Facultad de Informática

Grado de Ingeniería Informática

■ **P**royecto **F**in de **G**rado ■

Ingeniería de Software

SocialMusFest: una red social sobre festivales de música.

ALUMNO: Jon Junguitu Iturrospe

DIRECTOR: José Ángel Vadillo Zorita

Junio 2016

RESUMEN

En este documento se describe el Proyecto Fin de Grado (PFG) de Jon Junguitu Iturrospe, dentro de la titulación de Grado en Ingeniería Informática, especialidad de Ingeniería de Software, del curso académico 2015/2016.

El proyecto se denomina SocialMusfest, y se trata de desarrollar una aplicación web con la finalidad de red social sobre festivales de música. Así mismo, se da la opción a los creadores de festivales de organizar un evento para publicitarlo y poder realizar una gestión superficial del mismo. Para la elaboración del sitio web se ha utilizado una tecnología con cada vez más auge, Meteor.

La realización del proyecto ha sido en conjunto con Rachid Boudhar Tannaoui, el cual ha realizado la misma aplicación para sistemas Android. Ambos proyectos comparten la base de datos y los detalles sobre la implementación, con la finalidad de poder crear un sistema web accesible desde equipos PC y dispositivos móviles con sistema operativo Android.

La gestión de este proyecto se inicia realizando una planificación acorde a las características del proyecto. A continuación, se realiza un estudio de la tecnología a utilizar, debido a que antes no se había trabajado con ella, para finalmente implementar la aplicación. Todo este trabajo queda detallado a lo largo de este documento.

Actualmente la página web está alojada en la siguiente dirección: https://sozialmusfest.scalingo.io/.

AGRADECIMIENTOS

Agradecer a todas las personas que me han ayudado tanto durante la elaboración de este proyecto como en toda mi etapa universitaria. A José Ángel Vadillo como director del proyecto, por haberme ayudado en cualquier problema que me ha surgido, y a José Miguel Blanco por aportarme sugerencias e ideas. A mis compañeros de Vitoria-Gasteiz y San Sebastian, por su implicación y dedicación. A todos mis amigos, que han estado interesados en ayudarme y se han preocupado en todo momento de mí. Finalmente, a toda mi familia por haber estado encima y porque siempre estarán ahí. Gracias a todos.

ÍNDICE

LISTA DE FIGURAS Y TABLAS

FIGURAS

1

INTRODUCCIÓN

En este punto se va a describir lo realizado a lo largo del proyecto. El mismo se divide en diferentes fases, unas dependientes de otras, que permiten llegar a un resultado esperado y óptimo.

Se está desarrollando en paralelo otro PFG por el alumno Rachid Boudhar, el cual explotará en su aplicación el modelo de datos aquí implementado. Su aplicación, bajo el mismo dominio, se centrará en el desarrollo para Android con backend Java. Además, incorporará otras funcionalidades como geolocalización, etc...

En este caso, lo primero de todo es definir el contexto en el que se enmarca este proyecto, para posteriormente definir la propuesta a elaborar. Este proyecto toma como base que se quiere trabajar con una nueva tecnología, no muy utilizada pero con grandes perspectivas de futuro. Junto a ello, elaborar una red social acerca de festivales de música que, a pesar de ser un ámbito que mueve gran cantidad de personas, no está elaborado por el momento.

Por lo tanto, este punto comienza con los antecedentes del proyecto, donde se explican las necesidades de elaborar una red social de este tipo. A continuación se explica el contexto en el que se ubica el proyecto, sus motivaciones y se describe la propuesta. Finalmente se explica la organización de este documento.

1.1. ANTECEDENTES

Los festivales de música son grandes congregaciones de gente con el propósito de acudir a diferentes conciertos que se ofertan por parte de la organización. Algunos ejemplos conocidos son los siguientes:

- **País Vasco:** Azkena Rock (Vitoria-Gasteiz, Álava), BBK Live (Bilbao, Vizcaya), Getxo Folk (Getxo, Vizcaya), Jazzaldia (Donostia-San Sebastián), Big Festival (Biarritz, Lapurdi),...

- **España:** Viña Rock (Villarrobledo, Albacete), Sónar (Barcelona, Barcelona), Primavera Sound (Barcelona, Barcelona), Arenal Sound (Burriana, Castellón), FIB (Benicassim, Castellón), AlRumbo Festival (Chipiona, Cádiz),...

- **Internacional:** Tomorrowland (Bélgica), Ultra Music Festival (Estados Unidos), Festival de Viña del Mar (Chile), Rock in Rio (Brasil), Summer Sonic (Japón),...

Como se puede observar, estos festivales pueden ser de diferentes tipos y se organizan en todas las partes del mundo. Muchas veces es complicado promocionar un festival para que la gente interesada pueda asistir, por lo que se ha pensado en realizar dicha acción mediante un medio al que todo el mundo hoy en día puede acceder: Internet.

Como a estos eventos se suele acudir en grupo, una buena opción de relacionar gente, al igual que ocurre en otro tipo de casos, es creando una aplicación con la función de red social. Hoy en día, la mayoría de la gente dispone de algún tipo de red social, fáciles de utilizar y muy cómodas para poder comunicar gente. Estas redes sociales son de diferente tipo, en función de las necesidades que se quieran satisfacer:

- Amistad: Facebook, Tuenti...

- Información: Twitter...

- Comunicación: WhatsApp, Telegram...

- Trabajo/Estudios: Linkedin, Infojobs...

Este tipo de redes sociales congregan a gran cantidad de usuarios. Son aplicaciones con grandes comunidades de personas, pero hasta la fecha no hay ninguna aplicación que trabaje para poder unir esa gente entorno a festivales musicales. Por ello, esta puede ser una buena opción para llevarla a cabo.

1.2. CONTEXTO

Uno de los mayores problemas que existen en torno a los festivales de música es que sólo los mayores eventos, y entre ellos una reducida parte, promociona su evento de tal forma que llegue a todos los interesados. Para el resto de casos, son los propios asistentes los que tienen que realizar una búsqueda, que requiere mucho tiempo de anticipación, para poder acudir a festivales que puedan ser de su agrado.

Hoy en día, con el gran uso que se hace de las redes sociales por parte de la mayoría de la gente, sobre todo joven que es el tipo de gente que mayoritariamente acude a los festivales de música, este problema que surge en ambos lados, tanto en la parte de organizadores como en la de los asistentes, puede resolverse muy fácilmente.

Además, es fundamental disponer de la última tecnología en la elaboración de aplicaciones de cualquier tipo, y en concreto web. Los usuarios de cualquier aplicación cada vez tienen más experiencia con las nuevas tecnologías y son más exigentes, lo que obliga a los desarrolladores a satisfacer esas necesidades con aplicaciones sencillas, de fácil comprensión pero lo suficientemente potentes para poder tener funcionalidades lo más amplias posibles.

1.3. PROPUESTA

El proyecto tiene como objetivo la elaboración de un sitio web donde dar a conocer festivales de música de forma sencilla, con perspectivas de convertirse en red social colaborativa entre personas interesadas en este ámbito.

Sin llegar al punto de red social, la propuesta de este proyecto es la de realizar una aplicación que permita dejar la información del festival, así como formas de contacto para poder acudir al mismo. Para los interesados en acudir al evento, el objetivo es el de informar del mayor número posible de festivales, así como dar la opción de apuntarse en los mismos para conocer la demanda existente. Con esto, la opción de apuntarse permite poder gestionar los festivales a los que se van a acudir (fechas...).

Además, a través de este proyecto se quiere dar a conocer una nueva tecnología web con perspectivas de futuro, que aporte ventajas respecto a las herramientas ya existentes. El framework Meteor es la opción escogida, ya que trabaja con una base de datos no-relacional (MongoDB) y con el lenguaje JavaScript tanto en la parte cliente como en la parte servidor, ambos aspectos que tienen gran importancia, y se espera que continúen igual, en el mundo web.

Dado que es una herramienta no muy conocida por muchos desarrolladores debido a su novedad, en primer lugar se realiza una fase de estudio de la misma para que, posteriormente, se pueda elaborar una guía básica de los aspectos más importantes. Esto ayudará a comprender mejor la tecnología, ordenando las ideas respecto al mismo.

1.4. ORGANIZACIÓN DEL DOCUMENTO

Este documento queda estructurado siguiendo la forma en la que se ha trabajado durante el proyecto.

Para comenzar, tras este punto se describe el DOP (“Documento de Objetivos del Proyecto”) en el capítulo 2. En él se muestra una descripción del proyecto elaborado, así como el periodo de tiempo calculado para su realización.

A continuación, en el capítulo 3 titulado “Análisis”, se detalla en más profundidad el proyecto. Se describen los actores que participan en la aplicación, junto a los casos de uso. Cada uno de esos casos de uso hace referencia a una funcionalidad de la aplicación, describiendo detalladamente su uso.

En el capítulo 4, “Diseño”, se presentan los diagramas de contexto, clases y secuencia. A través de estos diagramas se hace una muestra detallada de toda la aplicación, de su uso y de todos los aspectos participantes en la misma.

Seguido, en el capítulo 5 llamado “Tecnología” se describe en profundidad los puntos de Meteor elaborados durante el proyecto. Estos aspectos son básicos para poder elaborar un sitio web con esta tecnología, permitiendo disfrutar de muchos de los beneficios que ofrece.

Posteriormente, en el capítulo 6, “Implementación”, se muestran todos los puntos elaborados en el desarrollo de la aplicación. En este punto se describe la implementación de la aplicación SocialMusFest.

Finalmente, en el capítulo 7 titulado “Conclusiones” se hace un resumen, con las virtudes y defectos hallados durante la elaboración del proyecto, tanto en tema de gestión como en tecnología. Así mismo, se aporta un punto de líneas futuras a seguir, siguiendo una idea principal pensada pero no cerrada a nuevas aportaciones.

Como último punto, se aporta una bibliografía con enlaces utilizados durante el proyecto, que son de interés general.

2

DOCUMENTO DE OBJETIVOS DEL PROYECTO

Para poder llevar a cabo el proyecto, se ha realizado una planificación inicial con el fin de conseguir el resultado esperado. Esa planificación se ajusta a las características del proyecto, detallando a qué se quiere llegar y el tiempo a invertir para ello.

2.1. ALCANCE

A través del alcance del proyecto se define hasta que punto se quiere realizar el proyecto SocialMusFest. Las características especiales del proyecto hacen que la finalidad se adecue a las mismas, creando un proyecto real ante las exigencias presentadas.

El producto a desarrollar va a ser una aplicación web creada mediante el framework Meteor a modo de red social de festivales de música, en la que los usuarios de la misma podrán apuntarse en los eventos organizados mediante esta herramienta para así poder estipular el número de personas que acudirán al festival. Además de esta funcionalidad, el sitio sirve para realizar una gestión por parte de los organizadores de los festivales, publicitando el mismo y así llegando a más gente.

2.1.1. ESQUEMA

Para poder realizar una buena gestión, se ha realizado un análisis previo de lo que debe contener la aplicación y sus posibles actores o tipos de usuarios:

- Usuarios:
  - Anónimos.
  - Registrados.
- Secciones de la aplicación:
  - Página principal.
    - Apartado con festivales que han sido organizados mediante la aplicación, a modo de información.
    - Opción para registrarse.
    - Opción de inicio de sesión.
  - Usuarios.
    - Apartado para organizar un festival, con opción de generar una propuesta de horarios.
    - Apartado para cambiar la información del perfil del usuario.
    - Apartado de seguidores, gente a la que sigo y encuentra gente, en los que habrá usuarios de la aplicación. Para ello, todos los usuarios tendrán la opción de seguir a otros usuarios.
    - Apartado con los festivales organizados por el usuario. La aparición de esta información será en base a la fecha de inicio del festival, de forma ascendente.
    - Apartado con los festivales en los que participará el usuario. La aparición de esta información será en base a la fecha de inicio del festival, de forma ascendente.

- Apartado para ver los festivales que se ofertan. En esta lista aparecen todos los festivales de la aplicación (los creados por el usuario, a los que ha confirmado asistencia y el resto). La aparición de esta información será en base a la fecha de inicio del festival, de forma ascendente.

La estructura del sitio web se muestra a través de la siguiente imagen:

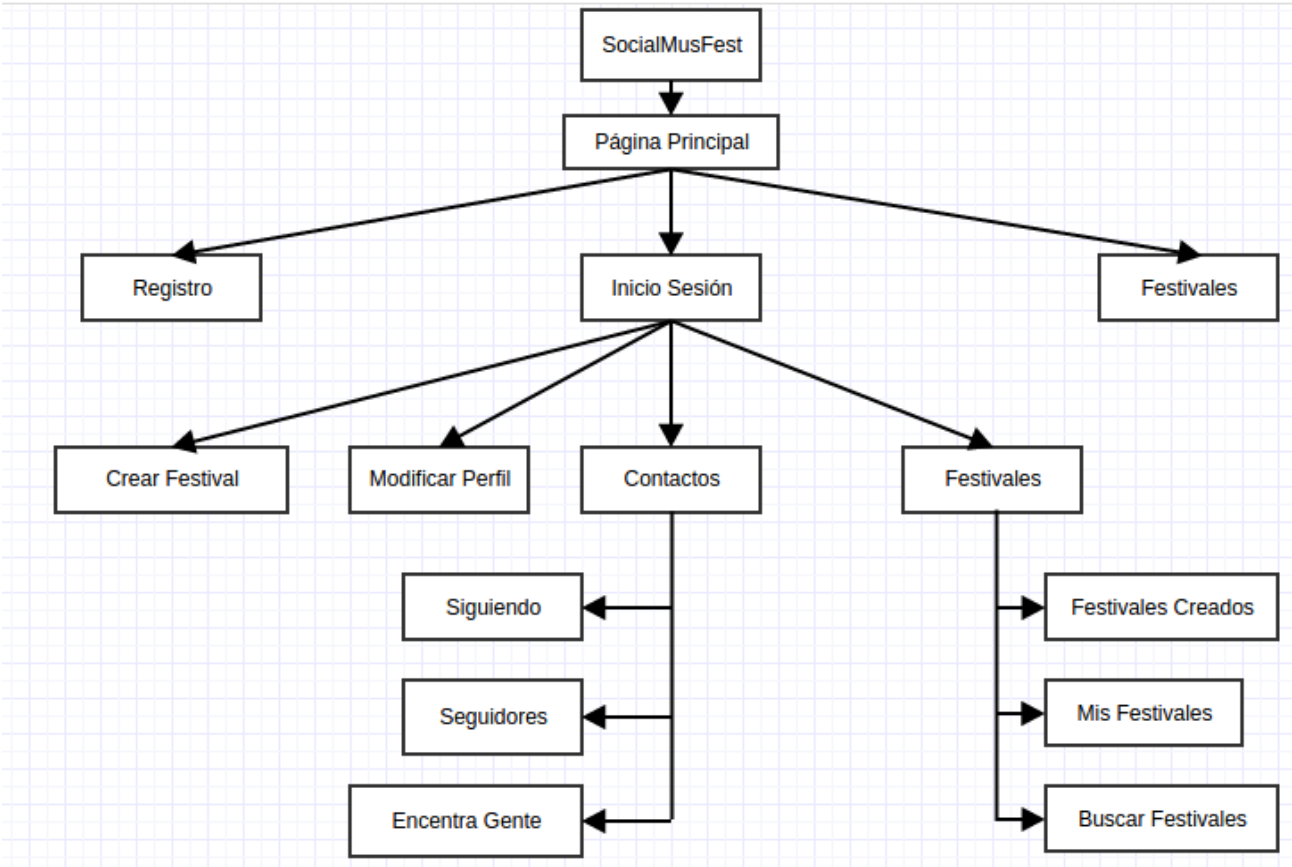

Figura 1 Esquema del proyecto SocialMusFest

2.1.2. DIAGRAMA WBS/EDT

En el diagrama de Estructura de Descomposición del Trabajo (EDT), o en inglés Work Breakdown Structure (WBS), que aparece en la figura 2, se hace una descomposición jerárquica orientada al entregable. Esta descomposición se divide en 2 fases (gestión y producto), siendo la primera correspondiente a la planificación y seguimiento y control del proyecto y la segunda orientada a crear la aplicación web y presentarla.

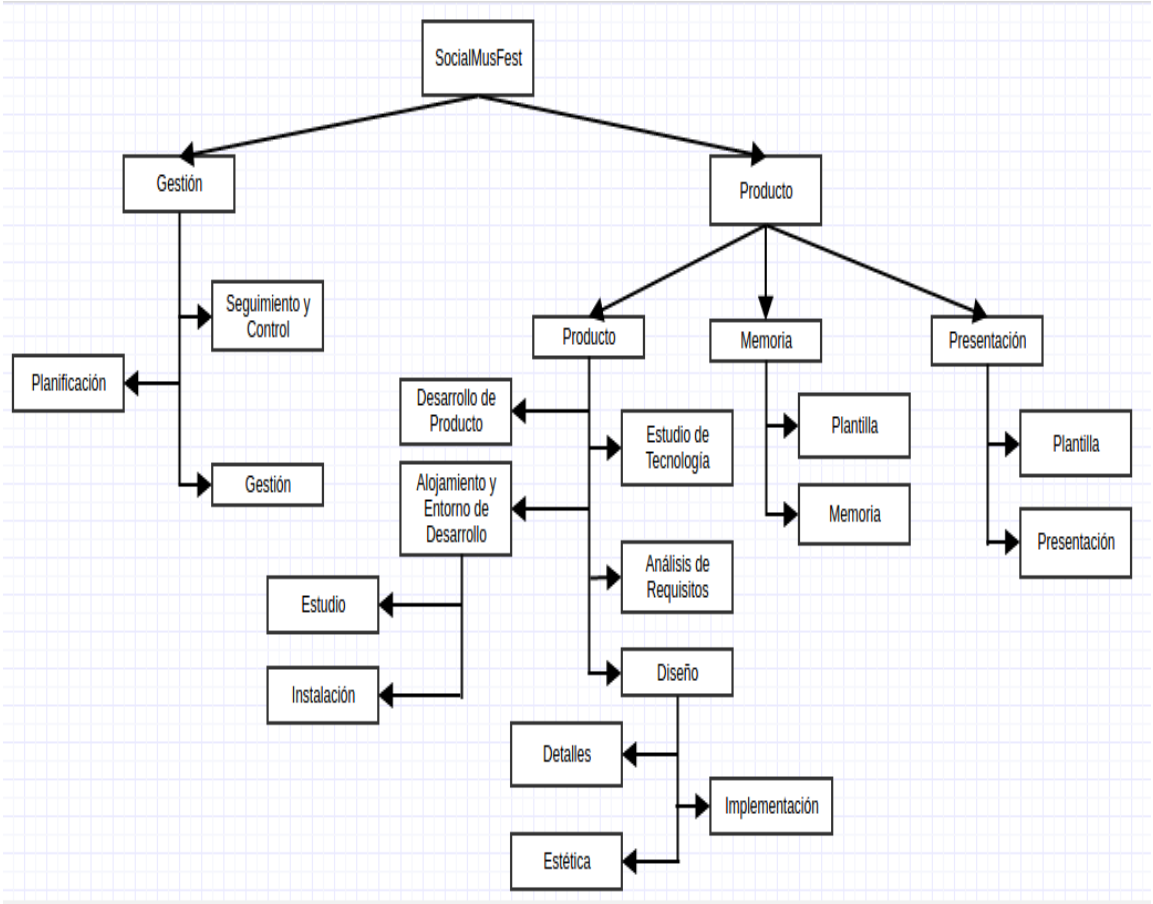

Figura 2 Diagrama WBS/EDT

2.2. PERIODOS

Para poder llevar a cabo este proyecto, se han fijado unos periodos e hitos de realización con el fin de lograr llegar a los objetivos para la fecha prevista.

2.2.1. HITOS DEL DESARROLLO

- **GESTIÓN:**
    - **Tarea:** Planificación
        - **Inicio:** Lunes 25/01/2016.
        - **Finalización:** Jueves 25/02/2016.
    - **Tarea:** Gestión
        - **Inicio:** Viernes 05/02/2016.
        - **Finalización:** Miércoles 15/06/2016.
    - **Tarea:** Seguimiento y Control
        - **Inicio:** Jueves 04/02/2016.
        - **Finalización:** Miércoles 15/06/2016.

- **PRODUCTO:**
    - **DESAROLLO PRODUCTO:**
        - **ALOJAMIENTO Y ENTORNO DE DESARROLLO:**
            - **Tarea:** Estudio
                - **Inicio:** Lunes 08/02/2016.
                - **Finalización:** Sábado 05/03/2016.
            - **Tarea:** Instalación
                - **Inicio:** Lunes 08/02/2016.

        - **Finalización:** Jueves 24/05/2016.
    - **Tarea:** Estudio de Tecnología
        - **Inicio:** Miércoles 03/02/2016.
        - **Finalización:** Sábado 02/04/2016.
- **DISEÑO:**
    - **Tarea:** Estética
        - **Inicio:** Martes 15/03/2016.
        - **Finalización:** Viernes 18/03/2016.
    - **Tarea:** Implementación
        - **Inicio:** Miércoles 16/03/2016.
        - **Finalización:** Lunes 02/05/2016.
    - **Tarea:** Detalles
        - **Inicio:** Lunes 04/04/2016.
        - **Finalización:** Domingo 29/05/2016.

- **MEMORIA DEL PROYECTO:**
    - **Tarea:** Plantilla
        - **Inicio:** Sábado 14/05/2016.
        - **Finalización:** Sábado 14/05/2016.
    - **Tarea:** Memoria
        - **Inicio:** Lunes 16/05/2016.
        - **Finalización:** Miércoles 15/06/2016.
- **PRESENTACIÓN DEL PROYECTO:**
    - **Tarea:** Plantilla
        - **Inicio:** Viernes 17/06/2016.
        - **Finalización:** Viernes 17/06/2016.

- **Tarea:** Presentación
  - **Inicio:** Sábado 18/06/2016.
  - **Finalización:** Jueves 23/06/2016.

2.2.2. DIAGRAMA DE GANTT

El diagrama de Gantt es una herramienta que permite modelar la planificación de las tareas necesarias para la realización de un proyecto. Como las tareas y sus plazos ya se han definido anteriormente, a través de la figura 3 se muestra la distribución ordenada de esos datos.

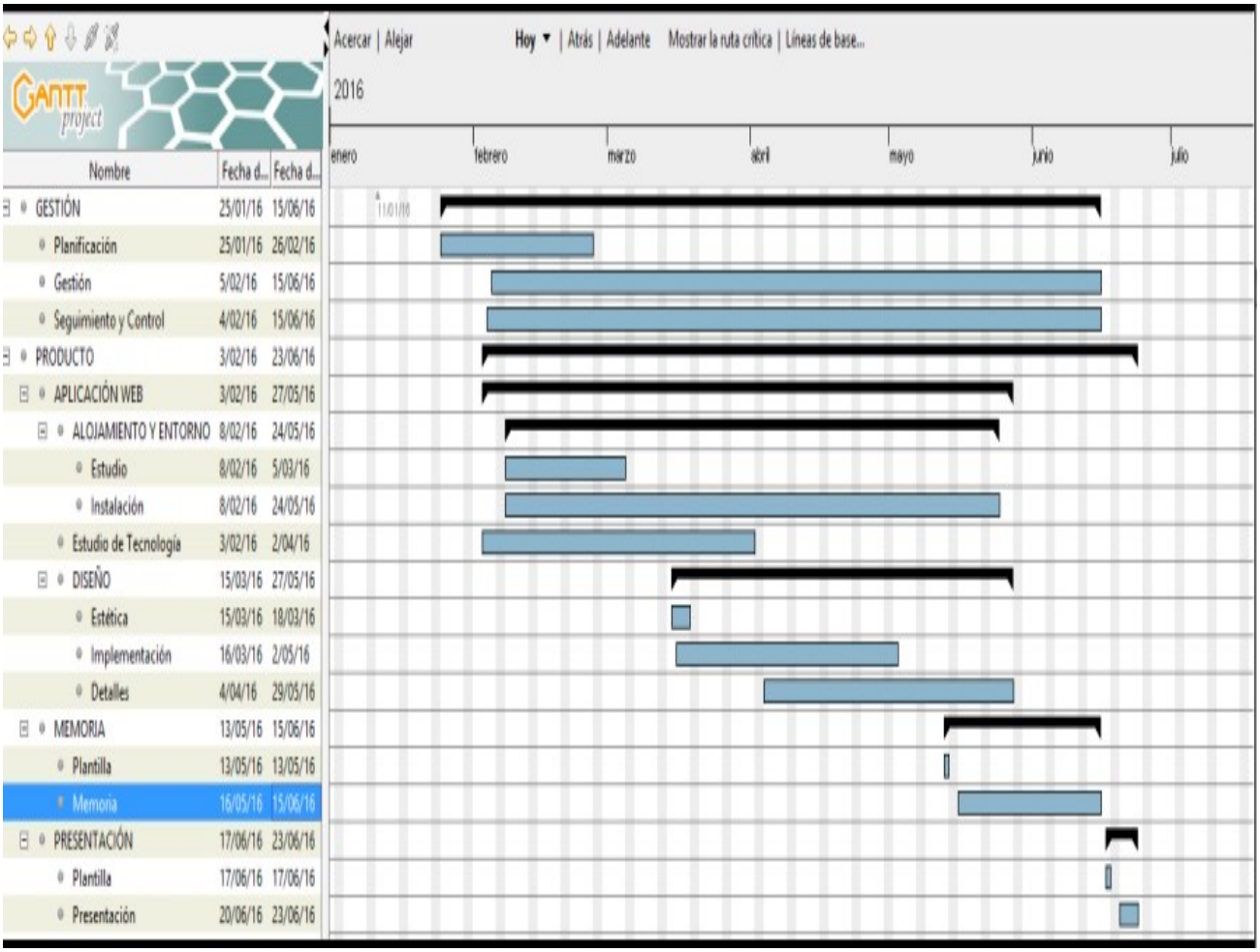

Figura 3 Diagrama Gantt

3

ANÁLISIS

En este punto se tiene por objetivo entender y describir qué es exactamente lo que se quiere conseguir con el proyecto y describir su funcionalidad. El análisis especifica qué debe hacer el sitio web para cumplir los requisitos especificados.

3.1. CAPTURA DE REQUISITOS

Antes de empezar a realizar cualquier análisis de las funcionalidades, se realiza un análisis de los requisitos sobre las necesidades de la aplicación web. Esos requisitos se han agrupado en dos tipos:

- Requisitos no-funcionales:
  - El frontend es un equipo PC.
  - Hacer uso de los puntos fuertes de Meteor que ayuden a gestionar funcionalidades esenciales de una aplicación web.
  - Probar el uso de la base de datos.
- Requisitos funcionales:
  - Cualquier persona podrá registrarse en la aplicación.
  - Un usuario ya registrado podrá iniciar sesión en la aplicación.

- La información sobre el perfil del usuario puede ser modificada en cualquier momento por el propietario de ese perfil.
- Todos los usuarios pueden acceder a ver la información de los festivales creados en la aplicación.
- Todos los usuarios registrados pueden crear un festival. Dentro de esta opción, se puede generar una propuesta de posible calendario para dicho festival.
- Los usuarios que hayan creado un festival pueden eliminarlo cuando lo vean conveniente.
- Todos los usuarios registrados pueden apuntarse o desapuntarse de los festivales.
- Para los usuarios registrados, se crea una sección con los seguidores, gente a los que sigue el usuario y todos los usuarios registrados de la aplicación.
- Los usuarios registrados pueden acceder a la información sobre el perfil de otros usuarios. Dentro, se puede seguir o dejar de seguir a esos usuarios.

3.2. CASOS DE USO

Siguiendo con los requisitos planteados en el punto anterior, a continuación se especifican todos los casos de uso para los diferentes actores de la aplicación.

Para definir bien el proyecto, primero hay que definir los actores que participan en la aplicación. En este caso se trabaja con dos tipos; por un lado los anónimos o no registrados, y por otro los registrados. Las acciones que cada uno de esos actores podrá realizar en la aplicación quedan definidos en la figura 4.

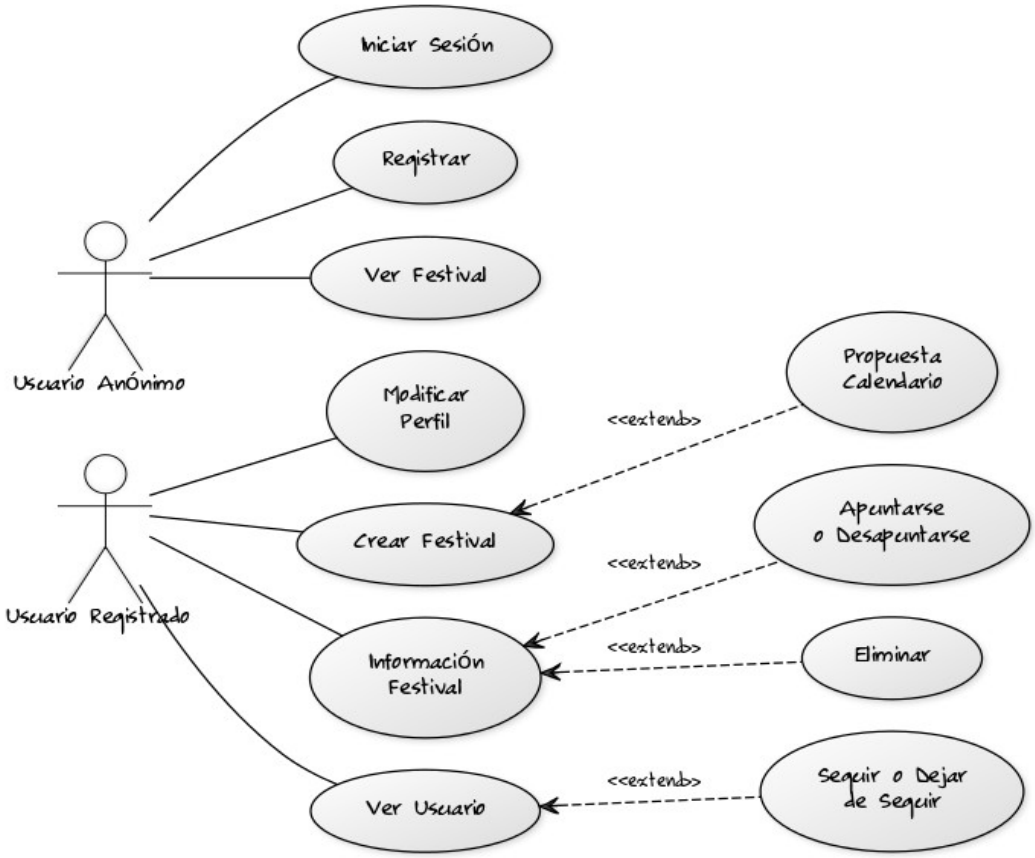

Figura 4 Diagrama de Casos de Uso

3.2.1. INICIAR SESIÓN

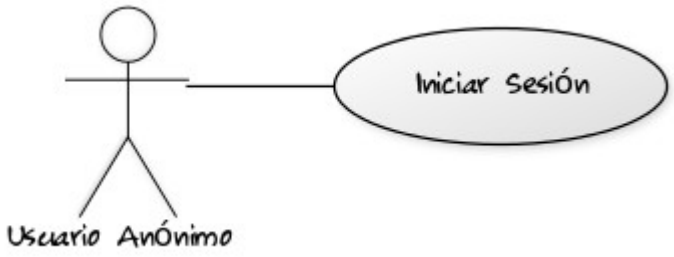

Figura 5 Caso de Uso: Iniciar Sesión

DESCRIPCIÓN: Proceso para acceder a la cuenta personal del usuario que lo solicite.

PRE-CONDICIÓN: Tener acceso a la página principal de la aplicación, así como tener una cuenta ya registrada en la aplicación.

FLUJO DE EVENTOS:

1. En la página principal acceder a la sección de iniciar sesión. Introducir los datos que se han proporcionado en el registro de la cuenta.

    - Si al introducir esos datos se comete alguna equivocación, se comunica al usuario y este vuelve a la opción de iniciar sesión. Si se introducen correctamente, se pasa a la página principal del propio usuario.

POST-CONDICIÓN: Accede a la página principal del usuario que lo ha solicitado. Esa página está personalizada a la situación actual del usuario en cuestión.

3.2.2. REGISTRAR

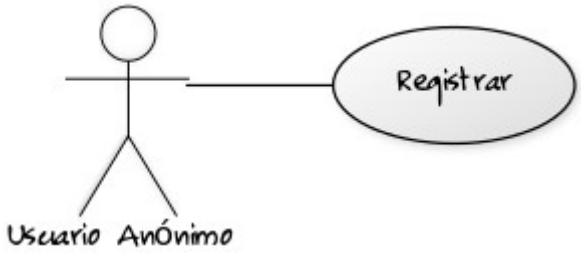

Figura 6 Caso de Uso: Registrar

DESCRIPCIÓN: Proceso para que un usuario genere una cuenta personal.

PRE-CONDICIÓN: Tener acceso a la aplicación web, en concreto a la página inicial del sitio.

FLUJO DE EVENTOS:

1. Dentro de la página principal, acceder al apartado para el registro.

2. Esa opción lleva a un formulario que el usuario rellena. Los campos a rellenar son nombre de usuario, email y contraseña, con un mínimo nivel de seguridad para todos ellos.

3. Una vez introducidos esos datos, el perfil del usuario queda almacenado en la aplicación. Se inicia sesión para la cuenta recién creada.

POST-CONDICIÓN: Se almacenan los datos introducidos en la base de datos, para poder consultarlos cuando el usuario lo requiera. También se hace uso de los mismos al iniciar sesión.

3.2.3. CONSULTAR FESTIVAL

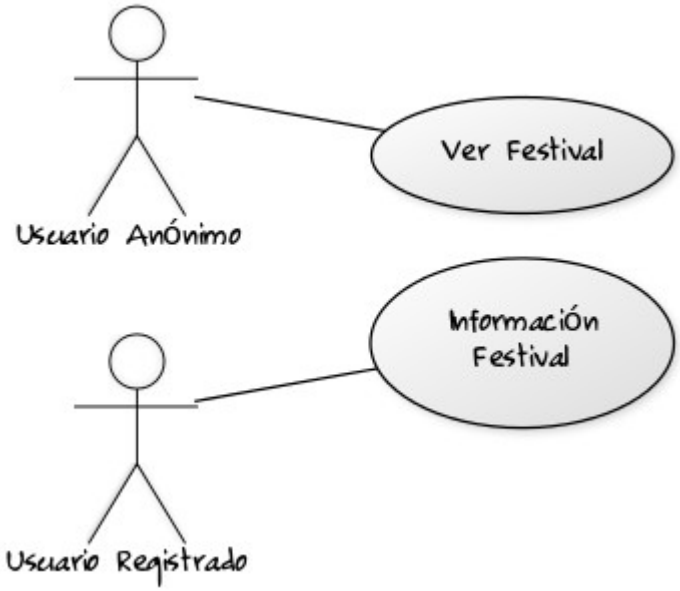

Figura 7 Caso de Uso: Consultar Festival

DESCRIPCIÓN: Todos los usuarios de la aplicación tienen la opción de mirar la información sobre un festival.

PRE-CONDICIÓN: Cualquier usuario de la aplicación, tanto anónimos como registrados, pueden acceder a esta opción. Los anónimos acceden a la página inicial de la aplicación, y los usuarios que han iniciado sesión a la página principal de su cuenta.

FLUJO DE EVENTOS:

1. En las páginas mencionadas, dependiendo la situación, el usuario encuentra todos los festivales que se han creado en la aplicación, ordenados por la fecha de inicio de forma ascendente.

2. Una vez escogido el festival, se presiona sobre el mismo y aparece una página con toda la información asociada a dicho festival. Los festivales no aparecen si ya ha pasado la fecha de inicio.

POST-CONDICIÓN: El usuario tiene accesible toda la información sobre el festival que le interesa.

3.2.4. APUNTAR O DESAPUNTAR DE FESTIVAL

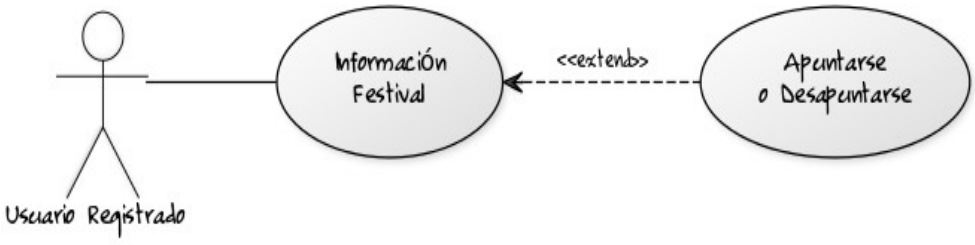

Figura 8 Caso de Uso: Apuntar o Desapuntar de Festival

DESCRIPCIÓN: Los usuarios registrados pueden indicar su asistencia al evento, a modo de información ya que no garantiza la reserva.

PRE-CONDICIÓN: Los usuarios tienen que estar registrados en la aplicación, con sesión iniciada.

FLUJO DE EVENTOS:

1. Dentro de la página principal del usuario, acceder a la información del festival al que desea asistir, como se ha explicado en un punto anterior. Se puede acceder a cualquier festival que esté tanto en la sección “Festivales Creados”, “Mis Festivales” o “Buscar Festivales”.

2. Dentro de la información del festival hay un cuadrado con la información del aforo del sitio, el número de asistentes, y dos flechas (una hacia arriba y otra hacia abajo).

    - Si se presiona sobre la flecha hacia arriba se suma uno al número de asistentes, y ese festival queda registrado en la sección “Mis Festivales” del usuario conectado. En cambio, si se presiona sobre la flecha hacia abajo se resta uno al número de asistentes, y ese festival queda eliminado de la sección “Mis festivales” del usuario conectado. Por lo tanto, si no ha indicado su asistencia sólo se puede presionar la flecha positiva, y si previamente se había indicado asistencia la flecha negativa. Al llegar al aforo máximo, no deja que más usuarios se apunten al festival.

POST-CONDICIÓN: El usuario se apunta o desapunta del festival, ajustando el número de asistentes al mismo.

3.2.5. ELIMINAR FESTIVAL

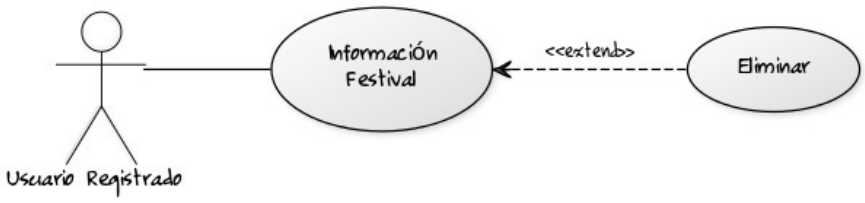

Figura 9 Caso de Uso: Eliminar Festival

DESCRIPCIÓN: Proceso para eliminar un festival de la aplicación.

PRE-CONDICIÓN: Los usuarios que quieran acceder a esta opción tienen que estar registrados en la aplicación y tener iniciada sesión en la aplicación. Así mismo, la aplicación a eliminar tiene que haber sido creada por el usuario que quiere realizar la acción.

FLUJO DE EVENTOS:

1. Dentro de la página del usuario, en la sección de “Festivales Creados”, se busca el festival que se quiere eliminar (en los otros apartados también se puede encontrar, pero la búsqueda es más complicada). Una vez encontrado, acceder a la información de dicho festival.

2. Como creador del festival en cuestión, junto a la opción de “Salir” aparece la de “Eliminar”. Presionar el último botón mencionado y el festival queda eliminado.

POST-CONDICIÓN: Se elimina el festival deseado de la aplicación.

3.2.6. CREAR FESTIVAL

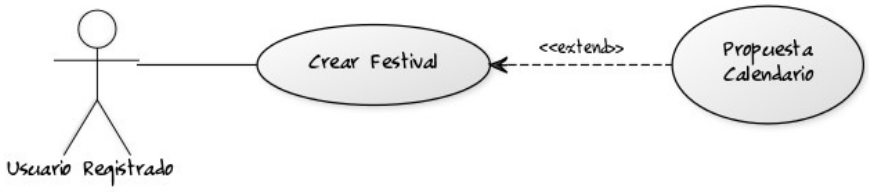

Figura 10 Caso de Uso: Crear Festival

DESCRIPCIÓN: Proceso para crear un evento y mostrar su información, con opción de recibir una propuesta de horarios.

PRE-CONDICIÓN: Los usuarios que quieren acceder a esta opción tienen que estar registrados y tener iniciada sesión en la aplicación.

FLUJO DE EVENTOS:

1. Dentro de la página principal del usuario, existe una opción llamada “Crear Festival”. Entrar en la misma.

2. Dentro hay un formulario con unos parámetros a introducir, de los cuales algunos son obligatorios (están marcados con un asterisco).

    - Dentro del formulario, existe la opción de generar una propuesta de horarios una vez se hayan introducido las fechas del festival. Dentro, en primer lugar se solicita introducir el número de grupos, horarios y escenarios, para posteriormente introducir los nombres de cada uno de los grupos, horarios y escenarios, junto a una puntuación para cada uno de ellos. Finalmente, al presionar el botón de “Generar” se presenta una propuesta por parte de la aplicación, que no se almacena.

3. Dar a la opción de “Guardar” para almacenar los datos introducidos (la propuesta no). Si no se quiere guardar se presiona el botón “Cancelar”.

POST-CONDICIÓN: Se genera una página dedicada al festival, con la información que el usuario ha decidido introducir. Dicho festival queda accesible para cualquier usuario de la aplicación.

3.2.7. MODIFICAR EL PERFIL

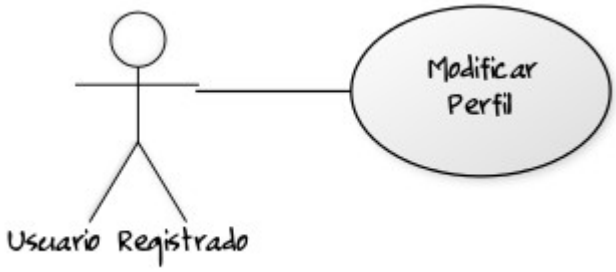

Figura 11 Caso de Uso: Modificar el Perfil

DESCRIPCIÓN: Proceso para modificar los datos personales del usuario conectado. Estos datos son visibles para el resto de usuarios de la aplicación.

PRE-CONDICIÓN: Los usuarios que quieren acceder a esta opción tienen que estar registrados en la aplicación, con sesión iniciada.

FLUJO DE EVENTOS:

1. Una vez esté en su página personal, el usuario accede a la opción de “Modificar Perfil”.

2. Dentro de esa opción, para modificar los datos, aparecen todos los campos de los atributos correspondientes al perfil del usuario. En comparación con el registro, aquí aparecen mas atributos.

3. Los datos que no se rellenen se mantendrán como estaban. Al cambiar esos datos, se presiona un botón para confirmarlos y quedan registrados en la base de datos. Al usuario se le comunica que se han realizado esos cambios correctamente.

POST-CONDICIÓN: Cambio de la información del perfil del usuario, actualizado a las nuevas exigencias del usuario.

3.2.8. CONSULTAR USUARIO

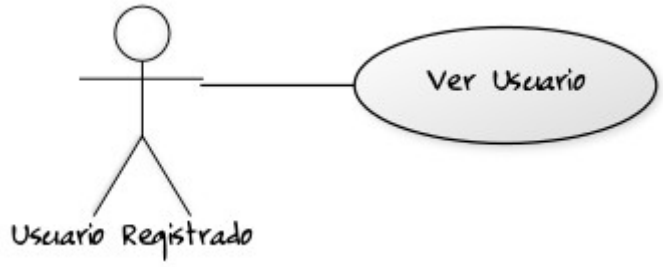

Figura 12 Caso de Uso: Consultar Usuario

DESCRIPCIÓN: Los usuarios registrados pueden acceder a la información sobre el resto de usuarios registrados en la aplicación.

PRE-CONDICIÓN: Tener una cuenta registrada en la aplicación, y tener una sesión iniciada con dicha cuenta.

FLUJO DE EVENTOS:

1. Dentro de la página principal de la cuenta del usuario, este encuentra una sección con el título “Contactos”. En la misma, hay tres apartados: “Siguiendo”, “Seguidores” y “Encuentra Gente”, en las que aparecen los nombres de usuario de los usuarios registrados en la aplicación. Se busca a la persona que se quiere consultar en cualquiera de los tres desplegables mencionados (los usuarios que aparecen en “Siguiendo” y “Seguidores” también aparecen en “Encuentra Gente”).

2. Una vez encontrada la persona deseada, presionar sobre su nombre de usuario y esa acción lleva a la página con la información asociada a dicho usuario (la que el propio usuario ha introducido en la aplicación).

POST-CONDICIÓN: El usuario visualiza la información asociada a otro usuario.

3.2.9. SEGUIR O DEJAR DE SEGUIR USUARIO

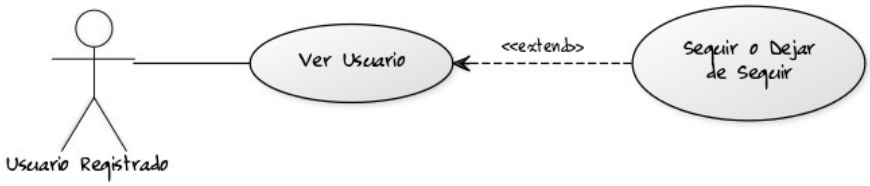

Figura 13 Caso de Uso: Seguir Usuario

DESCRIPCIÓN: Cada usuario puede seguir a otro(s) usuario(s) que también estén registrados en la aplicación.

PRE-CONDICIÓN: Tener una cuenta registrada en la aplicación, y tener una sesión iniciada con dicha cuenta.

FLUJO DE EVENTOS:

1. Dentro de la página principal de la cuenta del usuario, este encuentra una sección con el título “Contactos”. En la misma, se busca al usuario que se quiere seguir o dejar de seguir, a través del nombre de usuario del mismo, en “Seguidores”, “Siguiendo” o “Encuentra Gente”. Una vez encontrado, presionar sobre el nombre del usuario.

2. Dentro de la información del usuario hay unos botones para “Seguir” o “Dejar de Seguir” a ese usuario.

    - Si se presiona el botón “Seguir” se añade ese usuario al grupo de “Siguiendo”. Además, al usuario conectado se le añade al grupo de “Seguidores” del usuario a seguir. Si se presiona sobre el botón “Dejar de Seguir” el usuario en cuestión desaparece del grupo de “Siguiendo” y, por consecuencia, el usuario conectado desaparece del grupo de “Seguidores” del usuario al que se le ha dado a dejar de seguir. Por lo tanto, si no se seguía a ese usuario sólo se puede presionar el botón “Seguir”, y si previamente se había indicado que se seguía el botón “Dejar de Seguir”.

POST-CONDICIÓN: El usuario sigue o deja de seguir a otro de los registrados en la aplicación. Así mismo, en el caso de que se empiece a seguir, al otro usuario le aparece el usuario conectado entre su grupo de seguidores.

4

DISEÑO

A continuación, se describe el diseño de la aplicación mediante los diagramas de contexto, de clases y de secuencia. En este apartado se muestran los diagramas correspondientes a las funcionalidades desarrolladas en el proyecto.

4.1. DIAGRAMA DE CONTEXTO

Mediante el diagrama de contexto se muestran las interacciones existentes entre los agentes externos y el sistema a través de flujos de datos. El sistema de información se representa como un único proceso de muy alto nivel con entradas y salidas hacia los agentes externos que lo limitan.

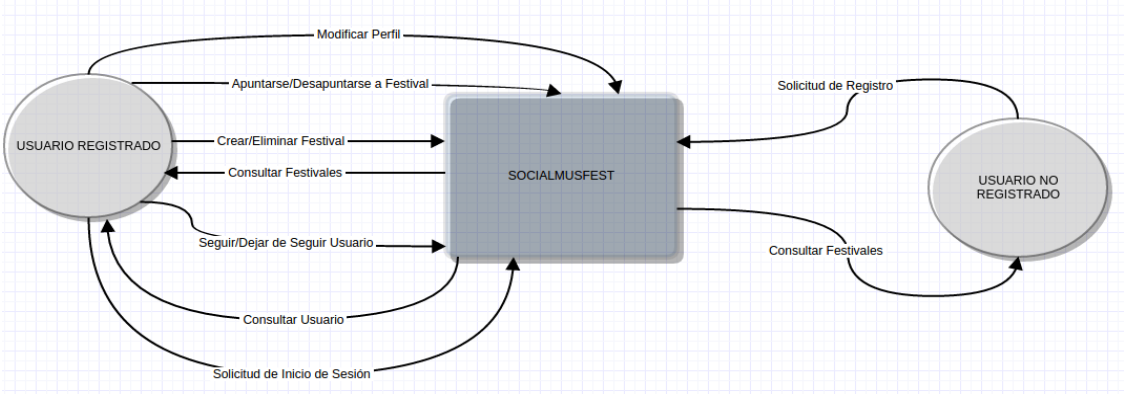

Figura 14 Diagrama de Contexto

4.2. DIAGRAMA DE CLASES

Mediante el diagrama de clases se consigue visualizar las relaciones entre las clases involucradas en el sistema. La representación de este estilo muestra los atributos, métodos y visibilidad de dichas clases, junto a las relaciones entre las mismas.

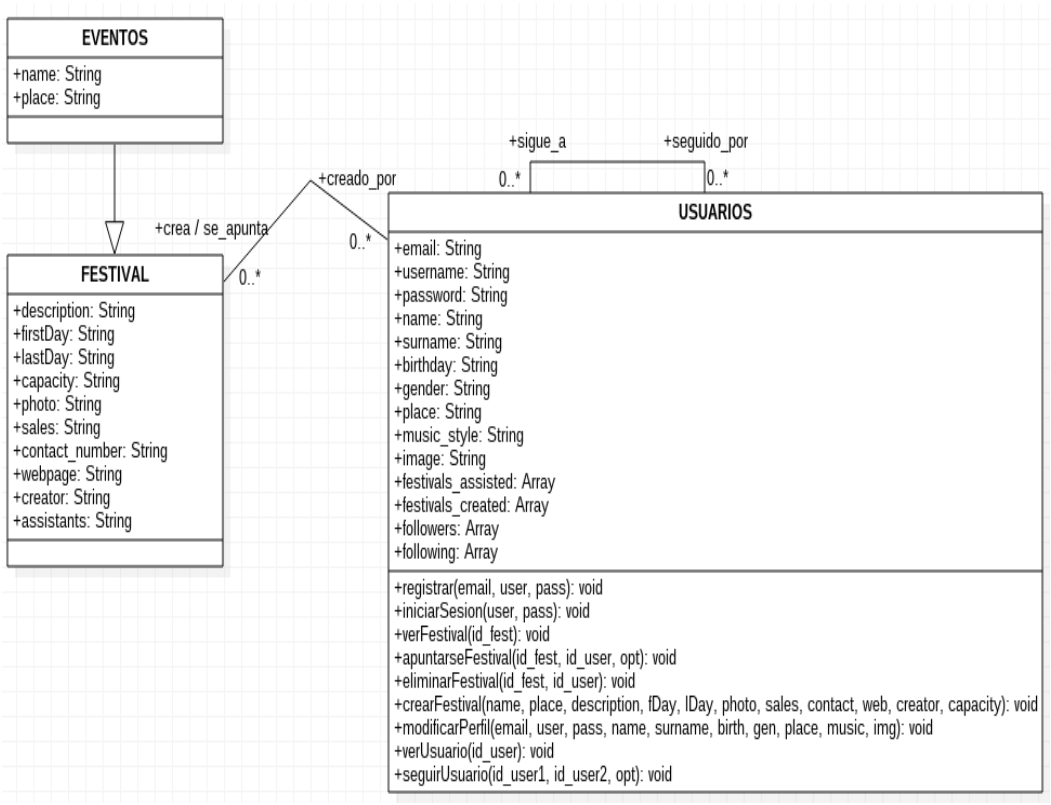

Figura 15 Diagrama de Clases

4.3. DIAGRAMA DE SECUENCIA

Este tipo de diagramas ayudan a mostrar los objetos envueltos en un escenario y la secuencia de mensajes intercambiados entre los objetos necesarios para realizar la funcionalidad. Por lo tanto, para cada una de las funcionalidades de la aplicación se crea un diagrama,

4.3.1. INICIAR SESIÓN

Esta funcionalidad da acceso a los usuarios a sus cuentas personales dentro de la aplicación, permitiendo realizar las acciones pertinentes en este lado y gestionar sus datos personales.

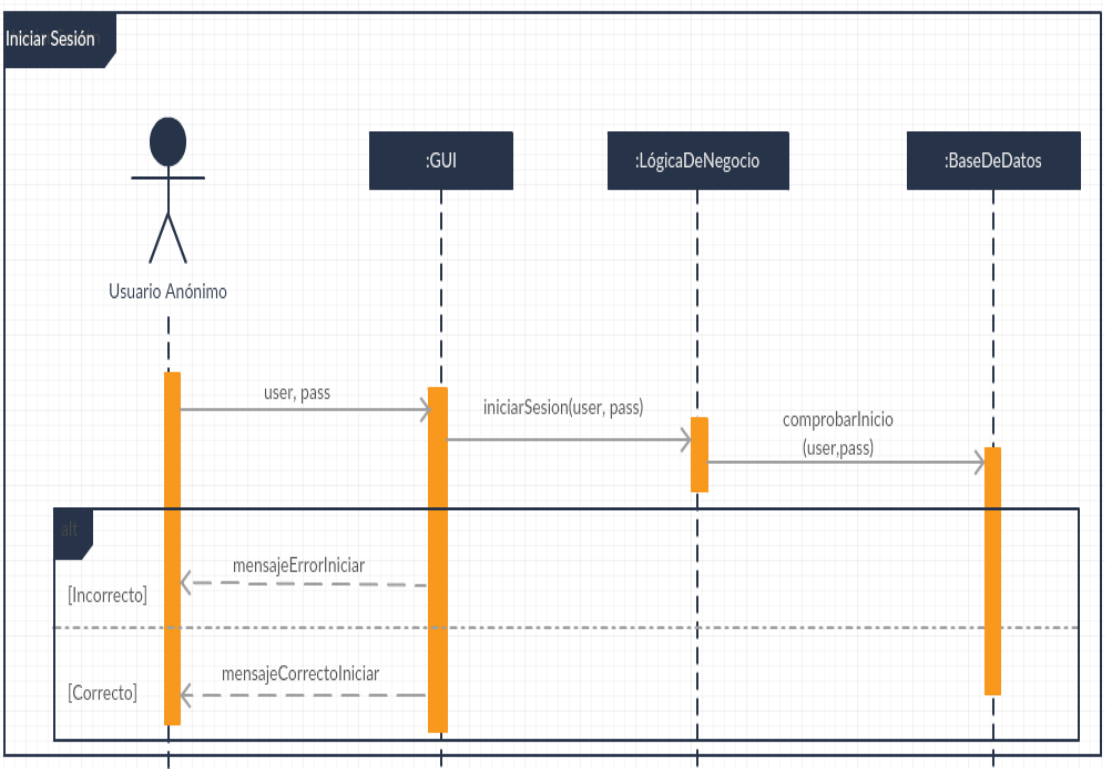

Figura 16 Diagrama de Secuencia: Iniciar Sesión

4.3.2. REGISTRAR

Mediante el registro, los usuarios anónimos pueden crear su perfil en la aplicación y tener acceso a funciones propias de los usuarios registrados.

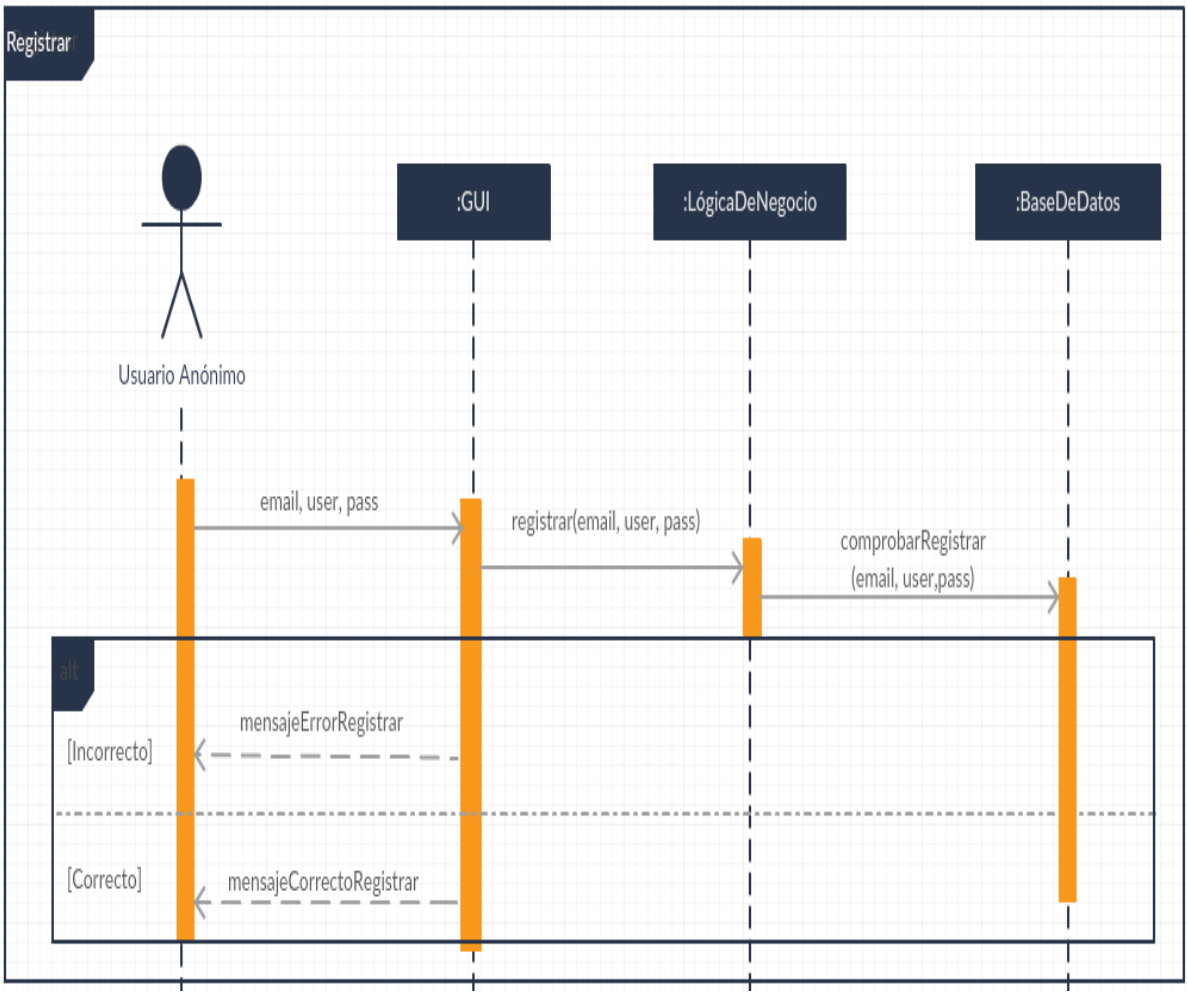

Figura 17 Diagrama de Secuencia: Registrar

4.3.3. CONSULTAR FESTIVAL

Cualquier usuario puede consultar la información sobre cualquier festival que esté registrado en la aplicación, accediendo a datos que el creador de dicho festival ha dejado accesibles en la página del festival.

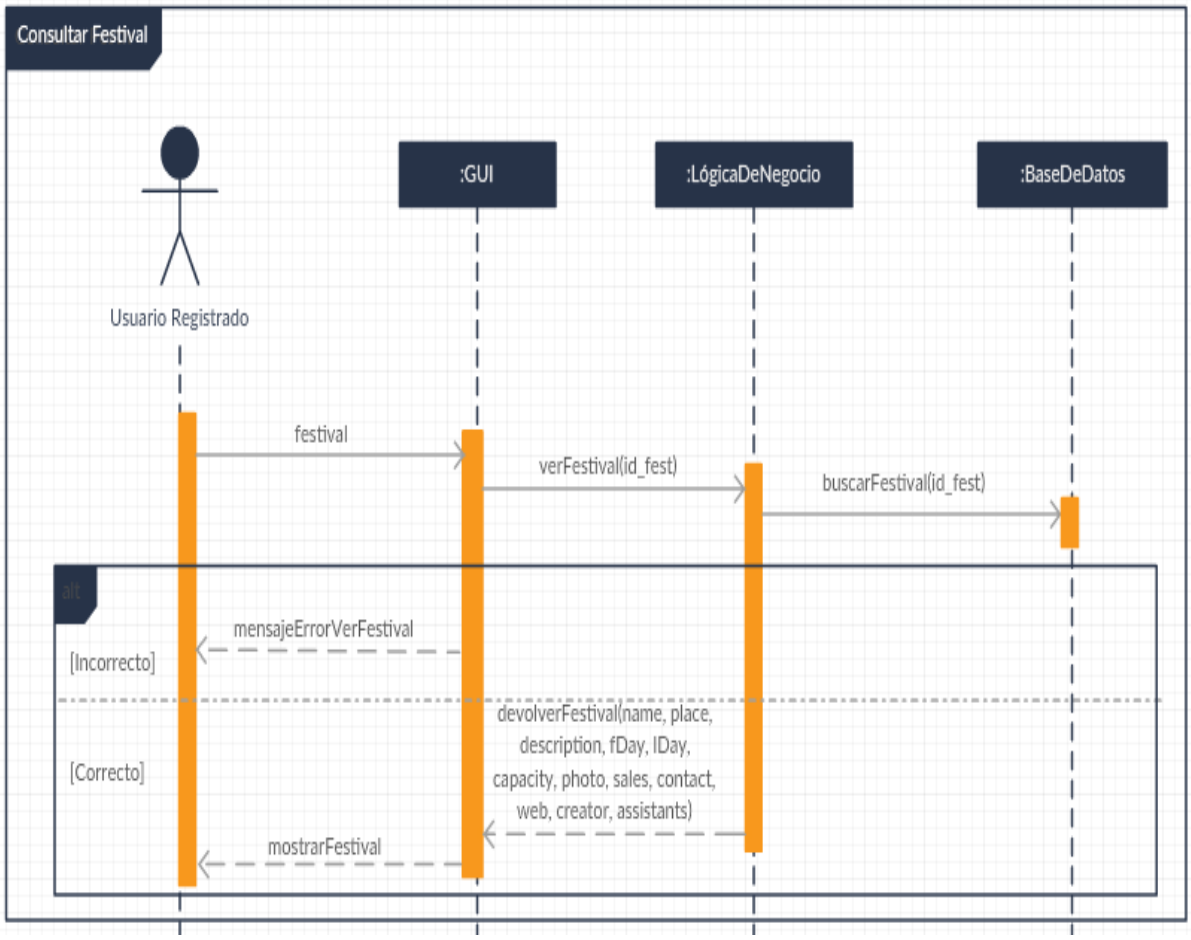

Figura 18 Diagrama de Secuencia: Consultar Festival

4.3.4. APUNTAR O DESAPUNTAR DE FESTIVAL

Dentro de las páginas de cada uno de los festivales registrados en la aplicación, existe la opción para los usuarios registrados de poder apuntarse o, en caso de estar ya apuntado, desapuntarse. Con esta acción los festivales en los que se ha apuntado un usuario registrado quedarán agrupados, y se puede hacer una estimación de la gente que va a asistir a dicho festival.

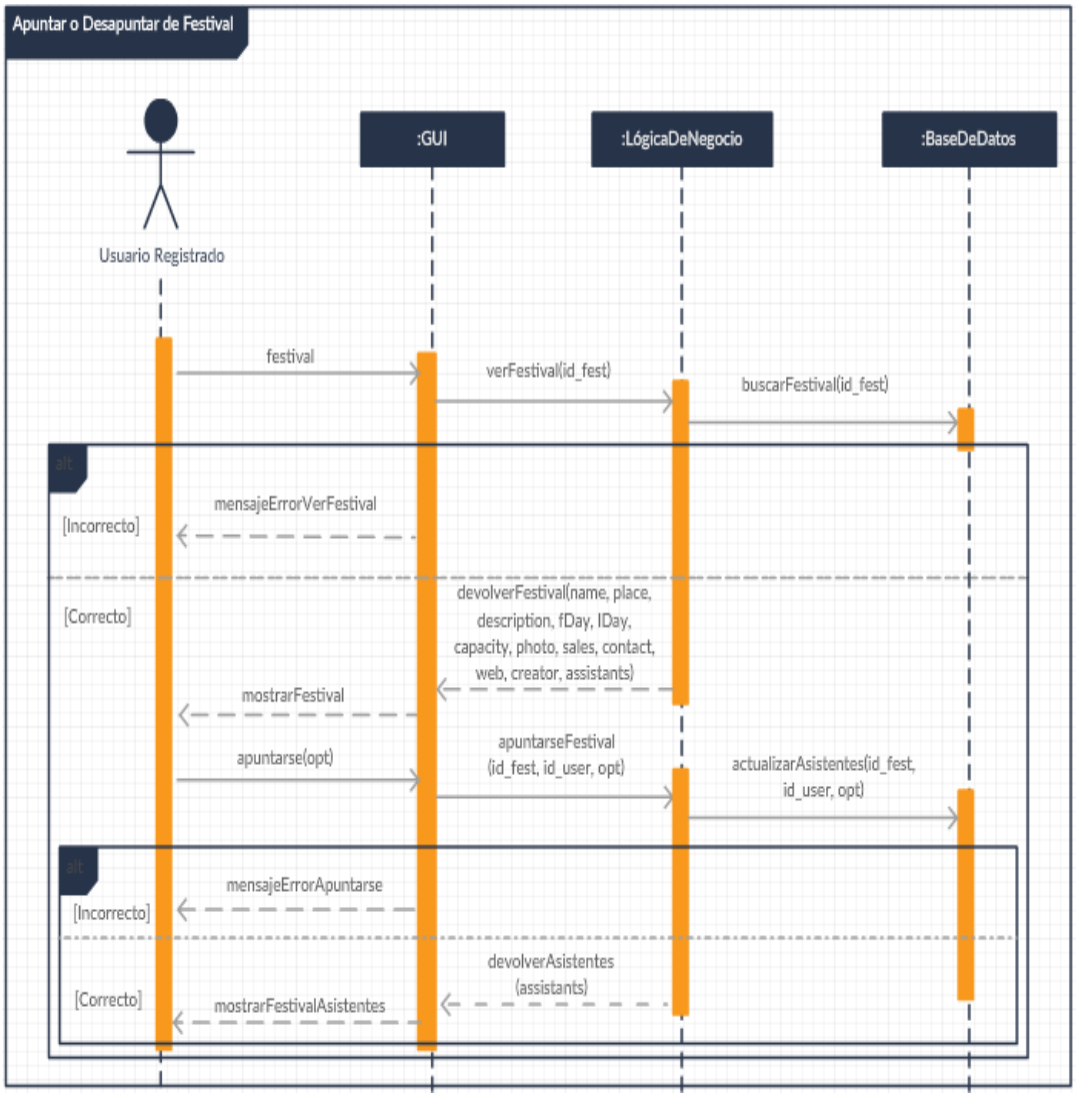

Figura 19 Diagrama de Secuencia: Apuntar o Desapuntar Festival

4.3.5. ELIMINAR FESTIVAL

Los usuarios que tengan un festival creado pueden eliminarlo cuando lo deseen. Esto hace que no se vuelva a mostrar de nuevo en la aplicación, dejando de mostrar toda la información asociada.

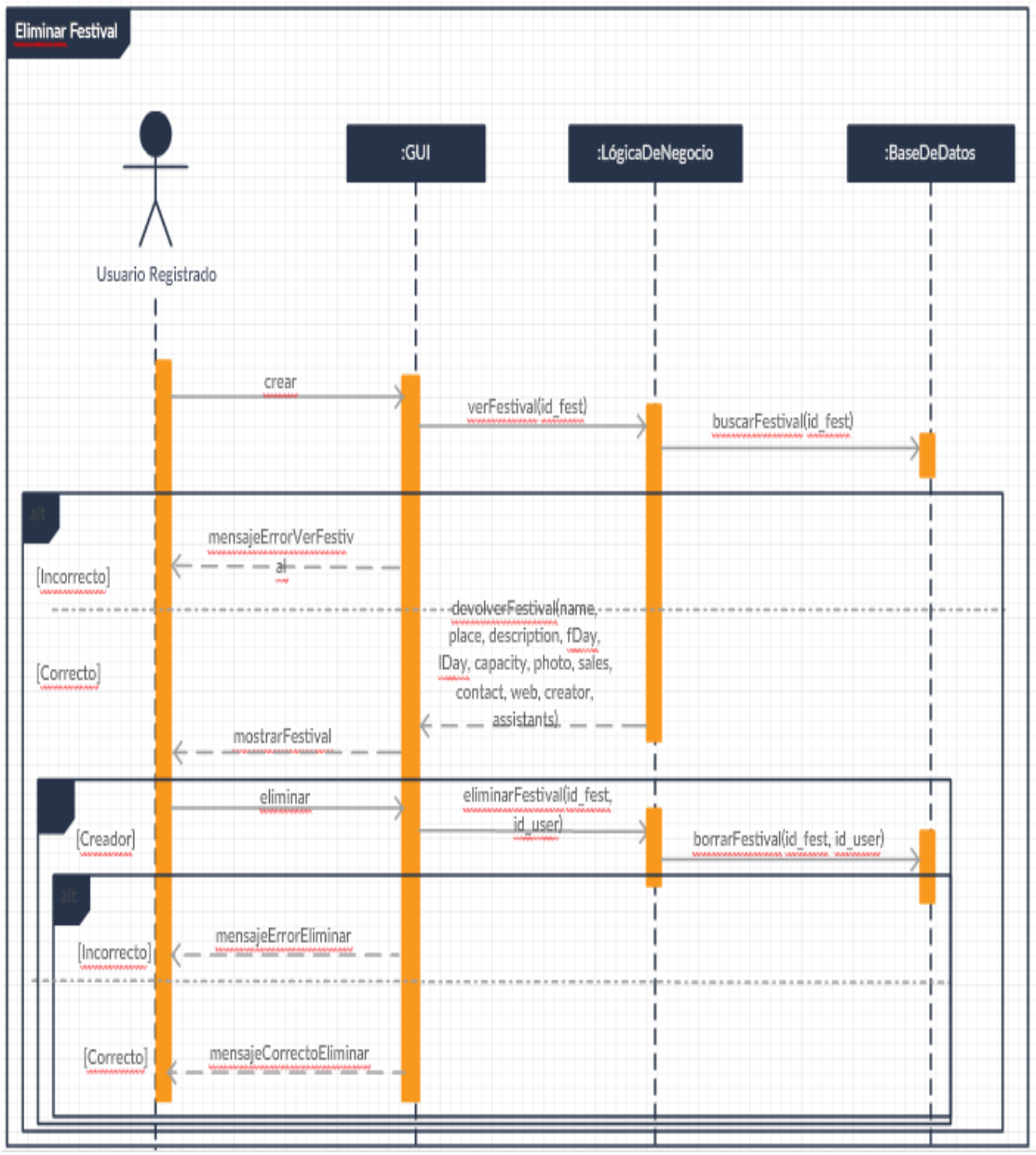

Figura 20 Diagrama de Secuencia: Eliminar Festival

4.3.6. CREAR FESTIVAL

Todos los usuarios registrados en la aplicación pueden crear un festival; es decir, introducir información asociada a un festival para ser mostrada al resto de usuarios de la aplicación. No se permite introducir más de un festival con el mismo nombre, pero cada uno de los usuarios tiene la opción de crear más de un festival.

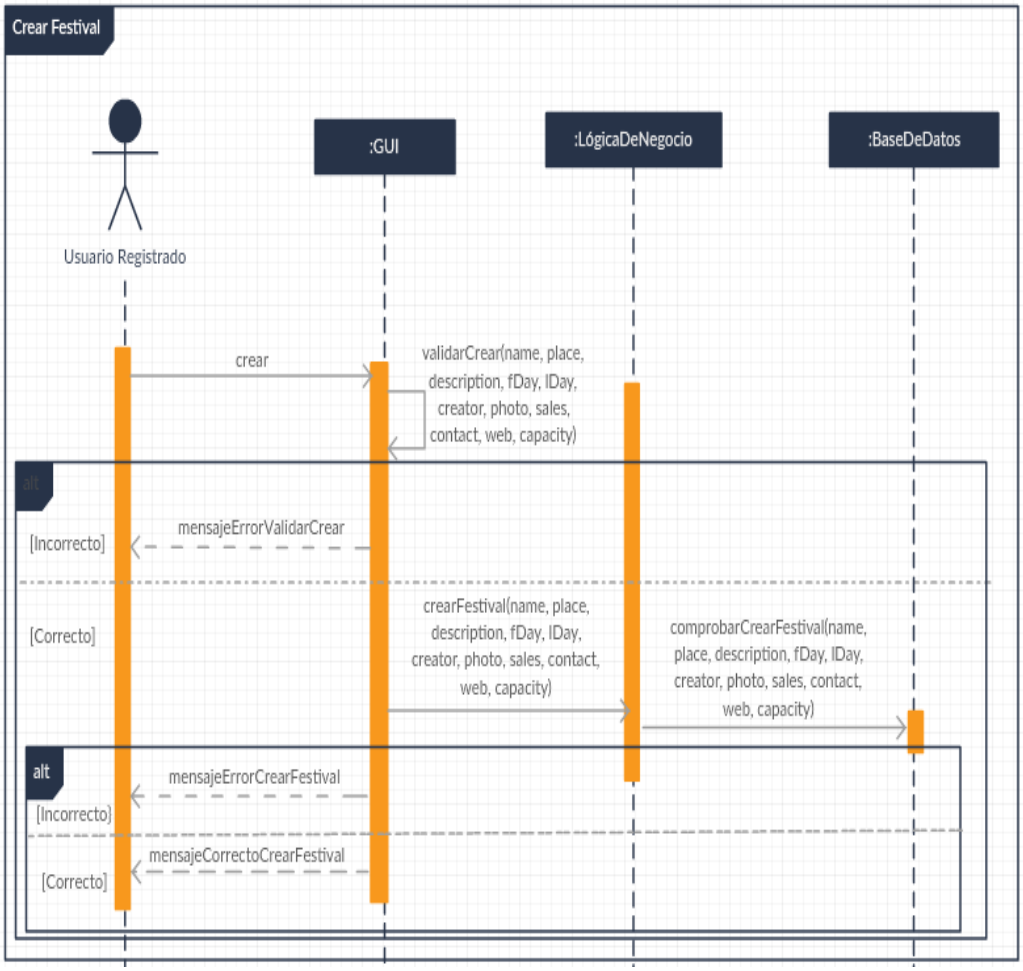

Figura 21 Diagrama de Secuencia: Crear Festival

4.3.7. MODIFICAR EL PERFIL

Los usuarios registrados en la aplicación pueden cambiar su información del perfil en cualquier momento. Esos cambios se llevan a cabo a través de un formulario en el cual, si no se introduce ningún dato sobre algún campo, este se mantiene tal y como estaba.

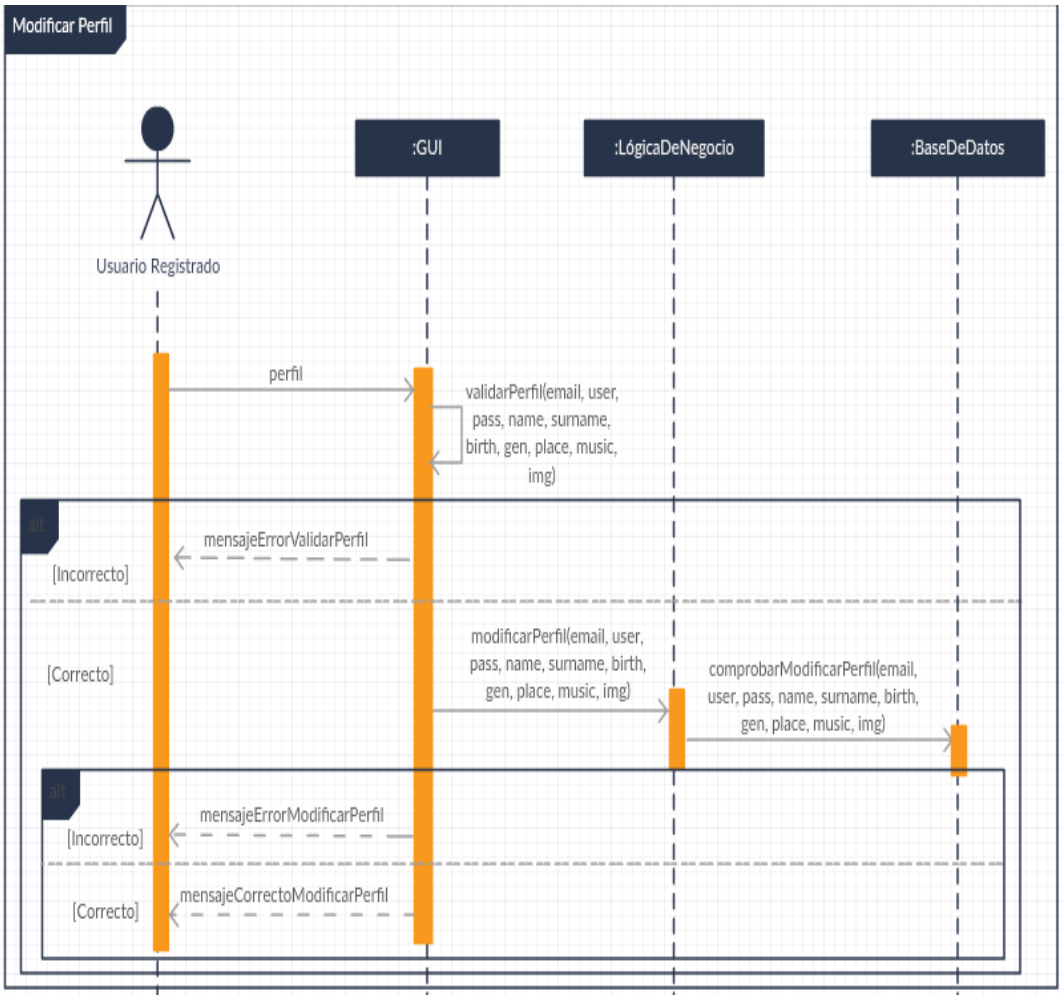

Figura 22 Diagrama de Secuencia: Modificar Perfil

4.3.8. CONSULTAR USUARIO

Sólo los usuarios registrados pueden consultar la información sobre el resto de usuarios de la aplicación, accediendo a la información que cada usuario ha introducido de su perfil.

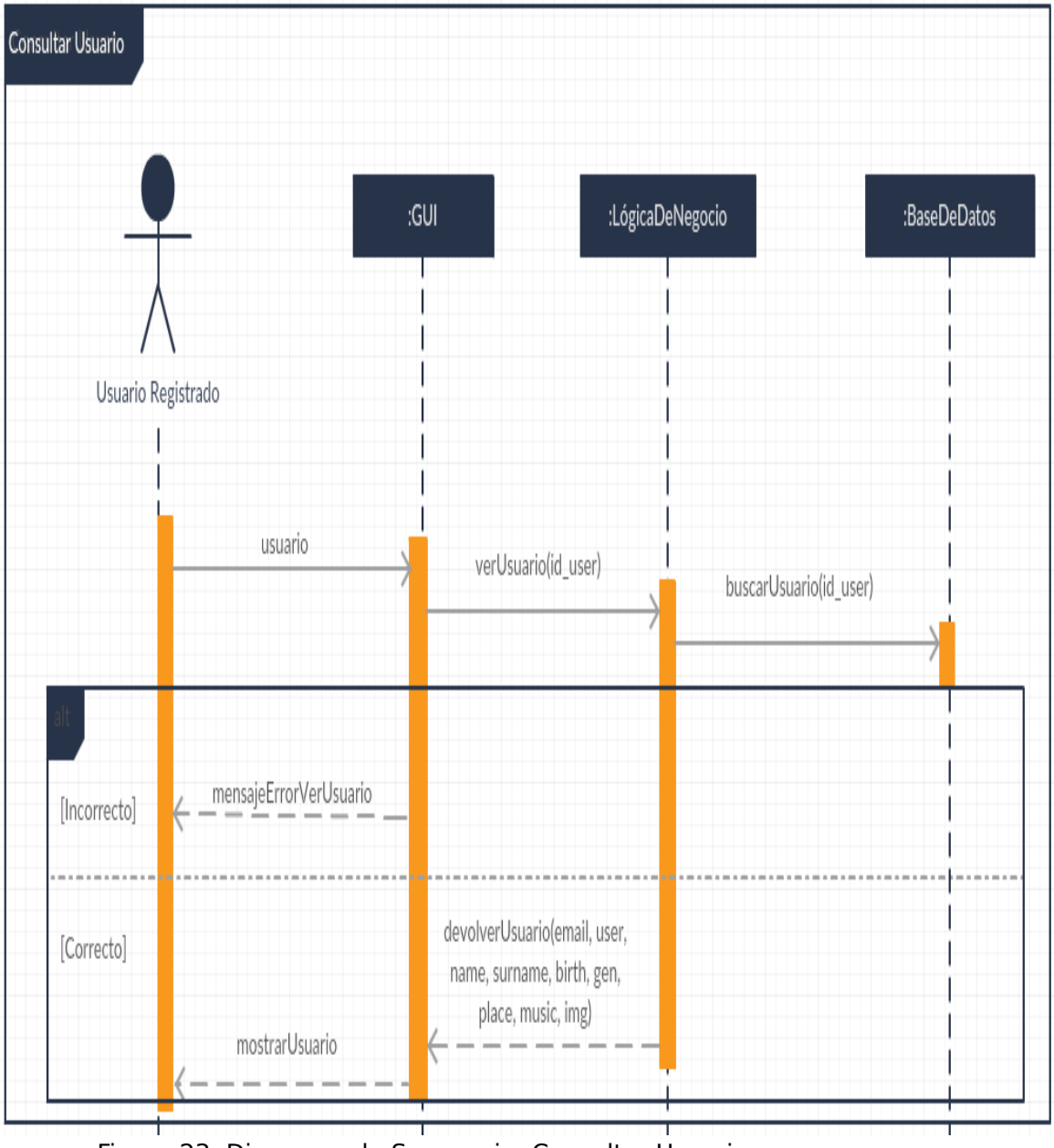

Figura 23 Diagrama de Secuencia: Consultar Usuario

4.3.9. SEGUIR O DEJAR DE SEGUIR USUARIO

Dentro de las páginas de cada uno de los usuarios registrados en la aplicación, existe la opción de poder seguir o, si ya se sigue dejarle de seguir, a dichos usuarios. Con esta acción se pueden mostrar usuarios que se siguen y usuarios que siguen al usuario conectado, facilitando para un futuro el sistema de red social.

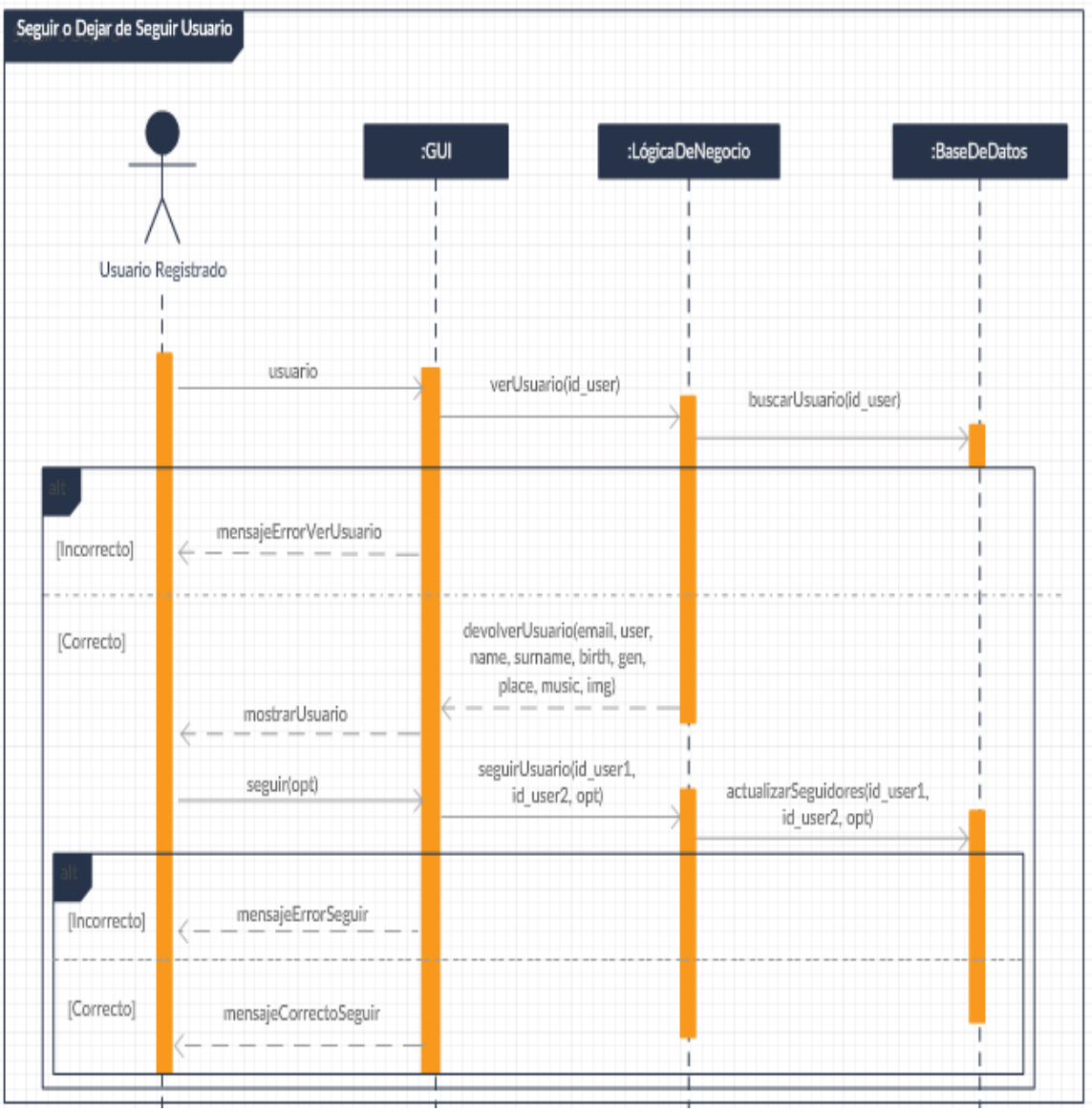

Figura 24 Diagrama de Secuencia: Seguir o Dejar de Seguir Usuario

5

TECNOLOGÍA

En este capítulo de la memoria se describe el resultado de un análisis profundo de la tecnología utilizada en este proyecto. Dicha tecnología es Meteor, una plataforma para crear aplicaciones web en tiempo real construida sobre Node.js.

Node.js es un entorno JavaScript del lado del servidor, basado en eventos. Node.js ejecuta JavaScript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. Aprovechando ese motor, Node.js proporciona un entorno de ejecución del lado del servidor que compila y ejecuta JavaScript a velocidades increíbles.

Continuando con Meteor, al ser un framework de código abierto su futuro se prevé que está garantizado, dada la gran comunidad que le acompaña y su gran actividad [4].

Junto a muchas de sus virtudes, existen una serie de fortalezas muy positivas y de gran utilidad para los desarrolladores de aplicaciones web, ya que gracias a su sistema de paquetes muchas funcionalidades quedan en mano del propio framework. Esos paquetes están desarrollados por la comunidad de Meteor, y son muy fáciles de instalar dentro de los proyectos; con un sólo comando se tiene la funcionalidad de dicho paquete. Gracias a ello, los desarrolladores de aplicaciones web pueden “olvidarse” de ciertos aspectos, invirtiendo menos tiempo para crear los productos. Este tema se explica con más detalles dentro de este capítulo.

5.1. METEOR

Este tipo de tecnología surgió de un grupo de desarrolladores formado por ex-empleados de Google, Asana y el creador de Etherpad, en 2011. Con el tiempo, varias empresas (Ikea, Mazda, Honeywell, Qualcomm, PGA...) han confiado en Meteor como su framework de trabajo, aumentando así el número de usuarios de la misma. Según Google Trends, la búsqueda sobre este framework ha ido aumentado desde 2012, estando actualmente entre lo más buscado.

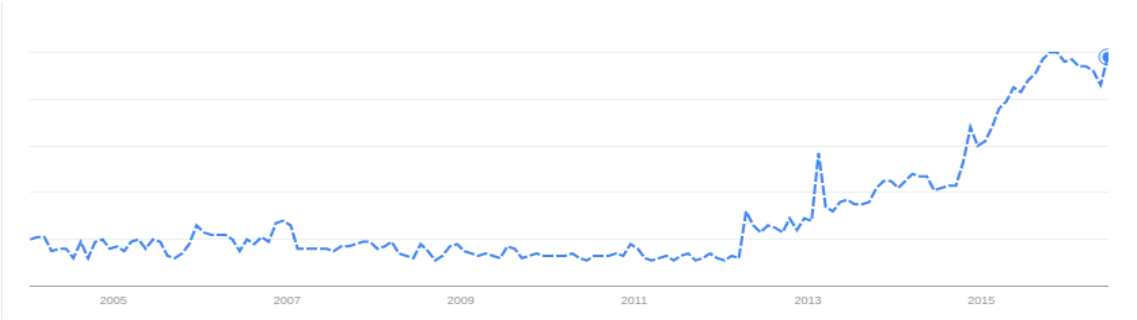

Figura 25 Gráfico de interés a lo largo del tiempo de Meteor

Meteor es un framework cliente-base de datos–servidor, escrito por completo en JavaScript. La parte de servidor de Meteor funciona sobre Node.js, pero eliminando muchos de los problemas existentes en esta tecnología.

Este framework contiene grandes ventajas para los desarrolladores que lo utilizan. Las aplicaciones Meteor están escritas en JavaScript, CSS y HTML, por lo que no se tiene que aprender un nuevo lenguaje específico de servidor. Además, proporciona un entorno de desarrollo fácil de instalar, que incluye un servidor web y una base de datos de servidor. Incluye un sistema de empaquetado fácil de usar y una fuente de datos reactiva, con un modelo de datos distribuido confortable, lo que significa que las aplicaciones dan una impresión de rapidez para múltiples usuarios.

Meteor incluye la capacidad de generar aplicaciones nativas iOS y Android desde la misma base de JavaScript. Así mismo, implementa un modelo isomórfico, lo que significa que las aplicaciones corren el mismo código, basado tanto en el servidor como en el cliente. Finalmente, este framework se basa en Node.js para ejecutar las

aplicaciones, y utiliza MongoDB como base de datos. Con todas estas características, el resultado es una plataforma muy potente y muy sencilla ya que Meteor abstrae muchas de las molestias y dificultades que se encuentran habitualmente en el desarrollo de aplicaciones web. Meteor permite crear una aplicación web en tiempo real en cuestión de horas.

5.2. PUNTOS FUERTES DE METEOR

Como ya se ha explicado en el punto anterior, Meteor dispone de varias características que lo hacen un framework sencillo pero a la vez potente. Tanto para los usuarios de las aplicaciones como para los desarrolladores de las mismas, este framework es una gran oportunidad para crear aplicaciones a medida para cualquier exigencia.

A continuación se mencionan algunas de las fortalezas que ofrece Meteor para los desarrolladores, haciéndolo competitivo ante otras opciones de desarrollo web.

- Meteor es una herramienta para la creación de aplicaciones web o móvil, tanto iOS como Android.

- Actualizaciones en tiempo real. Cualquier cambio que se realice en el proyecto (HTML, JavaScript, CSS, base de datos) se verá reflejado en tiempo real en cada uno de los navegadores que estén utilizando la aplicación.

- Compensación de retardo. Cuando algún cliente modifica o agrega datos, Meteor los muestra como si ya se hubieran guardado sin ninguna demora. Luego, si la conexión del cliente falla en alcanzar el servidor o algún otro inconveniente impide el guardado de datos, el usuario es informado y los cambios revertidos. Así se logra la sensación de que la aplicación funciona sin ningún tipo de demora.

- Toda la funcionalidad es escrita en un sólo lenguaje, JavaScript, que además es un lenguaje que tiene gran importancia dentro del desarrollo de aplicaciones web.

- Un sólo lenguaje entre servidor y cliente. Al estar basado en Node.Js, Meteor utiliza Javascript tanto en el lado del servidor

como en el cliente. De esta forma, permite crear productos más rápido y con menos errores.

- Beneficia tanto a los desarrolladores como a los clientes, ya que menos código se traduce en menos errores y mejor calidad.
- Es una herramienta muy fácil de entender. Su uso es muy sencillo, con una utilización de lenguajes conocidas por cualquier desarrollador.
- Un ecosistema de paquetes que acelera el desarrollo. Meteor tiene un repositorio oficial con paquetes y librerías que agregan funcionalidad extra, y esos paquetes están constantemente actualizados debido a que Meteor es una herramienta de código abierto y grandes desarrolladores alrededor del mundo trabajan en él.
- No presenta problemas de estabilidad. Sin embargo, no es inmune, ya que cualquier aplicación grande que sea muy accedida es susceptible a presentar problemas.
- Es escalable, el servidor se puede escalar a cualquier número de nodos.

5.2.1. GESTIÓN DE USUARIOS

Hasta la fecha, para cualquier desarrollador la gestión de usuarios era una tarea que él mismo tenía que realizar manualmente. Meteor sigue permitiendo esa opción, pero dado que al fin y al cabo puede ser un proceso repetitivo, los colaboradores de este framework han creado un paquete a través del cual se gestiona internamente esta tarea, haciendo que sea Meteor el que se ocupe del registro e inicio de sesión de los usuarios.

Dicho paquete es muy fácil de instalar. Mediante el comando *meteor add accounts-ui* se consigue introducir en el proyecto la funcionalidad, y dentro de la parte cliente se introduce, en la parte que mas se desee, el comando *{{> loginButtons}}* que está asociado a introducir en la aplicación los botones con las funcionalidades mencionadas.

Este último comando crea el apartado para el registro e inicio de sesión de usuarios. Como es algo genérico, que puede ser utilizada por cualquier aplicación, los campos solicitados en el mismo, tanto para el registro como en el inicio de sesión, son limitados. Más adelante se explica como introducir más atributos para la gestión de los usuarios, pero el paquete *accounts.ui* tiene dos opciones a escoger; o bien se trabaja con los atributos email y password, o sino con email, username y password.

La seguridad de todos esos campos es también gestionada por Meteor. Aunque dicha gestión no es muy exhaustiva, sí que se verifica que no haya dos usuarios con username y email iguales (si no hay username no se gestiona). Además, también se comprueba que en el campo del email se tenga el formato de un correo electrónico (“@server.com”). En el caso de la contraseña, esta tiene que ser de mínimo seis caracteres, dando igual de que tipo sean esos.

Además de este paquete, también cabe destacar otros paquetes que dan otras opciones de gestionar los usuarios. Como en otras plataformas, aquí también se pueden utilizar los enlaces con Google, Facebook, Twitter, GitHub, Meetup o Meteor Developers, todas aplicaciones utilizadas por los usuarios de Internet y de diferente ámbito, lo que abre muchas opciones dependiendo del tipo de usuario al que se quiere llegar. La configuración en esos casos es la misma que se realiza en otras plataformas, como por ejemplo Android. Así mismo, también existe otro modo de registro, menos utilizado pero que puede ser interesante; mediante el login basado en la seguridad de la contraseña (*accounts-password*) [8].

Por lo tanto, aunque siempre queda abierta la opción de gestionar manualmente los usuarios, mediante los paquetes creados por la comunidad de Meteor esta gestión puede quedar en manos del framework. Los paquetes que se pueden usar son los siguientes:

- meteor add accounts-ui
- meteor add accounts-password
- meteor add accounts-facebook
- meteor add accounts-google
- meteor add accounts-github
- meteor add accounts-twitter
- meteor add accounts-meetup
- meteor add accounts-meteor-developer

5.3. ARQUITECTURA DE METEOR

Meteor es una plataforma full stack cliente-servidor que posee una copia propia de la base de datos de la aplicación en cada lado (cliente y servidor) y se encarga que ambos estén sincronizados [9]. Esta arquitectura lleva usándose para el desarrollo de aplicaciones web desde el principio y sigue siendo muy utilizada hoy en día.

Para garantizar la seguridad de la aplicación, sólo el código en el servidor tiene acceso a la base de datos principal. Además, en este tipo de arquitectura de una aplicación web, el renderizado suele hacerse en el servidor, aunque las aplicaciones web modernas han llevado la parte de renderizado al cliente.

Como ya se ha comentado, en Meteor se trabaja con Node.js. Creado en 2009, este lenguaje ha supuesto muchos cambios en el desarrollo web. Grandes empresas han movido sus servicios a Node.js y pese a no ser la mejor opción para todos los casos, aporta suficientes ventajas como para ser tenido en cuenta en muchos proyectos actualmente. Desarrollar una aplicación cliente-servidor con Node.js supone que ahora la parte del servidor también va a estar programada en JavaScript, lo que supone una oportunidad de utilizar el mismo lenguaje para reducir la barrera entre ambas partes.

Meteor se planteó desde el inicio como una plataforma isomórfica, es decir, debía funcionar en cliente y servidor. Esta idea más tarde se amplió para incluir dispositivos móviles. Una vez se disponía de una API compartida entre cliente y servidor, todo bajo un mismo lenguaje, había algo que faltaba: el acceso a la base de datos.

En una aplicación web moderna no desarrollada bajo Meteor el cliente “pide” al servidor el listado de usuarios y es el servidor el que conecta a la base de datos, extrae los mismos y los envía codificados al cliente. Este sistema tiene un gran problema: el tiempo de carga. Es imposible evitar que la información consuma tiempo en viajar del cliente al servidor y realice el recorrido inverso, lo que provoca que la experiencia del usuario sea inferior. Por esta razón Meteor lleva la base de datos también al cliente y la mantiene sincronizada con el servidor, todo utilizando un protocolo extremadamente sencillo llamado DDP [10].

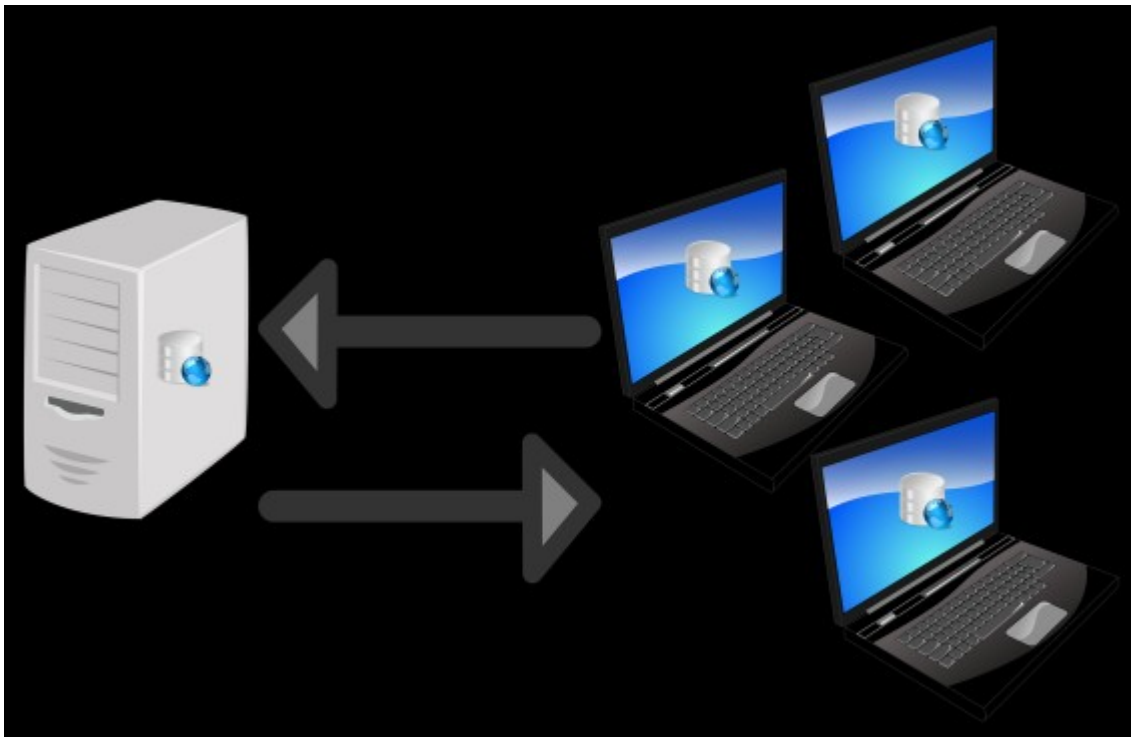

Figura 26 Arquitectura Meteor

Detrás de las decisiones en el desarrollo de Meteor, se siguen los principios mencionados a continuación:

- **Sólo transporta datos:** Las aplicaciones desarrolladas con Meteor llevan el renderizado al cliente. El protocolo que une la capa cliente con la capa servidor únicamente transporta datos, nunca transporta HTML, CSS...

- **Lenguaje único (JavaScript):** Ya que el lenguaje obligatorio en cliente es JavaScript, llevarlo al servidor es la única opción para compartir un único lenguaje. Javascript es además un lenguaje en continua evolución, con mucho presente y un futuro brillante.

- **Bases de Datos en todas partes:** Llevar la base de datos al cliente supone que el usuario deja de esperar al servidor. Todo es más rápido y los datos siempre están sincronizados. La base de datos del cliente es una versión reducida, adaptada al usuario que está conectado a ese momento.

- **Compensación de latencia:** Tener la base de datos en el cliente permite que con distintos métodos se compense la latencia del tráfico entre cliente y servidor de forma que el

usuario de la aplicación no “vea” el tiempo que se pierde. Para ello se trabaja siempre con los datos en cliente, tanto para lectura como para escritura. Todos los cambios que se hacen se producen de forma instantánea en el cliente, en lo que se denomina “simulación”, y de este modo no es necesario esperar al servidor. Cuando se hacen estos cambios “simulados” en cliente, este se encarga de avisar al servidor que a su vez verifica que esa operación en concreto esté permitida y realiza los cambios en la base de datos real. Así se elimina la latencia y se mantiene la seguridad.

- **Reactividad:** Una función reactiva es una función que vuelve a calcular su resultado cuando una de sus fuentes cambia. Muchas partes de Meteor utilizan reactividad de forma transparente y también es posible utilizarla bajo el control del desarrollador.

- **Adopción del ecosistema:** Meteor es un framework que pretende unir distintas tecnologías. El ecosistema web es enorme y se hace imprescindible aprovecharlo, por ello es posible utilizar Angular, React, GraphQL, Blaze, paquetes npm… Cada día Meteor es más flexible y sus partes más independientes.

- **Simplicidad y Productividad:** La API y todo el ecosistema que rodea a Meteor intentan ser lo más sencillos posibles. Por ello Meteor integra la sincronización en tiempo real de base y sin ninguna complicación; crear una aplicación de Meteor es tan sencillo como ejecutar un comando en el terminal.

- **Plataforma online:** los servidores de Meteor simplifican la implementación, la escalabilidad y el monitoreo de las aplicaciones conectadas con el cliente.

5.3.1. SISTEMA DE PAQUETES

Al contrario de la forma “tradicional” en la que incluir recursos externos requería de agregar enlaces a ficheros, Meteor lo hace mediante el sistema de los paquetes. Meteor puede usar cinco tipos básicos de paquetes:

- El mismo núcleo de Meteor está dividido en diferentes paquetes de la plataforma Meteor. Están incluidos en cada app y probablemente el desarrollador nunca va a tener que preocuparse por ellos.

- Los paquetes ordinarios se conocen como paquetes isomórficos (paquetes que funcionan en los dos lados, cliente y servidor). Los paquetes First-party, como *accounts-ui*, están mantenidos por los desarrolladores de Meteor.

- Los paquetes de terceros son paquetes isopacks que otros usuarios han subido al servidor de paquetes de Meteor.

- Los paquetes locales son paquetes personalizados que se pueden crear manualmente y colocarlos en el directorio */packages*.

- Los paquetes npm (Node.js Packaged Modules) son paquetes de Node.js. A pesar de que no funcionan por defecto con Meteor, pueden ser utilizados por los tipos de paquete anteriores.

5.3.2. LADO CLIENTE

Como ya se ha mencionado, la parte cliente se desarrolla con JavaScript y HTML. En el lado del cliente no hay problemas, porque los navegadores web llevan mucho tiempo con la facultad de ejecutar JavaScript. Respecto al HTML, Meteor envía desde la parte servidor datos, no HTML, y el cliente los renderiza. Ese HTML se ejecuta mediante la etiqueta *Template*, la cual sirve para crear las "páginas" de las aplicaciones.

El HTML gestiona los datos que vienen de la parte servidor mediante los *Handlebars*. Esos templates *Handlebars* aparecen como HTML regular, con expresiones *handlebars* incrustados. Una expresión *handlebars* se escribe mediante "{{", seguido del contenido, y acabando con "}}".

Meteor utiliza *Spacebars* para los templates. Esos *Spacebars* son un lenguaje de los template de Meteor, inspirado por *Handlebars*. Estos comparten algunos detalles de la sintaxis de los *Handlebars*, pero han sido adaptados para producir templates reactivos de Meteor cuando son compiladas. Cada cambio en el código es automáticamente aplicado en la aplicación real, sin tener que refrescar.

En el manejo que se hace de la aplicación real, los *Tem*plate *helpers* proporcionan datos a los templates (*Template.nombre-template.helpers()*). Esos helpers son llamados en el Template desde HTML, y las acciones sobre elementos de la parte HTML se controlan desde la parte JavaScript del cliente. Para ello, se tiene que crear una clase dentro del elemento a controlar (*class="nombre-clase"*). Posteriormente, el evento es creado en el archivo JavaScript:

```
Template.nombre-template.events({
        'click .nombre-clase': function(event){
            //actions
        }
)
```

Finalmente, recordar que en cada cliente Meteor se dispone de una “copia” de la base de datos, para hacer mas ágil la conexión con la parte servidor. Para establecer una conexión entre ambas partes cuando se gestiona manualmente algún dato, es necesario que el cliente se subscriba a la publicación que hace el lado servidor: *Meteor.subscribe('nombre-publicitado')*.

5.3.3. LADO SERVIDOR

Desde el lado servidor se gestionan los datos y permisos que se da a cada uno de los usuarios. Como ya se ha mencionado, en este lado, al igual que en el cliente se utiliza JavaScript, facilitando la comunicación entre ambos lados y con ello el tránsito de información.

El manejo de los datos se realiza en conjunto con la base de datos MongoDB. Aunque en la parte cliente se dispone de “copias” de la base de datos del servidor, la principal es la que se encuentra en el servidor, y todo cambio o consulta que se hace en la de los clientes viene seguida de la consulta sobre la principal.

Desde la parte servidor de Meteor se pueden crear nuevas colecciones (se llaman así en MongoDB a las tablas de las bases de datos relacionales). Esto se realiza mediante el comando: *Variable = new Mongo.Collection("nombre-coleccion")*. Esta colección queda registrada en la base de datos principal (y propagada a los clientes).

Para poder realizar consultas, modificaciones,... sobre dicha colección recién creada, es necesario que desde el servidor se le de los permisos apropiados a los clientes. Controlar la seguridad es imprescindible para un correcto funcionamiento de las aplicaciones, y con ello un correcto uso de la base de datos. Por ello, los permisos son revocados en un primer momento a todo usuario, y hay que indicar explícitamente qué acción puede hacer sobre qué colección qué determinado usuario:

```
Variable.allow({
   //update,remove,insert
})
```

Desde el servidor también se pueden inicializar determinados datos, para que estos puedan ser introducidos en la base de datos (no se necesitan permisos para ello, ya que es el propio servidor el que lo mete): *Variable.insert({})*.

Además de las nuevas colecciones que se pueden introducir, anteriormente se ha hablado de la opción que da Meteor de gestionar los usuarios mediante los paquetes que se quieran introducir. Esa gestión, para la base de datos, queda registrada en una colección llamada “Accounts”, la cual tendrá de inicio algunos atributos ya mencionados en el apartado correspondiente. Para añadir nuevos atributos a los usuarios, es necesario que se indique en el servidor, y comunicarle ese cambio a la parte cliente. En el servidor, se pueden introducir esos atributos al crear el usuario, incluso darle un valor concreto a los mismos, para almacenarlo en la base de datos. El inicio de esos atributos se hace con el siguiente comando:

```
Accounts.onCreateUser(function(options, user) {
    user.atributo = '';
    return user;
});
```

Finalmente, como se ha mencionado, para conectar el lado cliente con el lado servidor, este último tiene que publicitar para que el cliente pueda acceder a determinada información del mismo. Esto se realiza mediante la expresión: *Meteor.publish('nombre-publicitado', function() {})*.

5.3.4. BASE DE DATOS

La base de datos utilizada en Meteor es MongoDB (que proviene de «humongous») la cual es la base de datos NoSQL mas usada actualmente dentro del desarrollo web. Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales.

A diferencia de esas bases de datos tradicionales, MongoDB no necesita seguir un esquema; almacena los datos en colecciones. MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Es algo bastante lógico, dado que los propios documentos se almacenan en BSON, utilizando Strings, Arrays, Números,...

MongoDB viene de serie con una consola desde la que se pueden ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones propias de JavaScript, se pueden utilizar muchas de las funciones de MongoDB. En la consola también se pueden definir variables, funciones o utilizar bucles [13].

En el servidor, la colección tiene la tarea de hablar con la base de datos MongoDB y leer y escribir cualquier cambio. En este sentido, se puede comparar con una librería de base de datos estándar. En el cliente sin embargo, la colección es una copia de un subconjunto de la colección canónica. La colección del lado del cliente se mantiene actualizada, de forma constante y (normalmente) trasparente con ese subconjunto de datos en tiempo real. Los documentos se almacenan en la memoria del navegador, lo que significa que el acceso a ellos es prácticamente instantáneo.

La parte más importante de todo esto es cómo se sincronizan los datos de la colección del cliente con la colección del mismo nombre del servidor. Los datos viajan hasta la base de datos sin escribir una sola línea de código para enlazar cliente con el servidor (bueno, en sentido estricto, hay que escribir una línea de código: *new Mongo.Collection("nombre-colection")*). Lo que pasa es que la colección del cliente informa de una modificación a la colección del servidor, que inmediatamente se pone a distribuirlo en la base de datos Mongo y a todos los clientes conectados a la colección afectada.

Meteor tiene habilitado por defecto el paquete autopublish, algo que no es conveniente para aplicaciones en producción. Este paquete indica que las colecciones son compartidas en su totalidad con cada cliente conectado. Para poder darle acceso a los usuarios de forma que tengan referencia a lo que el cliente necesita, se utiliza la función ya mencionada *publish()*, y el cliente subscribirse a dicha publicación con el método *Meteor.subscribe()*.

5.3.5. ENRUTAMIENTO

Mediante el enrutamiento se permite controlar qué partes de la aplicación aparecen en determinados momentos, haciendo una recarga dinámica de los ficheros HTML.

Para trabajar con este aspecto, Meteor proporciona un paquete denominado *iron:router*. En el archivo HTML se crean los template para cada página, proporcionándoles un nombre, y en el JavaScript se controla el enrutamiento entre dichas páginas.

```
Router.route('/referencia-url', function () {
    this.render('nombre-template');
});
```

Se pueden enviar algunas opciones de configuración en el sistema de enrutamiento, creando en el HTML un template dedicado a ello. Por ejemplo:

```
Router.configure({
    layoutTemplate: 'ApplicationLayout'
});

<template name="ApplicationLayout">
 {{> yield "navbar"}}
 {{> yield "main"}}
</template>
```

Yield es una instrucción para el *router* indicándole que hay un punto en el template *layout* que puede renderizar sub-templates.

```
Router.route('/referencia-url', function () {
    this.render('navbar', {
        to:"navbar"
   });
   this.render('nombre-template', {
       to:"main"
   });
});
```

5.4. ENTORNO DE DESARROLLO

El entorno de desarrollo en Meteor es muy amplio y sencillo de instalar; con sólo un comando de una línea se instala una base de datos, un servidor de desarrollo, y un programa de línea de comandos, que se encarga de la creación y el despliegue de las aplicaciones. Con todo ello, las aplicaciones se desarrollan rápidamente.

5.4.1. LOCAL

Las aplicaciones de Meteor, antes de ser accesibles para todo el mundo, se pueden desarrollar localmente. Eso indica que sólo se podrá ejecutar la aplicación en el equipos en el que se está desarrollando el proyecto.

Para lograrlo, lo primero que hay que hacer es instalar Meteor en el equipo donde desarrollar la aplicación. La instalación es muy sencilla, utilizando la terminal (y en este caso, un sistema Linux). Primero se introduce el siguiente comando: *curl https://install.meteor.com/ | sh*. A continuación, se crea la aplicación con el nombre deseado, introduciendo en la terminal: *meteor create nombre-aplicación*. Esto crea una carpeta con el nombre que se le ha dado a la aplicación, la cual contiene tres ficheros: HTML, CSS y JavaScript. Son los ficheros base sobre los que se creará la aplicación. Para iniciar la aplicación se introduce el siguiente comando dentro de la carpeta de la aplicación: *meteor*.

Una vez inicializada la aplicación, desde el propio terminal se indica a que dirección se tiene que acceder desde el navegador; esa dirección será *localhost:3000*. Como bien indica la URL, se ejecuta localmente, pero puede ser la mejor opción si se están haciendo modificaciones sobre la aplicación, para ir comprobando el estado de la misma.

Desde el terminal se pueden ejecutar muchos comandos propios de Meteor. Para trabajar sobre el proyecto, hay que ponerse dentro de la carpeta de dicho proyecto e introducir el comando deseado. Ya se ha comentado como introducir paquetes en una aplicación, introduciendo el comando *meteor add nombre-paquete*, y se pueden consultar todos los paquetes introducidos en el proyecto con el comando *meteor search* .. Otro comando muy utilizado puede

ser el de resetear la aplicación, inicializando de nuevo toda la base de datos. Esta acción se realiza con el comando: *meteor reset*. Para poder realizar cualquier acción, se pueden consultar otros comandos que da Meteor.

5.4.2. SERVIDOR

Para que la aplicación esté accesible a todo el mundo, la misma hay que dejarla en un servidor. Las aplicaciones de Meteor, a pesar de ser una tecnología con grandes expectativas de futuro, no están aún aceptadas en algunos servidores.

Primero hay que destacar que Meteor proporciona servidores propios para poder alojar las aplicaciones. Aunque los mismos al principio eran gratuitos, durante el desarrollo del proyecto han pasado a ser de pago. Las opciones que proporciona Meteor son:

- La forma mas sencilla de dejar operativa la app es mediante lo denominado **Galaxy**, un servicio creado por Meteor Development Group especializado en ejecutar aplicaciones Meteor. Galaxy es un sistema distribuido que se ejecuta en Amazon AWS. Dada su forma de arrancar las aplicaciones y su manera de trabajar, Galaxy ayuda a ahorrar mucho tiempo y problemas. La mayoría de aplicaciones de Meteor se alojan mediante Galaxy hoy en día, y muchas de ellas han cambiado de soluciones personalizadas que utilizaban antes del lanzamiento del Galaxy a esta opción. Para utilizar esta opción se necesita registrar una cuenta aquí, y además proporcionar una base de datos MongoDB externa. Hecho esto, se añaden algunas variables de entorno al archivo de configuración para apuntar a MongoDB, y ya se puede subir la aplicación con: *DEPLOY_HOSTNAME=galaxy.meteor.com meteor deploy your-app.com --settings production-settings.json*. Si por algún caso se quiere trabajar con un dominio propio, se necesita preparar el DNS para apuntar a Galaxy

- **Meteor Up**, o mup, es una herramienta “open source” que puede ser utilizada para suministrar aplicaciones Meteor a cualquier servidor online mediante SSH. Mup se ocupa de algunos de los requisitos esenciales de “deployment”. Se puede obtener un servidor que trabaje en Ubuntu o Debian de diferentes proveedores genéricos de hosting. Mup puede

acceder mediante SSH al servidor con las claves que proporciona el proveedor.

- Si se quiere buscar una solución creándolo desde cero, la herramienta de Meteor tiene un comando *meteor build* que crea un paquete de instalación que contiene una aplicación de Node.js. Cualquier dependencia de npm (descrita anteriormente) tiene que ser instalada antes de introducir el comando mencionado, para poder ser incluida en el paquete. Se puede alojar la aplicación en donde se desee.

Aún así, existen servidores en la red que también pueden ser interesantes de utilizar, de cara a tener un dominio propio. Como se ha comentado, la mayoría de servidores no soportan Meteor como framework, incluso algunos que utilizan Node.js tampoco lo aceptan. Uno que sí lo soporta, y no es de pago, es Scalingo, la opción escogida para este proyecto. La cuenta creada para este proyecto es gratuita y tiene una fecha de caducidad.

6

IMPLEMENTACIÓN

Una vez realizado el estudio de la tecnología, se pasa a realizar la implementación de la aplicación “SocialMusFest”, a través de la cual se podrán aplicar los conocimientos adquiridos de Meteor. Además, se quiere apoyar una necesidad de mercado que aún no se ha trabajado en ninguna aplicación; una red social de festivales de música.

6.1. LADO CLIENTE

Para empezar este análisis, se observa uno de los lados que forman parte de este tipo de aplicaciones; el lado cliente. Esto indica que la ejecución de los programas o scripts mencionados en esta sección se realiza en el navegador del usuario.

La implementación de este apartado se puede dividir en dos secciones principales; la parte de interfaz de usuarios (HTML), a través de la cual se organiza la forma de visualizar la información, y la parte de gestión de datos (JavaScript), dentro de la cual se pueden diferenciar los subapartados de enrutamiento, helpers y eventos. Todas estas partes se explican a continuación, con ejemplos de los mismos.

6.1.1. PLANTILLA

Para este proyecto, en primer lugar se ha dibujado un esquema de lo que se quiere mostrar en la aplicación web. Logrado el esquema con todos los elementos y sus interacciones, en vez de crear un diseño propio, se decide buscar uno de los tantos que hay en Internet que mejor se adecue a las exigencias planteadas.

En cuanto a plantillas para aplicaciones web, un lugar muy utilizado y con gran variedad de diseños es Bootstrap. Estas plantillas no son adaptadas a Meteor; es decir, no siguen una estructura de *Templates* que sigue este framework. Por lo tanto, una vez escogido el diseño que más se adecua a las exigencias impuestas en el voceto inicial, se tiene que adaptar la misma a las características particulares de Meteor.

Como la plantilla escogida contiene muchos detalles nada relevantes para la aplicación SocialMusFest, lo que sobra hay que eliminarlo también. En cuanto al CSS, se decide que los colores, distribución... que viene en la propia plantilla son los adecuados de momento, por lo que no se toca nada en cuanto a ello. Sí que se elimina la parte de “Footer”, ya que para un proyecto de estas características no se puede poner mucha información relevante en la misma.

Con todo esto decidido, en las siguientes figuras (27-28-29-30-31-32-33-34-35) se muestra como ha quedado el diseño final del sitio web:

Figura 27 Página Principal

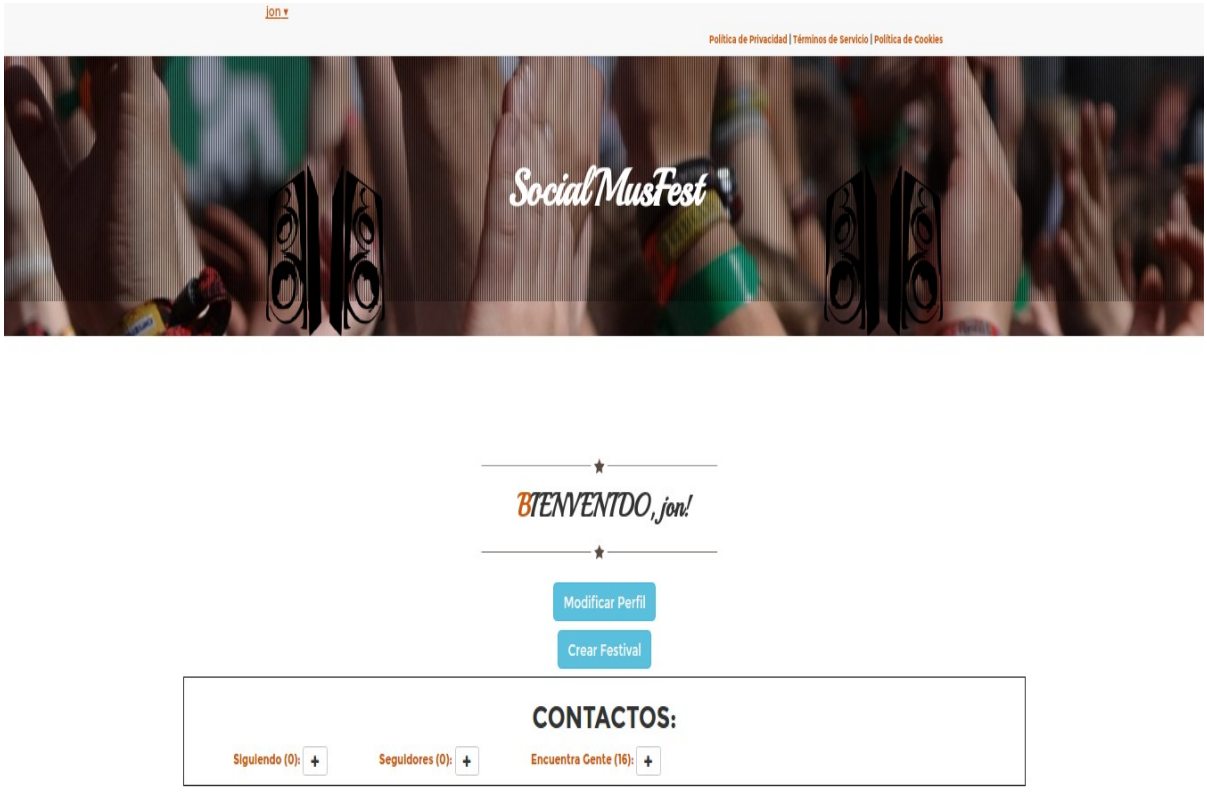

Figura 28 Página Usuario 1

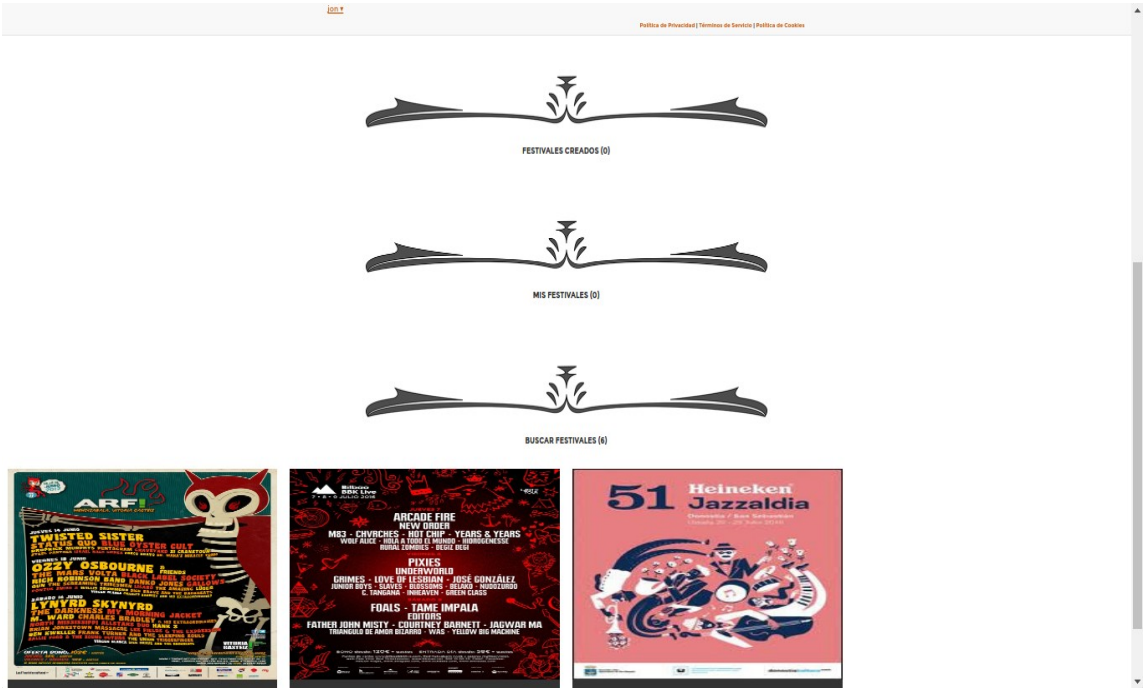

Figura 29 Página Usuario 2

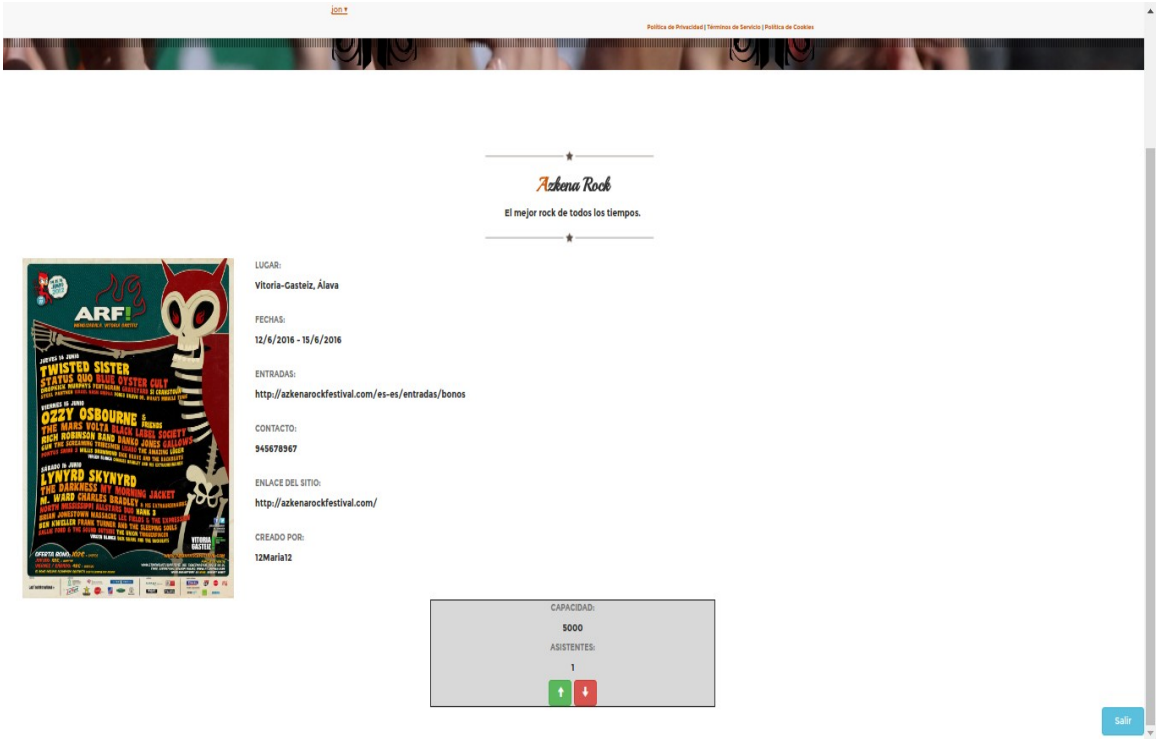

Figura 30 Página Festival

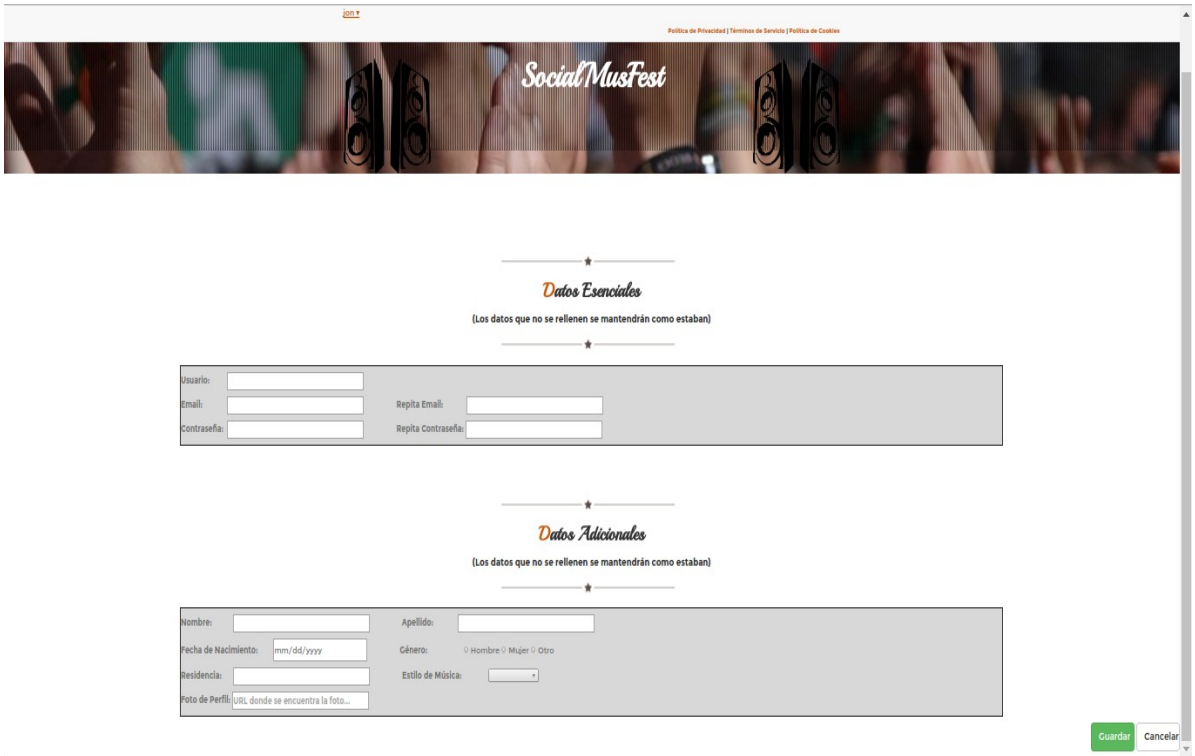

Figura 31 Página Modificar Perfil

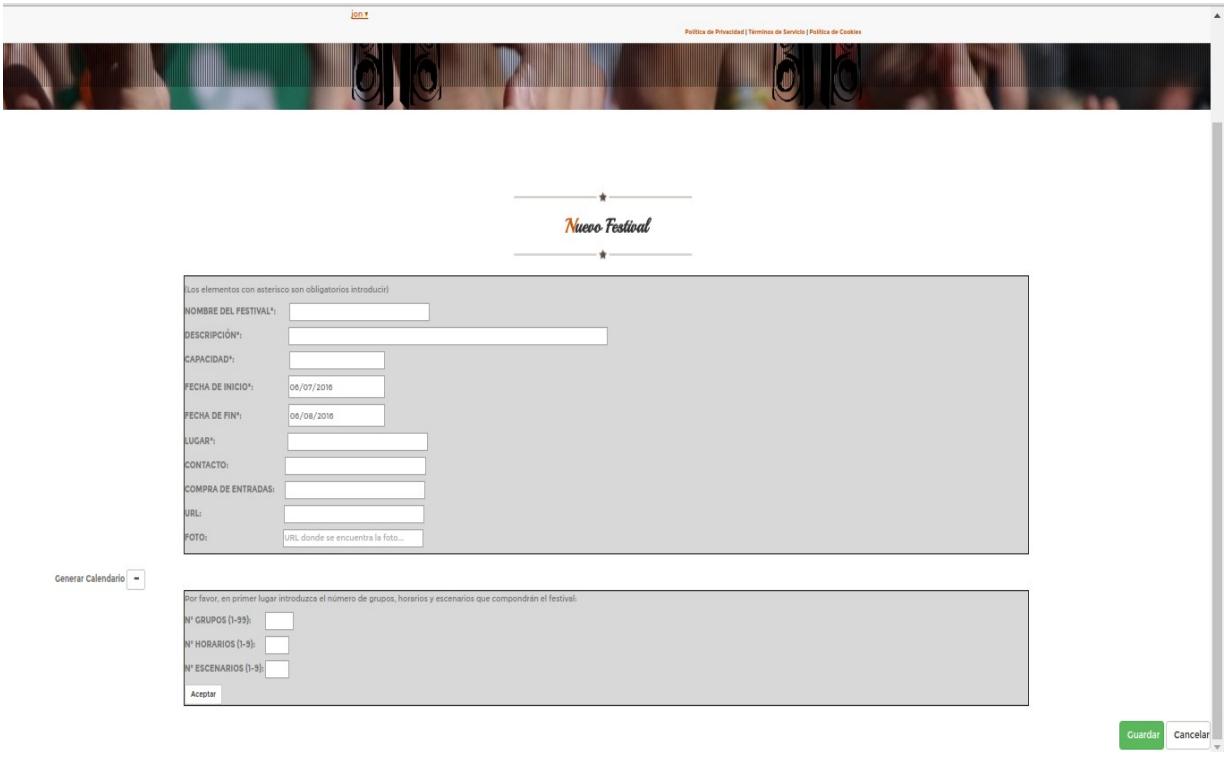

Figura 32 Página Crear Festival

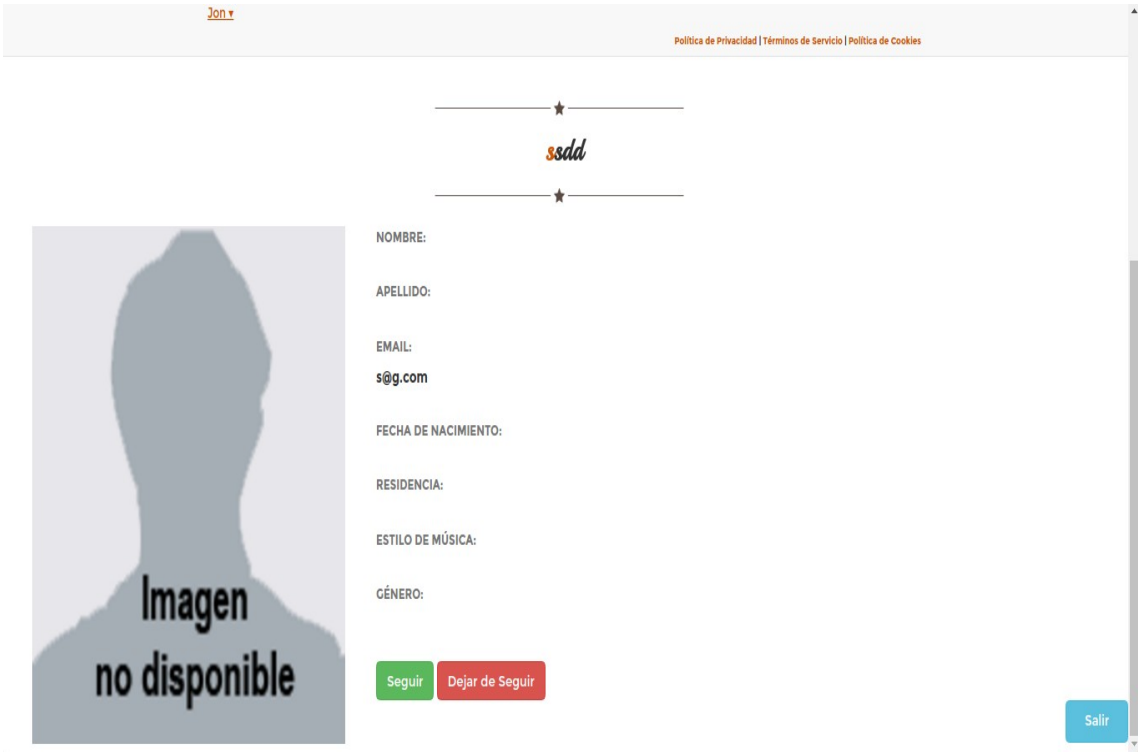

Figura 33 Página Usuario

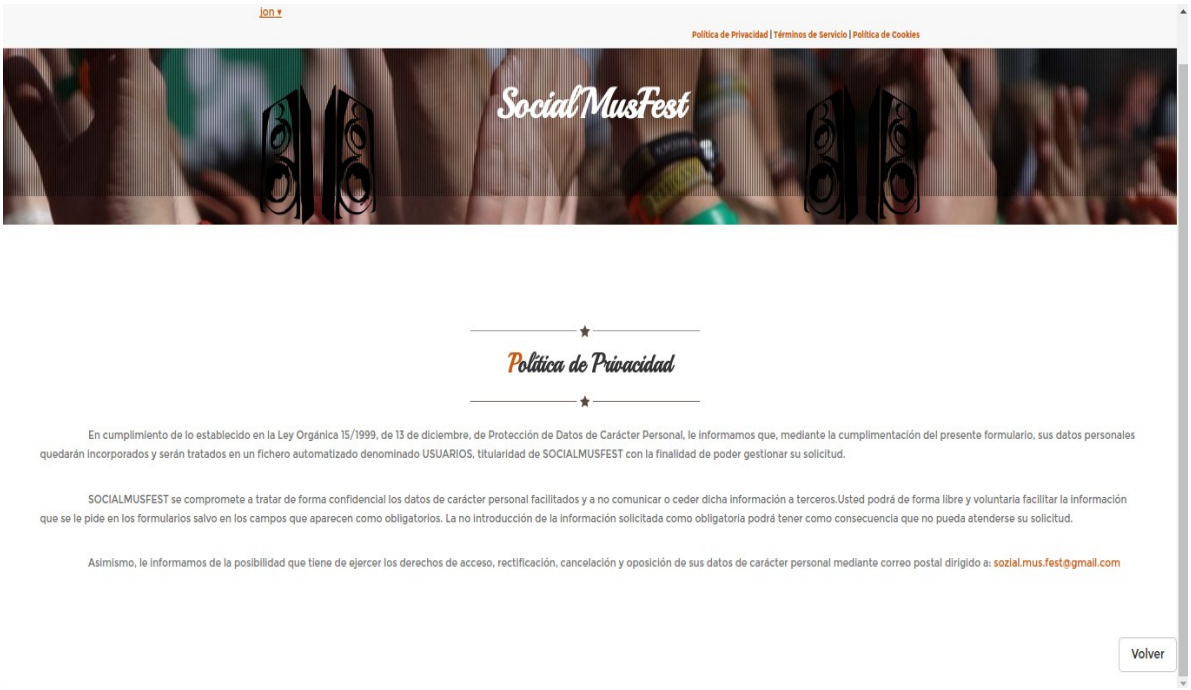

Figura 34 Página Política de Privacidad

Figura 35 Página Términos de Servicio

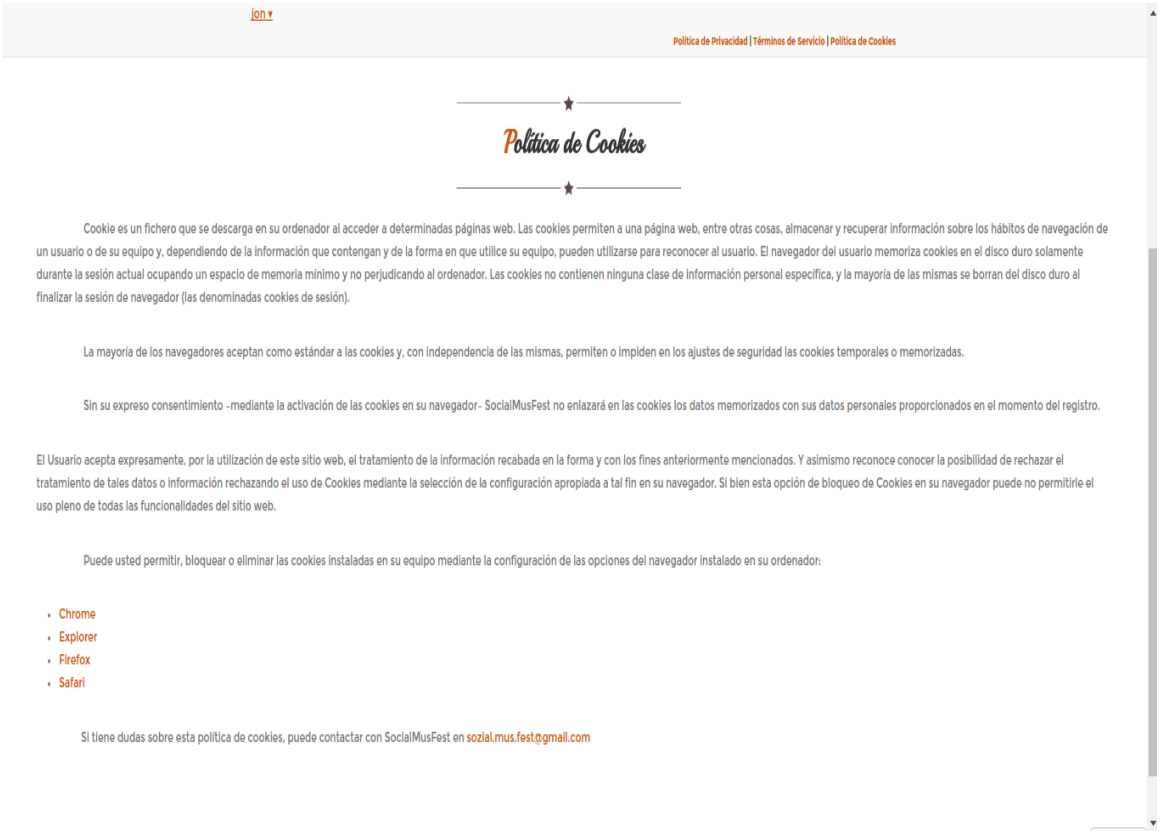

Figura 36 Página Política de Cookies

6.1.2. INTERFAZ DE USUARIO

En esta parte se especifica la forma de mostrar los datos. La estructura viene definida por una plantilla escogida en Bootstrap, la cual se ha adaptado para poder personalizar al gusto del desarrollador dicha página.

A diferencia de los ficheros HTML vistos hasta la fecha, no se implementa el contenido de los elementos dentro de las etiquetas *<body></body>* de HTML. En Meteor, como se ha mencionado, cada una de las páginas que constituyen la aplicación se introduce dentro de las etiquetas *<template></template>*.

A cada uno de esos *template* se le tiene que dar un nombre para posteriormente identificar a las diferentes páginas. Dentro de esas etiquetas, la creación de los elementos HTML es igual que en cualquier otra aplicación que use esta tecnología, siempre respetando la plantilla escogida.

Otra diferencia con los documentos HTML clásicos es la gestión de los datos que se hace. Es una gestión dinámica; si hay algún cambio en los datos manejados de la aplicación, esta quedará registrada en la interfaz, mostrando lo datos actualizados a los usuarios de la aplicación. Como se ha comentado anteriormente, esta gestión se hace mediante los *Handlebars* y los *Spacebars*. Estos elementos están en constante conexión con la gestión de datos, haciendo que cualquier actualización se refleje sin demora de tiempo.

```
            <!------------------------ FESTIVAL PAGE ------------------------->
<template name="festival">

    <div class="wrapper" id="wrapper">

        {{> header}}

        <div class="heading text-center">
            <img class="dividerline" src="img/sep.png" alt="">
            <h2>{{name}}</h2>
            <h4>{{description}}</h4>
            <img class="dividerline" src="img/sep.png" alt="">
        </div>

        <img src="{{photo}}" alt="" width="500" height="600" align="left" hspace="50"/>
        <strong>LUGAR: </strong> <h4>{{place}}</h4>
        <br />
        <strong>FECHAS: </strong> <h4>{{firstDay}} - {{lastDay}}</h4>
        <br />
        <strong>ENTRADAS: </strong> <h4>{{sales}}</h4>
        <br />
        <strong>CONTACTO: </strong> <h4>{{contact_number}}</h4>
        <br />
        <strong>ENLACE DEL SITIO: </strong> <h4>{{webpage}}</h4>
        <br />
        <strong>CREADO POR: </strong> <h4>{{creator}}</h4>
        <br />
```

Figura 37 Código de página de información de “Festival”

En el código anterior se observa otro detalle importante; hay una llamada a otro template del fichero en el que se encuentra este.

{{> header}}

Ese comando introduce en el punto mencionado el código del template con nombre *header*.

Además de lo mostrado en el código anterior, se pueden hacer consultas a la parte de gestión. Esas consultas se abordan en el siguiente punto.

6.1.3. ENRUTAMIENTO

A continuación se analiza el enrutamiento, un concepto analizado previamente y muy necesario para que realmente el usuario pueda navegar entre las páginas.

En el punto anterior se ha mencionado que las etiquetas *template* hacen mención a cada página de la aplicación, por lo que con el nombre que se les asigna se puede acceder a dicha página.

El enrutamiento se realiza mediante el paquete *iron:router*. Desde la terminal, dentro del fichero del proyecto, se añade el paquete mencionado con el comando *meteor add iron:router*. Esto ya da permiso a usar el paquete de enrutamiento, y con sólo hacerle una llamada a *Router* se accede a sus funciones y características.

```
/// route to access to an specific page of a user profile
Router.route('/perfil/:username', function () {
    this.render('infoPerfil', {
        to:"main",
        data:function(){
            return Meteor.users.findOne({username:this.params.username});
        }
    });
});
```

Figura 38 Código enrutamiento

En la figura 27 se puede observar que la opción para poder enrutar es con *Router*, y en concreto con su función *route()*. Esta función consta de dos variables; la primera es la ruta en concreto a la que se quiere llegar (en este caso */perfil/:username*, donde *:username* hace referencia a una variable dinámica que cambiará de valor en base al usuario que esté registrado). La segunda es la función para gestionar dicha URL; en concreto, se indica a qué *template* se va a hacer referencia. Además, dentro de esa redirección, se pueden obtener datos creando funciones; en el caso de la figura 27, se obtiene un usuario a través de un Username.

Para cada una de las rutas a utilizar en la aplicación se tiene que hacer esta tarea. Si una URL no está definida en este grupo, la misma no será funcional en la aplicación, imposibilitando su uso.

6.1.4. HELPERS

Los *Helpers* son funciones JavaScript utilizadas por los *templates* y permiten ejecutar código desde dentro de la interfaz. La estructura de estos *helpers* son muy sencillas, y hacen referencia a los *templates* ya creados. Por lo tanto, si un *template* tiene que utilizar una función para hacer alguna consulta, esta primero se creará en la parte JavaScript:

```
/// helper function that returns the people the current user is following
Template.following_list.helpers({
    user:function(){
        var users = Meteor.user().following;
        var user = new Array();
        var cont = 0;
        for(var i=0; i<users.length; i++){
            if(Meteor.users.findOne({username:users[i]})){
                user[cont] = Meteor.users.findOne({username:users[i]});
                cont++;
            }
        }
        return user;
    }
});
```

Figura 39 Código Helpers JavaScript

En el ejemplo de la figura 28, se observa que se hace referencia al *Template* llamado f*ollowing_list*. Dentro del mismo sólo hay una función, denominada *user*, a la cual se le llamará desde el template f*ollowing_list*. Esa llamada se hace mediante el siguiente código:

```
<!-- template that displays only the users that the current user is following -->
<template name="following_list">
    {{#each user}}
        <li>
            <a href="/perfil/{{username}}">{{username}}</a>
        </li>
    {{/each}}
</template>
```

Figura 40 Código Helpers HTML

En este caso, se realiza una consulta con el que se va recorriendo (*each*) todos los user; esos user ya se han explicado como se obtienen en la parte JavaScript. Por cada uno de los mismos, se muestra el atributo username del usuario en cuestión. Así pues, se ha utilizado el *Helper* para poder obtener una lista de todos los usuarios a los que sigue el usuario conectado, para poder ir obteniendo cada uno de sus username.

Al igual que *each*, existen sentencias de programación aplicables a la parte HTML en sus *templates*. Esas sentencias ayudan a mostrar determinada información, o dar acceso a algunos usuarios a determinados recursos.

- **IF:** esta sentencia es muy útil para dar acceso a determinados usuarios, y a determinados recursos en determinados momentos: *{{#if display}}*. Además, en este proyecto como se tienen usuarios registrados y no registrados, para que los no registrados no puedan acceder a recursos reservados a los registrados se utiliza la sentencia *{{#if currentUser}}*.

- **EACH:** esta sentencia sirve para recorrer listas, en las que se accederá a cada uno de los elementos individualmente. La sentencia es la siguiente: *{{#each user}}*.

En este proyecto se han utilizado esas dos sentencias en los templates, aunque la comunidad de Meteor está en constante movimiento y puede que aparezcan nuevas.

6.1.5. EVENTS

Los e*vents* se encargan de controlar eventos que se realizan en la parte de HTML. Esos eventos pueden ser de cualquier tipo que exista en el desarrollo web (click, submit,...), y gestionan el modo en que se comportan en la aplicación. En este proyecto, los eventos utilizados han sido “click” y “submit”, los cuales han sido utilizados para gestionar los eventos de los botones, realizando las acciones que le corresponden a los mismos.

La estructura de estos eventos es muy parecida a la mencionada de los *helpers*; aquí también se crean eventos por cada uno de los *templates* creados en la parte HTML. Para cada uno de ellos hay que hacer una llamada a dicho t*emplate* indicando que es un evento. Para cada una de esos *templates*, dentro se pueden indicar los eventos que se quiera. Por lo tanto, como en este proyecto se tienen mas de un botón en cada página, no se tiene ningún problema en controlar cada uno de esos botones, indicando el nombre asignado al botón.

```
Template.creator.events({
    "click .js-create-calendar":function(event){

        // verify that some data is introduced
        if(diaInicio.value=="" || diaFin.value==""){
            $("#notDates").show('slow');
            return false;
        }
        else{
            $("#notDates").hide('slow');
            $("#calendar_create").show('slow');
            $("#create").hide();
            $("#notcreate").show();
        }
    },
    "click .js-notcreate-calendar":function(event){
        $("#calendar_create").hide('slow');
        $("#calendar_create_2").hide('slow');
        $("#create").show();
        $("#notcreate").hide();
        $("#propuesta").empty();
        $("#advise").hide();
```

Figura 41 Código Events JavaScript

En el código de la figura 41 se ve como se crea un grupo de eventos para el *template* llamado *creator*. Ese *template*, al igual que en el resto, se gestionan botones; es decir, se llama al evento *click* del botón llamado *js-create-calendar* en primer lugar, y posteriormente a *js.notcreate.calendar*. Esas son funciones que gestionan recursos, movimientos,... de la página. Esos recursos, como se ve, tienen que tener asignado un nombre.

```
<strong>Generar Calendario </strong>
<a id="create" class="btn btn-default js-create-calendar">
    <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>
</a>
<a id="notcreate" class="btn btn-default js-notcreate-calendar" style="display:none">
    <span class="glyphicon glyphicon-minus" aria-hidden="true"></span>
</a>
```

Figura 42 Código Events HTML

En la figura 42 se observan los dos recursos con la llamada a eventos en el atributo *class*. Desde JavaScript, como se ha explicado, se gestionan esos eventos.

6.2. LADO SERVIDOR

Analizada la parte del cliente o frontend del sistema, en este apartado se describen los aspectos más relevantes de la implementación del lado servidor o backend. Esta parte es fundamental para garantizar la integridad de todos los datos, salvaguardándolos de posibles acciones por parte de los clientes que puedan dañar la integridad del sitio web.

En esta proyecto, la parte de servidor se encarga de dar permisos y gestionar la base de datos, para que no quede vulnerable. Sin la correcta utilización de estos recursos, la aplicación quedaría vulnerable a posibles ataques.

6.2.1. COMUNICACIÓN CON LADO CLIENTE

En una aplicación web es esencial que la parte cliente y la parte servidor se comuniquen correctamente. Esa comunicación servirá para poder gestionar la parte cliente.

Para empezar, lo que deben hacer es compartir información entre ambas partes. Como se ha indicado anteriormente, las colecciones son creadas en la parte servidor, y estas envían los datos a las base de datos. La parte cliente también tiene “copias” de la base de datos, y esas copias se gestionan internamente por Meteor. Aún así, se pueden gestionar datos fuera de las colecciones que también necesitan ser accesibles por los clientes.

En este proyecto, esos datos fuera de las colecciones son los nuevos atributos para los usuarios de la aplicación. La gestión de los usuarios se hace internamente, pero para poder tener usuarios personalizados a las exigencias del sitio se añaden nuevos atributos. La forma de introducir esos datos se explica a continuación, pero mencionar que para que el cliente pueda modificar los atributos fuera de la colección se utiliza el método *publish()* para que los clientes puedan acceder a esos datos.

```
Meteor.publish('userData', function() {
  if(!this.userId) return null;
  return Meteor.users.find();
});
```

Figura 43 Código Publish Servidor

Los datos a introducir (en este caso los nuevos atributos de los usuarios) se almacenan dentro de la variable “userData”, la cual se publicita para ser accedida por parte de los clientes. Esos clientes tienen que suscribirse a la publicación mencionada ahora mismo:

```
/// access to read the publish things for new attributes on users
Deps.autorun(function(){
    Meteor.subscribe('userData');
});
```

Figura 44 Código Subscribe Cliente

En la figura 44 se observa como el cliente se ha subscrito a la publicación llamada “userData” de la parte servidor. Esto se hace nada más arrancar el cliente, mediante una variable llamada “Deps” que gestiona el método “autorun()”.

6.2.2. PERMISOS

En cualquier aplicación web con más de un tipo de usuario es esencial restringir el acceso a cada grupo de usuarios. Los usuarios con diferentes privilegios tendrán acceso a diferentes acciones de la base de datos.

De forma determinada, Meteor no da acceso a las acciones sobre la base de datos; sólo se puede acceder a consultar los datos. Tanto para modificar, añadir o eliminar información, se necesita que desde la parte servidor se de acceso a las mismas.

```
// set up security on festivals collection
Festivals.allow({

    // to be able to update festivals for ratings.
    update:function(userId, doc){
        console.log("testing security on festival update");
        if (Meteor.user()){// they are logged in
            return true;
        } else {// user not logged in - do not let them update (rate) the festival.
            return false;
        }
    },

    insert:function(userId, doc){
        console.log("testing security on festival insert");
        if (Meteor.user()){// they are logged in
            return true;
        }
        else {// user not logged in
            return false;
        }
    },

    remove:function(userId, doc){
        console.log("testing security on festival insert");
        if (Meteor.user()){// they are logged in
            return true;
        }
        else {// user not logged in
            return false;
```

Figura 45 Código Permisos MongoDB

Como se observa en la figura 45, los permisos se dan sobre cada una de las colecciones de la parte cliente. En este caso, esa colección se almacena en la variable “Festivals”. Y por cada uno de los permisos que se otorgan mediante el método *allow()* se indica qué usuarios tienen acceso a la función mencionada; en este caso, como los usuarios que tendrán acceso a dichas acciones sólo serán los que están registrados, mediante el método *Meteor.user()* se indica que si el usuario está conectado (es un método del paquete de gestión de usuarios de Meteor) tendrá acceso a la acción mencionada (return true), el resto de usuarios no tendrán acceso porque no están registrados (return false).

6.3. BASE DE DATOS

Hasta el momento ya se ha explicado todo lo relacionado con la base de datos MongoDB utilizada en este proyecto. Como ya se ha indicado, se ha utilizado el paquete de gestión de usuarios de Meteor para controlarlos. Además, se ha creado una nueva colección llamada “festivals” que almacena la información de dichos festivales.

Recordar que este tipo de base de datos hace muy dinámico el uso de los atributos, por lo que si se quiere se pueden ir modificando los datos añadiendo nuevos atributos si se considera necesario, una vez creada la base de datos y las colecciones. Así mismo, salvo petición expresa por parte de la aplicación de los datos, todos pueden ser vacíos aunque, para evitar esto, se han acordado con Rachid Boudhar una serie de datos requeridos.

6.3.1. USUARIOS

El paquete de gestión de usuarios ayuda en la creación y acceso de usuarios, pero al ser un paquete genérico, este es muy limitado. Entre las opciones que da, en este proyecto se incluye la siguiente configuración:

```
/// accounts configuration
Accounts.ui.config({
    passwordSignupFields: "USERNAME_AND_EMAIL"
});
```

Figura 46 Código configuración usuarios

En los usuarios, se decide usar la opción de utilizar los atributos username, email y password por decisión conjunta con Rachid Boudhar, ya que esta configuración es la más adecuada para poder mostrar información de los usuarios cuando se solicite. Aún así, estos atributos no son suficientes, ya que la información que se quiere mostrar tiene que ser mayor.

```
{{> loginButtons}}
```

Figura 47 Código botones registro

Con el código de la figura 47 se añaden los botones de registro de los usuarios. Esos botones se acceden mediante los paquetes que se han instalado de Meteor, que en este caso son tres; *accounts-ui* para los registros “normales”, *accounts-facebook* para el registro mediante Facebook, y *accounts-google* para el registro mediante Google. Estos modos de registro se han acordado con Rachid Boudhar, ya que se entienden son los más adecuados y populares para una aplicación de este estilo.

Ante este problema se plantean dos posibles soluciones; por un lado, se puede tener en cuenta la opción de dejar de lado el paquete de Meteor y crear los usuarios manualmente, pero como este proyecto quiere ser de aprendizaje, se desecha.

La otra opción es la de añadir los atributos que no están a los ya existentes. Esta vía hace que los atributos tengan que ser accedidos por el cliente al iniciarse, suscribiéndose a la publicación donde se guardan los datos. Como esta vía sí que permite la utilización del paquete de Meteor, se opta por utilizarla.

```
// add new attributes to users
Accounts.onCreateUser(function(options, user) {
    user.name = '';
    user.surname = '';
    user.birthday = '';
    user.gender = '';
    user.place = '';
    user.music_style = '';
    user.image = '';
    user.festivals_assisted = [];
    user.festivals_created = [];
    user.followers = [];
    user.following = [];
    user.facebook_id = '';
    user.google_id = '';
    return user;
});
```

Figura 48 Código añadir atributos a usuarios

Los atributos añadidos son inicializados (en este caso, quedan vacíos). La forma de trabajar es muy sencilla, al crear un usuario se rellenan los atributos del paquete (username, email y password), y además se crean los "campos" que aparecen en la figura 48 (mediante el método *onCreateUser())*. Esos campos están vacíos, pero a la hora de utilizar la opción de "Editar Perfil" se pueden modificar para que puedan ser actualizados.

Aunque los usuarios tengan ya los atributos asignados, es necesario dar permisos a los usuarios para que puedan modificarlos, ya que como se ha mencionado no tienen acceso a ello por defecto.

La colección, por lo tanto, queda de la siguiente forma:

Tabla Users

DATOS BÁSICOS:

- _id = (PK) (text)
- emails (PK) (text)
- username (nick, PK) (text)
- password (password)

DATOS ADICIONALES:

- name (text)
- surname (text)
- birthday (date)
- gender (text)
- place (text)
- music_style (text)
- image (text)

- festivals_assisted = [(sólo nombres de festivales)];
- festivals_created = [(sólo nombres de festivales)];

- followers = [(sólo username)];
- following = [(sólo username)];

Figura 49 Tabla de Usuarios

El atributo "_id" es un campo que Meteor lo asigna automáticamente para poder gestionar por su cuenta los usuarios. Ese valor, por lo tanto, no es gestionada por los desarrolladores. Además, el atributo del email lo almacena como un array, aunque para esta aplicación sólo se utiliza un correo por usuario.

6.3.2. FESTIVALES

La colección de los festivales, a diferencia de la de los usuarios, tiene que ser creada manualmente por parte de los desarrolladores. La creación es muy sencilla y simple.

En primer lugar, como se muestra en la figura 50, se declara una colección en la parte servidor mediante la función de MongoDB *Collection()*, a la cual se le asigna el nombre que se le quiere dar a la colección. Esa creación queda registrada en una variable llamada “Festivals”.

```
Festivals = new Mongo.Collection("festivals");
```

Figura 50 Código declaración de colección

Creada la colección, y almacenada en una variable, la gestión de los datos es algo que puede variar mucho; no hay una serie de atributos predefinidos, sino que a medida que se vayan haciendo acciones sobre la aplicación se pueden añadir nuevos, obviar otros... Esto tiene un peligro para los desarrolladores, que si al escribir código se confunden en el nombre de los atributos, MongoDB crea un atributo nuevo dejando el que se quería modificar tal y como estaba.

Para tener una ordenación en el código, es necesario concretar los parámetros a utilizar. Eso se ha hecho en esta aplicación, acordando con Rachid Boudhar los parámetros a utilizar.

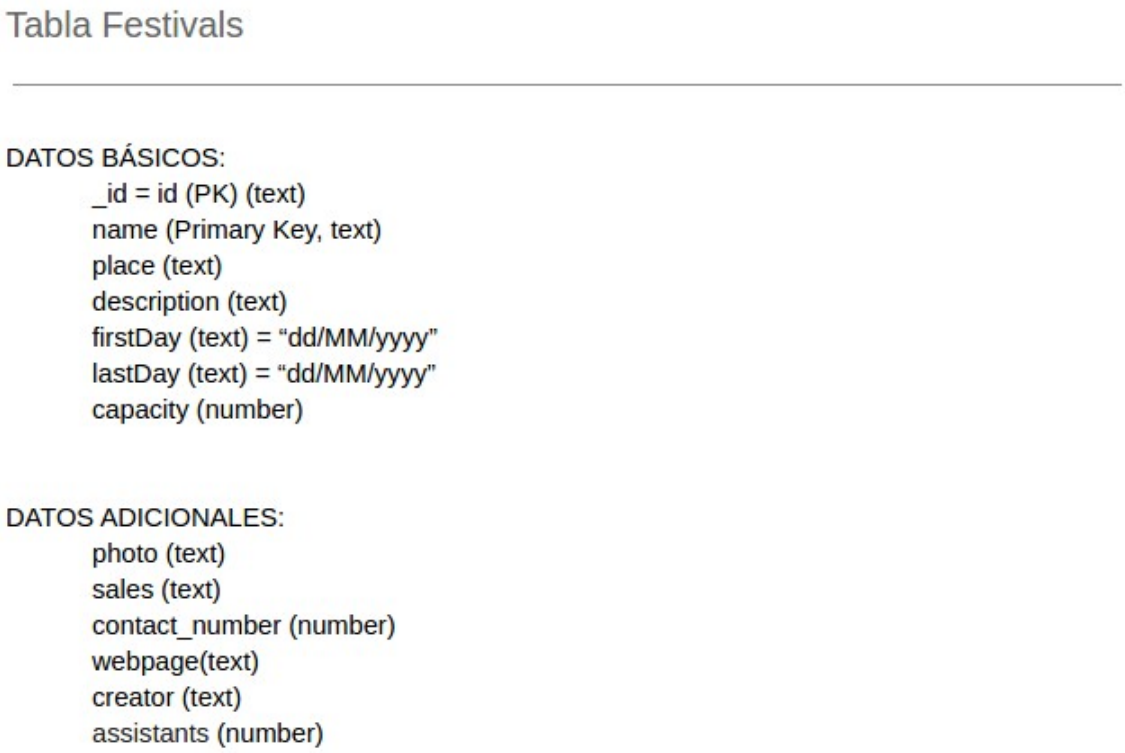

Figura 51 Tabla de Festivales

Al igual que pasaba en la tabla de los usuarios, el atributo “_id” es otro campo que Meteor rellena por sí solo y que los desarrolladores no tienen porque gestionar.

Así mismo, al iniciar el servidor se pueden crear datos dentro de la colección ahora mismo mencionada. El método *startup()* de Meteor permite rellenar la colección con datos que se quieran, y en esta aplicación se ha utilizado también:

```
// start up function that creates entries in the Websites databases.
 Meteor.startup(function () {
    // code to run on server at startup
    if (!Festivals.findOne()){

        ///CREATE FESTIVALS
        Festivals.insert({
            name: "Viña Rock 2016",
            place: "Villarobledo, Albacete",
            description: "Viña rock!!!!!!!!!!!",
            firstDay: "28/4/2016",
            lastDay: "30/4/2016",
            photo: "http://www.dodmagazine.es/wp-content/uploads/2015/01/cartel-vina-rock-2016.jpg",
            sales: "http://www.vina-rock.com/entradas/",
            contact_number: 949112233,
            webpage: "http://www.vina-rock.com/",
            creator: "Javi33",
            capacity: 70000,
            assistants: 0
        });
        Festivals.insert({
            name: "Azkena Rock",
            place: "Vitoria-Gasteiz, Álava",
            description: "El mejor rock de todos los tiempos.",
            firstDay: "12/6/2016",
```

Figura 52 Código para iniciar datos en colección

En este caso se siguen respetando los atributos acordados y mencionados. Entre los mismos, dado que no ha dado tiempo a acordar nada más, se ha decidido crear los datos de forma simple, con Strings, dado que desde la parte Android se tenía dificultad para acceder a los tipos Date y Number, aunque esos tipos son soportados sin problemas en Meteor.

El esquema de los atributos se mantiene siempre, incluso en la realización de acciones sobre las colecciones. Los ejemplos que van a aparecer en las figuras 53, 54 y 55 son códigos de modificación, inserción y eliminación de elementos de colecciones sobre la colección “festivals”, pero son aplicables de igual manera a la colección de usuarios.

```
Festivals.insert({
    name: document.getElementById('nombreFestival').value,
    place: document.getElementById('lugar').value,
    description: document.getElementById('descripcion').value,
    firstDay: fD,
    lastDay: fI,
    photo: p,
    sales: document.getElementById('entradasfestival').value,
    contact_number: n,
    webpage: document.getElementById('urlfestival').value,
    creator: Meteor.user().username,
    capacity: capacity,
    assistants: 0
});
```

Figura 53 Código para insertar datos en colección

En el caso de la inserción de datos, en el ejemplo se observa como se cogen los datos de los elementos de la parte HTML del cliente, mediante los identificadores ID de cada uno de ellos.

```
Festivals.update({_id:festival_id},
    {$inc: {assistants:1}});
```

Figura 54 Código para modificar datos en colección

En cuanto a las modificaciones, se realiza mediante el ID del festival al que se está accediendo (primer campo). En el ejemplo se utiliza el método *$inc* para incrementar en 1 el valor existente en el atributo “assistants”, aunque existen una serie de métodos a utilizar para trabajar sobre las colecciones; también se pueden modificar directamente como se hace normalmente.

```
Festivals.remove({_id:festival_id});
```

Figura 55 Código para eliminar datos en colección

Finalmente, para eliminar un elemento de una colección, con identificarlo adecuadamente basta. En este caso, se utiliza el ID del festival para lograr ese elemento y eliminarlo. Para acceder al dato sobre el ID se utiliza la sentencia *var festival_id = this._id*.

6.4. INFORMACIÓN LEGAL

Aunque en la elaboración de este proyecto no ha dado tiempo para elaborar un informe sobre la información legal del sitio, y no se han gestionado todos los movimientos a realizar para poder tener un sitio real que cumpla las exigencias legales, sí que se ha reflejado toda esta información en una serie de apartados reservados para posibles futuras implementaciones reales que requieran de este apartado. Son tres apartados los dedicados:

- **Política de Privacidad:** el objetivo de este apartado es informar sobre los datos que se recogen, los motivos por los que se recogen y lo que se hace con ellos. Esta información es importante, por lo que es importante dedicarle tiempo para leerlo.

- **Términos de Servicio:** los términos de servicio son los compromisos de uso que se comprometen a respetar todos los miembros de la aplicación. Estos quedan reflejados en diferentes apartados, haciendo que se mencionen todos los aspectos que puedan suscitar conflictos.

- **Política de Cookies:** las cookies son utilizadas habitualmente por los servidores web para diferenciar usuarios y para actuar de diferente forma dependiendo de éstos. Los usuarios normalmente se identifican introduciendo sus credenciales en una página de validación; las cookies permiten al servidor saber que el usuario ya está validado, y por lo tanto se le puede permitir acceder a servicios o realizar operaciones que están restringidas a usuarios no identificados. Las cookies se utilizan también para realizar seguimientos de usuarios a lo largo de un sitio web. El código para implementar una cookie viene en la página de http://politicadecookies.com/.

7

CONCLUSIONES

Estas conclusiones se dividen en dos apartados. Por un lado en cuanto a la gestión del PFG, en el que se puede diferenciar lo planificado y esperado con lo ocurrido realmente. Por otro lado, las conclusiones sobre la tecnología utilizada que, al ser una herramienta novedosa, puede ayudar a diferenciar las ventajas y desventajas con respecto a otras tecnologías.

7.1. GESTIÓN

La gestión del proyecto había sido planificada en primera instancia para poder seguir unas características similares a las exigencias del proyecto. Aunque se han seguido en la medida de los posible los puntos mencionados en dicha planificación inicial, durante el desarrollo se han decidido modificar algunos puntos para poder obtener un producto final acorde a las exigencias del proyecto de estas características.

En cuanto a tareas realizadas, la planificación ha sido la primera parte. En la misma se han descrito todos los puntos a desarrollar en el proyecto, limitando igualmente algunos aspectos que podían ser de conflicto. Además, se hizo una estimación de tiempos para poder concretar las horas a invertir en el mismo, teniendo en cuenta el aspecto de que este proyecto corresponde a 12 ECTS; es decir, 300 horas de trabajo. Aunque la estimación inicial y la final han cumplido con esta expectativa, la variación entre ambos tiempos queda reflejada en esta tabla de la figura 55:

TAREA			HORAS PREVISTAS	HORAS TOTALES	HORAS REALES	HORAS TOTALES REALES
Planificación	Inicial		6H	15H	5H 30min	16H
	Modificaciones		9H		10H 30min	
Aplicación Web	Estudio de tecnología	General	20H	77H	5H 30min	63H 30min
		MongoDB	12H		5H	
		BackEnd (NodeJS + Express)	20H		9H 30min	
		FrontEnd (AngularJS)	20H		20H 30min	
		Meteor	5H		23H	
	Alojamiento y Entorno	Estudio	6H	8H	3H 30min	28H 30min
		Instalación	2H		25H	
	Diseño	Estética	10H	85H	12H	157H 30min
		Implementación	55H		103H	
		Detalles	20H		42H 30min	
	Seguridad	Estudio	8H	20H		
		Implementación	12H			
	Sitio Web	Prueba 1	7H	19H		
		Prueba 2	6H			
		Prueba 3	6H			
Memoria	Plantilla		2H	42H	3H 30min	88H
	Memoria		40H		84H 30min	
Presentación	Plantilla		2H	24H		
	Presentación		22H			
Seguimiento y Control			15H	15H	8H	8H
Gestión			40H	40H	8H 30min	8H 30min
TOTAL			**345H**		**361H**	

Figura 56 Tabla dedicaciones

En la tabla de la figura 54 se observa la diferencia entre las tareas realizadas. Como se trata de una tecnología nueva para el desarrollador, se prepara una fase de aprendizaje de la misma en la que al principio se dedica tiempo al estudio de la tecnología MEAN pero que finalmente se cambia a Meteor. Ese cambio es a consecuencia de que tras iniciarse ese estudio surgió la oportunidad

de trabajar una nueva tecnología, mas moderna y que hasta la fecha no se había trabajado durante el Grado.

Aunque inicialmente se tenían previstos algunos tiempos de dedicación para los aspectos de seguridad y pruebas, la falta de tiempo ha impedido realizar este aspecto. Con ello, se ha dedicado más tiempo al diseño del sitio.

Las lecciones aprendidas es otro aspecto que no se ha abordado. Durante el desarrollo se han ido aprendiendo nuevos aspectos sobre la tecnología utilizada, pero se han introducido dentro de la implementación del sitio ya que se entendía que el aprendizaje residía en implantarlo.

Respecto a la elaboración de la memoria y la presentación de este proyecto, son dos aspectos que, aunque hayan variado horas, se han hecho como estaba previsto.

7.2. TECNOLOGÍA

En cuanto a la tecnología utilizada, Meteor es un framework que proporciona grandes ayudas a los desarrolladores de aplicaciones web. Esta tecnología ayuda a que muchas acciones que se pueden considerar genéricas para todo desarrollador se abstraiga y haga que dicha tarea quede en manos del framework, haciendo que el desarrollador pueda “olvidarse” de dichas tareas y centrarse en otros aspectos de la aplicación.

Los lenguajes y herramientas utilizadas son un aspecto conocido para cualquier desarrollador (JavaScript, HTML, MongoDB,...) o por lo menos de gran conocimiento en el ámbito, por lo que en este aspecto no se requiere conocimiento añadido.

En cuanto a la comunidad de Meteor, a pesar de ser un framework nuevo (2011) se trata de una herramienta de código abierto con gran actividad. Esto hace que la aplicación esté en mejora continua, arreglando los problemas que puedan surgir en determinado momento. Se trata de un modo de trabajo cooperativo y libre que cada vez se utiliza mas, y que ayuda a mantenerlo vivo y actualizado.

Aún así, en la búsqueda de determinadas dudas, errores, problemas... existe el riesgo de que no se encuentren soluciones fácilmente. Esto seguramente sea debido a la novedad aún del framework, pero se trata de un problema no extendido a todas las dudas que se puedan tener.

Además, otra dificultad que se ha encontrado ha sido a la hora de desplegar la aplicación. Al principio, para la tecnología MEAN, fue fácil encontrar el servidor donde alojarlo, pero Meteor genera muchos problemas ya que muchos de los servicios consultados no proporcionan la opción. Este paso era, además, necesario de realizar para que Rachid Boudhar, desde su aplicación Android, pueda conectarse a la base de datos creada en Meteor.

A pesar de estos problemas, en general la experiencia con Meteor ha sido muy positiva, siendo un framework con grandes perspectivas de futuro dada su gran simplicidad y fácil aprendizaje, y su gran ayuda para desarrolladores liberando de determinadas tareas.

7.3. LÍNEAS FUTURAS

Aparte de la tecnología y gestión utilizadas, en cuanto a la aplicación creada se trata de una nueva idea dentro de las redes sociales que mueve gran cantidad de gente. Como idea de futuro, puede ser una idea no tratada hasta la fecha, pero que dado que mucha gente se mueve en el mundo de los festivales puede ser una gran idea.

Hasta el momento, la aplicación es muy simple pero con grandes perspectivas de futuro. Se pueden realizar múltiples mejores en diferentes aspectos, para crear un sitio mas profesional y realista.

Una de las mejoras puede ser la de dividir los tipos de usuarios registrados; se puede crear un tipo de usuario organizador, que sea el encargado de proporcionar la oferta de festivales (promotores...) y otro tipo de usuario para el público que quiera conocer festivales y apuntarse a ellos.

Además, dentro de los usuarios para apuntarse, informarse... se puede crear una red social en base a las opciones ya implementadas de seguir a otros usuarios. Entre los usuarios se puede implementar la opción de crear grupos, para que un grupo de usuarios pueda interaccionar entre sí sobre determinados temas (hablar, proponer festivales...).

Siguiendo con este aspectos a implementar en el futuro, la aplicación puede crear una pasarela de compra de entradas, aunque ese tema sería algo a comentar con los organizadores de los festivales. La opción de generar el calendario, a su vez, puede adquirir mayor importancia dentro de la aplicación, haciendo que se genere realmente un calendario final que pueda ser almacenado por el usuario que se quiera apuntar (en Google Calendar, por ejemplo). Así, el usuario puede verificar en qué fechas tiene previsto asistir a festivales, ayudándole a poder organizarse.

Estas son algunas de las mejoras que se proponen. En base al uso que se vaya dando de la aplicación pueden surgir mas o incluso eliminar algunas, pero la línea a seguir está prevista que sea esa. Al intentar darle un toque de carácter de red social, la línea a seguir debería centrarse en conectar a los usuarios, para que puedan interaccionar entre ellos. Así mismo, es importante darle mas opciones de publicidad a los organizadores, para poder “vender” mejor su producto.

Finalmente, indicar que esta idea se puede trasladar a otros ámbitos: torneos deportivos, congresos... Con perspectiva de futuro, esta aplicación puede ayudar también a esas ideas o incluso realizar una aplicación común para todos los eventos. En principio sólo está pensada para los festivales de música, pero estas otras también necesitan ser publicitadas para poder llegar al mayor número de gente.

8

REFERENCIAS

[1] Tom Coleman, Sacha Greif: *Descrubriendo Meteor*. http://es.discovermeteor.com/

[2] Alvaro Martínez Guaita: *Meteor, una nueva forma de crear apps*. Desarrollo Web (27/07/2012, visto el 16/05/2016). http://www.desarrolloweb.com/actualidad/meteor-nueva-forma-crear-apps-7280.html

[3] Jorge Álvarez Navarro: *Meteor – Desarrollando aplicaciones web en el siglo XXI*. Visto el 16/05/2016. http://www.alvareznavarro.es/desarrollo-web/2014/4/meteor-desarrollando-aplicaciones-web-en-el-siglo-xxi

[4] Bitomule: *¿POR QUÉ METEOR?*. 24/01/2015 (visto el 27/05/2016). http://bitomule.com/por-que-meteor/

[5] Ytalo Elías Borja Mori: *Meteor*. 03/11/2015 (visto el 28/05/2016). http://es.slideshare.net/ytachi0026/meteor-54687818

[6] Leonel Viera: *Aplicaciones en tiempo real con Meteor*. 16/07/2013 (visto el 28/05/2016). http://www.smartgob.com/2013/07/meteor-javascript/

[7] Percolate Studio: *Atmosphere*. https://atmospherejs.com/

[8] Meteor Developers: *Users and Accounts*. Visto el 28/05/2016. http://guide.meteor.com/accounts.html

[9] Juan Manuel Mussachio: *Aplicaciones en tiempo real en una arquitectura orientada a microservicios*. 15/01/2016 (visto el 29/05/2016). http://itech.folderit.net/739/aplicaciones-en-tiempo-real-en-una-arquitectura-orientada-a-microservicios/

[10] Curso de Meteor JS: *¿Qué es Meteor?*. Visto el 29/05/2016. https://openwebinars.net/academia/22/curso-de-meteor-js/1034/que-es-meteor/preview/

[11] Max Villegas, {ida Blog: *Desarrollo de aplicaciones con Meteor*. 21/10/2015, visto el 29/05/2016. http://www.ida.cl/blog/desarrollo/desarrollo-de-aplicaciones-meteor/

[12] José Manuel Cristobal Vera Open Sistemas: *Meteor.js un paso más allá*. 03/08/2015, visto el 30/05/2016. http://blog.opensistemas.com/meteor-js-un-paso-mas-alla/

[13]MongoDB. Visto el 31/05/2016. https://www.mongodb.com/

[14]Rubén Fernández, Genbeta:dev. *MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no)*. 03/02/2014 (visto el 31/05/2016). http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

[15]Víctor Navarro Lázaro. *MongoDB: Introducción a Meteor*. 11/09/2015 (visto el 17/06/2016). http://es.slideshare.net/VctorNavarroLzaro/introduccin-a-meteor

9

ANEXOS

9.1. SEGUIMIENTO Y CONTROL

PLANIFICACIÓN:

- 26/01/2016 (3H):
  - Encontrar la plantilla utilizada en la asignatura “Gestión de Proyectos”, y prepararla para ser rellenada en base a este proyecto.
  - Rellenar los siguientes puntos del documento mencionado: alcance, objetivos, descripción y sistema de información, así como decidir la licencia a utilizar para este proyecto.

- 27/01/2016 (2H 30min):
  - Rellenar los siguientes puntos del documento mencionado el día anterior: entregables, EDT, calidad, riesgos, periodos de realización y estimaciones.

- 31/01/2016 (45min):
  - Revisar el documento de planificación. Durante esa revisión se ajustan las fechas del apartado “Aplicación Web → Sitio Web” para dar un tiempo para las pruebas. Con ello, todas las tareas posteriores a la misma son retrasadas y de nuevo ajustadas.

- 05/02/2016 (15min):
  - Revisión de la distribución de las tareas sobre “Aplicación Web → Estudio de la Tecnología”. En ella, se añade apartado “General” para contabilizar cuando se trabaja sobre toda la tecnología MEAN, no sólo en una parte de ella. Se decide apuntar 20 horas de trabajo.

- 17/02/2016(2H):
  - Añadir apartado de “Antecedentes” al documento de planificación.
  - Leer otros proyectos dirigidos por el tutor para comprender que añadir al documento.

- 24/02/2016(5H 30min):
  - Añadir apartado de “Análisis (Casos de Uso)” al documento de planificación.
  - Añadir apartado de “Modelo de Dominio” al documento de planificación.

- 25/02/2016(2H):
  - Repasar las fechas de entregas, así como todo el documento.
  - Entregar la planificación, en su segunda versión.

APLICACIÓN WEB:

ESTUDIO DE TECNOLOGÍA:

- 03/02/2016 (2H 15min - General):
  - Buscar material para poder realizar el estudio de la tecnología a utilizar. Se obtiene el manual creado por la empresa Magna SIS, y además se buscan cursos para practicar. Se encuentras cursos para MongoDB, Express + NodeJS y para AngularJS, así como otro para toda la tecnología MEAN.

- 05/02/2016 (4H 45min - AngularJS):
  - Realización del tutorial de Codecademy sobre AngularJS: una pequeña aplicación, teoría sobre directivas en AngularJS, teoría sobre servicios en AngularJS y teoría sobre rutas en AngularJS.
  - Redactar el documento sobre lo visto y aprendido en la realización de esta tarea.

- 08/02/2016 (30min - General | 15min - AngularJS):
  - Cambiar los tutoriales a realizar, ya que algunos son de pago y no son completos. Se añaden nuevos.
  - Inicio del tutorial de Curso de AngularJS: introducción al tema.

- 09/02/2016 (8H - AngularJS):
  - Continuación del tutorial sobre Curso de AngularJS: apartados AngularJS, Servicios, Más directivas y Filtros.
  - Redactar el documento sobre AngularJS (Curso AngularJS).

- 10/02/2016 (4H – AngularJS):
  - Terminar el tutorial Curso de AngularJS; apartados Promesas, Rutas, Formularios y Directivas.
  - Redactar el documento sobre AngularJS (Curso AngularJS).

- 11/02/2016 (4H 15min – ExpressJS):
  - Realizar el tutorial de Geeky Theory.
  - Realizar el tutorial de Node Hispano.
  - Redactar el documento sobre ExpressJS (Geeky Theory).
  - Redactar el documento sobre ExpressJS (Node Hispano).

    - Realizar la parte de “Getting Started” del tutorial de Express JS.
    - Redactar el documento sobre ExpressJS (Express).

- 12/02/2016 (2H – MongoDB):
    - Realizar el tutorial de Scotch.
    - Redactar el documento sobre MongoDB (Scotch).

- 14/02/2016 (3H – MongoDB):
    - Redactar el documento Tutorial MongoDB.

- 15/02/2016 (3H 30min – AngularJS | 2H 15min ExpressJS):
    - Redactar el documento Tutorial AngularJS.
    - Continuación del tutorial sobre Express JS: apartados Routing, Writing middleware, y Database Integration de la parte “Guide”.
    - Redactar el documento sobre ExpressJS (Express).

- 16/02/2016 (3H – ExpressJS):
    - Terminar el tutorial Express JS; apartados Error handling, Using template engines y Using middleware.
    - Redactar el documento sobre ExpressJS (Express).
    - Redactar el documento Tutorial Express.

- 29/02/2016 (1H – MEAN):
    - Crear la estructura, portada, y licencia del documento Tutorial.
    - Pasar la información de las tecnologías al documento Tutorial.

- 01/03/2016 (1H 15min – MEAN):
    - Añadir información de cabecera al documento Tutorial.
    - Añadir bibliografía al documento Tutorial.
    - Añadir detalles al documento Tutorial.
    - Reestructurar el documento Tutorial.

- 02/03/2016 (30min – MEAN):
    - Correcciones ortográficas del documento Tutorial.
    - Enviar el documento Tutorial al tutor.

- 10/03/2016 (3H 30min– Meteor):
    - Hacer tareas de Semana 1 del curso sobre Meteor.

- 11/03/2016 (3H – Meteor):
    - Hacer tareas de Semana 2 del curso sobre Meteor.

- 12/03/2016 (3H 30min – Meteor):
    - Hacer tareas de Semana 3 del curso sobre Meteor.

- 13/03/2016 (3H – Meteor):
    - Hacer tareas de Semana 4 del curso sobre Meteor (sin acabar).

- 02/04/2016 (3H – Meteor):
  - Acabar las tareas de la Semana 4 del curso sobre Meteor, y corregir proyectos.

- 03/04/2016 (7H):
  - Repasar todos los conceptos vistos y preparar un tutorial sobre Meteor.

ALOJAMIENTO Y ENTORNO:

- 08/02/2016(45min – Estudio | 45min - Instalación):
  - Análisis de las herramientas a utilizar en el desarrollo. Se intenta instalar Netbeans y Java JDK, pero tras probar con las últimas versiones estables da error de compatibilidad entre los mismos. Por lo tanto, se buscan nuevas vías. En el "Software Center" de Ubuntu se encuentra una versión de Netbeans a instalar (la que se estaba instalando era la 8, y hay se encuentra la 7). Se decide instalar esa para los tutoriales, y continuar con el análisis del entorno más adelante.
  - Elaboración de documento sobre el estudio realizado.

- 04/03/2016(2H – Estudio):
  - Añadir nueva herramienta.
  - Escoger la mejor herramienta.
  - Crear documento de Alojamiento y Entorno de Desarrollo.

- 05/03/2016(45min – Estudio | 30min – Instalación):
  - Mirar más documentación de Cloud9.
  - Añadir la parte de creación de cuenta en Cloud9.
  - Crear cuenta de Cloud9, y enlazarlo a cuenta de Github.

- 03/05/2016 (4H – Instalación):
  - Tras concretar con Rachid algunos detalles de la base de datos, intentar alojar el proyecto en un servidor para que esté accesible a todo el mundo. Se intenta primero con el deploy en el propio servidor de MeteorJS, pero da siempre un mismo error: "Error deploying application: Connection error (connect ETIMEDOUT)". Tras buscar soluciones en Internet, no se encuentra nada en claro ya que en los foros se indica que se tiene el mismo error, dando como respuesta que es culpa de los wizards de meteor. Algunas soluciones son el intentarlo 5 veces el deploy (se ha llegado a intentar hasta 12 veces sin resultado positivo) o iniciar el proyecto en una ventana del terminal y una vez arrancado realizar el deploy en otra ventana. Ninguna solución es adecuada. Desde el sitio web de Meteor tampoco se consigue añadirlo (se creó una cuenta con Rachid).
  - Otro intento que se ha realizado es el de alojarlo en un servidor ajeno a Meteor. Se intenta desde Hostinger, el cual se sube mediante FTP (ya se tenía una cuenta tras la

reunión con Rachid). Al principio no se tiene la dirección correcta, ya que el nombre del main HTML no es el adecuado, pero tras cambiarlo la página queda en blanco.
    - Como información adicional se decide mirar otros servidores, pero todos ofertan lo mismo que lo que ya se ha mirado.
    - Se decide concretar mañana una reunión con Rachid para lograr alojar el proyecto en algún servidor, y así que el pueda acceder a la base de datos.

- 04/05/2016 (4H – Instalación):
    - Se consigue recuperar la contraseña de Hostinger. Dentro, a través de FTP, se observa que se tienen todo los ficheros, junto a otro fichero con los ficheros replicados. Por lo tanto, se borra ese fichero con los replicados, pero sigue ocurriendo que la página queda en blanco.
    - Además se observa que la mejor forma de utilizar el servidor de Meteor es accediendo a la versión Galaxy (http://guide.meteor.com/deployment.html#environment). No se resuelve el problema.
- 12/05/2016 (5H 30min – Instalación):
    - Subir a Heroku el proyecto, para que Rachid pueda coger de hay. Primero se ha subido a Github (http://www.redeszone.net/2013/06/08/utilizar-git-y-github-para-subir-proyectos-desde-ubuntu-ii/) para desde hay conectarlo a Heroku. No se consigue ejecutar, por lo que se intenta subir mediante el Heroku Git (https://dzone.com/articles/meteor-nodejs-app-heroku-5). Tras este intento, Heroku reconoce que la aplicación subida es de Meteor, pero no se muestra mediante la URL. Se deja de momento, y se informa a Rachid.

- 13/05/2016 (1H – Instalación):
    - Rachid informa de que no puede acceder a los datos de la aplicación a través de Heroku. Se mira más opciones de intentar subirlo al servidor de Meteor, y para ello se indaga en las opciones de Meteor Galaxy y Meteor Up, dos opciones que aparecen por Internet recomendadas en las páginas oficiales de Meteor, pero tras la introducción de los comandos pertinentes aparecen errores. No se consiguen subsanar.

- 18/05/2016 (5H – Instalación):
    - Se siguen buscando alternativas junto a Rachid para colocar en servidor la aplicación. Se buscan opciones económicas (en Meteor hay que pagar 25$ al mes). No se encuentra ninguna solución.

- 19/05/2016 (4H 15min – Instalación):
    - Seguir buscando soluciones para dejar en servidor la aplicación. Heroku sigue sin funcionar. Rachid encuentra otra opción, Scalingo. Con una aplicación de prueba de Meteor funciona, así que se intenta con la aplicación real. Da problema con la base de datos (MONGO_URL), pero

instalando el addon de Scalingo MongoDB en la cuenta de Scalingo si que deja.
    - La aplicación está accesible en https://socialmusfest.scalingo.io/ y se miran alguna característica mas. El addon, al ser gratuito, tiene un máximo de 520MB de capacidad (suficiente para este proyecto). En la cuenta creada, también gratuita, sólo se acepta un proyecto, y está accesible hasta el 17 de junio (a partir de esa fecha hay que pagar). Se decide crear una nueva cuenta el 17 de junio para volver a subir la aplicación, y al ver que dura 1 mes la versión gratuita, ese tiempo será suficiente para entregar el proyecto y presentarlo.

DISEÑO:

- 15/03/2016 (6H - Estética):
    - Propuesta de la estética de las páginas, en papel: páginas Principal, Organizador, Usuario, Festival, Crear Festival, Modificar Festival, Configuración y Perfil.

- 16/03/2016 (4H - Implementación):
    - Elegir template de Bootstrap para la aplicación, así como introducirla en el proyecto que se ha creado. Introducir todos los paquetes en el proyecto de Meteor: para bootstrap (meteor add twbs:bootstrap) y para las rutas (meteor add iron:router).

- 17/03/2016 (3H 30min – Estética | 3H – Implementación):
    - Elegir template de bootstrap.
    - Aplicar template al proyecto.
    - Estructurar el código para Meteor.

- 18/03/2016 (2H 30min – Estética):
    - Estructurar la página principal.
    - Asignar las rutas.
    - Crear contenido en la base de datos, así como coger esos datos y mostrarlos en la página principal.

- 21/03/2016 (2H 30min – Implementación):
    - Cambiar footer y header de la página principal, cambiando tamaño de las mismas mediante el código HTML en el header y el CSS (.footer) en el footer. Cambio de foto en el header, y asignar un modo de registro e inicio de sesión con paquete de meteor (meteor add accounts-ui accounts-password). Cambiar links de contacto en footer.
    - Introducir más información en la base de datos.

- 22/03/2016 (2H 30min – Implementación):
    - Crear función para asignar un formato adecuado a las fechas (Date).

    - Crear templates dedicados específicamente a header y footer. Desde los templates de las páginas se llama a esos templates.
    - Crear ruta para poder acceder a la información de los festivales.
    - Crear página dedicada a un festival. Mostrar toda esa información en correcto formato. Crear botón para que vuelva a la página principal.

- 23/03/2016 (3H – Implementación):
    - Mirar la opción de almacenar más campos para los usuarios, para una mejora en la gestión en el futuro. No se obtienen resultados satisfactorios.
    - Anadir las opciones de registro y acceso a la aplicación mediante Facebook y Google.
    - Utilizar el mismo template principal para los usuarios registrados y no registrados. De momento, muestra los mismos festivales en ambos, pero en el futuro se personalizará la información para los usuarios registrados.

- 24/03/2016 (3H – Implementación):
    - Crear botón para acceso a la modificación de la información del perfil del usuario registrado. Hubo problemas con el acceso, ya que no detectaba correctamente la ruta y por lo tanto no mostraba el template. El problema se arregló introduciendo el paquete de rutas de nuevo en el proyecto, desde la terminal (meteor add iron:route).
    - Crear contenido de la página del perfil del usuario.

- 25/03/2016 (3H – Implementación):
    - Estructurar adecuadamente la información de la página del perfil. La funcionalidad aún no está implementada.
    - Crear acceso para la página de Crear Festival.
    - Crear el contenido de la opción de Crear Festival, sin estructurar definitivamente. La funcionalidad aún no está implementada.

- 28/03/2016 (5H – Implementación):
    - Añadir valor de capacidad a los datos de los festivales.
    - Adaptar la página de festivales para mostrar la información adecuada a cada uno de los posibles usuarios del sitio.
    - Gestionar los usuarios (http://es.discovermeteor.com/chapters/adding-users/) (http://bugui-do.blogspot.com.es/2012/12/meteor-cuentas-de-usuario.html) (http://stackoverflow.com/questions/15260243/accounts-oncreateuser-not-firing) (http://blogs.candoerz.com/question/101924/meteor-trying-to-add-another-field-to-the-user-profile-accountsoncreateuser.aspx) (http://stackoverflow.com/questions/16605549/cant-add-user-attribute-using-accounts-oncreateuser). Añadido nuevo parámetro “rol” para el usuario, accesible desde el cliente.

- 29/03/2016 2H 30min – Implementación):
  - Añadir todos los parámetros que describen el perfil del usuario. Esos valores iniciales serán vacíos, salvo el de “rol” que será “Usuario” como valor inicial.
  - Intentar leer los datos de la página de perfil y modificarlos. No se ha obtenido la solución, sólo se consigue mostrar mediante alert() el valor del cambo del nombre de usuario.

- 31/03/2016 (2H – Implementación):
  - Intentar poder modificar el valor de los atributos de los elementos de un usuario, pero no se ha conseguido resolver.

- 01/04/2016 (5H – Implementación):
  - Actualizar la versión actual de Meteor a la 1.3 (meteor update).
  - Arreglado el problema de cambiar datos del perfil de usuario (http://stackoverflow.com/questions/29012836/how-to-partly-update-meteor-users-profile). Para ello, es necesario darle acceso al usuario al cambio de esos datos (https://groups.google.com/forum/#!topic/meteor-talk/F4-xd7pXJfE).
  - Para los casos en los que no se agrega ningún valor nuevo a los atributos, se verifica que el valor que estaba antes no se borra y se mantiene.

- 04/04/2016 (3H – Implementación | 2H - Detalles):
  - Almacenar los datos que se meten en la página de crear festival, creando así un nuevo festival dentro de la aplicación. No se ha creado la opción de generar un calendario aún.
  - Tapar acceso mediante URL a los usuarios que no tengan acceso a diferentes recursos (páginas). Esto se realiza directamente en el HTML, cerrando la opción a aquellos que no estén registrados. (http://stackoverflow.com/questions/15252754/if-statements-in-handlebars).
  - Comprobar que el registro no permite crear diferentes usuarios con los mismos datos. Esto queda verificado, quedando comprobado que no se pueden usar ni username, ni email iguales a alguno de los ya registrados. Así mismo, la contraseña tiene que ser de mínimo 6 caracteres, dando igual de que tipo sean esos.

- 05/04/2016 (1H – Detalles | 2h 30min – Implementación):
  - Eliminar footer, ya que la información que tenía no era demasiado relevante.
  - Cambiar la forma de recoger las fotos de los festivales, para que sólo se tenga que pasar la URL y estar accesible desde la red.
  - Hacer resumen de los detalles esenciales a revisar los siguiente días: mirar si se llega al máximo de aforo, mirar si

hay otro festival con ese nombre y almacenar foto de perfil usuario.
    - Arreglar el error que da al mostrar los botones de asistir a un festival. Si ya se ha indicado que se va a asistir, sólo debe mostrar la flecha de desasistir, y si no se ha indicado que se asiste el botón de asistir. Arreglado controlando este aspecto desde los botones.

- 06/04/2015 (2H – Detalles | 6H – Implementación):
    - Arreglar la opción de no dejar apuntarse a más usuarios si se ha llegado al máximo. Si es el caso, se muestra un div al lado de la asistencia indicando este aspecto. Sólo gestiona al intentar apuntarse al evento presionando el botón de asistencia, ya que es el único sitio donde se debe gestionar, y al desapuntarse se verifica que ese mensaje se elimina.
    - Apuntar nuevo detalle para implementar: dividir el mostrado de los festivales en usuarios.
    - Arreglar la opción de no dejar registrar un festival con un nombre igual al de alguno ya registrado. Se verifica buscando si el nombre del festival es igual a alguno de los introducidos, y si ese es el caso se muestra un mensaje.
    - Cambiar la gestión de las fotos de perfil de los usuarios. Se ha intentado buscar solución, pero no se ha encontrado nada por lo que se ha optado por que se pase una URL (como en el caso de las fotos de los festivales). En el futuro se valorará si cambiar la forma de gestionar las fotos.
    - Se ha dividido la forma de mostrar los festivales para los usuarios registrados, aplicando tres clasificaciones (los creados por el usuarios, a los que ha indicado la asistencia, y un buscador con todos los festivales (creados, asistidos y el resto)). Para el buscador en general, en el futuro se valora si eliminar de esta clasificación los festivales creados y asistidos. Además, como detalle se indica junto al título de cada clasificación el número de festivales que están dentro de dicha clasificación.

- 07/04/2016 (5H – Implementación):
    - Implementar la función de generar un calendario en base a los grupos, horarios y escenarios del festival. Se ha implementado la tabla para indicar el número de cada uno de esos datos, así como mostrar los input asociados a los valores indicados. Poner que el número de grupos este en un rango de 1 a 99 grupos posibles, los horarios entre 1 y 9, y los escenarios entre 1 y 9. Para las valoraciones de cada elemento, se intenta introducir unas estrellas, pero dadas las dificultades para encontrar un elemento como tal en HTML y que el ejemplo del tutorial sobre Meteor no es aplicable a esta aplicación, se opta de momento por valorar mediante un input de tipo number, con valores entre 0 y 10. (https://developer.mozilla.org/es/docs/Trazado_de_una_tabla_HTML_mediante_JavaScript_y_la_Interface_DOM)

- 11/04/2016 (3H – Implementación):

    - Mirar forma de calcular el número de días entre dos fechas para poder crear una correcta tabla. (http://blog.jmacoe.com/programacion/web_developer/javascript-numero-dias-entre-dos-fechas/) No resuelto.

- 12/04/2016 (4H – Implementación):
    - Arreglar problema de calcular el número de días entre dos fechas. El problema era que el separador era “-” en vez de “/”.
    - Guardar todos los datos de grupos, horarios y escenarios en un array (nombre + puntuación).
    - Conseguir meter en la tabla el nombre del grupo con la puntuación más alta.
    - Borrar elementos del array (grupos), para que así en la propuesta aparezcan todos los grupos y no se repitan.

- 13/04/2016 (2H 30min– Implementación):
    - Almacenar los datos del escenario en un array (cada escenario duplicado para todos los días que dure el festival).
    - Conseguir asociar a cada grupo un escenario (a los grupos con la puntuación más grande el escenario con la puntuación más alta).

- 14/04/2016 (2H 30min– Implementación):
    - Conseguir mostrar los horarios en la tabla (en la primera columna).
    - Intentar asociar de nuevo los escenarios a los grupos, ya que tras introducir los horarios no aparecen (o lo hacen de forma incorrecta). Se verifica que no hay problema de guardado de los escenarios, sino problema de obtener los datos del array. No se consigue arreglar.

- 15/04/2016 (2H 30min – Implementación):
    - Arreglar la asignación de los escenarios a los grupos. Ocurría un error con una variable (s), que no se inicializaba correctamente para los escenarios.

- 16/04/2016 (6H – Implementación):
    - Cambiar el tipo de input de los horarios de “text” a “time”.
    - Verificar que en la primera fila de la tabla de propuesta aparece el mejor horario, con los mejores grupos (desde el mejor para abajo) en los mejores escenarios.
    - Cambiar el formato de las fechas al formato español. Para ello, a la función utilizada para cambiar el formato de las fechas se le introduce el identificador de España (toLocaleDateString("es-es")).
    - Se introducen las fechas en la tabla de propuesta. Para ello, se crea un array donde se almacenan todas las fechas que hay entre la fecha de inicio y la de fin. A continuación, en la primera fila se introducen la fechas, cambiándoles el formato para poder mostrarlo de forma agradable.
    - Hacer una verificación de que, antes de que se guarden los datos de un festival, el usuario haya introducido el nombre

del festival, descripción, capacidad, día de inicio, día de fin y lugar. Así mismo, verificar que antes de acceder a la opción de generar un calendario se introduzcan los datos de las fechas, y dentro de esta opción que se introduzcan todos los elementos que se han pedidos (en base a los números asignados).
    - Eliminar las selecciones una vez que el usuario quiera tomar otra decisión en el generador de calendario (incluso eliminar esta opción de nuevo para que no quede almacenada). Mediante jQuery, y su función “empty()”, se elimina el contenido creado en los div para introducir los nombres y puntuaciones de grupos, horarios y escenarios, y la propuesta realizada. Con ello, se le da la opción al usuario de introducir nuevos datos. Con esa función empty queda eliminado toda tabla que se crea, quedando un espacio en blanco (no le aparece nada al usuario).
    - Se crea un botón para volver a la pantalla de introducir el número de grupos, horarios y escenarios (desde la pantalla de introducir los nombres y puntuaciones de los mismos).

- 17/04/2016 (3H 30min – Implementación):
    - Reordenar la tabla de propuesta, para que muestre las filas en función del orden en el que se meten los horarios. Se le avisa al usuario de que la tabla se mostrará en el orden en el que meta los horarios.
    - Verificar que el botón creado de volver de la introducción de los nombres y valoraciones de grupos, horarios y escenarios a la introducción del número de elementos de los mismos funcionaba mal. La llamada a la función no era la correcta, ya que el nombre no estaba bien asignado en el código JavaScript, por lo que se cambia al correcto y ya funciona perfectamente.
    - Dar permiso de “remove” a los usuarios, para que estos puedan borrar los festivales que han creado. Crear botón para eliminar un festival, y crear un evento que gestione esa acción. El mismo, elimina el festival de la base de datos y del array festivalesCreados del usuario. Además, hacer que el botón de eliminar sólo le aparezca al creador del festival, para que este sea el que tiene la opción.

- 18/04/2016 (4H– Implementación):
    - En la opción de crear un festival, se introduce la verificación de que las fechas que se introducen son correctas (fechaInicio =< fechaFin). Se enseña un mensaje en rojo indicando el error.
    - Se cambia el tipo de input de la capacidad de un festival a “number”, con un mínimo = 1 y sin máximo.
    - En la página del perfil, se cambian los mensajes que habían de contraseñas y emails correctos (que no coinciden). Antes se mostraba un alert, ahora un mensaje en rojo. Además, se introducen textos para indicar que los campos que no se rellenen se quedarán tal y como estaban.
    - En la página principal del usuario, se crean etiquetas y botones para mostrar a la gente de la aplicación, los

seguidores del usuario y a los que sigue el usuario. Se crea el evento para la gentes en general que, presionando un botón con un signo plus, se cree una tabla debajo (esa tabla pisa lo que va debajo de la etiqueta). También se puede cerrar esa tabla presionando un botón con un signo minus.

- 19/04/2016 (2H – Detalles):
  - Reunión con Rachid para acordar el contenido de la base de datos. Se hace una comparación de lo que se estaba almacenando cada uno, y se debaten sobre datos básicos (los que se pedirán como obligatorios) y los que son adicionales. Se acuerdan esos datos y se crea una cuenta en “mLab” donde se almacenará la base de datos. Se crea una colección “Evento” y se almacenan cuatro documentos dentro (uno por festival).

- 24/04/2016 (5H – Implementación):
  - Añadir al buscador general de perfiles los username de todos los usuarios registrados en la aplicación. Así mismo, se intenta eliminar de dicha lista el usuario en sí que está registrado, pero esta acción cuesta en hacerla y finalmente se deja de lado, como posible mejora en el futuro.
  - Se genera una ruta que, seleccionando uno de los nombres de username de la lista del buscador, saque la página del perfil de dicho usuario. La ruta no se realiza, porque es necesario indicar meteor add iron-route, pero al añadir este paquete da un error de “no such package”, por lo que se tiene que solucionar. Se comprueba que el error es la denominación del paquete, ya que es iron:route.
  - Conseguir que se redireccione al template correcto desde el username. Se tuvo error porque esa redirección se tiene que poner tras la del principal, y al tenerlo al final no se realizaba correctamente.

- 25/04/2016 (5H – Implementación):
  - Cambiar el diseño de la página sobre la información de un usuario. De momento, sólo se muestra la información de usuario que está conectado.
  - Cambiar diseño de la base de datos para que los atributos tanto de festivales como de usuarios sean acordes a lo acordado con Rachid. Al cambiarlo se realizaron modificaciones también en los eventos de modificación e inserción en la propia base de datos. Se trabajó con un error de que no se realizaba bien la creación de los atributos nuevos para los usuarios, pero tras una serie de intentos de arreglarlo (volver a instalar el paquete de autopublish, cambiar la forma de suscribirse a una publicación,...), se verificó que el error residía en que para asignar los valores iniciales se utilizaba el signo “:” y hay que usar “=”.

- 26/04/2016 (6H 30min – Implementación):
  - Asignar los valores del número de seguidores, siguiendo y gente total del usuario registrado. Esos datos aparecen

junto a las etiquetas correspondientes en la página del usuario.
    - Mostrar en la página del usuario los username de los seguidores y siguiendo del usuario actual. El enlace lleva a la página del usuario en cuestión.
    - Cambiar la url de acceso a la página del perfil de un usuario. Antes aparecía la extensión “/username”, pero ahora aparece el username del usuario en cuestión (“/{{username}}”).
    - Conseguir que se muestre el username correcto en la página del perfil. De momento, el resto de parámetros no se muestran. Además, hay problemas con las rutas; o muestra bien la del perfil del usuarios, o sino la del resto, pero sólo una de las opciones. Se verifica que el problema está en el lugar en el que se asigna la ruta del perfil del usuario, ya que quedan incorrectas todas las que aparecen debajo de la misma. Al ponerla la última es la propia ruta a la información del perfil la que falla.

- 02/05/2016 (7H 30min – Implementación):
    - Arreglar el problema de las rutas. Ahora se realizan bien las asignaciones de la URL, el problema residía en que al poner dos asignaciones (_id y username) en el mismo nivel, sólo se leía la que aparecía en primer lugar. Por lo tanto, para arreglar este problema a la ruta de la información del perfil se le ha añadido “perfil”, quedando como “/perfil/{{username}}” (el del id queda de la misma forma que estaba).
    - Solucionar el problema de que un usuario obtenga todos los datos de otros usuarios para mostrarlos en la información del mismo. Tras observar que el problema era de que los usuarios no devolvían toda la información cuando se realizaba el Meteor.user.findOne() en la ruta, se ha investigado por Internet. En ella, tras mirar varios sitios, se encuentra que el problema era de que a la hora de realizar el publish desde el cliente no se leían todos los atributos en el cliente, por lo que se encuentra la solución de que desde el cliente sólo se devuelva el find del user que se hace sin pasar ningún atributo: return Meteor.users.find(); (https://forums.meteor.com/t/meteor-users-findone-selector-help/8104/2) (http://www.colbycheeze.com/blog/2015/06/publishing-user-data.html).
    - Se realiza la acción de seguir y dejar de seguir a personas. Para ello, se les da funcionamiento a los dos botones creados, realizando las mismas acciones que se hacían a la hora de apuntarse a un festival. Es decir, se comprueba que la persona que se quiere seguir no sea ella misma (en ese caso aparece un mensaje indicando la imposibilidad de realizar dicha acción), siguiendo por confirmar que todavía no se sigue a esa persona (o si ya se sigue si se quiere empezar a seguir) y añadiendo el username de la persona a seguir en la lista following del actual usuario, así como añadir a la lista followers de la persona a seguir el

username del usuario actual. Se comprueba que aparecen correctamente los nombres de los usuarios en las búsquedas de “Siguiendo”, “Seguidores” y “Buscar Gente”.

- 10/05/2016(9H – Detalles):
  - A la hora de buscar gente, se eliminan de entre todos los usuarios que aparecen en “Encuentra Gente” al usuario que está conectado actualmente (a si mismo). Así mismo, se resta uno al número de esa opción (ya que el usuario en concreto no cuenta). A su vez, se pregunta si los usuarios que ya aparecen en “Seguidores” y “Siguiendo” se tienen que eliminar de “Encuentra Gente”. La misma situación se encuentra en los festivales, por lo que se pregunta a Rachid que implementación va a llevar a cabo en ese caso. Como él va a mostrar todos los festivales en la opción “Buscar Festival” se decide seguir la misma línea, así como con los usuarios. Jon Garrido también da su opinión, dejando ver que dependiendo la implementación que se le quiera dar las dos opciones son buenas. Se deja tal y como está.
  - Dentro de la página principal del usuario se cambia el encabezado, para que se diferencie de los encabezados de los festivales. Como al cambiar la etiqueta “h2” del encabezado se pierde el tipo de letra, se decide que por estética se deja esa para dar la bienvenida al usuario. Se cambia la de los festivales, rebajando a “h3” el tipo de letra (pierde la estética de los encabezados) y se cambian las estrellas que rodean el título por algo nuevo. Se encuentra uno por Internet, el cual sólo se coloca en la parte superior del texto del título por estética. Se pregunta a Rachid que tal queda, dando este su aprobado.
  - Se añade el footer ya creado al principio de la implementación. Se añade ese footer a todos los templates de la página. El footer no se ajusta al contenido de la página, ya que o bien deja mucho espacio con el contenido o pisa el mismo. El problema está en el CSS, en el cual hay una propiedad “margin-top” que coloca el footer al número de pixeles que se le indica. Recordad que este CSS viene de la plantilla que se ha utilizado, por lo que tras buscar alguna solución no se encuentra nada adecuado (si se arregla el espacio en blanco no el pisado y al revés), por lo que de momento se suprime el footer.
  - Mostrar los festivales ordenados por la fecha de inicio. Los más recientes aparecen en primer lugar, seguido por los más lejanos en esa fecha. Al no ser un Array, primero los elementos que se encuentran en el find se pasan a un Array, para poder ordenarlos mediante la función sort() por el atributo firstDay (sin meterlos en Array no se ejecutaba la función sort()). Se aplica este cambio para todas las opciones de mostrar los festivales.
  - Insertar icono de favicon (con la imagen del logo). La línea introducida en el head del HTML es: <link rel="icon" type="image/png" href="/img/logo.png" />
  - Redistribuir los botones de “Modificar Perfil” y “Crear Perfil” de la página principal del usuario, metiéndolos en div y

centrándolos en la página. Debajo de los mismos se crea una tabla con título “Contactos”, donde aparecen las etiquetas de la gente (posicionadas con espacios, no con la position que lo posiciona sobre la página en total). Al seleccionarlas, las tablas aparecen debajo ocupando toda la línea, por lo que aparecen unas debajo de las otras (no se ha conseguido poner una al lado de la otra).
    - Cambiar los botones de “Eliminar” y “Salir” de la página que muestra la información de los festivales. Se quitan la position y se alinean a la derecha de la página, uno al lado del otro, adaptando el tamaño y el color con el sentido de los mismos. Realizar la misma acción para los botones “Guardar” y “Cancelar” de la opción de modificar el perfil, los botones “Guardar” y “Cancelar” de la opción de crear un festival y el botón “Salir” de la página de la información de un usuario.
    - Para los festivales y usuarios que no tienen imagen asociada, se pone una imagen que indica que no se tiene imagen asociada. Esa imagen se guarda en la carpeta “public/img/noimg.gif”, y en cada llamada a la foto, mediante un método en Javascript se mira si el atributo de la imagen tiene una URL asociada o no (en caso de no se pone la imagen mencionada).
    - La información de la capacidad y asistentes a un festival (así como la opción de asistir o no asistir al mismo) dentro de la página del festival se guarda en un cuadro.

- 11/05/2016 (7H – Detalles):
    - Poner los layouts que muestran los festivales en las páginas principales de igual tamaño. Para ello, se genera un tamaño fijo para las imágenes, al igual que en las páginas que muestran la información del festival (con tamaño mas reducido). La muestra de los festivales se verifica que sigue correcta al crearse una nueva línea.
    - Comprobar el problema con la opción de recuperar la contraseña. Se detecta que los mensajes no llegan a enviarse, es necesario cambiar la variable MAIL_URL. Se busca en Internet solución (http://www.meteorpedia.com/read/Environment_Variables) (http://docs.meteor.com/#/full/http_del), y como en los tutoriales vistos siempre se utiliza Mailgun se crea una cuenta, pero a pesar de poner las credenciales sigue habiendo error (DeliveryError: Message delivery failed: 554 Sandbox subdomains are for test purposes only).

- 18/05/2016 (30min – Detalles):
    - Tras hablar con Rachid, se decide que los festivales anteriores a la fecha actual no se le muestren a los usuarios. Esos festivales sólo serán accedidos para los creadores (no se implementa la opción de eliminar el festival una vez finalizado, esa tarea la hará el propio creador). Se implementa el no visionado de los festivales pasados para los apartados “Mis festivales” y “Buscar festivales”.

- 19/05/2016 (3H 30min – Detalles):
  - Se comprueba que Rachid tiene problemas para gestionar los datos de tipo Date(). Por lo tanto, y tras debatir entre los dos posibles soluciones, se decide cambiar esos datos de tipo Date a String, y por lo tanto también cambiar las acciones que se hacen sobre los mismos.
  - Desde la aplicación de Android se verifica que algunos datos que deben ser de tipo Number se almacenan como String. Se comprueba que eso es debido a que, al coger los datos de un elemento (document.getElementById().value) esto devuelve String. Se cambia los mismos con un parseInt para convertirlos a Number.

- 20/05/2016 (6H – Detalles):
  - Mirando posibles soluciones de mejora con Rachid, se decide cambiar que si el usuario no introduce foto se almacene directamente en la base de datos la URL de la no imagen, para que luego no se tengan que hacer comprobaciones. Así mismo, se decide mantener la foto actual de No Imagen para las fotos de perfil, pero cambiar la foto para los festivales. Se crea una imagen con el logo de la aplicación, y encima un texto de No Imagen.
  - Se comenta la necesidad de indicar tres apartados importantes en el sitio web: política de privacidad, términos de uso y aviso de cookies. Desde la aplicación de Android se dispondrá de enlaces que lleven a las páginas creadas en la web para estos apartados. Para el aviso de cookies, http://politicadecookies.com/descargas.php, http://blog.legisconsulting.com/2013/09/modelo-de-aviso-y-politica-de-cookies-segun-la-agpd/, se obtiene el código de la la página que el Gobierno de España tiene para el tema, adaptando el texto al uso actual.
  - Se genera la página del aviso de las cookies. Esa página se genera en base al ejemplo que da la AGPD.
  - Se generan los enlaces a la Política de Privacidad, Términos de Servicio y Aviso de Cookies. En la parte de footer se observa que se tienen problemas de colocar, por lo que se encuentra un sitio adecuado en el header.

- 21/05/2016 (6H – Detalles):
  - Se crea la página de Política de Privacidad, y se crea la ruta correspondiente. https://www.agpd.es/portalwebAGPD/index-ides-idphp.php. http://www.susdatos.es/proteccion-datos-documentos-gratis/documentos/modelos-de-clausulas-para-paginas-web/modelo-de-politica-de-privacidad-para-pagina-web-con-formulario. http://ayudaleyprotecciondatos.es/2012/03/07/modelo-politica-privacidad-web/. Como no se tiene fichero en la AGPD, dicho dato de momento no es correcto.
  - Se crea la página de Términos de Servicio, y se crea la ruta correspondiente. http://www.timeinc.net/subs/privacy/termsofservice/pptos.ht

ml. https://formaldocs.com/2014/12/Modelo-de-condiciones-de-uso-de-una-pagina-web.

- 23/05/2016 (3H 30min – Detalles):
  - Cambiar la ubicación del mensaje de las cookies. Como cada vez que se carga la página principal sale el mensaje (la página principal es donde se ubica ahora el mensaje, debajo del header) a pesar de darle al OK, mediante JavaScript y un display se mira si es la primera vez que se carga la página. Se utiliza una variable global inicializada a 0 que, si es 0 saca el mensaje y si no no lo saca. Al mostrarlo se suma uno a esa variable global.

MEMORIA:

PLANTILLA:

- 14/05/2016 (3H 30min):
  - Buscar la plantilla que proporciona la Facultad de Informática para las memorias de PFG. Así mismo, se buscan los PFG que ha llevado a cabo el director de este proyecto, José Ángel Vadillo, y se obtienen. Rachid también proporciona algunas memorias sobre web, Android y otros que le han parecido interesantes. Se revisan, y se crea la portada y el inicio del índice de la memoria. Se deja el resto del índice para rellenar durante la elaboración del documento. Se decide mantener los tipos de letras y demás del documento que proporciona la Facultad de Informática en su plantilla de memoria.

MEMORIA:

- 16/05/2016 (5H):
  - Redactar la sección de “Resumen” del documento, basado en otros resúmenes de memorias de proyecto dirigidos por José Ángel Vadillo.
  - Redactar la sección de “Introducción” del documento. Tras consultar otros PFG dirigidos por José Ángel Vadillo, en esta sección se introduce la motivación (antecedentes) del proyecto. Este apartado está en el documento de planificación.
  - Redactar la sección “Documento de Objetivos del Proyecto”. Se reestructura esta sección por el cambio de introducir los antecedentes en la sección anterior. Se miran otros PFG dirigidos por José Ángel Vadillo, y entre los puntos ya introducidos se utilizan el “Alcance”, dentro definiendo los objetivos, el esquema y el EDT. Los esquemas de la parte de la estructura del proyecto y EDT se modifican para adaptarlos.

- 17/05/2016 (3H):

    - Redactar la sección de “Periodos”, dentro de la sección de “Documento de Objetivos del Producto”. Se restablecen los periodos para poder aplicarlos a la realidad, y se crea el diagrama de Gantt acorde. Ese diagrama se realiza a través de la aplicación Gantt Project, ya que se observa que otros PFG utilizaban dicha herramienta.

- 18/05/2016 (2H 30min):
    - Redactar la sección de “Análisis”. En base a los casos de uso ya redactados, y mirando otras memoria, se crea una subsección “Actores” con un diagrama de los actores y las acciones que estos pueden hacer. En la siguiente subsección se introducen los casos de uso, cambiando diferentes aspectos de los mismos y creando diagramas de las acciones que se puedan hacer. No se acaba con los casos de uso.

- 19/05/2016 (2H):
    - Acabar de redactar los casos de uso, en la sección de “Análisis”.

- 21/05/2016 (2H 30min):
    - Repasar todo el documento redactado, y adaptar los contenidos a las páginas.

- 24/05/2016 (4H):
    - Crear los siguientes puntos principales de la memoria, en base a otras memorias de otros proyectos. Esos puntos son “Diseño”, “Arquitectura”, “Implementación” y “Conclusiones”.
    - Dentro de la sección “Diseño”, se crean tres subgrupos, en base a otras memorias: “Diagrama de Contexto”, “Diagrama de Clases” y “Diagrama de Secuencia”.
    - Se crea el Diagrama de Contexto, redactando dicho punto y creando el propio diagrama.
    - Se crea el Diagrama de Clases, redactando dicho punto y creando el propio diagrama.

- 25/05/2016 (4H):
    - Hacer todos los diagramas de secuencia de las acciones que realiza la aplicación. Primero se buscan ejemplos, tanto en otros PFG como en Internet, para finalmente descargar la aplicación StarUML en Ubuntu y crear los diagramas de la aplicación.

- 28/05/2016 (6H 30min):
    - Crear la estructura de los apartados de los temas 5 (Tecnología), 6 (Implementación) y 7 (Conclusiones), en base a lo elaborado a lo largo del proyecto.
    - Dentro del apartado de “Tecnología”, se escribe la introducción del mismo, y los subapartados “Meteor” y “Fortalezas de Meteor” (dentro de este último también se escribe sobre la “Gestión de Usuarios”).

- 29/05/2016 (5H):
  - Se escribe el apartado de “Arquitectura de Meteor”, dentro de la sección “Tecnología”.

- 01/06/2016 (4H):
  - Se escribe el apartado de “Entorno de Desarrollo”, dentro de la sección “Tecnología”.
  - Repaso de las sección 5 “Tecnología”. Enviar proyecto a Jose Angel Vadillo.

- 02/06/2016 (3H):
  - Redactar el apartado de “Implementación” los apartados del “Lado Cliente” que son “Interfaz de Usuario” y “Enrutamiento”.

- 03/06/2016 (2H):
  - Redactar el apartado de “Entorno de Desarrollo”, dentro del cual se ha definido los subapartados “Local” y Servidor”

- 06/06/2016 (6H):
  - Redactar los apartados de “Implementación” llamados “Lado Cliente”, “Lado Servidor” y “Base de Datos”, y sus correspondientes subapartados.

- 07/06/2016 (3H 30min):
  - Redactar los apartados de “Implementación” llamados “Plantilla” e “Información Legal”.
  - Redactar, dentro del apartado “Conclusiones”, las secciones de “Conclusiones” y “Gestión”.

- 08/06/2016 (3H):
  - Redactar, dentro del apartado “Conclusiones”, las secciones de “Tecnología” y “Líneas Futuras”.
  - Tras mirar otros PFG, se decide subir el tamaño de la letra redactada de los puntos de 11 a 12, para una mejor visión para el usuario.

- 09/06/2016 (2H):
  - Arreglar el problema de la sangría para todos los párrafos, haciendo que sea igual en todas.
  - Arreglar los espacios este apartados, imágenes.
  - En el índice, indicar las páginas de los apartados y subapartados de la memoria. Actualizar a su vez las páginas de las figuras.

- 11/06/2016 (4H):
  - Cambiar los enlaces de la imagen de los actores. Además, actualizar la información de las imágenes de los casos de uso particulares y cambiar el orden de esos particulares, haciendo que sigan un orden más lógico.
  - Revisar a papel todos los diagramas de secuencia. Se queda en papel para pasarlo al documento de memoria.

    - Cambiar en el EDT “Aplicación Web” por “Producto”.

- 12/06/2016 (2H):
    - Arreglar los diagramas de secuencia en base a lo realizado en papel. Se utiliza la herramienta “Creately”.

- 13/06/2016 (3H 30min):
    - Revisar el apartado “Resumen”. No se cambia nada, no se ve necesidad.
    - Cambiar el apartado “Introducción”, en base a otras memorias de PFG.
    - Se elimina el apartado “Objetivos” dentro del capítulo 2. Con lo explicado en otros puntos se cree arreglado.
    - En la sección “Análisis” se revisa la introducción y se añade un subapartado de “Captura de Requisitos” para definir los puntos a tratar.
    - Volver a ordenar el documento de memoria, para asemejarlo a otros ejemplos.

- 14/06/2016 (3H 30min):
    - Revisar los 3 primeros capítulos de la memoria.
    - Arreglar los puntos: cambiar diagrama de clases (eliminar la clase “Festivales” y poner “Eventos”, así como especificar más detalladamente los atributos y operaciones), añadir “Análisis de los Requisitos” en el EDT (dentro del Producto) y cambiar nombres en el punto sobre requisitos.

- 15/06/2016 (2H 30min):
    - Añadir texto a los capítulos que solo son una imagen.
    - Cambiar coloquialismos.
    - Enviar los cambios de los primeros 4 capítulos a Jose Angel Vadillo.

- 16/06/2016 (3H):
    - Revisar los capítulos del 1-4.

- 17/06/2016 (2H):
    - Arreglar detalles indicados por Jose Angel Vadillo de los temas 5-6-7
    - Reestructurar los puntos, acorde a la propuesta de mejora de Jose Angel Vadillo.

- 18/06/2016 (2H 30min):
    - Añadir anexo de Seguimiento y Control.
    - Añadir todas las actas como anexos.
    - Añadir a la tabla de dedicaciones dos filas para comparar el tiempo total previsto y el tiempo total invertido en el proyecto.

- 19/06/2016 (3H 30min):
    - Añadir anexo sobre el tutorial de Meteor.
    - Revisar los capítulos 5-7 y anexos.

- Enviar versión a Jose Angel Vadillo.

SEGUIMIENTO Y CONTROL:

- 04/02/2016 (1H):
  - Crear documento de “Seguimiento y Control” para ir apuntando lo realizado, y el tiempo invertido, en cada día.
  - Se rellena lo realizado hasta el momento, a día 4 de febrero de 2016.

- 18/02/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 03/03/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 17/03/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 31/03/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 14/04/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 28/04/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 12/05/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 26/05/2016 (30min):

Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 09/06/2016 (30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

- 19/03/2016 (2H 30min):
  - Hacer un seguimiento de todo lo realizado, contabilizando las horas invertidas y previstas hasta la fecha.

GESTIÓN:

- 05/02/2016 (15min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 11/02/2016 (45min):
  - Rellenar propuesta en el GAUR, como paso para la inscripción del proyecto.

- 17/02/2016 (15min):
  - Organizar la carpeta de “Planificación” en base a versiones, en Google Drive y en local.
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 02/03/2016 (1H 15min):
  - Pasar por el despacho del tutor.
  - Organizar la carpeta de “Seguimiento y Control” en base a versiones, en Google Drive y en local.
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 14/03/2016 (15min):
  - Acordar con el profesor el cambio a la tecnología Meteor.
  - Actualizar documento de Seguimiento y Control.
  - Cambiar tabla del Seguimiento y Control, para adaptarla a las nueva situación.

- 26/03/2016 (15min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 13/04/2016 (30min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 18/04/2016 (30min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 26/04/2016 (30min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 02/05/2016 (30min):
  - Hacer copia de seguridad del proyecto en Google Drive, Dropbox y local.

- 18/05/2016 (45min):
  - Ir a secretaría a preguntar por la matrícula del proyecto. Mirar con Rachid en el GAUR como se haría.

    - Ir al despacho del José Ángel Vadillo. Comentarle como vamos tanto Rachid como yo, y quedamos pendientes de ir enviándole partes de la memoria a medida que vayamos avanzando con la misma, y a pasar la semana que viene por su despacho para enseñarle la aplicación. Comenta que ahora los servidores de Meteor son de pago, tenemos que buscar nuevas opciones. Indica que no hay que subir código a ningún lado, lo importante es la memoria.

- 21/05/2016 (15min):
    - Pasar la primera versión de la memoria a José Ángel Vadillo. Así mismo, indicarle las fechas de posible reunión (semana del 30 de mayo) y facilitarle la URL para que pueda ver la aplicación.
- 01/06/2016 (45min):
    - Reunión con José Argel Vadillo. Se indican algunos cambios a realizar en los casos de uso, y la introducción de algunas secciones nuevas.

- 09/06/2016 (30min):
    - Mirar bien el documento de la memoria y enviársela por correo a José Ángel Vadillo. Se le indica que no es una versión definitiva, y que se seguirá revisando.

- 10/06/2016 (45min):
    - Concretar con José Ángel Vadillo detalles a mejorar en la memoria. Junto a ello, concretar una reunión con él para el martes 14.
    - Hacer copia de seguridad del proyecto.
- 14/06/2016 (30min):

    - Reunión con Jose Angel Vadillo en el que se indican algunos errores de los 4 primeros capítulos.

9.2. ACTAS DE REUNIONES

ACTA DE REUNIÓN 02/02/2016

Reunidos en la Facultad de Informática de San Sebastian, el día 02 de Febrero de 2016 a las 12:00 horas, con la asistencia de los participantes enumerados a continuación, la reunión sobre el proyecto SocialMusFest trató y acordó las siguientes cuestiones:

ASISTENTES:

- Jon Junguitu
- Rachid Boudhar
- José Ángel Vadillo

ORDEN DEL DÍA:

1. Propuesta de PFG.
2. División de trabajo.

ACUERDOS:

1. Se presenta por parte de Jon Junguitu y Rachid Boudhar una propuesta de PFG conjunto. La idea es la de realizar una aplicación, tanto en web como en Android, sobre festivales de música. La motivación de realizar la aplicación está presente en un documento que se le hace llegar a José Argel Vadillo, el cual lo ha leído y acepta la idea. Se discuten algunos detalles de la aplicación. Finalmente, se acepta por parte de todos los participantes el documento existente.

2. De cara a realizar un proyecto en conjunto, Rachid Boudhar indica que él hace la aplicación para dispositivos Android y, por consiguiente, Jon Junguitu lo realiza para web. Este último propone la idea de MEAN, la cual es aceptada por José Ángel Vadillo.

Tratados todos los temas, finaliza la reunión a las 12:45 horas del día citado, estando todos los presentes de acuerdo con las decisiones tomadas.

ACTA DE REUNIÓN 19/04/2016

Reunidos en la Facultad de Informática de San Sebastian, el día 19 de Abril de 2016 a las 10:00 horas, con la asistencia de los participantes enumerados a continuación, la reunión sobre el proyecto SocialMusFest trató y acordó las siguientes cuestiones:

ASISTENTES:

- Jon Junguitu
- Rachid Boudhar

ORDEN DEL DÍA:

1. Acordar detalles conjuntos de las aplicaciones.
2. Acordar una base de datos conjunta.
3. Acordar lugar para alojar la aplicación web.

ACUERDOS:

1. Se discuten los detalles de la aplicación.
2. En cuanto a la base de datos, se comparan los atributos introducidos en ambos proyectos. Primero se escogen los atributos más importantes. Una vez escogidas las comunes, se discuten el resto. Tras debatir, se crea una estructura común de base de datos que se crea en la aplicación web, y a la que tendrá acceso la aplicación Android mediante el acceso al servidor donde esté alojada la aplicación web.
3. En cuanto al alojamiento, se escogen una serie de opciones para ir comprobando su posible uso. Durante el transcurso de los siguientes días, entre los dos asistentes a la reunión se van a intentar las opciones planteadas.
    - Meteor Server
    - Hostinger
    - Cloud9
    - Heroku
    - Scalingo

Tratados todos los temas, finaliza la reunión a las 12:30 horas del día citado, estando todos los presentes de acuerdo con las decisiones tomadas.

ACTA DE REUNIÓN 18/05/2016

Reunidos en la Facultad de Informática de San Sebastian, el día 18 de Mayo de 2016 a las 15:30 horas, con la asistencia de los participantes enumerados a continuación, la reunión sobre el proyecto SocialMusFest trató y acordó las siguientes cuestiones:

ASISTENTES:

- Jon Junguitu
- José Ángel Vadillo

ORDEN DEL DÍA:

1. Seguimiento del PFG.
2. Siguientes pasos del PFG.

ACUERDOS:

1. Jon Junguitu le presenta la aplicación realizada hasta la fecha a José Ángel Vadillo. El alumno le deja un enlace al director para que este pueda interaccionar con la aplicación. Queda pendiente por parte de José Ángel Vadillo interaccionar con la aplicación para valorarla.

2. José Ángel Vadillo le indica a Jon Junguitu que se ponga con la memoria del PFG. Indica que la memoria es el documento a presentar, y que en base a que el alumno vaya generándolo le envíe partes para poder ir revisándolo.

Tratados todos los temas, finaliza la reunión a las 16:00 horas del día citado, estando todos los presentes de acuerdo con las decisiones tomadas.

ACTA DE REUNIÓN 01/06/2016

Reunidos en la Facultad de Informática de San Sebastian, el día 01 de Junio de 2016 a las 15:30 horas, con la asistencia de los participantes enumerados a continuación, la reunión sobre el proyecto SocialMusFest trató y acordó las siguientes cuestiones:

ASISTENTES:

- Jon Junguitu
- José Ángel Vadillo

ORDEN DEL DÍA:

1. Seguimiento del PFG.
2. Cambios en la memoria.

ACUERDOS:

1. José Ángel Vadillo le indica a Jon Junguitu que ha estado revisando lo enviado hasta la fecha. Entre los cambios a realizar, lo crucial es que los diagramas de casos de uso y los diagramas de secuencia no son correctos, por lo que se tienen que cambiar. En cuanto a los casos de uso, José Ángel Vadillo indica que las funcionalidades no están bien indicadas, y que los enlaces (extends vs includes) no son correctos, por lo que se le explica a Jon Junguitu.
2. La lista total de cambios a realizar es la siguiente.
    - Punto “2.1.1. Objetivos”.
    - Punto “3. Análisis”.
    - Diagramas de Secuencia (no repetir iniciar sesión en todos los procesos, dar nombre a los actores, introducir capa de presentación).
    - Cambiar los enlaces en la imagen sobre los actores de la aplicación.
    - Meter la sección de “Antecedentes”.
    - Cambiar en el EDT “Aplicación Web” por “Producto”.
    - Actualizar y mejorar los puntos “Resumen” e “Introducción”.

Tratados todos los temas, finaliza la reunión a las 16:00 horas del día citado, estando todos los presentes de acuerdo con las decisiones tomadas.

ACTA DE REUNIÓN 14/06/2016

Reunidos en la Facultad de Informática de San Sebastian, el día 14 de Junio de 2016 a las 15:30 horas, con la asistencia de los participantes enumerados a continuación, la reunión sobre el proyecto SocialMusFest trató y acordó las siguientes cuestiones:

ASISTENTES:

- Jon Junguitu
- José Ángel Vadillo

ORDEN DEL DÍA:

1. Seguimiento del PFG.
2. Cambios en la memoria.

ACUERDOS:

1. José Ángel Vadillo indica que tras la primera revisión global de la memoria, ahora ha corregido los cuatro primeros capítulos. El resto de capítulos los revisa los siguientes días y por correo avisa a Jon Junguitu de los detalles a modificar.
2. Dentro de la corrección de los primeros capítulos, estos son los detalles indicados por José Ángel Vadillo.
    - Muchos coloquialismos.
    - Introducir el punto de “Análisis de Requisitos” en el EDT.
    - Explicar los puntos que solo tienen una imagen.
    - En el punto sobre el “Análisis de Requisitos”, referirse a la clasificación como puntos funcionales y no funcionales.
    - En el diagrama de clases, no esta bien la clase “Festivales”. Se puede o eliminar o poner “Eventos”.
    - Cambios en los diagramas de secuencia. Se proporciona un ejemplo en papel para el diagrama de de “Iniciar Sesión”.

Tratados todos los temas, finaliza la reunión a las 16:00 horas del día citado, estando todos los presentes de acuerdo con las decisiones tomadas.

9.3. MANUAL DE METEOR

A través de este manual se va a mostrar una forma sencilla de crear una aplicación web completa y multiusuario utilizando el framework Meteor y MongoDB. Se van a implementar la autenticación de usuarios, algunas características de seguridad, templates reactivas y el uso del enrutamiento. También se llevarán a cabo operaciones clave para las bases de datos (inserción, eliminación y modificación).

MOTIVACIÓN

Meteor es el framework de desarrollo que se va a utilizar para desarrollar la aplicación. Esta elección se debe a los siguientes motivos:

1. Las aplicaciones Meteor están escritas en JavaScript, CSS y HTML, así que no hay que aprender ningún lenguaje nuevo.
2. Proporciona un entorno de desarrollo fácil de instalar que incluye un servidor web y un servidor de base de datos.
3. Incluye un sistema de empaquetado muy sencillo de utilizar.
4. Proporciona fuentes de datos reactivas, con un ordenado modelo de datos distribuido, lo que significa que la aplicación dará una sensación de rapidez para múltiples usuarios.
5. Meteor proporciona servidores para poder alojar las aplicaciones.
6. Meteor incluye la capacidad de generar aplicaciones nativas de iOS y Android.
7. Implementa un modelo isomórfico, lo que significa que tanto el cliente como el servidor utilizan el mismo código base.

PRIMEROS PASOS

INSTALACIÓN Y CREACIÓN

Iniciar una aplicación con Meteor en local es muy sencillo. Los pasos a seguir que se muestran en este manual son para sistemas Linux, pero Meteor también puede usarse para otros sistemas operativos. Los comandos a introducir son mediante la terminal del equipo en el que se trabaja.

INSTALAR METEOR (LINUX): *curl https://install.meteor.com/ | sh*

CREAR UNA APLICACIÓN SENCILLA: *meteor create image_share* (se crean tres ficheros, uno HTML otro CSS y otro JavaScript)

INICIALIZAR APLICACIÓN: *meteor* (dentro de la carpeta donde están los ficheros creados con el comando anterior)

EDITAR UN TEMPLATE

Meteor utiliza los “Spacebars” para trabajar los templates. Esos elementos se basan en los “Handlebars”, y cualquier cambio que se produzca en el código afecta automáticamente en la aplicación, sin tener que refrescar.

ENVIAR DATOS A LOS TEMPLATES UTILIZANDO HELPERS

Tal y como está creada la aplicación al principio, dentro del fichero JavaScript se encuentra el modo en el que se divide el código del servidor y del cliente. La estructura es la siguiente:

```
if (Meteor.isClient) {

}
if (Meteor.isServer) {
}
```

Los “helpers” proporcionan datos a los templates. En el fichero JavaScript se puede crear el siguiente ejemplo, donde se trabaja con una variable que describe una imagen:

```
if (Meteor.isClient) {
        var img_data = {
                img_src: "kRSbkVbH.jpg",
                img_alt: "My image"
        }
        Template.images.helpers(img_data);

}
```

Una vez creado en la parte JavaScript, desde el template de HTML se puede llamar al mismo:

```
<template name="images">
  <img src="{{img_src}}" alt="{{img_alt}}"/>
</template>
```

Como apunte, también se puede trabajar con arrays:

```
if (Meteor.isClient) {
	var img_data = [
	{
		img_src: "kRSbkVbH.jpg",
		img_alt: "My image"
	},
	{
		img_src: "imagen_corporativa.jpg",
		img_alt: "Corporative image"
	},

	{
		img_src: "imagen-leon-fotografia.jpg",
		img_alt: "Lion image"
	},

	];

	Template.images.helpers({images: img_data});

}
```

En este caso, en HTML se llama de la siguiente forma:

```
<template name="images">
	{{#each images}}
		<img src="{{img_src}}" alt="{{img_alt}}"/>
	{{/each}}
</template>
```

CAMBIAR ESTILOS

Meteor proporciona paquetes adicionales de funcionalidad. Para poder ver todos esos paquetes se utiliza el siguiente comando:

```
meteor search .
```

Con esto, para añadir un paquete a un proyecto se utiliza el siguiente comando en el que, para este caso, se añade un paquete de estilos de Meteor:

```
meteor add twbs:bootstrap
```

Meteor tiene la opción con estos paquetes de poder adaptar el diseño a las exigencias del desarrollador:

```
<div class="col-xs-12 col-md-3">
```

RESPONDER A ACCIONES DEL USUARIO

Un acción sobre un elemento HTML es, por ejemplo, el siguiente onclick():

```
<img onclick="alert('hello!')" src="{{img_src}}" alt="{{img_alt}}"/>
```

Para gestionar acciones en JavaScript, dentro del HTML se tiene que crear una clase dentro del propio elemento al que afecta la acción:

```
<img class="js-image" src="{{img_src}}" alt="{{img_alt}}"/>
```

El evento es creado dentro del fichero JavaScript:

```
Template.images.events({
        'click .js-image': function(event){
                alert('hello!!');
}
```

La forma de gestionar esas acciones se adecuan a las necesidades del momento. Por ejemplo, para cambiar la anchura de la imagen, event.target proporciona acceso a la etiqueta “img”, y el evento quedaría tal que así:

```
Template.images.events({
        'click .js-image': function(event){
                $(event.target).css("width", "50px");
        }
});
```

BASE DE DATOS

En este punto se trabaja con la creación de las Mongo Collections y la utilización las operaciones de inserción y búsqueda de Mongo. Además, también se trabaja el control de un modelo Bootstrap de Meteor.

MODELO DE DATOS DISTRIBUIDOS DE METEOR

Mongo es la base de datos, donde se almacenan los datos en colecciones. Como novedad, Meteor tiene almacenados datos locales en el navegador (y, como en otros casos, tiene en el servidor todos los datos), lo que ayuda a que los usuarios puedan observar los cambios directamente.

CREAR UNA COLECCIÓN

En el archivo JavaScript visto hasta ahora, todo lo que este fuera de la sentencia “if” se ejecutará tanto en la parte cliente como servidor. Las colecciones pueden ser creadas a través de la función de Mongo para ello, como se muestra a continuación:

```
Images = new Mongo.Collection("images");
console.log(Images.find().count());
```

Si se está ejecutando en el servidor, se pueden crean los datos a inicializar:

```
if(Meteor.isServer){
	Meteor.startup(function(){
		if(Images.find().count() == 0){
			Images.insert(
				{
					img_src: "kRSbkVbH.jpg",
					img_alt: "My image"
				},
			);
		}//end of if have no images
	});
}
```

ELIMINAR DATOS DE UNA COLECCIÓN Y OTRAS SENTENCIAS

Una sentencia conocida para recorrer arrays es “for”:

```
for(var i=1; i<23; i++){
	Images.insert(
	{
		img_src: "img_"+i+".jpg",
		img_alt: "image "+i
	},
	);
}
```

Para mostrar los datos, existe la siguiente sentencia:

```
console.log("startup.js says: "+Images.find().count());
```

En cuanto a la eliminación de elementos de la colección, por ejemplo se puede asignar esa acción a un botón:

```
<button class="js-del-image btn btn-warning">delete</button>
```

En el archivo JavaScript se especifica la acción a realizar:

```
Template.images.events({
	'click .js-image': function(event){
		$(event.target).css("width", "50px");
	},
	'click .js-del-image': function(event){
		var image_id = this._id;
		console.log(image_id);
		$("#"+image_id).hide('slow', function(){
			Images.remove({"_id":image_id});
		})
	}
});
```

En cuanto a aspectos de la sentencia anterior:

- ‘this’: se refiere a los datos del template al que hace referencia (una imagen).
- ‘_id’: se refiere a un único identificador para un ítem en la colección Mongo.
- ‘{_id: image_id}”: es un filtro de Mongo.

MODIFICAR DATOS DE UNA COLECCIÓN

La última opción básica es la de modificar:

```
Images.update({_id:image:id},
	{$set: {rating:rating}});
```

El primer parámetro hace referencia al elemento que se quiere modificar (utilizando el identificador id), y los siguiente parámetros son los que se modifican. Para mejorar la interacción, se pueden ordenar las imágenes en base a las calificaciones que se les van dando:

```
Template.images.helpers({images:
	Images.find({}, {sort:{rating:-1}})}
);
```

AUTENTICACIÓN

En este punto se analiza la forma de introducir la autenticación de usuarios a una aplicación web y a aprender a usar filtros de Mongo.

USAR LA AUTENTICACIÓN CON METEOR

Para la autenticación de usuarios, se tiene que añadir un nuevo paquete en el proyecto:

meteor add accounts-ui

En este caso hay algunos elementos muy interesantes a utilizar en el archivo HTML, como por ejemplo:

{{> loginButtons}}

Así hay una forma de registrarse en la aplicación, y entre las opciones de la misma hay una que es para recuperar la contraseña olvidada. Esta envía un email al email que se mencione. Se ejecuta cuando la aplicación está en un servidor real (no localmente). La contraseña se reinicia funcionalmente como viene en el paquete.

ORDENAR EL DISEÑO CON UN NAVBAR

Por ejemplo, en el caso de que la imagen ocupe mucho espacio para poder mostrar texto en la parte inferior, en el CSS se puede poner una altura máxima:

```
.thumbnail-img{
	max-height:200px;
}
```

Para tener un estilo personalizado en el inicio del sitio, se puede poner un estilo de bootstrap en el archivo HTML:

```
<nav class="navbar navbar-default navbar-fixed-top">
	<div class="container">
		{{> loginButtons}}
	</div>
</nav>
```

El navbar obtiene todo el espacio superior de la página, y en el CSS, para poder mostrar el texto del titulo, se puede añadir:

```
body{
	padding-top:60px;
}
```

ACCEDIENDO A INFORMACION DE USUARIO

Para personalizar el sitio, se pueden usar algunos datos del usuario. Por ejemplo, para dar la bienvenida al usuario se puede poner en el titulo:

```
<h1>Welcome to image share {{username}}!</h1>
```

Se necesita un “helper template” llamado ‘username’. En el archivo JavaScript, poner:

```
Template.body.helpers({username:function(){
	if (Meteor.user()){
		return Meteor.user().emails[0].address;
	}
	else {
		return "anonymous internet user";
	}
}
});
```

El “helper template” se ejecuta dos veces; primero Meteor ejecuta el template tan pronto como puede, pero a esas alturas aún no se ha creado la variable “username”. Luego el template es reactivo y renderizado, así que los datos van cambiando dinámicamente.

Meteor.user() es una fuente reactiva de datos. Tiene toda la información sobre el inicio de sesión para los usuarios. “currentUser” es un “template helper” incorporado accesible en cualquier template.

CUSTOMIZANDO LOS FORMULARIOS PARA EL REGISTRO DE USUARIOS

El formulario para el registro en la aplicación puede ser personalizado. Con el paquete instalado previamente, se realiza esta acción en el fichero JavaScript:

```
Accounts.ui.config({
	passwordSignupFields: "USERNAME_AND_EMAIL"
});
```

Esto crea un formulario que pregunta por el “username”, “email” y “password”. Ahora se puede trabajar con esta información en JavaScript:

```
Template.body.helpers({username:function(){
	if (Meteor.user()){
		return Meteor.user().username;
		 //return Meteor.user().emails[0].address;
	}
	else {
		return "anonymous internet user";
```

```
        }
  }
  });
```

ASIGNAR DATOS

```
Template.image_add_form.events({
        'submit .js-add-image':function(event){
        var img_src, img_alt;

        img_src = event.target.img_src.value;
        img_alt = event.target.img_alt.value;
        console.log("src: "+img_src+" alt:"+img_alt);

        if (Meteor.user()){
                Images.insert({
                        img_src:img_src,
                        img_alt:img_alt,
                        createdOn:new Date(),
                        createdBy:Meteor.user()._id
                });
        }

        $("#image_add_form").modal('hide');
        return false;
}
});
getUser:function(user_id){
        var user = Meteor.users.findOne({_id:user_id});
        if (user){
                return user.username;
        }
        else {
                return "anon";

}
```

En el HTML:

```
<p>
        User: {{getUser createdBy}}
</p>
```

FILTRANDO POR PARÁMETROS

Para acceder a la información de los usuarios, se tiene que hacer una referencia al mismo. En HTML:

```
<p>
        User:
          <a href="#" class="js-set-image-filter">{{getUser
createdBy}}</a>
```

```
</p>
```

Para tener un evento aquí, se puede crear en el archivo JavaScript:

```
'click .js-set-image-filter':function(event){
        Session.set("userFilter", this.createdBy);
}
```

Aquí la aplicación rememora que el usuario esta registrado. ‘this’ es el dato para el template donde ocurre el evento. Para filtrar sólo las imágenes del usuario seleccionado, en el template se cambia el código y se pone el siguiente en el archivo JavaScript:

```
Template.images.helpers({
        images:function(){
                if (Session.get("userFilter")){// they set a filter!
                        return
Images.find({createdBy:Session.get("userFilter")}, {sort:{createdOn: -1,
rating:-1}});
                }
                else {
                        return Images.find({}, {sort:{createdOn: -1, rating:-1});
                }
        },
}
```

El filtro de Mongo, en este caso, es:

```
createdBy:Session.get("userFilter")}.
```

ELIMINANDO EL FITRADO

Para eliminar el filtrado creado, se añade al HTML:

```
<h2>
        {{#if filtering_images}}
                Showing images by user {{getFilterUser}}. <a href="#"
class="js-unset-image-filter">Show all images</a>
        {{/if}}
</h2>
```

Pero esto no funciona ya que necesita una función template en el archivo JavaScript:

```
filtering_images:function(){
        if (Session.get("userFilter")){// they set a filter!
                return true;
        }
        else {
                return false;
```

```
        }
},

getFilterUser:function(){
        if (Session.get("userFilter")){// they set a filter!
                var user = Meteor.users.findOne(
                        {_id:Session.get("userFilter")});
                return user.username;
        }
        else {
                return false;
        }
},
```

La variable de la sesión es usado como filtro. En la página aparece un enlace para eliminar el filtro. En el JavaScript, se crea el evento para ello:

```
'click .js-unset-image-filter':function(event){
        Session.set("userFilter", undefined);
},
```

SCROLL INFINITO

Meteor tiene eventos sobre scroll. En el archivo JavaScript, en la parte cliente, colocar:

```
Session.set("imageLimit", 8);

lastScrollTop = 0;
$(window).scroll(function(event){
        // test if we are near the bottom of the window
        if($(window).scrollTop() + $(window).height() > $(document).height()
- 100) {
                // where are we in the page?
                var scrollTop = $(this).scrollTop();
                // test if we are going down
                if (scrollTop > lastScrollTop){
                        // yes we are heading down...
                        Session.set("imageLimit", Session.get("imageLimit") +
4);
                }
                lastScrollTop = scrollTop;
        }
})
```

Primero se limita el número de imágenes obtenidos. Para empezar, se tienen 8 y cuando se llega al final de la página se añaden 4 más.

```
Template.images.helpers({
        images:function(){
                if (Session.get("userFilter")){// they set a filter!
```

```
                return
Images.find({createdBy:Session.get("userFilter")}, {sort:{createdOn: -1,
rating:-1}});
            }
            else {
                return Images.find({}, {sort:{createdOn: -1, rating:-1},
limit:Session.get("imageLimit")});
            }
        },
```

ACCESIBILIDAD

En este punto se muestra la forma de realizar un testeo básico de seguridad en la aplicación y cómo implementar características básicas sobre datos.

ACCESIBLES EN APLICACIONES METEOR

Se pueden hacer las aplicaciones accesibles para otras personas (no sólo localmente). Para tener la aplicación en el servidor de Meteor se utiliza el comando:

meteor deploy myk_image.meteor.com

Ahora se tiene una nueva dirección de la aplicación, accesible para cualquier persona que quiera acceder. Para eliminar la URL, utilizar:

meteor deploy --delete myk_image.meteor.com

COMO ORGANIZAR EL CODIGO

El fichero público es utilizado para los recursos estáticos. Además de este, otros ficheros especiales son:

- cliente: todo el código de la parte cliente. Los archivos CSS y HTML van en la parte cliente.
- servidor: todo el código de la parte servidor.
- lib: todo lo que está dentro se ejecuta antes que nada. Aquí hay información sobre las colecciones.

HACER EL SITIO MAS SEGURO

En este punto, se elimina un paquete de Meteor, que hace la aplicación mas fácil de desarrollar porque no se tienen que buscar posibles puertas trasera:

Meteor remove insecure

Ahora Meteor cierra todas las puertas, y es el desarrollador quien tiene que abrir esas puertas. Esto se gestiona en el servidor:

```
// set up security on Images collection
Images.allow({

	// we need to be able to update images for ratings.
	update:function(userId, doc){
		console.log("testing security on image update");
		if (Meteor.user()){// they are logged in
			return true;
		} else {// user not logged in - do not let them update  (rate) the image.
			return false;
		}
	},

	insert:function(userId, doc){
		console.log("testing security on image insert");
		if (Meteor.user()){// they are logged in
			if (userId != doc.createdBy){// the user is messing about
				return false;
			}
			else {// the user is logged in, the image has the correct user id
				return true;
			}
		}
		else {// user not logged in
			return false;
		}
	},
	remove:function(userId, doc){
		return true;
	}
```

True permite realizar la acción asociada, y false lo deniega. Así que, aquí se comprueba que la persona que quiera insertar sea la misma que la del creador de la imagen.

ENRUTANDO CON IRON:ROUTER

El enrutamiento permite controlar qué partes aparecen en determinados momentos. Primero se añade el paquete iron:router:

meteor add iron:router

En el archivo HTML, se crea un template para cada componente del "body". En el archivo JavaScript principal se controla el enrutamiento de páginas:

```
Router.route('/', function () {
	this.render('welcome');
});

Router.route('/images', function () {
	this.render('navbar');
	this.render('images');
});
```

Se pueden colocar todos los templates que se deseen como se desee en las rutas. Además, se pueden enviar algunas opciones de configuración en el enrutamiento del sistema:

```
Router.configure({
	layoutTemplate: 'ApplicationLayout'
});
```

En el archivo HTML se crea un template para ello:

```
<template name="ApplicationLayout">
	{{> yield "navbar"}}
	{{> yield "main"}}
</template>
```

"Yield" es una instrucción de enrutamiento que indica que hay un punto en el template "layout" para el cual pueda renderizar sub-templates. El enrutamiento seria tal que así:

```
Router.route('/', function () {
	this.render('welcome', {
		to:"main"
	});
});

Router.route('/images', function () {
	this.render('navbar', {
		to:"navbar"
	});
	this.render('images', {
		to:"main"
	});
});
```

```
Router.route('/image/:_id', function () {
	this.render('navbar', {
		to:"navbar"
	});
	this.render('image', {
		to:"main",
		data:function(){
			return Images.findOne({_id:this.params._id});
		}
	});
```