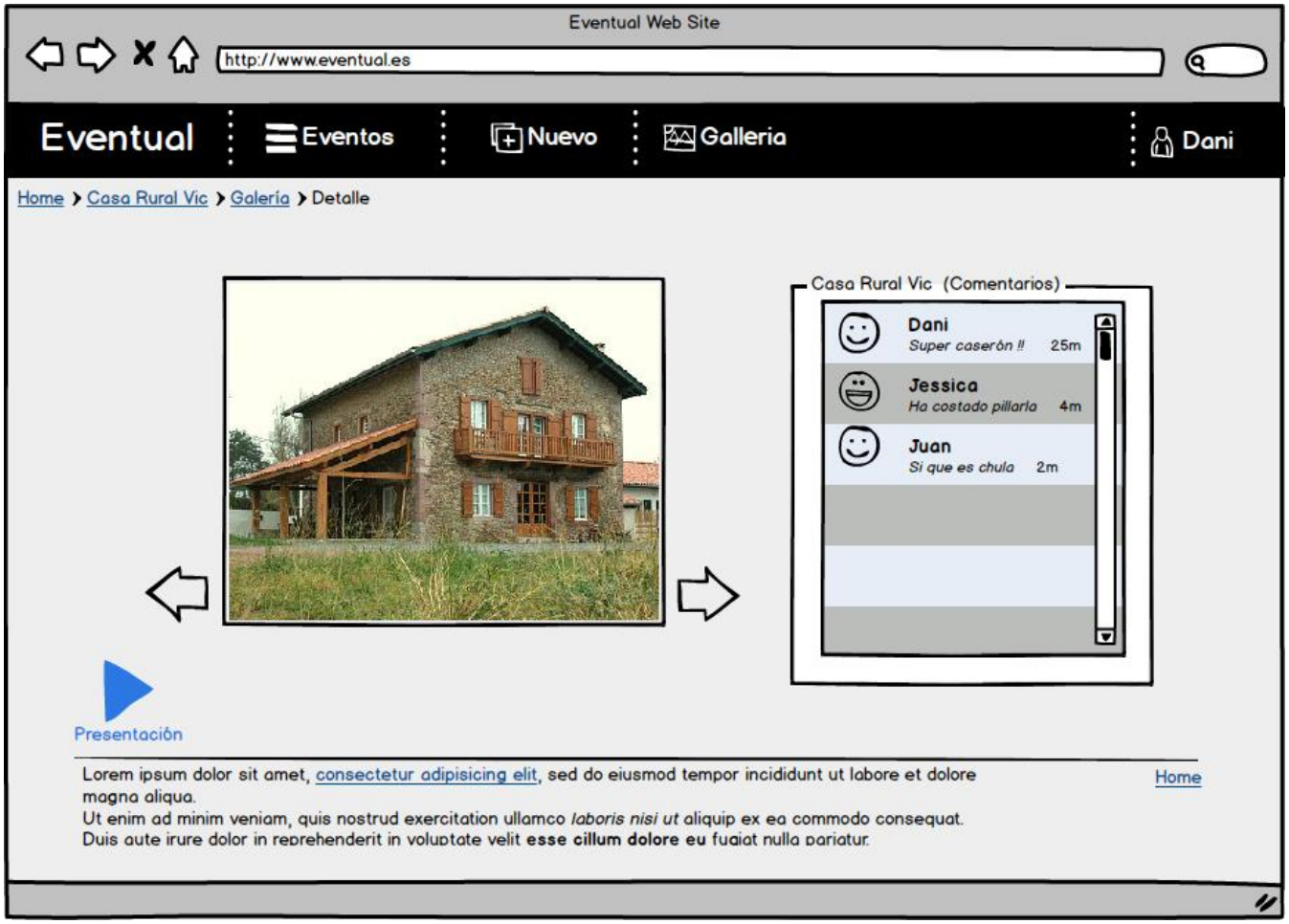


Interfaz Web: Detalle de imagen

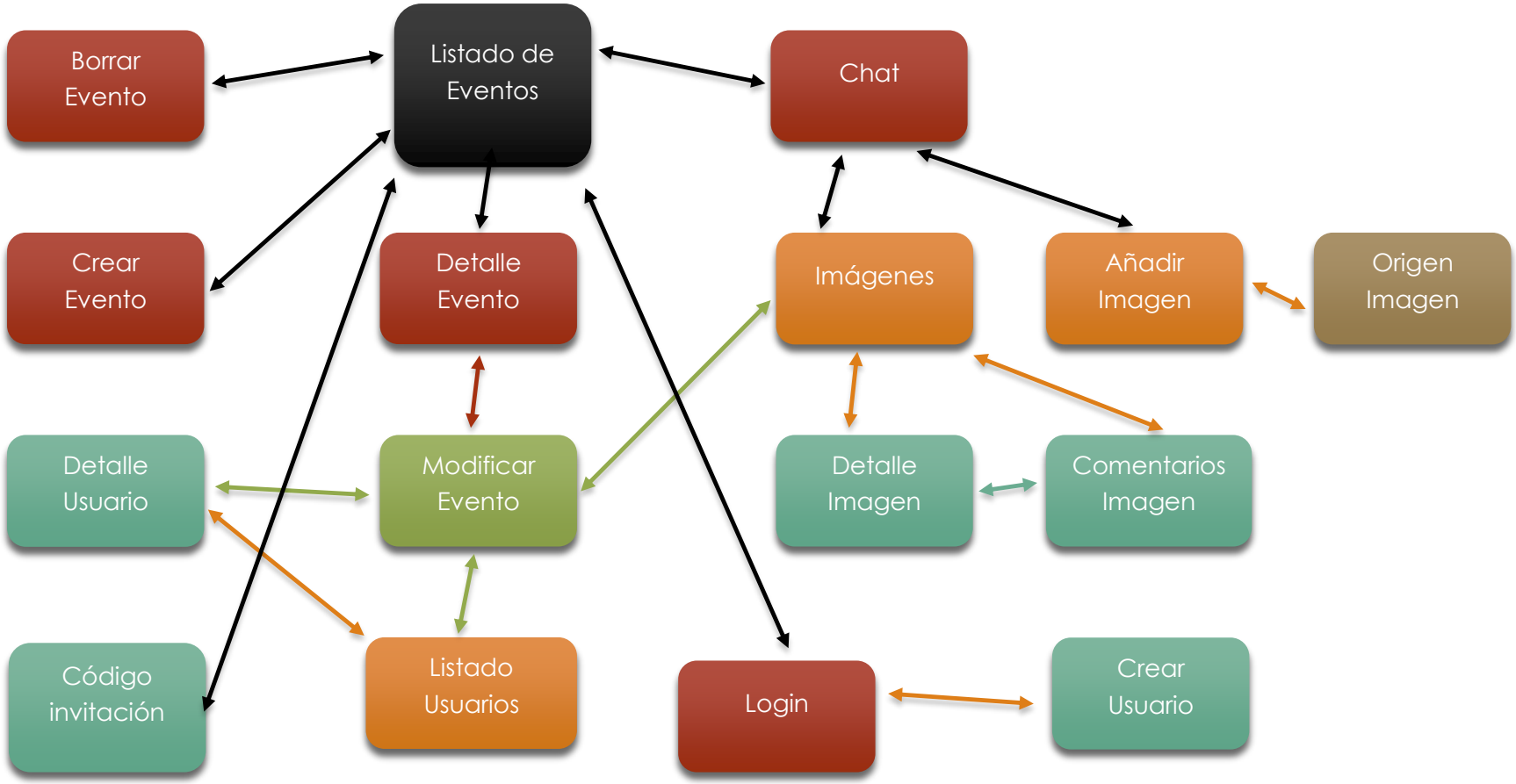
Pantalla: Detalle de imagen

En esta pantalla se podrá visualizar la imagen en mayor tamaño y se podrá visualizar los comentarios creados por los usuarios.

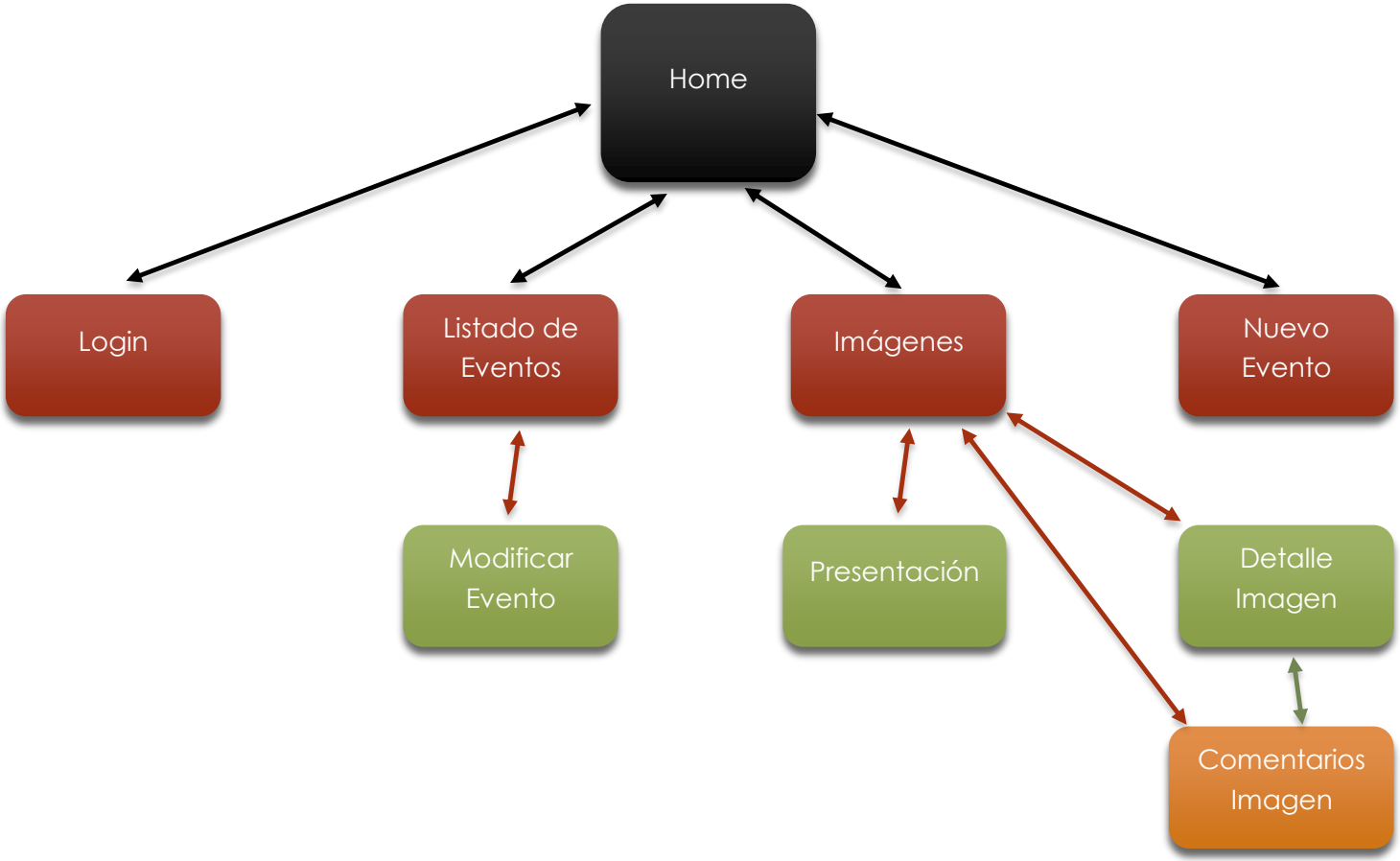
En la parte inferior izquierda de la pantalla se habilitará un botón, *play*, para poder visualizar la presentación de las imágenes en pantalla completa.



9. Análisis y diseño: Flujo navegación Móvil



Análisis y diseño: Flujo navegación Web



10. Análisis y diseño: Historias de Usuario: Aplicación Móvil

A continuación se exponen de forma ordenada, de más prioritaria a menos, las historias de usuarios clave que servirán para mostrar la funcionalidad de la aplicación. Para cada una de las historias se incluyen la carga de trabajo necesaria que se ha estimado para implementarla y que ésta funcione correctamente.

Título: Listado de Eventos	**Móvil**
Historia: *Como usuario puedo acceder a mi listado de eventos, y poder ver los eventos disponibles en la aplicación*	
Pantallas: Listado de eventos	
Pruebas Validación: • Validar que el usuario puede listar los eventos • Validar que los datos del listado son correctos	
Carga de trabajo: 1 Semana	**Prioridad:** Alta

Título: Crear Evento	**Móvil**
Historia: *Como usuario quiero crear un evento y poder elegir qué usuarios pueden acceder a él. Podré elegir de forma fácil quienes serán los participantes, de entre los usuarios existentes en la aplicación.*	
Pantallas: Creación evento, Crear Usuario, Listado de Usuarios	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de creación de usuario desde el listado de eventos. • Validar que los datos se visualizan y almacenan correctamente. • Validar que el usuario puede añadir un nuevo usuario accediendo al listado de usuarios • Validar que el usuario se añade a la lista de participantes del evento • Validar que al volver al listado de eventos, éste contiene el nuevo evento recién creado.	
Carga de trabajo: 2 Semanas	**Prioridad:** Alta

Título: Crear Usuario	**Móvil**
Historia: *Como usuario todavía no registrado a la aplicación quiero poder crear mi usuario y registrarme de forma sencilla mediante un nombre de usuario y dirección de correo.*	
Pantallas: Crear Usuario	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de creación de usuario. • Validar que los datos modificados se almacenan correctamente. • Validar que el usuario creado aparecerá en el listado de usuarios.	
Carga de trabajo: 1 Semana	**Prioridad:** Alta

Título: Modificar Evento	**Móvil**
Historia: *Como usuario poder modificar datos de un evento, añadir nuevos participantes o borrar existentes.*	
Pantallas: Detalle de Evento, Modificar Evento, Listado de Usuarios	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de detalle de usuario desde el listado de eventos. • Validar que los datos modificados se almacenan correctamente. • Validar que el usuario puede añadir un nuevo usuario accediendo al listado de usuarios • Validar que el usuario se añade a la lista de participantes del evento • Validar que al volver al listado de eventos, éste contiene los datos modificados.	
Carga de trabajo: 1 Semana	**Prioridad:** Alta

Título: Compartir fotos / comentarios	**Móvil**
Historia: *Como participante de un evento quiero poder compartir fotos y comentarios de forma fácil con el resto de participantes.*	
Pantallas: Chat, Imágenes, Añadir Imagen, Origen Imagen, Comentarios Imagen	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de Chat, y se muestran los comentarios y fotografías de los participantes del evento. • Validar que el usuario puede añadir nuevos comentarios y estos se visualizan en el chat. • Validar que el usuario puede añadir una nueva fotografía. • Validar que la fotografía se visualiza en la pantalla de chat. • Validar que el usuario puede añadir comentarios a fotografías existentes.	
Carga de trabajo: 5 Semanas	**Prioridad:** Alta

Título: Presentación Imágenes	**Web**
Historia: *Como participante o creador de un evento quiero poder visualizar todas las imágenes de un evento en modo presentación.*	
Pantallas: Imágenes, Presentación	
Pruebas Validación: • Validar que las imágenes mostradas pertenecen al evento seleccionado • Validar que las imágenes se visualizan de forma correcta	
Carga de trabajo: 1 Semana	**Prioridad:** Alta

Título: Listado de Eventos	**Web**
Historia: *Como usuario puedo acceder a un listado de eventos y poder ver los eventos disponibles en la aplicación.*	
Pantallas: Listado de eventos	
Pruebas Validación: • Validar que el usuario puede listar los eventos • Validar que los datos del listado son correctos	
Carga de trabajo: 1 Semana	**Prioridad:** Media

Título: Crear Evento	**Web**
Historia: *Como usuario quiero crear un evento y poder elegir qué usuarios pueden acceder a él.*	
Pantallas: Creación evento	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de creación de usuario desde el listado de eventos. • Validar que los datos se visualizan y almacenan correctamente.	
Carga de trabajo: 1 Semana	**Prioridad:** Media

Título: Modificar Evento	**Web**
Historia: *Como usuario poder modificar datos de un evento.*	
Pantallas: Detalle de Evento	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de detalle de usuario desde el listado de eventos. • Validar que los datos modificados se almacenan correctamente. • Validar que al volver al listado de eventos, éste contiene los datos modificados.	
Carga de trabajo: 1 Semana	**Prioridad:** Baja

Título: Usar token invitación	**Web**
Historia: *Como usuario quiero utilizar un token o código de invitación generado previamente.*	
Pantallas: Invitaciones	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de invitaciones. • Validar que los una vez introducido el token generado en la versión web, el usuario se añadirá al evento correspondiente.	
Carga de trabajo: 1 Semana	**Prioridad:** Media

Título: Generar invitación Evento	**Web**
Historia: *Como usuario quiero poder generar un "token" de invitación a un evento concreto para ser usado por cualquier usuario en la versión móvil.*	
Pantallas: Detalle de Evento	
Pruebas Validación: • Validar que el usuario puede acceder a la pantalla de detalle de evento desde el listado de eventos. • Validar que los datos modificados se almacenan correctamente.	
Carga de trabajo: 1 Semana	**Prioridad:** Media

Diagrama clases API REST

La comunicación que se ha realizado entre la aplicación móvil y web, se realizará mediante llamadas JSON a una API REST que encapsulará las acciones necesarias para realizar el CRUD de las diferentes entidades.

La tecnología elegida para la parte servidor es Spring MVC y mediante los siguientes controladores y sus correspondientes mapeos entre la URL solicitada y el método a ejecutar se gestionarán todas las peticiones.

Cada controlador tiene sus mapeos específicos, pero delega los propios de una arquitectura REST a **RestController.**

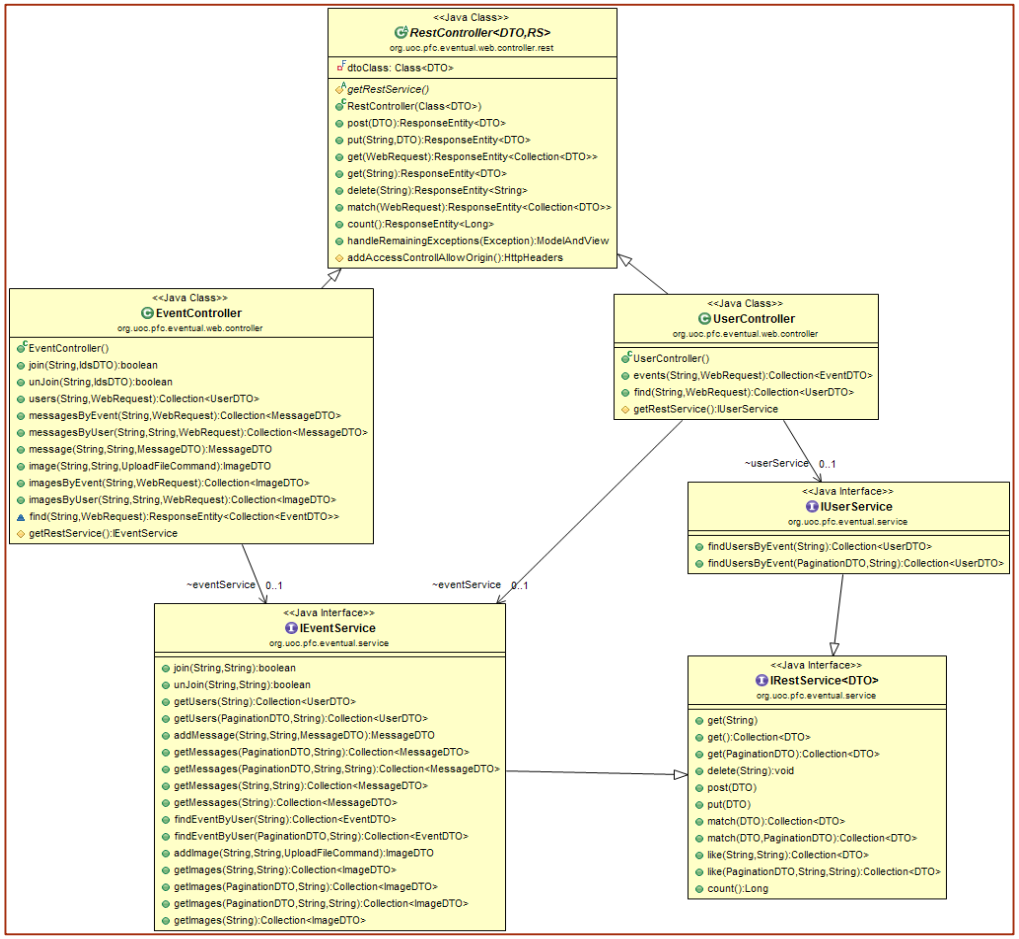


Diagrama de clases API REST

Diagrama clases modelo

Estas clases representan las entidades principales del modelo de datos de la aplicación.

Como BD se ha elegido **MongoDB** que es un BD **noSQL**, con lo que el modelado no es relacional.

Las principales entidades son Evento, Usuario, Mensaje e Imagen.

Las relaciones entre ellas variaran en función de si son relaciones o no. Habrá una relación por referencia entre *Usuario* y *Evento*, para modelar una relación **M:N**, mientras que en el caso de *Evento* y *Mensaje* se utilizará un entidad embebida, para modelar una relación **1:N**

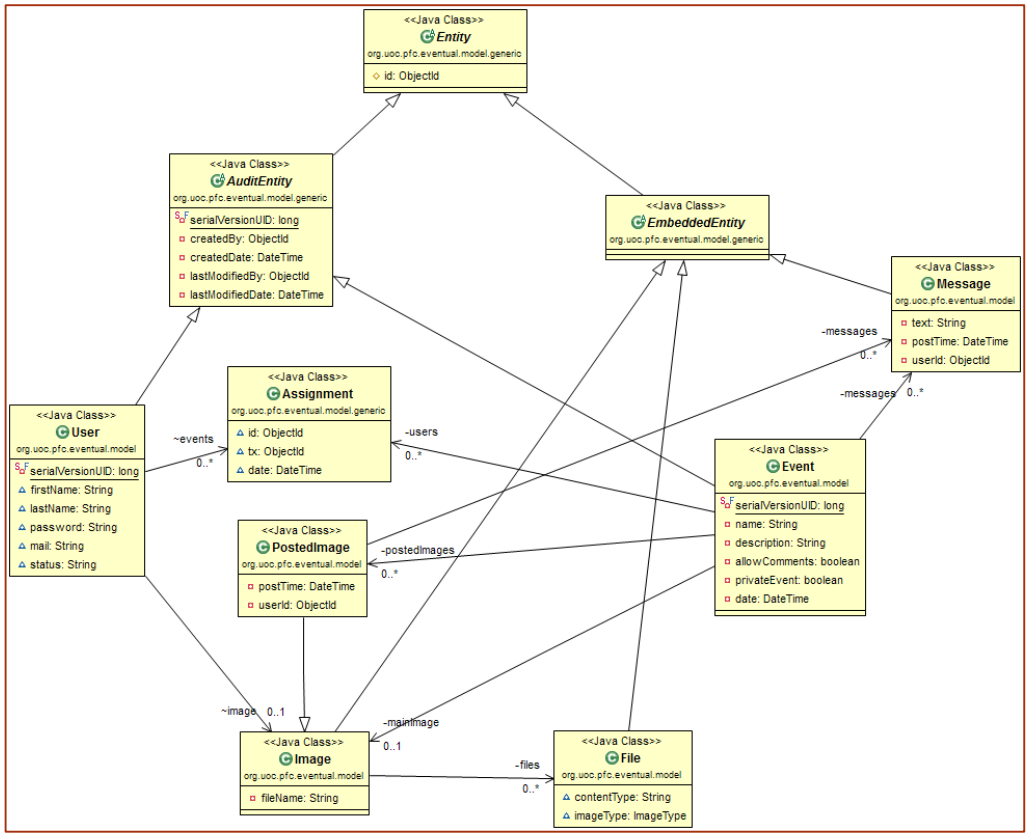


Diagrama de clases entidades modelo

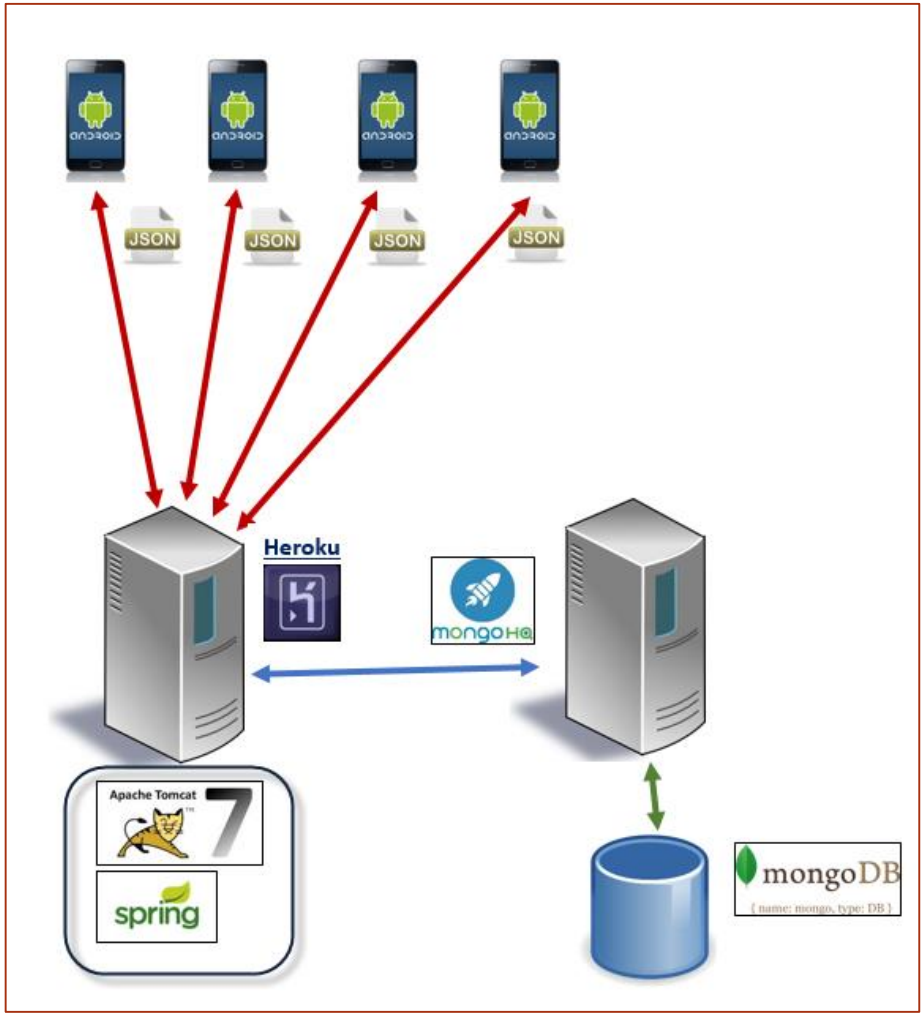
12. Arquitectura física de Eventual

La arquitectura física de la aplicación se muestra en la siguiente imagen:

La aplicación está compuesta por varios **clientes móviles** que se ejecutan en dispositivos **Android**. Estos clientes intercambian documentos JSON con el servidor.

El servidor que intercambia los datos con los clientes móviles está implementado íntegramente en **Java** utilizando la librería **Spring**. Está instalado en un contenedor de Servlets **Apache Tomcat**, que a su vez está alojado en el servicio de computación en la nube **Heroku**.

Heroku se comunicará con el servicio **MongoHQ** para realizar consultas y persistir los datos en la base de datos **MongoDB**, alojada por este último servicio.



Esquema arquitectura física Eventual

13. Implementación

Punto de entrada principal de la aplicación cliente

```
package com.uoc.pfc.eventual;

import org.apache.cordova.DroidGap;

import android.os.Bundle;
import android.view.Menu;

import com.uoc.pfc.eventual.ws.WebSocketFactory;

public class MainActivity extends DroidGap {

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		super.setIntegerProperty("loadUrlTimeoutValue", 70000);
		super.loadUrl("file:///android_asset/www/index.html");

		// attach websocket factory
		appView.addJavascriptInterface(new WebSocketFactory(appView),
		"WebSocketFactory");
	}

	@Override
	public boolean onCreateOptionsMenu(Menu menu) {
		getMenuInflater().inflate(R.menu.main, menu);
		return true;
	}
}
```

El punto de entrada principal de la aplicación móvil está implementado por la siguiente clase Java. La aplicación cliente está compuesta principalmente por una actividad Android que crea un componente web sobre el cual renderizará el HTML correspondiente al fichero *file:///android_asset/www/index.html*. Este fichero es realmente el que contiene el grueso de la lógica de la aplicación, y son los ficheros JavaScript referenciados en él y el uso de librerías como BackBone y JqueryMobile, los que hacen todo el trabajo.

En la imagen se puede ver también como en lugar de heredar de la clase normal de Android, se está heredando de la clase DroidGap, la cual pertenece a la librería PhoneGap como se puede ver en la sección de importación de clases:
import org.apache.cordova.DroidGap;

Se ha aumentada el tiempo de carga de la aplicación hasta 70000 milisegundos para evitar errores en la aplicación al ser ejecutada en máquinas o emuladores demasiado lentos. Esto le dará más tiempo a PhoneGap para cargar las librerías necesarias.

Cabe comentar que Eventual utiliza WebSockets para la comunicación en tiempo real entre los diferentes clientes. Esta tecnología requiere de una implantación en la parte servidor y en la cliente. Android no soporta esta tecnología por defecto y se ha tenido que implementar ad-hoc:
appView.addJavascriptInterface(new WebSocketFactory(appView), "WebSocketFactory");

Implementación: Carga de Scripts

El fichero HTML que se cargará en el componente Web de la aplicación está formado por tres partes diferenciadas.

Referencia a los scripts y CSS de las librerías de terceros utilizadas en la aplicación. Aquí se referencia a las librerías como BackBone, Jquery, Jquery Mobile y sobretodo PhoneGap.

Esta sección estará situada en la sección "Head" del fichero HTML y será lo primero que se cargará.

Después se cargarán los ficheros CSS y JavaScript propios de nuestra aplicación: modelos, vistas, utilidades, configuración, librerías de ayuda, etc... Estos serán los encargados de ejecutar toda la lógica de negocio propia de la aplicación.

Finalmente el fichero HTML está compuesto por todo el código HTML, en forma de script de plantillas, que forman las diferentes vistas de la aplicación Eventual.

Carga de scripts de terceros en la aplicación cliente

```
<!--Css de terceros -->
<link rel="stylesheet" href="http://www.photoswipe.com/latest/photoswipe.css" />
<!--JqueryMobile-->
<link rel="stylesheet" type="text/css" href="libs/jquery.mobile/jquery.mobile-1.3.2.css"/>
<!--Controles de fechas-->
<link rel="stylesheet" type="text/css" href="libs/datebox/jqm-datebox.min.css"/>

<!--Css específicos de la aplicación -->
<link rel="stylesheet" type="text/css" href="css/custom.css"/>

<!--Scripts de terceros -->
<script src="libs/jquery-1.10.2.js"></script>
<script src="libs/jquery.scrollTo-min.js"></script>
<!--Configuración de jquery mobile-->
<script src="libs/jqm-config.js"></script>
<script src="libs/jquery.mobile/jquery.mobile-1.3.2.js"></script>
<script src="libs/cordova-2.9.0.js"></script>
<!--Controles de fechas-->
<script src="libs/datebox/jqm-datebox.core.min.js"></script>
<script src="libs/datebox/jqm-datebox.mode.calbox.min.js"></script>
<script src="libs/datebox/jquery.mobile.datebox.i18n.es-ES.utf8.js"></script>
<!--Librería para la gestión de fechas-->
<script src="libs/moment-with-langs.min.js"></script>
<!--Gestor de plantillas-->
<script src="libs/underscore.js"></script>
<!--Backbone-->
<script src="libs/backbone.js"></script>
<script src="libs/backbone-relational.js"></script>
<!--Serialización de formularios-->
<script src="libs/formparams.js"></script>
<!--Carga diferida de imagenes-->
<script src="libs/jquery.krioImageLoader.js"></script>
<!--Mensajes-->
<link href="libs/toastr.min.css" rel="stylesheet"/>
<script src="libs/toastr.min.js"></script>
<!--Galeria-->
<script src="libs/klass.min.js"></script>
<script src="libs/code.photoswipe.jquery-3.0.4.min.js"></script>
```

Backbone.js será el encargado de ir mostrando las diferentes plantillas renderizadas con los datos correspondientes, en función del estado del modelo de la aplicación o de la respuesta a las acciones generados por el usuario.

Implementación: Carga de Scripts / Plantillas

Carga de scripts propios de la aplicación

```
<!--Scripts específicos de la aplicación -->
<!--Configuración de la aplicación-->
<script src="js/config.js"></script>
<!--Hack para el uso de websockets android-->
<script src="libs/websockets.js"></script>
<script src="js/utils.js"></script>
<script src="js/phoneGap.js"></script>
<!--Vistas,models y rutas genericas-->
<script src="js/backBoneExtend.js"></script>
<!--Modelos de la aplicación-->
<script src="js/models.js"></script>
<!--Vistas de la aplicación-->
<script src="js/views/events.js"></script>
<script src="js/views/users.js"></script>
<script src="js/views/images.js"></script>
<script src="js/views/posts.js"></script>
<!--Rutas-->
<script src="js/router.js"></script>
<!--js principal-->
<script src="js/main.js"></script>
<title>Eventual</title>
<script type="text/javascript" charset="utf-8">

    //sólo iniciamos phonegap si estamos en un emulador o en un móvil real
    if (eventual.config.mode=='mobile'){
        eventual.phoneGapUtils.init();
    }
</script>
```

Sólo cargamos la librería PhoneGap al ejecutar la aplicación en un emulador o en un dispositivo real. No es necesario al realizar pruebas con un navegador WEB

Plantilla HTML correspondiente a la vista listado de eventos de la aplicación

```
<!--LISTADO DE EVENTOS TEMPLATE-->

<script type="text/template" id="events-search-template">
    <div data-role="header" id="header">
        <h2>Eventos</h2>
        <a href="#events/add" data-icon="add" id="add-event-button">Añadir</a>
    </div>
    <div data-role="content">
        <ul id="autocompleteSearch" data-role="listview" data-inset="true" data-theme="a" data-count-theme="b"
            <% _.each( events, function( event ){ %>
            <li>
                <% var imgUrl=event.image?baseUrl+"/"+event.image.files[0].url:"images/no-photo.jpg"%>
                <a href="#events/<%= event.id %>/posts">
                    <img src="<%=imgUrl%>"/>
                    <h1><%=event.name%></h1>
                    <p><%=event.date%></p></br></br>
                    <p><strong><%=event.description%></strong></p>
                    <span class="ui-li-count"><%=event.postsCount+event.imagesCount%> posts</span>
                </a>
            </li>
            <% }); %>
        </ul>
    </div>
    <div data-role="footer" data-position="fixed">
        <div data-role="navbar" data-iconpos="bottom">
            <ul>
                <li><a href="#"         data-icon="grid" class="ui-btn-active">Tus eventos</a></li>
                <li><a href="#promo"    data-icon="star" >Invitaciones</a></li>
                <li><a href="#profile"  data-icon="gear">Perfil</a></li>
            </ul>
        </div><!-- /navbar -->
    </div><!-- /footer -->
</script>
```

BackBone.js sustituirá los datos de la plantilla por el valor correspondiente, y mostrará correctamente el HML resultante.

Implementación: Seguridad

El siguiente fragmento de código se ejecutará cuando la aplicación esté preparada para mostrar el contenido.

```
$(document).ready(function () {

      console.log('document ready');
      eventual.token=new Date();
    //iniciamos la aplicación
    eventual.app = new eventual.App();
    eventual.app.start();

});
```

Las peticiones al servidor contienen autorización y autenticación, para evitar que un usuario realice un cambio en un evento que no le pertenece o que cualquiera pueda hacerlo simplemente haciendo una petición a la API correspondiente. Para dotar al sistema de seguridad, cada petición que se realiza contra el servidor se hace intercambiando un token generado por el servidor con una duración determinada. Esta token se incluye en la cabecera de la petición Ajax enviada al servidor.

```
//en todas las peticiones contra el servidor, incluiremos el token, generado por el servidor, como
//medida de seguridad
//si el token no existe, asociado al usuario
//devolverá un 401 y el usuario deberá volver a logarse
//las credenciales estarán guardadas en el localstorage del dispositivo, asi podrá logarse de forma transparente para
//el usuario, cuando la sesión caduque.

      $.ajaxSetup({
            headers : {'X-CSRF-Token' : eventual.token}
      });
```

Implementación: Seguridad

La aplicación servidor obtendrá la petición, comprobará si el token pertenece a un usuario logado, y en caso contrario generará un nuevo token, devolviendo un error de autorización 401 y obligará al cliente a logarse de nuevo. Los token's tienen una validez configurable, actualmente la configuración es de 30 minutos.

```
@Controller
@RequestMapping(value = "/api/auth")
public class AuthController {

	@RequestMapping(method = RequestMethod.GET)
	public @ResponseBody
	ResponseEntity<TokenDTO> isAuth(HttpServletRequest request) {

		HttpHeaders headers = CorsFilter.getAccessControllAllowOrigin();
		ResponseEntity<TokenDTO> response = null;

		// Obtenemos el token actual
		String csrf_token = request.getHeader(AppConst.Security.CSRF_TOKEN);

		// Comprobamos que el usuario exista en el repositorio de tokens
		TokenDTO tokenDTO = csrf_token != null ? securityService.findToken(csrf_token) : null;

		if (tokenDTO != null) {
			// Si existe devolvemos el token actual
			response = new ResponseEntity<TokenDTO>(tokenDTO, headers, HttpStatus.OK);

		} else {
			// Sino existe creamos una nueva respuesta solo con el nuevo token
			TokenDTO new_token = new TokenDTO(TokenGeneratorUtils.getRandomToken(32));
			response = new ResponseEntity<TokenDTO>(new_token, headers, HttpStatus.UNAUTHORIZED);
		}
		return response;
	}
```

Implementación: Seguridad

Obviamente si al usuario logado en la aplicación le saltase la página de login cada media hora, la aplicación sería totalmente inutilizable. Para evitarlo al logarse se almacenan las credenciales del usuario en el dispositivo móvil, mediante el uso del fileStorage de HTML5.
El token de seguridad se reiniciará en el servidor, la aplicación cliente utilizará los credenciales almacenados para autenticarse de forma trasparente para el usuario y guardará el nuevo token para las posteriores peticiones al servidor.
Para solucionar el problema del autologin, interceptaremos todas las peticiones al servidor sobrescribiendo el método fetch de los modelos gestionados por BackBone y en caso de fallar por motivos de autentificación (401), se obtendrán las credenciales del fileStorage, se logará contra el servidor y se reintentará de nuevo la petición original.

```
fetch: function(options) {

            options || (options = {});

            //comprobamos si existe una url
            if (typeof(options.url) !== "undefined"){
                  options.url=this.url()+options.url;
            }

            //preparamos la función de callback
            var that=this;
            var retryFunction = function (options) {
                  return Backbone.Model.prototype.fetch.call(that, options);
            }

            //preparamos los callback de vuelta en caso de éxito o fallo
            //en  caso de error reintentaremos ejecutar esta misma función
            eventual.utils.handleFetchSuccess(options);
            eventual.utils.handleFetchError("fetch",retryFunction,options);

            //seguimos con la ejecución normal
            return Backbone.Model.prototype.fetch.call(this, options);
      }
```

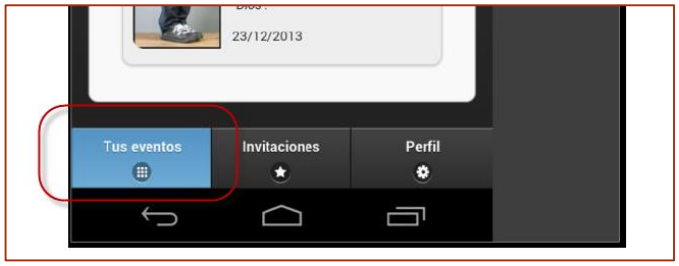
Implementación: Router

Tanto si el token enviado al servidor es correcto, y pertenece a un usuario logado, como si éste ha caducado y ha sido necesario autenticarse de nuevo contra el servidor, una vez que el usuario ya está logado, la clase Router de BackBone será la encargada de determinar que vista es la que se ha de cargar.

Esto se realiza manteniendo un mapeo entre la url solicitada y la vista a mostrar. A continuación se muestran todas las "rutas" definidas por la aplicación cliente.

```
routes : {
            "" : "events",
            "events"                        :"events",
            "events/add"                    :"addEvent",
            "events/:eventId/edit"          :"editEvent",
            "events/:eventId/post-image"    :"postImage",
            "events/:eventId/show"          :"showEvent",
            "events/:eventId/images"        :"images",
            "events/:eventId/posts"         :"posts",
            "events/:eventId/post/:postId"  :"comments",
            "events/:eventId/users2join"    :"users2join",
            "users/:userId/edit"            :"editUser",
            ":entity/:entityId/add-image"   :"addImage",
            "createUser"                    :"createUser",
            "login"                         :"showLogin",
            "profile"                       :"showProfile",
            "promo"                         :"promo",
            "back"                          :"back"
      },
```

Si la aplicación cliente detecta que el usuario ha pulsado en el listado de sus eventos, por ejemplo pulsando en el botón "Tus eventos" situado en parte inferior de la aplicación.



Implementación: Flujo completo

El mapeo que la clase Router ejecutaría será este

```
"events"                        :"events",
```

Ya que la plantilla HTML, correspondiente al footer de la página que incluye el botón "Tus eventos" así lo indica:

```
<div data-role="footer" data-position="fixed">
       <div data-role="navbar" data-iconpos="bottom">
       <ul>
              <li><a href="#events"     data-icon="grid" class="ui-btn-active">Tu eventos</a></li>
…
```

La función *events*, solicitará la siguiente petición contra el servidor http://server/eventual/api/by-user/xxxx, obtendrá el listado de eventos en los que participa el usuario xxxx y actualizará la vista con la plantilla correspondiente a la vista *EventsSearchView*, incluyendo la colección de eventos retornada por el servidor.

El método fetch de los modelos y colecciones de BackBone es el que hemos "interceptado" previamente para implementar la seguridad.

```
events : function() {
       var view;
           eventual.events = new eventual.Events();
           var self=this;
           var userId=eventual.auth.getData("user").replace(/"/g, '');
           eventual.events.fetch({
              //solo mostramos los eventos en los que participa el usuario logado por eso pasamos la url que filtra por usuario
              url:"/by-user/"+userId,
              success: (function (data) {
                     self.loadView(new eventual.EventsSearchView({collection: eventual.events,baseUrl: eventual.config.rest.url}));
            }),onGenericError:(function(model,xhr,options){})
        });
},
```

Implementación: Flujo completo

La API REST implementada en el servidor recibirá la petición y devolverá el objeto JSON correspondiente a los eventos en los que participa el usuario. Para eso delegará la lógica al servicio *UserService*, llamando a la función *byUser* que esta clase implementa. La anotación `@ResponseBody` convertirá el listado de objetos EventDTO en una lista de objetos JSON.

```
// Listado de ventos por usuario
	@RequestMapping(value = "/by-user/{userId}", method = RequestMethod.GET)
	public @ResponseBody
	ResponseEntity<Collection<EventDTO>> eventsByUser(@PathVariable("userId") String userId) {
		HttpHeaders headers = CorsFilter.getAccessControllAllowOrigin();
		ResponseEntity<Collection<EventDTO>> entity = new
		ResponseEntity<Collection<EventDTO>>(getRestService().byUser(userId), headers, HttpStatus.OK);
		return entity;
	}
```

A su vez, el servicio UserService delegará la lógica en el objeto EventRepository que será el encargado de acceder a base de datos y filtrar todos los elementos del tipo Event (mapeo en base de datos de la entidad), que pertenezcan al usuario solicitado.

```
public Collection<Event> byUser(ObjectId... userIds) {

		Collection<Event> events = in("assignment.id", userIds);
		// Ampliamos la información de la clase evento para cada ítem
		for (Event e : events) {
			expand(e);
		}
		return events;
	}
```

Implementación:Flujo completo

El método "in" es un método genérico implementando en la clase BaseRepository que equivaldría a ejecutar una sentencia "in" en SQL convencional, pero en este caso usando los métodos de acceso a la base de datos MongoDB que nos facilita la librería Spring Data para MongoDB.

```
public Collection<E> in(String field, ObjectId... ids) {
        Query query = new Query();
        query.addCriteria(Criteria.where(field).in((Object[]) ids));
        return getMongoOperations().find(query, entityClass);
}
```

Una vez devuelto el listado de eventos en formato JSON, Backbone ejecutará el método render de la vista EventsSearchView, que mediante Underscore cargará la plantilla events-search-template, y la dibujará a partir de los datos incluidos en la colección asociada a la vista.

```
eventual.EventsSearchView = eventual.BaseView.extend({
   baseUrl:null,
   initialize: function(options) {
      this.template = _.template($('#events-search-template').html());
            this.baseUrl=options.baseUrl;
      },
      render: function() {
            $(this.el).html(this.template( { "events": this.collection.toJSON(),"baseUrl":this.baseUrl}));
            return this;
    },
    changeItem: function(){
        this.collection.sort();
    }
});
```

Implementación: Flujo completo

La plantilla HTML contiene un listado de eventos, así como la información principal de los mismos. Esta sección de código HTML será mostrada en la pantalla de la aplicación, sustituyendo los datos marcados por el código <%%> por la representación JSON correspondiente al modelo o colección asociada a la vista.

```
<!--LISTADO DE EVENTOS TEMPLATE-->

<script type="text/template" id="events-search-template">
      <div data-role="header" id="header">
            <h2>Eventos</h2>
            <a href="#events/add" data-icon="add" id="add-event-button">Añadir</a>
      </div>
      <div data-role="content">
      <ul id="autocompleteSearch" data-role="listview" data-inset="true" data-theme="a" data-count-theme="b" data-
      filter="true" data-filter-reveal="false" data-filter-placeholder="Buscar un evento..." data-filter-theme="b">
            <% _.each( events, function( event ){ %>
            <li>
                  <% var imgUrl=event.image?baseUrl+"/"+event.image.files[0].url:"images/no-photo.jpg"%>
                        <a href="#events/<%= event.id %>/posts">
                                          <img src="<%=imgUrl%>"/>
                                          <h1><%=event.name%></h1>
                                          <p><%=event.date%></p></br></br>
                                          <p><strong><%=event.description%></strong></p>
                                          <span class="ui-li-count"><%=event.postsCount+event.imagesCount%> posts</span>
                                    </a>
                              </li>
                              <% }); %>
                        </ul>
                  </div>

      ...
```

Implementación: Flujo completo

Finalmente se obtendrá resultado que se puede visualizar en la imagen, una vez sustituido en la plantilla las etiquetas dinámicas por los valores correspondientes.

Uno de los errores más comunes que se pueden realizar al utilizar un framework como BackBone al ir realizando una navegación entre las diferentes páginas de la aplicación, es cambiar el HTML que se visualiza sin eliminar los datos de la vista ni los eventos asociados a ella, dejando vistas "zombie" abiertas, haciendo que la aplicación se comporte de forma impredecible.

Para evitar este problema se ha implementado el método LoadView, dentro de la clase Routers de la aplicación móvil, que garantiza que sólo se mantendrá una vista activa al mismo tiempo, ya que lo primero que realizará cuando se haga una navegación entre dos vistas diferentes será eliminar los datos de la anterior.

```
//métodos para cargar las vistas mediante backbone, no se corresponden a vistas reales
//sino que son objetos genéricos para la navegación entre vistas.
    loadView: function(view) {

            if (this.currentView) {
                  console.log('Detach the old view');
           this.currentView.close();
        }
          console.log('Render view after it is in the DOM (styles are applied)');
          this.currentView = view;
          this.currentView.render();
          $("#backbone-dynamic-content").html(this.currentView.el);
          $("#backbone-dynamic-content").trigger('pagecreate');
          $("#backbone-dynamic-content").krioImageLoader();
          //dejamos de mostrar el mensaje de "Cargando...”
            eventual.utils.hideLoading();
      },
```

Implementación: Captura y subida de imagenes

Una de las princípiales funcionalidades, sino la principal, de ***eventual*** es la captura de imágenes con la cámara del dispositivo y la subida de la misma al servidor para su posterior proceso (ajustar tamaños, crear miniaturas, persistencia en base de datos, etc.)

El código más significativo de estas funcionalidades es el siguiente:

```
// Iniciamos la cámara
        //
        captureImage:function () {
            // Lanzamos la ejecución de la cámara
            // Sólo permitimos capturar una imagen, cada vez
            navigator.device.capture.captureImage(eventual.utils.CaptureSuccess, eventual.utils.CaptureError, { limit: 1 });
        },
…

// Subimos el fichero al servidor
        uploadFile:function (params,onSuccess,onError,redirectionPostLogin) {

                var that=this;
                var retryFunction = function (params,onSuccess,onError) {
                        eventual.phoneGapUtils.uploadFile(params,onSuccess,onError,null);
                }

                 var ft = new FileTransfer(),options = new FileUploadOptions();
                 options.fileKey  = "fileData";
                 options.fileName = "file.jpg";
                 options.mimeType = "image/jpeg";
                 options.chunkedMode = false;
                 options.headers = { "Connection":"close",'X-CSRF-Token' : eventual.token};

         if (params.comments){
             options.params = {"comments": params.comments};
         }

         //progreso de subida
         ft.onprogress = function(progressEvent) {
                if (progressEvent.lengthComputable) {
                        //mostramos un mensaje con el porcentaje de la imagen subida
```

Implementación: Aplicación Cliente:Captura y subida de imágenes

```
var perc = Math.floor(progressEvent.loaded / progressEvent.total * 100);
        eventual.utils.showLoading("Subiendo imagen...",perc);
}

if (perc == 99){
        //empezamos las distintas transformaciones en función del tipo de imagen subida (thumnbail, detalle, grande)
            eventual.utils.showLoading("Procesando imagen...");
        }
};

 // Subimos la imagen..
    ft.upload(params.imageURI, params.url,
        //función éxito, la subida del fichero ha funcionado
        //el parámetro pasado es FileUploadResult, que contiene la información de subida del fichero
        //nos interesa pasar el objeto json devuelto  a la función onSucces
            function(r) {
                if (onSuccess && typeof(onSuccess) === "function") {
                        console.log('call custom success');
                        onSuccess(JSON.parse(r.response));
                }
        },
        function(f) {
                //hay una función callback error definida, la ejecutamos
                        if (onError && typeof(onError) === "function") {
                                console.log('call custom error');
                                onError(f.http_status,f.code);
                        }
                        //no hay función personalizada de error, ejecutamos una genérica
                        else{

eventual.utils.handleGenericErrorCustomSync(f.http_status,"uploadFile",retryFunction,params,onSuccess,redirectionPostLogin);
                        }
        },options
        );
```

14. Manual de Usuario

Como parte de los entregables junto a esta memoria se ha incluido un documento con el manual de usuario para utilizar la aplicación, al mismo tiempo que se ha creado un video mostrando las funcionalidades más importantes de la misma.

Este video se ha subido al @presenta de la asignatura y también se ha alojado en la plataforma de videos online **Vimeo,** ya que ésta permite subir videos de mayor tamaño y calidad.

El video es accesible online en la siguiente dirección:
https://vimeo.com/83458135

Asimismo tambíen se incluye una presentación en formato ppt de esta memoria, accesible tambien en formato video en @presenta y en la plataforma vimeo.

https://vimeo.com/83637871

El video se ha grabado utilizando la aplicación **RecordMyDesktop** durante la ejecución de la aplicación **eventual** en el emulador **GenyMotion**, en un entorno Ubuntu.

Se ha utilizado **QuickTime** en un sistema operativo Mac OS X Lion para realizar una segunda captura en otro emulador, para poder mostrar la funcionalidad de "chat" entre usuarios mediante el uso de WebSockets.

La edición del video se ha realizado utilizando el software **Kdenlive**, la obtención de la voz se ha realizado usando las voces en español del software **Ivona Voices**.

15. Conclusiones

Al iniciar el desarrollo de este proyecto de fin de carrera me había planteado dos propósitos principales, por una parte desarrollar una aplicación móvil con aspecto los más profesional posible utilizando técnicas de desarrollo web modernas, partiendo de una idea sencilla, y por otra, utilizar servicios y tecnologías que se usen en aplicaciones reales de éxito para implementarla.

El primer hito es por supuesto el más ambicioso ya que el alcance y la funcionalidad de la aplicación desarrollada quizás excedían el ámbito y el propósito de un PFC. La aplicación realizada dista mucho de ser explotable en un entorno productivo sin realizar bastantes ajustes, pero he intentado ser consecuente en todas las decisiones técnicas, funcionales y arquitectónicas para construir un producto agradable, intuitivo y escalable, siguiendo todas las "best practices" de la industria posibles.
En este sentido estoy bastante satisfecho con el resultado, aunque haya supuesto un esfuerzo considerable.

El segundo hito era poner en "producción" el producto utilizando servicios que usen empresas con presencia real en internet como son Heroku o MongoHQ. No quería tomarme el PFC sólo como una prueba de concepto, sino más bien como un producto que potencialmente pudiese ser usado por usuarios reales y enfrentarme a los problemas que pudiera encontrarme en el camino.

Es en esto punto en el que más satisfecho estoy, ya que el servicio es totalmente funcional y lo he podido comprobar en dispositivos reales, y espero me sirva de experiencia en aplicaciones futuras.

En lo estrictamente referentes al área del proyecto, "aplicaciones móviles HTML5", me llevo una muy buena sensación del potencial que tienen las aplicaciones hibridas, y del más que notable resultado obtenido. Me ha sorprendido gratamente lo relativamente fácil que es el uso de librerías como JqueryMobile o PhoneGap, teniendo en cuenta el ahorro de trabajo que representan al desarrollar una única aplicación para varias plataformas.

En contrapartida, creo que el rendimiento es todavía el talón de Aquiles de esta tecnología, y dista mucho de ser comparable a las aplicaciones nativas, sobretodo en dispositivos de gama baja. Aunque creo que esta diferencia se irá reduciendo.

En conclusión he cumplido mis propósitos iniciales. Me he familiarizado con técnicas de desarrollo web modernas como MVC en el cliente, algo que ha roto totalmente los esquemas de los que partía y he conseguido implementar una aplicación móvil hibrida con características de una nativa, utilizando HTML, CSS y JavaScript, utilizando servicios reales.

16. Referencias

Backbone.js: http://backbonejs.org/,http://en.wikipedia.org/wiki/Backbonejs

Jquery Mobile: http://jquerymobile.com/

PhoneGap: http://phonegap.com/

PhoneGap Build: *Adobe® PhoneGap™ Build* https://build.phonegap.com/

SPRING MVC: ***Spring M**odel **V**iew **C**ontroller* http://docs.spring.io/spring/docs/4.0.0.BUILD-SNAPSHOT/spring-framework-reference/html/mvc.html,http://es.wikipedia.org/wiki/Modelo_vista_controlador

No SQL: http://es.wikipedia.org/wiki/NoSQL, http://nosql-database.org/, http://www.nosql.es/blog/nosql/que-es-nosql.html

MongoDB: *Hu**mongo**us **D**ata**B**ase* http://es.wikipedia.org/wiki/MongoDB, http://www.mongodb.org/,http://projects.spring.io/spring-data-mongodb/

REST: ***R**epresentational **S**tate **T**ransfer* http://es.wikipedia.org/wiki/Representational_State_Transfer

Heroku: https://www.heroku.com/

MongoHQ: http://www.mongohq.com/home

Android: http://developer.android.com/index.html

Json: http://es.wikipedia.org/wiki/JSON

WebSockets: http://es.wikipedia.org/wiki/WebSocket

Assembla: https://www.assembla.com/home

Referencias

Guías y tutoriales:

http://coenraets.org/blog/2011/12/backbone-js-wine-cellar-tutorial-part-1-getting-started/

http://stevenpsmith.wordpress.com/2012/02/07/from-a-list-to-a-detail-view-using-jquery-mobile-and-backbone-js/

http://danialk.github.io/blog/2013/07/28/advanced-security-in-backbone-application/

http://blog.manishchhabra.com/2013/04/spring-data-mongodb-example-with-spring-mvc-3-2/

http://anismiles.wordpress.com/2011/02/03/websocket-support-in-android%E2%80%99s-phonegap-apps/

https://devcenter.heroku.com/articles/create-a-java-web-application-using-embedded-tomcat

http://coenraets.org/blog/2013/09/how-to-upload-pictures-from-a-phonegap-app-to-amazon-s3/

http://java-success.blogspot.com.es/2012/11/cors-and-jquery-with-spring-mvc-restful.html

http://lostechies.com/derickbailey/2011/09/15/zombies-run-managing-page-transitions-in-backbone-apps/

Referencias

Concepto original:

https://www.google.es/search?q=wedding+disposable+cameras&es_sm=93&source=lnms&tbm=isch&sa=X&ei=BDdRUrXBGKjF7AbyuICoBA&sqi=2&ved=0CAcQ_AUoAQ&biw=1527&bih=829&dpr=1

Imagen portada: http://www.christinemanfield.com/images/events_bg.jpg

Eventual: **Event**o Soci**al**, Even**2**all, **Event** to **All**

http://event2all.herokuapp.com/eventual/

https://github.com/driegor/event2all