UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Arquitectura ágil web en la red social para buceadores Scube (ISTI_107/1314)

Víctor Gil Molina

Tutor: José Antonio Clavijo Blázquez

Ponente: Francisco Sáiz

Junio 2014

Resumen

La penetración en redes sociales ha alcanzado su madurez. Según IAB Spain 8 de cada 10 internautas, de entre 18 y 55 años utilizan redes sociales. Se ha producido una subida en el número de usuarios, aunque su uso se ha mantenido. El uso de redes sociales temáticas, como Instagram o Flirk, ha crecido rápidamente a pesar de su reciente poco tiempo de vida en el mercado.

Los ordenadores han sido sustituidos por los *smartphones* y las *tablets*, lo que ha ayudado al usuario a acceder a las redes sociales desde cualquier parte. Su utilización se ha convertido en algo cotidiano en la sociedad, dando paso a que usuarios con los mismos gustos se reúnan alrededor de redes sociales temáticas.

El objetivo de nuestro proyecto es desarrollar una red social para personas aficionadas al buceo. También los centros de buceo y otros negocios relacionados con esta práctica formarán parte de la red social. Su finalidad es que los usuarios puedan relacionarse entre ellos y sobretodo que puedan ver la actividad de sus amigos. Por su parte, los negocios podrán publicitarse y actuar como un usuario más de la red.

A lo largo de este documento se hace un repaso por todas las fases del proyecto. Primero se verán las tecnologías utilizadas, a continuación el análisis, después el diseño, el desarrollo, las pruebas y por último las conclusiones. Al finalizar la lectura de este documento, se tendrá una idea clara de la arquitectura completa del sistema y su implementación.

Palabras Claves: Redes sociales, redes sociales temáticas, buceo.

Summary

The penetration of social networking has reached maturity. According to IAB Spain, 8 out of 10 internet users, between 18 and 55 use social networks. There has been a rise in the number of users, although its use has remained the same. The use of thematic social networks, as Instagram or Flirk, has grown quickly despite of his recent short life on the market.

Computers have been replaced by smartphones and tablets, which has helped the user to get access to social networks from anywhere. Its use has become a part of normal everyday life, leading to users with the same interests gather around thematic social networks.

The goal of our proyect is to develop a social network for people who like diving. Also dive centers and other business related to this activity will be part of the social network. Its purpose is to allow the users to interact with each other and above all they can see the activity of his friends. On the other hand, the companies can advertise and act as a regular network user.

The document 's scope includes all the phases in the project. First you will see the technologies used, then the analysis, next the design, the development, the testing and finally the conclusions. After reading this document you will have a clear idea of the entire system architecture and its implementation.

Key Words: Social networks, thematic social networks, dive.

Índice

Índice de Figuras

Índice de Tablas

Glosario

AngularJS: framework JavaScript de código abierto que está patrocinado y mantenido por Google.

API (*Application Programming Interface*): es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software.

Back-End: es la parte de nuestro sistema que procesara las peticiones provenientes de la interfaz.

BootStrap: es un framework JavaScript UI y CSS de código abierto creado por dos de los mayores desarrolladores de Twitter.

CSS (*Cascading Style Sheets*): es un lenguaje que describe la presentación de los documentos estructurados en hojas de estilo.

DOM (*Document Objet Model*): estructura de objetos que genera el navegador cuando se carga un documento.

Font-End: es la parte del sistema que interactúa directamente con el usuario.

Framework: es una estructura conceptual y tecnológica de soporte definido que puede servir de base para la organización y desarrollo de software.

HTML (*HyperText Markup Language*): es un lenguaje que utilizado para establecer la estructura y el contenido de un sitio web.

JavaScript: es un lenguaje de programación orientado a objetos, se utiliza principalmente en desarrollo web para el lado del cliente.

JSON (*JavaScript Object Notation*): es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

JDBC (*Java Database Connectivity*): es un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación.

MVC (*Modelo Vista Controlador*): es un patrón de arquitectura de las aplicaciones software que separa la lógica de negocio de la interfaz de usuario.

MVCC (*Multi Version Concurrency Control*): es una técnica de concurrencia en donde ninguna tarea o hilo es bloqueado mientras se realiza una operación en la tabla, porque el otro hilo usa su propia copia (versión) del objeto dentro de una transacción.

MYSQL: es un sistema de gestión de bases de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

Plugin: es un complemento que se relaciona con una aplicación para aportarla una función nueva y generalmente muy específica.

PostgreSql: es un sistema de gestión de bases de datos objeto-relacional y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado.

SEO (*Search Engine Optimization*): posicionamiento orgánico en español, es el conjunto de técnicas utilizadas para aumentar el tráfico de calidad hacia un sitio web mediante la mejora del posicionamiento de una web en las páginas de resultados de un motor de búsqueda.

Servlet: son módulos escritos en Java que se utilizan en un servidor para extender sus capacidades de respuesta a los clientes al utilizar las potencialidades de Java.

Single Page Application: es una aplicación ejecutada desde un navegador que no necesita recargar la página durante su uso mejorando la usabilidad y fluidez de la aplicación

1 INTRODUCCIÓN

1.1 MOTIVACIÓN

Debido a la expansión de Internet a los dispositivos móviles, el uso de las redes sociales ha tenido un aumento considerable. Gracias a ello también han aparecido nuevas redes sociales que acercan a usuarios con los mismos gustos y aficiones. Por eso es el momento idóneo de encontrar un grupo de usuarios con los mismos intereses y juntarlos en una sola plataforma.

La motivación de este trabajo viene determinada por la necesidad de una red social que reúna tanto a usuarios aficionados al buceo, como a la industria relacionada. Actualmente existen otras redes sociales enfocadas este mundo, sin embargo, en muchas de ellas los negocios relacionados con el buceo no tienen un lugar donde conseguir nuevos clientes.

Este proyecto parte de un prototipo realizado durante las prácticas en empresa en Delonia, durante el primer cuatrimestre. Durante ese periodo se consiguió un pequeño conjunto de requisitos bien definidos con el que comenzar a trabajar a base de maquetas y pruebas. El desarrollo de este proyecto se ha realizado de nuevo en Delonia bajo una beca durante los últimos meses para presentarlo como proyecto fin de grado.

1.2 OBJETIVOS DEL TFG

El objetivo del proyecto es el diseño, desarrollo y documentación de una arquitectura ligera de referencia para el desarrollo ágil de proyectos web Single-Page Applications (SPA) utilizando Angular como framework JavaScript, BootStrap como framework CSS, comunicaciones con el servidor basadas en JSON, así como su aplicación en el desarrollo de la aplicación Scube (de tipo red social) que Delonia tiene en cartera y ha superado la fase inicial de análisis y prototipado.

Scube, la red social que cubre este proyecto, tiene como objetivo poner en contacto a buceadores entre ellos, compartir mensajes y fotografías, así como llevar el registro de las inmersiones que ha realizado. También pone en contacto a los buceadores con centros de buceo, agencias de viaje, tiendas de equipamiento, etc., permitiendo a éstos ofrecer servicios a los buceadores, tales como cursos, viajes, certificación de inmersiones, etc.

1.3 ESTRUCTURA DE LA MEMORIA

La estructura del proyecto sigue las líneas marcadas para el desarrollo del TFG. Se descompone en los siguientes capítulos:

- **Estado del arte:** En este capítulo se analizará las principales tecnologías utilizadas para la desarrollo de Scube.
- **Análisis:** A lo largo de este capítulo se realizará una definición del proyecto y se enumerarán los requisitos funcionales y no funcionales que se extrajeron del prototipo. En este capítulo también se verá el ciclo de vida elegido.
- **Diseño:** Después del análisis tenemos el diseño, capitulo en el que se mostrará un esquema de la arquitectura del sistema. También se verá un diagrama de dominio. Por último se explicará cómo se ha realizado el diseño del front-end.
- **Desarrollo:** En este capítulo se aborda la implementación del proyecto, desde la estructura hasta las librerías adicionales utilizadas. También se dedicará especial atención al desarrollo de la interfaz y el back-end, así como a los frameworks utilizados.
- **Pruebas**: Durante la realización de ese proyecto se han realizado pruebas de compatibilidad con los navegadores y de validación. En este capítulo se expondrán los resultados de dichas pruebas.
- **Conclusiones:** En este último capítulo se realizará una valoración general del trabajo realizado.

2 ESTADO DEL ARTE

En este capítulo se va a hablar de las tecnologías empleadas para desarrollar la aplicación Scube[1]. Así se podrá tener una visión global de los componentes principales del sistema. En concreto se abarcará los temas relacionados con la estructura de la aplicación (Single Page Application), El desarrollo del Front-End (AngularJS y BootStrap) y la base de datos (PostgreSQL).

2.1 SINGLE PAGE APPLICATION

Una **SPA(Single Page Application)** es una aplicación ejecutada desde un navegador que no necesita recargar la página durante su uso, mejorando la usabilidad y fluidez de la aplicación. Normalmente una SPA se puede ver como un cliente pesado que se carga desde el servidor web. (1)

El desarrollo de Single Page Applications trae consigo las siguientes ventajas:

1. Ofrecen una separación clara entre el cliente y el servidor. (1)
2. Las SPAs actualizan únicamente las partes que necesita cambiar dentro de la página. En contraposición, las Webs tradicionales recargan toda la página en la mayoría de acciones de un usuario. (1)
3. Minimizan el tiempo de respuesta desplazando parte del procesamiento del servidor al navegador. (1)

[1] www.scubeworld.com

4. Cuando una SPA tiene que esperar al servidor puede mostrar una barra de progreso que informa al usuario del proceso de carga. Por su parte, los usuarios en las webs tradicionales tienen que adivinar cuándo la página está cargada y lista para ser usada. (1)
5. A diferencia de la mayoría de aplicaciones de escritorio, los usuarios pueden acceder a una SPA desde cualquier conexión web con un navegador decente. Hoy en día, la lista incluye *Smartphone*, *tables*, televisiones y ordenadores. (1)

Esta tecnología no es perfecta y por ello es necesario también tener en cuenta los inconvenientes que conlleva utilizarla. Algunos de ellos son:

1. La carga de la primera página puede ser lenta. Esta condición puede afectar a su posicionamiento en buscadores.
2. Si la página no está bien formada el SEO[2] se vuelve más complejo.

A la hora de desarrollar una aplicación de este tipo existen varias plataformas, las tres más populares son: Java Applets, Flash y JavaScript. De las tres, la que aporta mayores beneficios es JavaScript por varias razones.

En primer lugar el usuario puede acceder a la aplicación sin la necesidad de instalar plugins y sin preocuparse por la compatibilidad de su Sistema Operativo. Por ejemplo todos los dispositivos móviles Android y iOS cuentan con un entorno robusto de ejecución de JavaScript. Además, también trae ventajas a la hora de desarrollar el sistema ya que simplifica el trabajo al utiliza JavaScript como único lenguaje de programación para la lógica del cliente. Una aplicación en JavaScript puede ser desarrollada usando Windows, Mac OS X o Linux, y se puede desplegar no sólo en ordenadores si no también en móviles y *tablets*.

En un sistema SPA una gran parte de la aplicación web se mueve al navegador, por lo que los requerimientos del servidor son reducidos significativamente. La

Figura 1 ilustran como la lógica del negocio y la generación de HTML han sido desplazadas desde el servidor al cliente.

[2] SEO : Search Engine Optimization

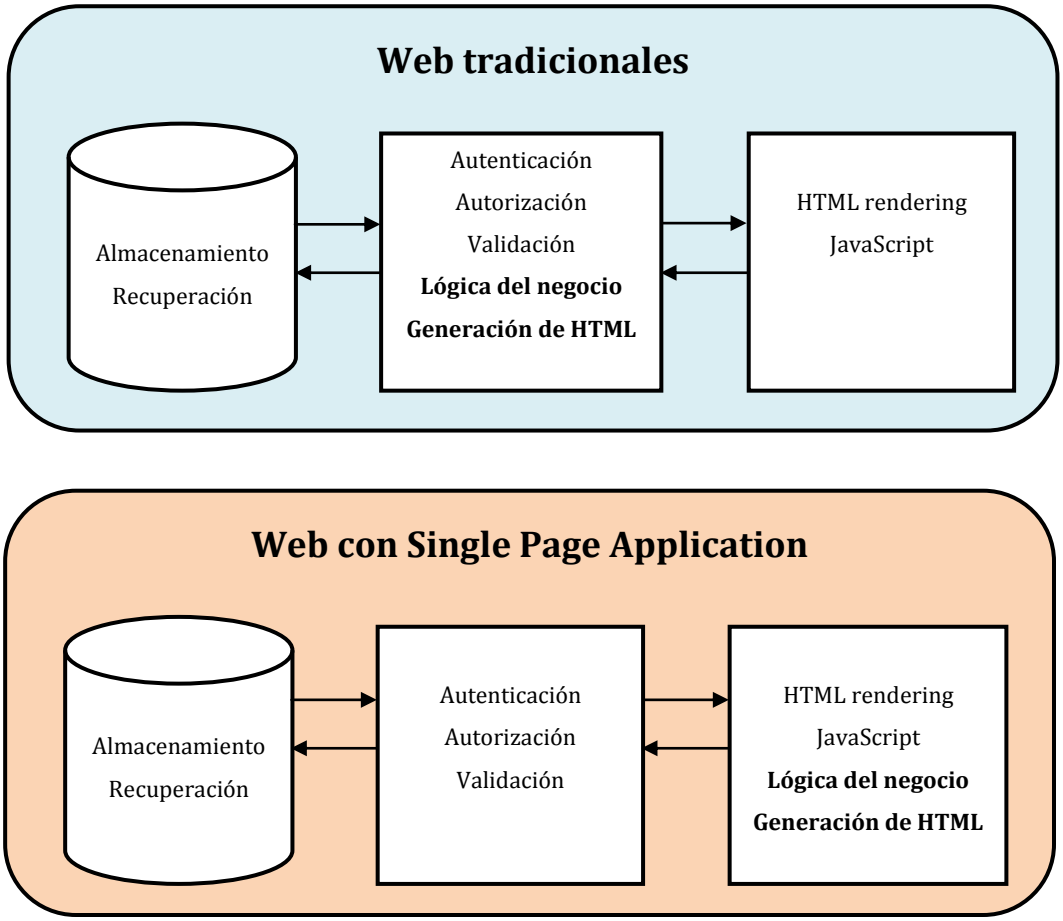

Figura 1-Arquitectura Web tradicional vs Single Page Application

2.2 FRAMEWORKS

Un framework es una estructura conceptual y tecnológica de soporte definido que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. (2)

2.2.1 ANGULAR JS

AngularJS es un framework JavaScript de código abierto que está patrocinado y mantenido por Google. Se ha utilizado en algunas de las aplicaciones web más grandes y más complejas. Las aplicaciones que utilizan AngularJS están construidas alrededor del

patrón de diseño Modelo Vista controlador (MVC), el cual hace énfasis en crear aplicaciones que sean extensibles, sostenibles, comprobable y estandarizadas. (3)

Las aplicaciones que hacen uso de AngularJS se dividen en tres partes, por un lado está la vista, que son ficheros HTML con directivas AngularJS que permiten modificar su contenido dinámicamente en función de los datos. Por otra parte están los ficheros JavaScript asociado a cada HTML que reciben el nombre de controladores. Estos ficheros contienen la lógica. En último lugar tenemos los datos que se almacenan en las propiedades del objeto. (4)

La utilización de AngularJS proporciona varias ventajas, una de ellas es la dinamización de los documentos HTML, lo que comúnmente se conoce como DHTML (*Dynamic HTML*), mediante la vinculación de elementos de nuestro documento HTML con nuestro modelo de datos. Otra de las características más destacadas que proporciona AngularJS es la de facilitar la manipulación de los elementos del DOM y sus propiedades. Esta es una tarea que por norma general es bastante tediosa y a medida que la lógica crece suele ser complicado escribir código que pueda mantenerse con facilidad. Por último cabe destacar que este framework tiene el respaldo de una gran comunidad con una documentación extensa y multitud de módulos y plugins ya implementados.

2.2.2 BOOTSTRAP

BootStrap es un framework JavaScript UI y CSS de código abierto creado por Mark Otto y Jacob Thornton, dos de los mayores desarrolladores de Twitter, para asegurarse que tanto la apariencia como las sensaciones de la interfaz de Twitter fueran consistentes y uniformes en todos sus módulos (5).

En su lanzamiento se presentó el proyecto de esta forma:

"En el pasado de Twitter, los ingenieros usaban casi cada librería con la que estaban familiarizados para cumplir los requisitos del front-end. Las inconsistencias entre las aplicaciones individuales dificultaban su escalabilidad y mantenimiento. BootStrap comenzó como una respuesta a estos desafíos y se aceleró rápidamente durante el primer HackWeek de Twitter. Al final de la HackWeek tuvimos una versión estable que los ingenieros podrían utilizar en toda la compañía"

-Mark Otto

https://dev.twitter.com/

Desde que BootStrap fue lanzado en Agosto de 2011, ha ido creciendo en popularidad rápidamente. Ha evolucionado de ser un proyecto totalmente CSS a incluir una gran cantidad de plugins JavaScipt así como iconos que van de la mano con las formas y botones. En su base permite diseños web adaptables a través de su disposición 12 en columnas, las cuáles se redimensionan en función del tamaño disponible. Uno de los puntos más importante de desarrollar con BootStrap es que puede ser personalizado totalmente para adaptarse a las necesidades de cada diseño web y está soportado en todos los navegadores populares. Todo esto además de su facilidad de uso hace de esté framework una herramienta perfecta para la creación de web con un diseño dinámico y adaptable (6).

2.3 POSTGRESQL

PostgreSQL es un sistema de gestión de base de datos objeto-relacional. Es el sistema de gestión de base de datos de código abierto más potente del mercado y sus últimas versiones no tienen nada que envidiar a las base de datos comerciales. PosgreSQL utiliza un modelo un modelo cliente/servidor con multiprocesos en vez de multihilos para garantizar la estabilidad del sistema.

Este gestor de base de datos tiene varias ventajas frente al resto de gestores presentes en el mercado, algunas de ellas son:

1. **Estabilidad y confiabilidad.** En contraste con muchos sistemas de bases de datos comerciales, es extremadamente común que compañías reporten que PostgreSQL nunca ha presentado caídas tras varios años de alta actividad. (7)
2. **Ahorros considerables de costos de operación**. PostgreSQL ha sido diseñado para tener un mantenimiento y ajuste menor que los productos de proveedores comerciales, conservando todas las características, estabilidad y rendimiento. (7)
3. **Diseñado para ambientes de alto volumen.** Utilizando una estrategia de almacenamiento de datos en varias filas denominado MVCC[3] consigue una mejor respuesta en grandes volúmenes. (7)
4. **Extensible.** El código fuente de PosgreSQL está disponible y es posible personalizarlo o ampliarlo con un mínimo de esfuerzo y sin costes adicionales.

[3]MVCC(MultiversionConcurrencyContro) : modelo utilizado para asegurar la consistencia de los datos. Mientras se consulta la base de datos, cada transacción ve una instantánea de los datos sin tener en cuenta el estado actual de dichos datos. Esto protege a la transacción de ver datos inconsistentes.

5. **Soporte para múltiples lenguajes.** Permite escribir procedimientos almacenados y funciones en varios lenguajes de programación. (8)

Como se ha comentado PostgreSQL tiene muchas ventajas, es un gestor muy competente para ámbitos profesionales. Sin embargo también tiene algunas limitaciones:

1. **Consume más recursos que MYSQL** por lo que se necesitan mayores características de hardware para ejecutarlo. (9)
2. **No existe una ayuda obligatoria**. Al tratarse de un producto Open Source no tiene un soporte oficial pero dispone de soporte en línea y foros oficiales.

3 ANÁLISIS

En este capítulo se dará una definición del sistema. También se mostrará un listado de requisitos, funcionales y no funcionales, definidos para nuestra aplicación. Con el fin de comprender como se irá desarrollando el proyecto al final de capítulo se hablará sobre el ciclo de vida elegido.

3.1 DEFINICIÓN DEL SISTEMA

Este proyecto ha nacido con el nombre de Scube[4], y se centra en la construcción de una red social especializada en el sector del buceo. Esta red social pretende cubrir algunos de los requisitos que se encuentran actualmente en redes más populares tales como compartir fotos, mandar menajes, etc., y además añadirle la parte especializada en el buceo.

3.1.1 OBJETIVOS Y FUNCIONALIDAD

Este proyecto tiene dos objetivos principales, en primer lugar la red debe funcionar como un mecanismo de comunicación y compartición de contenido entre gente aficionada al buceo. El segundo objetivo fundamental es el de proporcionar a los centros de buceo y demás negocios del sector una plataforma para publicar ofertas y atraer nuevos clientes. Como los requisitos no estaban tan claro como los objetivos se comenzó a trabajar sobre el prototipo, realizado durante las prácticas en empresa. A partir de ahí se

[4] www.scubeworld.com

fueron realizando iteraciones, según el ciclo de vida de prototipado, para extraer el resto de requisitos.

3.2 REQUISITOS FUNCIONALES

El sistema tendrá diferentes funcionalidades dependiendo de los tres tipos de usuarios que puede soportar:

- Buceadores: Son los usuarios normales de la aplicación.
- Centros de Buceo: Estos usuarios están asociados con negocios físicos dedicados al buceo.
- Administrador: Usuario administrador de la red, su principal cometido es el de gestionar el contenido y registrar centros en el sistema.
- Otros: En un futuro el sistema podría tener también usuarios especiales para agencias de viaje, centros de alquiler de equipo, etc. aunque no se encuentra dentro del alcance del proyecto.

Requisitos comunes para Buceadores y Centros

1. **RF1:** Todos los usuarios tendrán un muro donde se podrá publicar mensajes, tanto él como el resto de sus amigos.
2. **RF2:** Envío de mensajes privados.
3. **RF3:** Se debe poder crear inmersiones[5].
4. **RF4:** Todos los usuarios tendrán un perfil con sus datos. El perfil debe contener como mínimo un nombre, apellido, nombre de usuario y fotografía.
5. **RF5:** Tanto los centro de buceo como los buceadores tendrán una galería en la que podrán subir imágenes.

Requisitos para buceadores

1. **RF6:** Lo buceadores podrán añadir certificados[6] de buceo a través de un formulario y una foto.
2. **RF7:** Podrán añadid relaciones de amigo con otros usuarios de la red.

[5]Una inmersión representa una sesión de submarinismo.

[6]Acreditación emitida por una autoridad de certificación de buceo que certifica que el buceador posee conocimiento y experiencia suficiente para el nivel exigido por la acreditación

Requisitos para los centros de buceo

1. **RF8:** Los centro de buceo además de poder crear inmersiones, podrán añadir a otros usuarios de la red como participantes.
2. **RF9:** Podrán crear platillas para las inmersiones, de manera que cuando quieran rellenar una nueva inmersión, solo necesiten introducir un pequeño número de datos como la fecha y la hora.
3. **RF10:** Los centros podrán ofertar cursos de diferentes tipos (ocio, técnicos y otros).
4. **RF11:** Los centro no sólo tendrán relaciones de amigo con otros usuarios de la red, también podrán tener instructores y administradores.
5. **RF12:** Podrán enviar invitaciones de registro por correo electrónico.

Requisitos para el Administrador del sistema

1. **RF13:** El usuario administrador podrá asumir la identidad de cualquiera de los centros de buceo registrados en la red para gestionarlo.
2. **RF14:** Podrá crear nuevos centros y enviarles invitaciones por correo electrónico.

Estos requisitos son muy generales y su implementación admite muchas variantes diferentes. Por ello se trabajaba bajo el control del cliente, que validaba cada poco tiempo si las cosas se realizaban correctamente.

3.3 REQUISITOS NO FUNCIONALES

A continuación se expondrán los requisitos no funcionales de nuestra aplicación. Estos requisitos hacen referencia a aspectos no relacionados con la funcionalidad.

1. **RNF1:** El tiempo de carga para la carga de la primera página debe ser menos a 5 segundos. El resto de páginas debe tener una respuesta menor a 3 segundos.
2. **RNF2:** Debe estar disponible entre el 100% y el 98% del tiempo.
3. **RNF3:** El sistema debe estar capacitado para permitir añadir, modificar o eliminar funcionalidad después de su construcción y puesta en marcha.
4. **RNF4:** La aplicación debe poder ejecutarse en cualquier plataforma como ordenadores, tabletas o móviles.
5. **RNF5:** El sistema debe ser de fácil uso a pesar da la gran cantidad de funcionalidades que implementará.

6. **RNF6:** El sistema debe informar en todo momento presentando mensajes que notifican al usuario si ha sucedido algún evento inesperado, como por ejemplo errores de acceso a páginas no autorizadas.
7. **RNF7:** El acceso al sistema está restringido por un paso previo de autentificación. Solo podrán ingresar al sistema las personas registradas. Cada usuario será clasificado en los tres tipos antes mencionados. Cada uno de los tipos tendrá sus roles y permisos especiales.
8. **RNF8:** El sistema debe permitir el registro de las actividades con la identificación del usuario que las realizaron.

3.4 CICLO DE VIDA

"Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software"

-IEEE 1074

Esa es la forma en que define el ciclo de vida la norma IEEE 1074, pero también se puede ver como el conjunto de fases por la que pasa el sistema que se está desarrollando, desde que nace la idea inicial hasta que el software es retirado o reemplazado. (10)

Para poder elegir el ciclo de vida que se usará para este proyecto es necesario tener en consideración los siguientes aspectos:

1. **¿Están bien definidos los requisitos?** En este proyecto se empezó a trabajar sin tener el desglose completo de los requisitos. Se partía de una base de requisitos extraídos del prototipo que debían ir creciendo y desarrollándose según las indicaciones del cliente.
2. **¿Cuál es el tiempo límite de entrega del proyecto?** Para el proyecto se fijo como fecha límite los primeros días de junio, en estas fechas debía haber una versión con la funcionalidad básica de la web. Sin embargo, ya que los requisitos no estaban definidos desde el principio, no se sabía con exactitud si se podría cumplir esta premisa.
3. **¿Cuál es la dimensión del proyecto?** De nuevo nos encontramos en una incertidumbre. El cliente no tenía claro cuál es el alcance ni la dimensión del proyecto.

A la vista de la indefinición que tenía el proyecto por parte del cliente, se opto por utilizar un desarrollo de prototipado evolutivo. Este modelo se basa en la idea de desarrollar una implementación inicial exponiéndola a los comentarios del cliente. De esta forma se van añadiendo nuevos requisitos y refinando el sistema a través de las diferentes versiones hasta que se llega a un sistema adecuado. La

Figura 2-Diagrama muestra un gráfico con las fases del desarrollo de prototipado evolutivo.

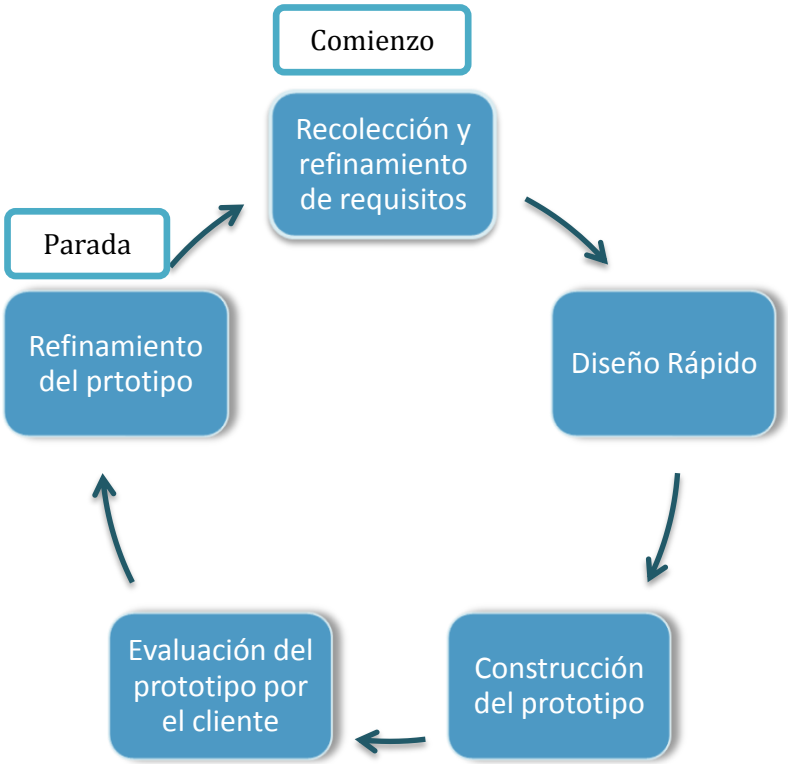

Figura 2-Diagrama de desarrollo de prototipo evolutivo

Si bien el modelo de prototipos evolutivos, fácilmente modificables y ampliables es muy usado, en muchos casos puede usarse prototipos descartables para esclarecer aquellos aspectos del sistema que no se comprenden bien. (11)

Este tipo de desarrollo nos ha permitido ir extrayendo los requisitos a medida que el proyecto avanza, esto en ocasiones ha propiciado que el tiempo desarrollo aumentara al tener que desechar y rediseñar ciertos requisitos. A pesar de ello, era uno de los que más nos ayudaban a un avance seguro del proyecto.

4 DISEÑO

En este capítulo se va a exponer el diseño del sistema. Para ello se empezará concretando la arquitectura completa del sistema, haciendo énfasis en sus tres partes principales el navegador, el servidor y la base de datos. A continuación se mostrará un diagrama de dominio que representa la estructura de los datos en la aplicación. Al final del capítulo se hablará del diseño de la parte visual de la aplicación, el Front-End.

4.1 ARQUITECTURA DEL SISTEMA

En esta sección se va a mostrar la arquitectura del sistema completo. Además se realizará una explicación detallada de cada uno de los componentes prestando especial atención a las capas de software del el servidor, con el fin de poder entender como está organizado el sistema completo.

La arquitectura de un sistema define los componentes y la interacción existente entre ellos. En la Figura 3- ARQUITECTURA del sistema se muestra una visión simplificada de la arquitectura del sistema completo en la que se pueden observar tres partes diferenciadas. Por un lado se tiene el navegador que es el cliente del sistema, el servidor y por último la base de datos.

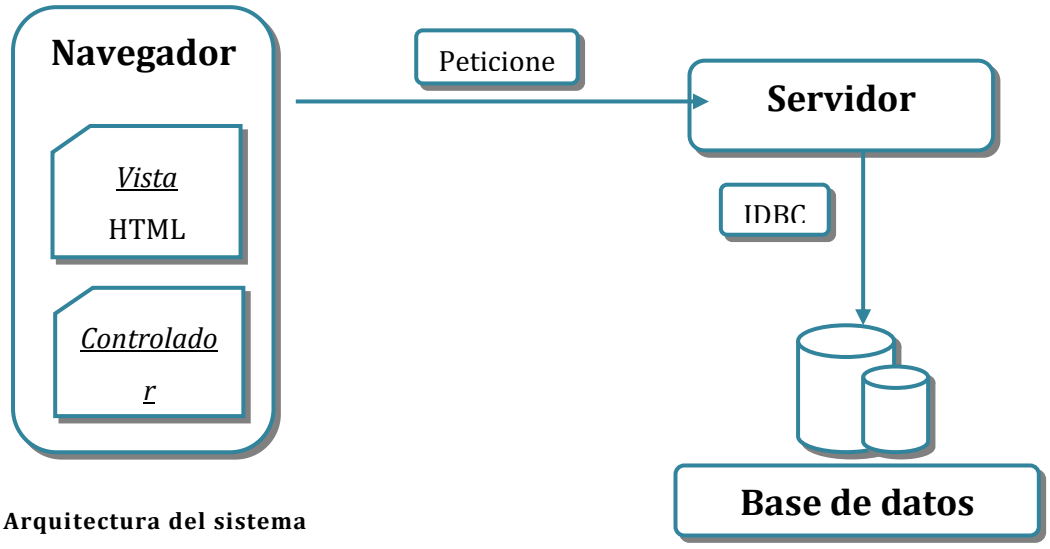

Figura 3- Arquitectura del sistema

4.1.1 NAVEGADOR

El navegador va a realizar la comunicación con el servidor. Como se muestra en la Figura 4- Peticiones del navegador al servidor, el navegador pedirá los archivos HTML y JavaScript que conforman la vista y el controlador respectivamente. Esta separación ha sido facilitada en gran parte por el framework AngularJS. Debido a la implantación del sistema Single Page Application (SPA) el navegador solo pedirá los archivos que sean necesarios para mostrar la vista solicitada. Una vez que ha cargado un archivo HTML o JavaScript éste se guarda en caché para no tener que volver a pedirlo en caso que el usuario pida de nuevo el mismo contenido.

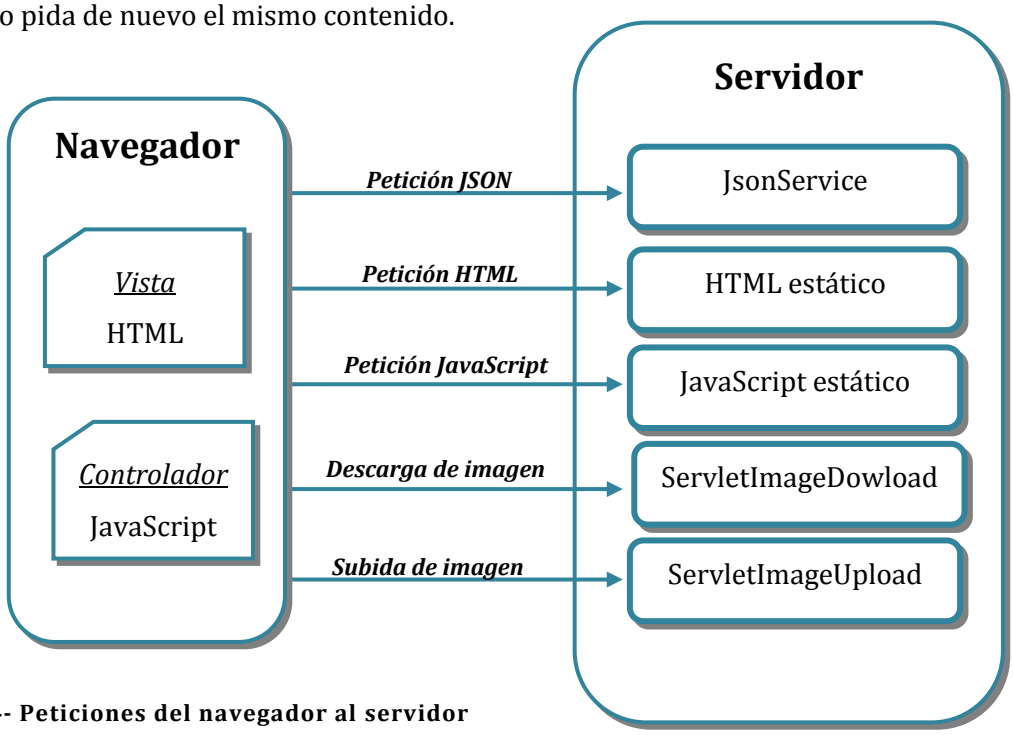

Figura 4- Peticiones del navegador al servidor

Como se ve en la Figura 4- Peticiones del navegador al servidor el navegador además de pedir la vista y controlador también realizará peticiones JSON que ejecutarán operaciones en el servidor. Por último también podrá subir imágenes y descargarlas, esta última acción se realiza a través de una URL donde se incluye como mínimo el id de la imagen en base de datos y un tipo que indica el contexto desde donde se están pidiendo.

4.1.2 SERVIDOR

Otro componente de nuestra arquitectura es el servidor. Éste se centra en responder a las peticiones del navegador, acceder a la base de datos y servir contenido, como por ejemplo lo ficheros estáticos HTML y JavaScript.

El servidor también gestiona las peticiones JSON a través de un conjunto de capas como se muestra en la Figura 5- Estructura del servidor. Cuando un navegador envía una petición JSON al servidor, éste la recibe en el módulo JsonService. En esta capa se comprueba que el usuario tiene los permisos necesarios para realizar la operación que se solicita. Si no los tiene el servidor rechaza la petición escribiendo en el log el fallo producido y redirigiendo a una página de error 403 (Privilegios insuficientes) si el usuario se había autenticado ó 401 (Necesario Login) si el usuario no se había autenticado.

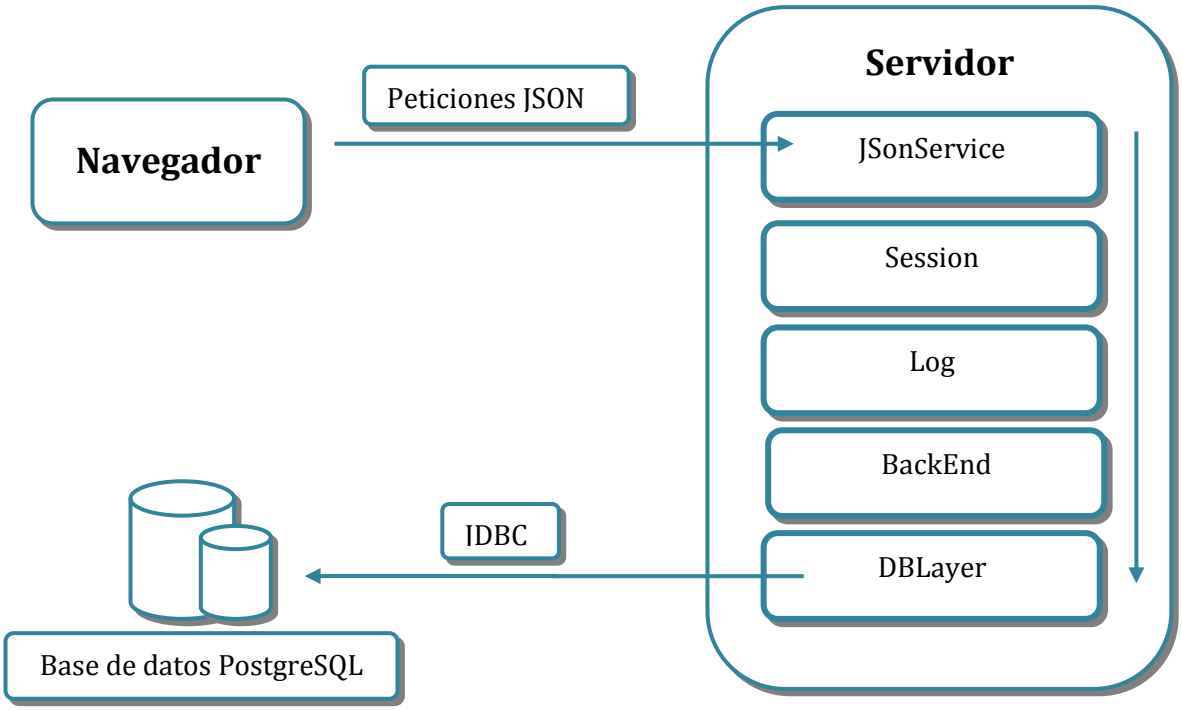

Figura 5- Estructura del servidor

En caso de que tenga permiso la petición continúa a la capa de sesión (*Session*). Esta capa se encarga de establecer la identidad del usuario en la operación, es decir, si un usuario envía una petición, en la capa sesión se marcará con el identificador único del usuario, para que de ésta forma no pueda suplantar la identidad de otro usuario de la red.

Una vez que se ha confirmado la identidad del usuario y se han comprobado sus permisos, el servidor dirige la petición hasta el BackEnd. En esta capa se encuentra la lógica de la operación si la tiene. Por ejemplo una operación que necesite cálculos o información complementaria de la base de datos para dar una respuesta tendrá su lógica en esta capa.

No todas las peticiones se detienen en el BackEnd, las operaciones simples a base de datos como por ejemplo un select o delete pasan directamente hasta la capa DBLayer. Esta última capa del servidor se encarga de acceder a la base de datos y realizar la operación solicitada. Puede ocurrir que esta capa reciba varias operaciones de base de datos desde el BackEnd como consecuencia de la ejecución de la lógica de la operación solicitada. Una vez que se ha ejecutado la operación el servidor devuelve una respuesta al navegador en formato JSON.

El servidor además del servlet JsonService que procesa las peticiones JSON contiene otros dos que se encargan de la subida y descarga de imágenes como se muestra en la Figura 4- Peticiones del navegador al servidor.

4.1.3 BASE DE DATOS

El último componente de nuestro sistema es la base de datos, es la encargada de almacenar todos los datos e información que maneja la aplicación. Está una implementada en PostgreSQL por los motivos descritos en la sección 2.3. La base de datos se conecta con la capa DBLayer del servidor a través de una conexión JDBC (*Java Data Base Connectivity*).

4.2 FRONT-END

El front-end es la parte del sistema que interactúa directamente con el usuario. La separación del sistema en front-end y back-end es un tipo de abstracción que ayuda a mantener las diferentes partes del sistema separadas. En la arquitectura de este proyecto los dos componentes que forman el front-end son los archivos HTML y JavaScript.

Como se ha comentado en la sección anterior los archivos HTML forman la vista y contienen los elementos gráficos que se muestran al usuario. Estos archivos hacen uso del framework BootStrap, que proporciona elementos gráficos HTML adaptables, y del framework AngularJS para interaccionar dinámicamente con el controlador. Los archivos JavaScript o controladores se encargan de manejar los cambios en la interfaz y de realizar las peticiones necesarias al servidor. La separación de vista y controlador nos permite cambiar una de las partes con mucha facilidad sin que la otra se vea afectada.

La información del front-end se va a cargando gradualmente. Cuando una acción de un usuario implica cargar un nuevo contenido en primer lugar se comprueba si está en cache, si no lo encuentra realiza la petición de los archivos JavaScript y HTML al servidor. Cuando el navegador los recibe solo recargará las partes de la vista que lo necesiten, esto supone una mejora frente a las web tradicionales en las que se recarga la página completa cada vez que se tiene que refrescar algún contenido.

Otra parte que cabe destaca del front-end son los diálogos emergentes, éstos al igual que el resto de páginas están separados en vista y controlador. Los diálogos se invocan desde cualquier página y se les puede pasar un parámetro de entrada. Dentro del JavaScript del diálogo puede realizar peticiones al servidor y cuando termina su cometido puede retornar un valor a la página que los invocó. Se utilizan principalmente para realizar búsqueda de usuarios y para mostrar mensajes de error, confirmación o información.

4.3 DIAGRAMA DE DOMINIO

En el diagrama de dominio que se verá a continuación se van a representar las entidades que forman los datos de Scube[7], la red social orientada a buceadores de la que trata el proyecto.

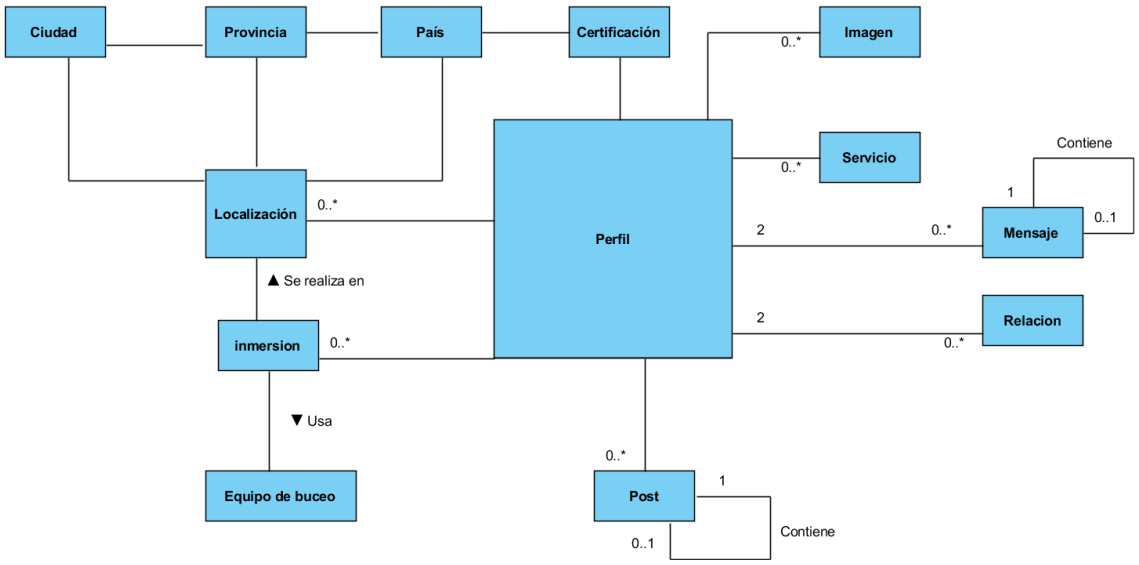

Figura 6-Diagrama de dominio de la aplicación

En la Figura 6-Diagrama de dominio de la aplicación se muestra el diagrama de dominio completo para la aplicación. A continuación se va a hacer un repaso de las entidades más importantes para comprender que representa cada una.

Clase Perfil: es la clase central de la aplicación ya que representa los usuarios. Contiene el tipo de usuario y sus datos como login, contraseña, nombre, apellidos...

Clase Localización: representa un lugar donde realizar buceo. Contiene datos como el país, ciudad, coordenadas GPS, tipo de buceo que se puede realizar, imágenes... De los tres usuarios sólo los centros pueden relacionarse directamente con esta clase. La relación de localización con centro representa las localizaciones en las que el centro realiza actividades.

Clase Inmersión: representa una sesión de submarinismo. A las inmersiones creadas por un buceador se les puede añadir información del equipo de buceo utilizado. Los centros crean las inmersiones con el objetivo de añadir a los usuarios que la

[7] www.scubeworld.com

realizaron. Contiene diferentes datos dependiendo quien la crea pero en general tiene una localización, fecha, hora de entrada y salida...

Clase Certificación: representa una acreditación emitida por una autoridad que certifica que el buceador posee conocimiento y experiencia suficiente para el nivel exigido por la acreditación. Sólo los buceadores pueden tener certificaciones.

Clase Servicio: representa la oferta de cursos y otros servicios por parte de los centros. Es una clase con la que sólo podrán tener relación los centros de buceo.

Clase Mensaje: representa los mensajes privados de la aplicación. Esta clase es común para todos los usuarios. Varios mensajes pueden tener el mismo padre, de esta forma se crean conversaciones.

Clase Relación: representa la relación entre dos usuarios, si la hubiera. Existen varios tipos de relaciones como amigo y bloqueado, para las relaciones con un centro de buceo también hay instructor y administrador.

Clase Post: representa un mensaje publicado en el sistema para que sea visualizado por otros usuarios. Esta es una clase que se relaciona con todos los tipos de usuario. Al igual que los mensajes también permite agrupación en conversaciones.

Aunque no se muestra en el diagrama, existen relaciones auxiliares de algunas clases con las imágenes. Algunas de las que tienen relación con las imágenes son por ejemplo los post, que pueden incluir imágenes en cada entrada. También las localizaciones, inmersiones y mensajes pueden tener imágenes asociadas.

Con todo lo explicado anteriormente ya se tiene una idea general de las clases en las que se compone la aplicación. A lo largo de este proyecto no se ha incluido más detalle sobre la base de datos o el modelo de clases ya que entraba en conflicto con la política de privacidad del proyecto.

4.4 CONCLUSIONES

Al terminar este capítulo ya se tiene una idea de las tres partes que forman el diseño de la arquitectura del sistema, haciendo especial atención a las capas del servidor. Además se ha visto los ficheros que contendrán el front-end y como éste interacciona con el resto de componentes. Por últimos ha visto el conjunto de datos que forman la aplicación representados en un diagrama de dominio.

5 DESARROLLO

Este capítulo abarca el desarrollo del proyecto. Para que se vaya teniendo un conocimiento incremental del desarrollo, se va a organizar en las siguientes secciones:

- Estructura del código: Organización del código en tres módulos.
- Single Page Application: Implementación de este sistema.
- Frameworks: AngularJS y BootStrap como herramientas para desarrollar el front-end.
- Seguridad: Medidas adoptadas para la protección frente a ataques maliciosos.
- Back-End: Implementación de las capas del servidor.
- APIS adicionales: Herramientas extras utilizadas para ciertas funcionalidades.

5.1 ESTRUCTURA DEL CÓDIGO

En esta sección se va a describir como se ha estructurado el código de nuestra aplicación. El proyecto se llama Scube por lo que todos sus módulos tendrán el nombre scube-nombre_del_módulo.

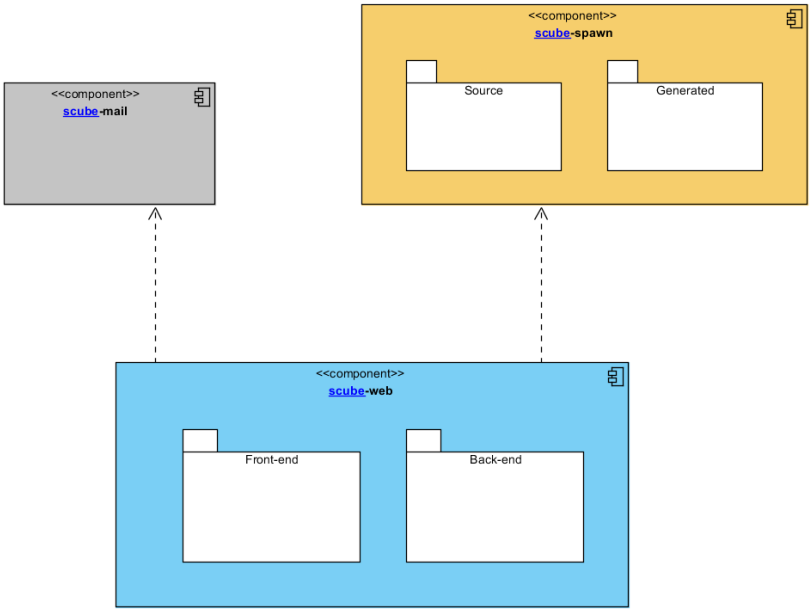

Figura 7-Estructura del proyecto

Como muestra la Figura 7-Estructura del proyecto la aplicación está divida en tres módulos diferentes scube-mail, scube-spawn y scube-web, cada uno de ellos tiene una funcionalidad específica.

5.1.1 SCUBE-MAIL

Este módulo es la parte de la aplicación encargada del envío de correos electrónicos. Contiene además de la lógica para enviar correos un archivo de configuración en el que se establece los credenciales con los que se van a enviar los emails.

Cuando un usuario intenta enviar varios emails, de invitación por ejemplo, el proceso de envío puede dejar el navegador inactivo durante algunos debido al retraso que se produce. Por ello el sistema de scube-mail va almacenando todos los emails que deben ser enviados en la base de datos. Cada cierto tiempo empieza un proceso en el que se busca en la base de datos todos los emails que están sin mandar y se envían. Esta acción la realiza el servidor por lo que los usuarios no sufren ningún retraso.

5.1.2 SCUBE-SPAWN

Este módulo genera el código necesario para utilizar la base de datos en nuestra aplicación. Es un trabajo pesado y repetitivo tener que crear las clases y las operaciones manualmente, por ello este módulo optimiza el trabajo para solo tener que definir las

entidades que hacen uso de la base de datos y sus operaciones, y con ello, generar todo el código necesario.

Este proyecto no cubre el desarrollo de éste módulo, sin embargo, se verá un ejemplo sencillo de su utilización para poder comprender en general como se han realizado las operaciones a base de datos. Para ello en primer lugar se tiene que hablar de su estructura interna. Las dos carpetas más importantes de éste módulo son source y generated.

En la carpeta source es donde se crean las clases java que contendrán las entidades y operaciones, las cuales una vez compiladas, generarán el resto de clases y archivos necesarios. Ya que la base de datos es una de las partes más sensible de la aplicación en cuanto a seguridad se refiere, vamos a utilizar un ejemplo muy simplificado para hacernos una idea de sus posibilidades.

En primer lugar vamos a crear una nueva entidad que actuará sobre la tabla de usuario. Para empezar creamos un fichero _usuario.java dentro de la carpeta source. Se ha situado un guión bajo delante del nombre para que así el módulo sea capaz de diferenciar las clases generadas de las creadas por nosotros. Un ejemplo de la clase _usuario.java podría ser el siguiente:

```
@OperationModule
@MyBatisMapper
@Doc({"Clase para la gestion de los usuarios del sistema"})
Public class _user{

	@Entity
	@MappedName("user_db")
	class _User{

		@Id
		Integer id;
		String login;
		String password;
	}

	@SelectOne(result= _User.class)
	class _SelectUserWithLogin{
		String login;
	}
}
```

Tabla 1-Código de ejemplo para la creación de una entidad y una operación.

Este código ilustra la creación de la entidad _User, que contiene los mismos atributos que la base de datos, y una operación SelectUserWithLogin que devuelve un usuario a partir del login. Cuando se compile este módulo con la clase _user.java nos generará los siguientes archivos en la carpeta generated:

- Dentro de la carpeta model generará la clase user.java que implementa serializable y contiene simplemente los atributos que se han definido para user con sus getters y setters.
- En la carpeta myBatis se generarán dos archivos UserMapper.java y UserMapper.xml.

  UserMaper.xml contiene las operaciones de base de datos en sql como se muestra a continuación.

```
<select id='selectUserWithLogin' resultType='User'>
Select
        id,
        login,
        password
from
        scube_user
where
        login = #{login,jdbcType=VARCHAR}
</select>
```

Tabla 2-Query generada para la operación SelectUserWithLogin

En el código se muestra la generación de la operación SelectUserWithLogin que se ha creado antes en _user.java. Como se puede observar esta operación recibe el parámetro login para hacer la comparación.

UserMaper.java contiene las llamadas a las operaciones de base de datos definidos en _user.java. También contiene la función executor que se utiliza para ejecutar las operaciones haciendo uso del patrón command. El código de de la función que llama a la operación SelectUserWitLogin es el siguiente.

```
@Override
public UserselectUserWithLogin(SelectUserWithLoginoperation) {
      User result=(User) getSession().selectOne(
"com.delonia.scube.myBatis.UserMapper.selectUserWithLogin" ,operation);
      returnr result;
}
```

Tabla 3- Código generado para la ejecución de la operación

- Por último en la carpeta operations se ha creado una clase llamada selectUserWitLogin.java que contiene como atributo el parámetro que recibe la operación y una función executor que llama a la operación de userMapper.java.

Este módulo es muy flexible y con él se pueden crear multitud de operaciones a través de etiquetas como @SelectMany, @SelectOne, @Update, @Delete… Además puedes crear tus propias consultas con la etiqueta @CustomSqlQuery.

5.1.3 SCUBE-WEB

Este es el módulo que contiene toda la aplicación web, desde los archivos HTML y JavaScript, que forman el front-end, hasta las capas en las que divide el servidor o back-end. A continuación se verá este módulo con un poco más de detalle para terminar de formar el esquema de la aplicación. Las principales carpetas de éste módulo son WebPages y Source. Estas carpetas separan el front-end y back-end respectivamente, ambas se encuentran en el mismo módulo para facilitar el desarrollo, sin embargo, podrían separarse sin mucha dificultad.

La carpeta WebPages contiene los archivos que forman la parte visible de la aplicación, también tiene las librerías externas utilizadas y las imágenes estáticas. Dentro de esta carpeta hay varias carpetas, una de ellas es public que contiene las páginas web accesibles sin necesidad de registrarse. Por otro lado está la carpeta private que contiene las paginas accesibles una vez se ha hecho login. La carpeta private a su vez contiene una carpeta por cada tipo de usuario y otras que albergan las páginas comunes para todos los usuarios. Para cada página de la aplicación, independientemente de la carpeta en la que se encuentre, hay un archivo HTML y otro JavaScript que forman la vista y el controlado respectivamente, ambos agrupados en la misma carpeta.

La otra carpeta fundamental de este módulo es source, contiene principalmente las capas del servidor explicadas en la sección 5.6, la lógica de las operaciones, la comprobación de permisos para cada operación y los servlets de subida y descarga de imágenes.

5.2 SINGLE PAGE APPLICATION

Esta aplicación está desarrollada basándose en del paradigma Single Page Application(SPA). La idea detrás de esto es hacer peticiones únicamente del contenido nuevo que se desea mostrar sin recargar el resto de la página.

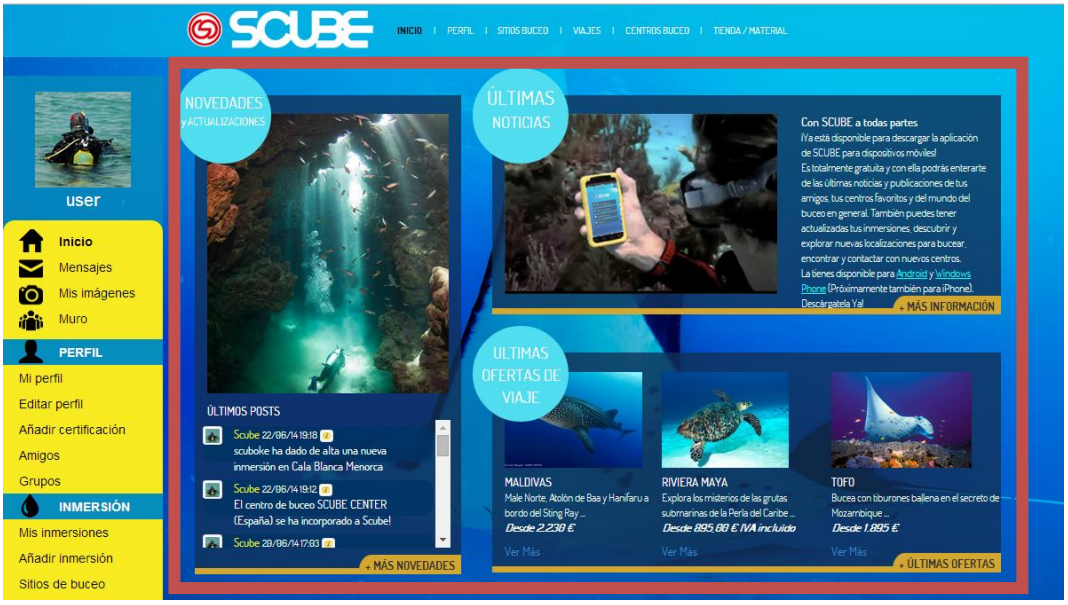

Figura 8-Página de inicio para buceadores

En la Figura 8-Página de inicio para buceadores se puede ver la página principal tras hacer login en la aplicación. Tanto el menú como la cabecera se van a mantener fijos para todas las páginas, sin embargo, el contenido marcado con un cuadro rojo, irá cambiando dependiendo de la sección en la que se encuentre el usuario.

Cuando el usuario cambia a otra sección del menú, por ejemplo mi perfil, internamente ocurre lo que se ilustra en la Figura 9, se supone que ninguno de los elementos se encuentra en la caché.

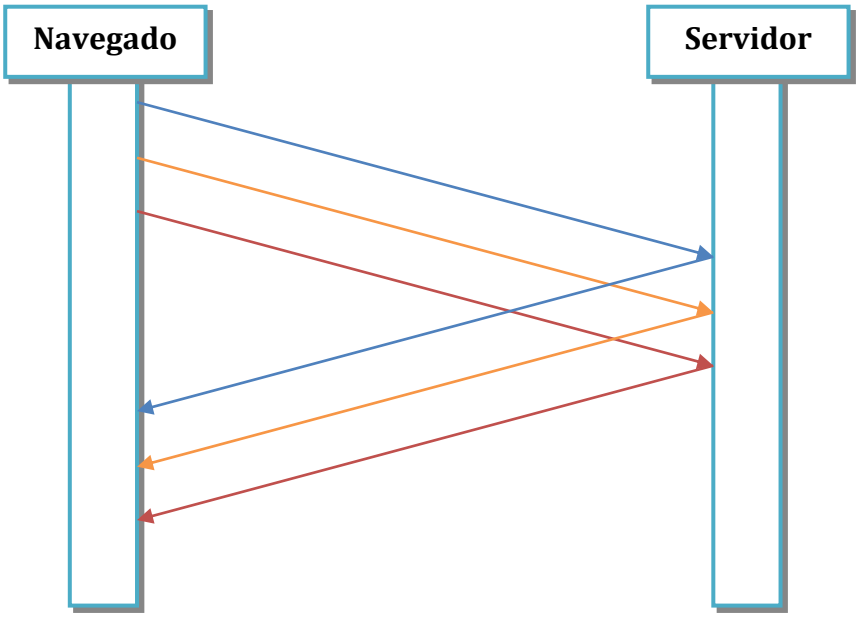

Figura 9-Peticiones en paralelo al servidor

En primer lugar el navegador envía una petición HTML en la que se indica la ruta de la plantilla que se necesita. Para nuestro caso se pide …/private/user/profile/profile.html.

En paralelo a la petición HTML, se envía la solicitud de la plantilla JavaScript que corresponde. Igual que en caso anterior se solicita a través de la ruta, en nuestro caso sería …/private/user/profile/profileController.js.

Por último puede ser necesario realizar alguna operación a base de datos para cargar información, por ejemplo los datos del perfil del usuario. Para ello se envía una petición JSON, en paralelo a las anteriores, en la que se especifica el tipo de operación y los parámetros si los tiene. El servidor comprueba si el usuario tiene permitido realizar la operación, si todo es correcto devuelve los datos en un objeto JSON.

El objeto JSON se recibe a través de los parámetros del controlador. Dentro de éste, sea signará a una variable. Como angular permite vincular datos entre el JavaScript y el HTML al cargar los datos en el controlador se muestran inmediatamente en el HTML. Para comprender este mecanismo vamos a ilustrar un ejemplo real simplificado.

Después de enviar la petición del HTML el archivo recibido es el que muestra a continuación:

```
<table>
<tr><td>Usuario</td><td><b>{{profile.username}}</b></td></tr>
<tr><td>Nombre</td><td><b>{{profile.name}}</b></td></tr>
<tr><td>Apellidos</td><td>{{profile.surname}}</td></tr>
<tr><td>E-mail</td><td>{{profile.email}}</td></tr>
<tr><td>País</td><td>{{profile.country}}</td></tr>
<tr><td>Provincia</td><td>{{profile.province}}</td></tr>
<tr><td>Ciudad</td><td>{{profile.city}}</td></tr>
<tr><td>Inmersiones</td><td>{{profile.dives}}</td></tr>
</table>
```

Tabla 4-HTML que muestra los datos de usario

Como sabemos el navegador también pide la plantilla JavaScript o controlador, un ejemplo podría ser el siguiente.

```
function ProfileController($scope,profileInfo){
    $scope.profile=profileInfo;

//El resto de las funciones se implementarán a partir de aquí

}
```

Tabla 5-Controlador JavaScript

El controlador recibe por parámetros $scope y profileInfo. Es fácil adivinar que en el objeto profileInfo se almacenará la información del perfil que se solicitó al servidor. $scope por su parte es un poco más difícil de entender. Podemos imaginar que $scope representa el alcance del controlador y la vista. En él se pueden guardar todas las variables y funciones que se necesiten vincular entre los archivos JavaScript y HTML. La información recibida en profileInfo se guarda en la variable profile dentro del $scope. Una vez hecha la asignación, el HTML podrá mostrar los datos. El ejemplo dará lugar al siguiente resultado:

Figura 10-Resultado de mostrar los datos de perfil

En conclusión este método nos permite hacer de una manera muy flexible la carga del contenido, además como en muchas ocasiones el contenido es pequeño, la descarga es muy rápida, lo que agiliza la carga del contenido. Este sistema es fácil de utilizar una vez que se ha comprendido por completo su funcionamiento y su estructura. Sin embargo, el sistema hubiera sido más complicado de implementar si no se hubiera usado el framework AngularJS.

5.3 UTILIZACIÓN DE ANGULAR

Como ya se ha mencionado en la sección del estado del arte, AnglarJS es un framework JavaScript que permite una separación fácil entre la parte visible y el controlador. AngularJS hace de intermediario entre los dos, permitiendo vincular datos entre ellos. Su utilización ha conllevado una optimización del tiempo de desarrollo ya que es sencillo de entender y manipular, sin embargo, también es cierto que es demasiado extenso para el uso que se le ha dado.

Al ser un framework de código abierto permite la modificación completa de cualquiera de sus características. Para nuestro sistema, por ejemplo, se ha añadido la funcionalidad de pedir al servidor archivos JavaScript. Además existe un gran soporte para este framework, hay multitud de módulos ya implementados que pueden resultar de mucha utilidad. En la sección 5.7 se hablará de las librerías adicionales que se han utilizado.

En este proyecto estamos utilizando Single Page Application, el hecho de unir este sistema con Angular, nos proporciona libertad absoluta a la hora de cambiar dinámicamente los contenidos que se muestran en cada momento.

Con la idea en mete de un ciclo de vida basado en prototipos, usar este framework era una opción muy buena ya que nos permite separa completamente la vista del controlador. Gracias a esta separación se pude modificar una de las partes, por ejemplo la vista, sin alterar en absoluto la otra.

Para terminar de comprender el framework se van a plasmar algunos ejemplos del sistema que el ilustra el uso de plantillas como ng-repeat, ng-show o ng-change.

Ng-repeat: se utiliza para recorrer un listado de elementos en el documento HTML. Es una herramienta muy utilizada en nuestra aplicación para mostrar cualquier tipo de lista como mensajes, posts, amigos, etc. En el siguiente se ejemplo se muestra el código html utilizado para el listado de mensajes privados que se muestra en la **¡Error! No**

```
<div class="row compact col-xs-12" ng-repeat="message in messages" >
  <div class="text-left">
     <div class="col-xs-1">
        <img ng-src="ImagesDownload?type=profileId={{message.sender}}" alt="foto
usuario">
     </div>
     <div class=" col-xs-5 compact">
        <a>
        {{message.senderName}}{{message.senderSsurname}}{{message.senderUsername}})
        </a>
        <br>
        <small>{{message.date| date:'short'}}</small>
     </div>
     <div class="row compact">
        <h4>{{aMessage.message}}</h4>
    </div>
  </div>
</div>
```

Tabla 6-Ejemplo de HTML con ng-repeat para mostrar mensajes

e encuentra el origen de la referencia.

Figura 11-Listado de mensajes privados

Ng-show: otra de las funciones de Angula que más se han utilizado, como su nombre da a entender se utiliza para mostrar u ocultar componentes HTML. Por ejemplo cuando se quiere ocultar cierto contenido del perfil si el visitante no es su amigo. Su utilización es trivial, basta con colocar la etiqueta ng-show=”variable” dentro de un para tener la posibilidad desde el controlador de ocultar o mostrar el contenido cambiando el valor de la variable.

Ng-change: esta etiqueta nos proporciona un callback aplicable a muchos elementos HTML. Por ejemplo en nuestra aplicación hay implementados varios SearchLive, esto se consigue fácilmente con ésta etiqueta de la siguiente forma.

```
<div class="input-group">
    <input type="text"
           id="location"
           class="form-control"
           name="location"
           ng-change="change(searchText)"
           ng-model="searchText"
           data-live-search="true"
           ng-maxlength="255" />
</div>
```

Tabla 7-Ejemplo de HTML con ng-change

Su funcionamiento consiste en que cuando cambia algo en el campo de entrada, llama a la función change() y le pasa como argumento el texto que contiene. En este caso la función buscará localizaciones que coincidan con el texto.

En conclusión vemos que este framwork ha facilitado en gran medida a la creación y remodelación de ciertos requisitos, aspecto que era fundamental para que la aplicación funcionara.

5.4 SEGURIDAD

La seguridad en cualquier aplicación web es un apartado muy importante, en este capítulo se abordaran bastantes de las medidas de seguridad adoptadas para mantener protegido nuestro sistema.

Una parte muy importante de la seguridad era el control de acceso a las distintas páginas paras los tres tipos diferentes de usuarios: buceadores, centros de buceo y administrador. Además era importante que no todos los usuarios pudieran realizar las mismas acciones, incluso una misma acción realizada por dos usuarios distintos puede tener un comportamiento diferente.

En primer lugar la mayor parte de la aplicación tiene el acceso restringido a usuarios no registrados. Una vez que el usuario ha hecho login en la aplicación se le establece una cookie, en ella se incluirán todos los roles que posee el usuario para que sean utilizados desde el navegador.

Los roles pueden ser desde roles generales como registrado o no registrado, hasta roles muy específicos que atribuyen por ejemplo al usuario administrador la posibilidad de eliminar centros dentro de la red. Estos roles no solo acreditan a un usuario para realizar ciertas operaciones, además permiten que la interfaz muestre el contenido acorde a ellos.

Puede ponerse como ejemplo el rol "*puede_eliminar_centro*" que solo posee el usuario administrador. Si cualquier usuario que no sea el administrador accede a una página donde se muestren todos los centros de aplicación, tendrán la opción de ver los detalles, sin embargo, el usuario administrador verá un botón para eliminarlos. Como además de la interfaz las operaciones también están restringidas, aunque un usuario no autorizado consiguiera ver el botón, nunca podría conseguir que esa operación se

ejecutara en el servidor. Es en la primera capa del Servidor, llamada JsonService, donde se comprueba si el usuario tiene permisos en la sesión para ejecutar la operación solicitada. Si no es así es rechazado y se escribe el error asociado al usuario.

Otro apartado importante en cuanto a seguridad es la contraseña. Se ha utilizado un cifrado de contraseña no reversible, al que se le han añadido varios tokens para amentar su robustez. Debido a esta irreversibilidad es necesario establece un mecanismo seguro para realizar cambios de contraseña. La estrategia utilizada para recuperar la contraseña es enviar un enlace con un código hash al correo del usuario. Este código permite el cambio de contraseña durante un periodo de tiempo determinado.

5.4.1 JSON VULNERABILITY PROTECTION

Como se ha comentado varias veces a lo largo de este documento, el navegador utiliza JSON para comunicarse con el servidor. Una web maliciosa puede aprovechar esta situación y conseguir tus datos de sesión sin mucho problema de la siguiente manera.

Suponemos que la víctima del ataque tras hacer login en www.scubeworld.com recibe en un objeto JSON la siguiente información {username:"pepe", password: "pepe"}. Tras un rato navegando en Scube, la victima navega a otra web. Si esta otra web fuera "maliciosa" y tuviera un script en su cabecera con el que accediera a www.scubeworld.com y guardara el objeto JSON devuelto, tendría en tu posesión el nombre y contraseña de la víctima.

Para este problema de seguridad AngularJS proporciona una solución fácil, consiste en colocar un prefijo en todas los objetos JSON. La siguiente imagen muestra JSON seguro y un JSON inseguro.

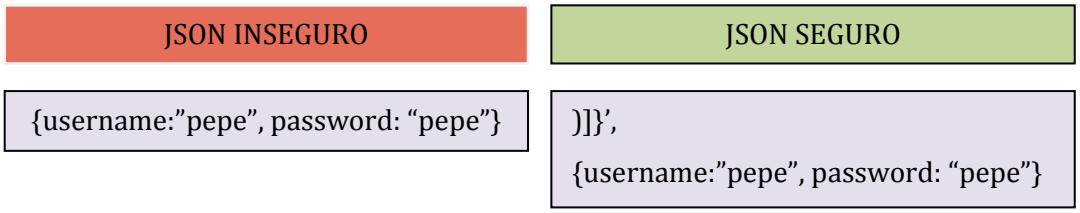

Figura 12-JSON inseguro vs JSON seguro

Cuando AngularJS recibe el objeto JSON seguro trata la primera línea como texto y las siguientes las procesa como un objeto JSON. Por su parte la web "maliciosa", cuando recibe el objeto seguro lo interpretará como un JavaScript e intentará parsearlo, el prefijo hace que se produzca un fallo de sintaxis y no pueda continuar leyendo el objeto impidiendo así el acceso a los datos.

5.4.2 CROSS SITE REQUEST FORGERY (XSRF) PROTECTION

En español significa falsificación de petición en sitios cruzados, es un tipo de ataque por el cual una web maliciosa realiza peticiones a una web en la cual la víctima ya ha estado autenticada. De esta manera un atacante puede acceder a la funcionalidad de la web a través del navegador ya autenticado de la víctima. Uno de los principales objetivos para este tipo de ataque son las redes sociales por lo que era necesario erradicarlo.

AngularJS proporciona un mecanismo para contrarrestar XSRF. Al realizar peticiones, el servicio $http lee un token de una cookie (por defecto XSRF-TOKEN) y lo establece como una cabecera HTTP. Dado que sólo el JavaScript que se ejecuta en su dominio podía leer la cookie, el servidor puede estar seguro de que la petición vino del JavaScript que se ejecuta en su dominio.

Para tomar ventaja, el servidor tiene que establecer un token en la cookie de sesión llamada XSRF-TOKEN del JavaScript en su primera petición HTTP GET. En las peticiones posteriores el servidor puede verificar que la cookie coincide con la cabecera, por lo que se asegura que sólo el JavaScript que se ejecuta en su dominio pudo haber enviado la solicitud. El token debe ser único para cada usuario y debe ser verificable por el servidor.

Si una web “maliciosa” intentará acceder a la funcionalidad de nuestra página se comprobaría si el token de sesión coincide con el que se encuentra en la cabecera de la petición. Este token es imposible de generar por una web que no sea la nuestra por lo que nunca serán iguales y no podrá acceder a ninguna funcionalidad.

5.4.3 PROTECCIÓN CONTRA INYECCIÓN HTML Y SQL

Un ataque por inyección por HTML se basa en que un atacante introduce código HTML malicioso en nuestra página. Normalmente este tipo de ataques se realizan a través de los campos de entrada en páginas que no filtran adecuadamente las etiquetas HTML.

Para prevenir este ataque AngularJS nos permite seleccionar el tipo de entrada para un campo, si este campo se marcara como texto sería imposible introducir HTML y que éste actuara como tal. Se puede dar el caso en el que se necesite que el tipo para la entrada de un campo sea HTML, para estos casos AngularJS nos proporciona $santize. $santize verifica la entrada mediante el análisis del código HTML en tokens. Todos los tokens de seguridad se vuelven a serializar para conseguir un código HTML limpio y de esta forma evitar un ataque de este tipo.

Otro tipo de ataque muy común es de inyección SQL. Consiste en introducir consultas SQL en campos de entrada o a través de la URL para tener acceso a la base de datos. Este ataque es más fácil de realizar cuando la página web atacada utiliza concatenación de cadenas para formar una consulta SQL.

En Scube, utilizamos myBatis para definir en las consultas el tipo de parámetro que debe sustituirse, nunca concatenarse. Como se especifica el tipo y no se hace concatenación, es imposible que un atacante pueda realizar la inyección de SQL. Sin embargo, si hay un tipo de consultas que utilizan la concatenación de cadenas, las que contienen ORDER BY. Para subsanar el problema que esto podría acarrear, todas las consultas con ORDER BY tienen una función que antes de realizar la consulta, comprueba si lo introducido a continuación del ORDER BY coincide exactamente con alguno de los parámetros en el select, en caso de que no sea así la consulta nunca se llega a ejecutar.

Por ejemplo si tuviéramos una consulta como esta **select nombre from user order by ${nombre,jdbcType=VARCHAR},** antes de ejecutar la consulta se comprobaría que la variable **${nombre,jdbcType=VARCHAR}** de tipo cadena coincide con "**nombre", "nombre asc" ó "nombre desc"** . Si la variable con coincide exactamente con alguna de esas cadena la consulta se rechaza.

Tras todo lo explicado se pude ver que la aplicación cuenta con bastantes mecanismos de seguridad para evitar ataques no deseados. A pesar de los esfuerzos puestos en este apartado, siempre puede haber algún agujero que permita la violación de la seguridad, que será más difícil de encontrar cuanto más controles de seguridad se hayan utilizado.

5.5 UTILIZACIÓN DE BOOTSTRAP

BootStrap es un framework CSS y JavaScript, es otro de los elementos importantes de la aplicación Scube. En anteriores secciones se ha hecho referencia a él, y en esta se mostrará en más detalle cuál ha sido su uso.

Para desarrollar la parte gráfica del sistema era necesario mantener una coherencia en toda la interfaz, desde los botones hasta los iconos. También era un requisito necesario que todos los elementos se pudieran modificar fácilmente y que el diseño fuera adaptable al tamaño de la pantalla. De los frameworks que pueden cubrir estas necesidades se eligió BootStrap.

En esta sección se va a hablar específicamente del diseño de la interfaz gráfica, en un primer momento se irá trabajando sobre prototipos los cuáles irán evolucionando hasta llegar al resultado final. Hay que tener en cuenta que la aplicación es compleja en cuanto al número de funciones que proporciona, por ello, es importante que la interfaz resulte lo más intuitiva posible para no desesperar al usuario.

BootStrap nos va a proporcional elementos gráficos uniformes para toda la interfaz que podrán ser modificados posteriormente para adaptarlos al diseño general de la aplicación. Además de eso, una de las opciones más interesantes que nos proporciona es la estructuración de los componentes en doce columnas. Esta forma de estructurar el contenido nos ayuda a realizar diseños adaptables a todos los tamaños de pantalla.

Algunas de las premisas que se han seguido para realizar la interfaz gráfica de la aplicación y conseguir los objetivos han sido:

1. El usuario debe ver una respuesta para cada acción que realice. Si por ejemplo envía un mensaje, éste debe aparecerle inmediatamente en la bandeja.
2. Como se comentó en el requisitos no funcional 6 (RNF6) los usuarios siempre deben tener información de los eventos inesperados que se produzcan durante su uso.
3. La interfaz se irá formando en función del tipo de usuarios y los roles que éste posea.
4. La interfaz debe ser coherente a lo largo de toda la aplicación. Todos los elementos que compartan una funcionalidad parecida deben estar en la misma disposición. Por ejemplo, todas las páginas que tengan un área de introducción de texto con un botón de enviar tendrán la misma estructura.

A continuación se muestra un esquema de cómo se ha implementado la pantalla de perfil de usuario con el fin de llegar a comprender la distribución en 12 columnas que nos proporciona BootStrap.

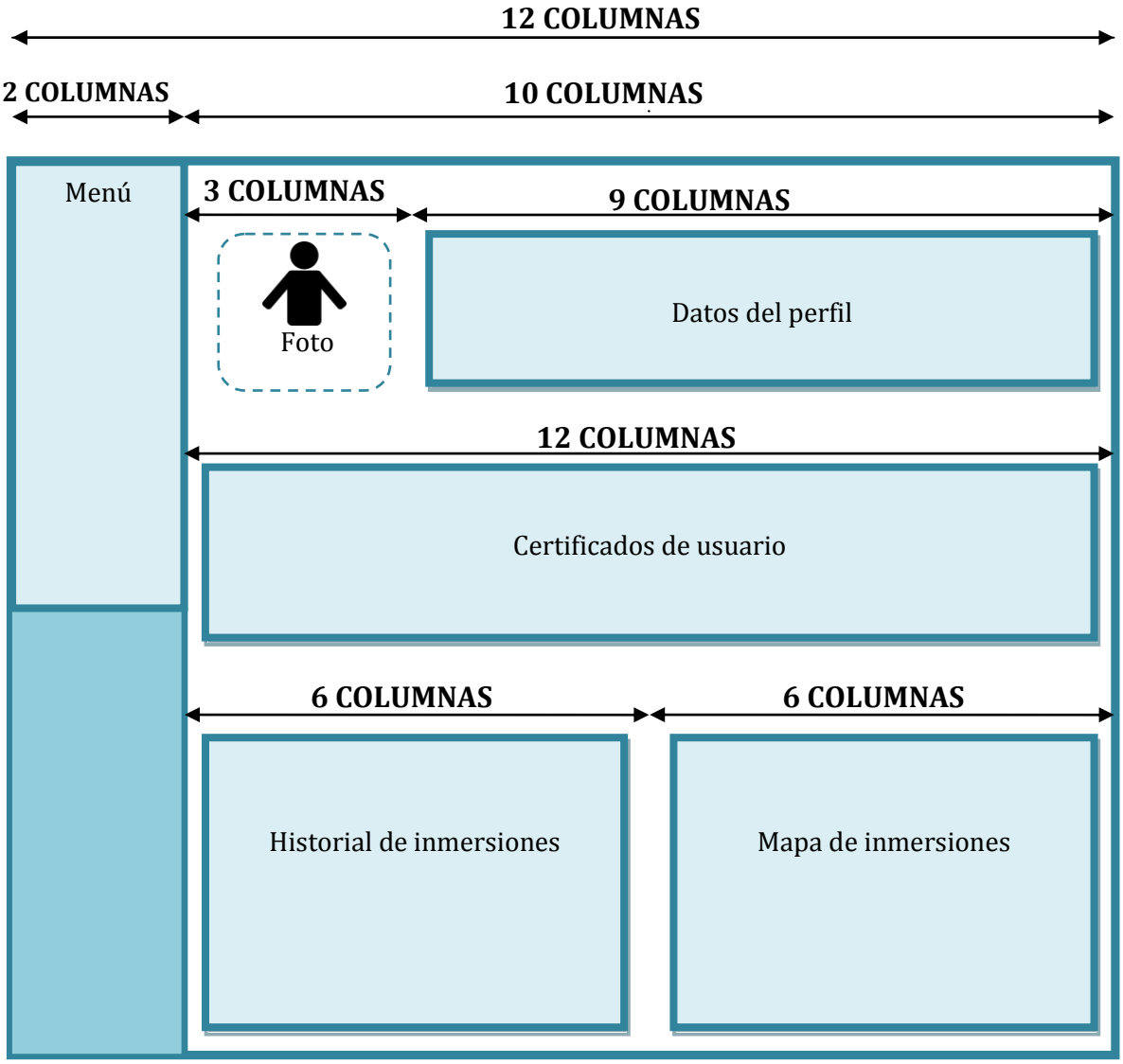

Figura 13- Utilización BootStrap para la maquetación de la página

Para el perfil del usuario dividimos la pantalla en 2 columnas para el menú y 10 para el contenido. Dentro de cada una de las dos divisiones se vuelven a tener 12 columnas para distribuir. Para el contenido en primer lugar dividimos en 3 columnas para la foto y 9 para los datos. Debajo estarán los certificados que ocuparan las 12 columnas del contenido (10 del total). Por último está el historial de inmersiones y el mapa donde se localizan, cada uno de los dos elementos cuenta con 6 columnas del contenido.

Esta ha sido la forma en la que se ha desarrollado toda la interfaz de la aplicación. Para mostrar la flexibilidad que ha proporcionado BootStrap junto con Angular a la hora de rediseñar la aplicación, en el ANEXO B se muestra el prototipo de algunas página junto a la del diseño actual.

5.6 BACK-END

El back-end es la parte de nuestro sistema que procesara las peticiones provenientes del front-end. A la hora de desarrollar las capas del back-end se ha tenido especial cuidado en que cada una sólo realizara la funcionalidad definida para ella, con el fin de mantener una funcionalidad independiente en cada una de las capas. Como se comento en la sección 4.1, el back-end se encuentra en el servidor y está dividido en cinco capas: JsonService, Session, Log, BackEnd y DBLayer.

5.6.1 JSONSERVICE

JsonService es el conector del back-end, en él se realizan los primeros controles de seguridad. Cuando el navegador envía una petición JSON el servidor la recibe en la capa JsonService. Lo primero que hace esta capa es comprobar la sesión, si existe saca de ella los datos del usuario y los almacena en una variable. A continuación, comprueba si el usuario tiene los permisos necesarios para poder ejecutar la operación de la siguiente manera:

1. Se compara la operación solicitada con las operaciones que no requieren ni permisos ni estar autenticado. Si alguna coincide, la operación pasa a la siguiente capa.
2. Si no coincide con ninguna de las anteriores, se comprueba si el usuario está autenticado en la aplicación, si no lo está se rechaza la petición y se escribe en el log la información.
3. Si el usuario se ha identificado se continúa comparando la operación solicitada con las que necesitan únicamente el rol de usuario registrado. Si coincide con alguna de éstas se pasa a la siguiente capa.
4. Las últimas operaciones con las que se comparará serán aquellas que requieren roles especiales. Si se encuentra la operación y el usuario tiene los roles necesarios se redirige la operación a la siguiente capa, si no, se rechaza la petición y se escribe en el log.
5. Puede darse el caso en el que la operación solicitada no se haya encontrado. Esto significa que es una operación inaccesible desde el navegador, por lo que se rechaza.

Cuando una petición pasa de JsonService a la siguiente capa se tiene claro que el usuario tiene los roles necesarios para ejecutarla, por lo que no es necesario volver a realizar comprobaciones de este tipo a lo largo del back-end.

5.6.2 SESSION

Session es una capa que tiene como finalidad es evitar la suplantación de identidad entre usuarios registrados. Para conseguirlo, intercepta la operación y coloca el identificador del usuario, obtenido de sesión, en la operación. De esta manera aunque un usuario enviara una petición con el identificador de otro usuario, en esta capa se sustituiría. Para comprender un poco mejor su funcionalidad se va a describir un pequeño ejemplo ilustrativo.

Supongamos que un usuario manda una operación para pedir todos sus mensajes privados. Dicha operación llevará como parámetro el identificador del propio usuario. Si el usuario consigue de alguna forma alterar esa petición y cambiar su identificador por el de otro usuario, podría tener acceso a los mensajes privados de otra persona. Evitar esto es justo el cometido de esta capa, ya que, independientemente del identificador que venga en la operación, se establecerá el que está en sesión.

5.6.3 LOG

Esta capa tiene como único cometido registra los errores que se producen durante la ejecución de una operación. En el mensaje de error se muestra el usuario, la operación realizada y la capa en la que se ha producido, para poder detectar donde se producen los errores durante la ejecución de una operación.

5.6.4 BACKEND

El BackEnd es la última capa antes de llegar a la capa de base de datos. Se utiliza para ejecutar la lógica de la operación si la tiene. No todas las peticiones se detienen en esta capa, sólo aquellas que necesiten una lógica adicional.

Un ejemplo claro para ilustrar el cometido de esta capa podría ser la operación Login. Cuando un usuario se autentica en la aplicación se manda la siguiente petición **{type:"Login",usuario:"pepe",contraseña="pepe"}.** La operación Login necesita comprobar en la base de datos que el usuario y la contraseña coinciden, y además, debe establecer los datos de sesión del usuario. Para ello la capa BackEnd intercepta la petición y realiza en primer lugar una consulta a base de datos para comprobar que el usuario y contraseña son correctos. En caso de acierto, se establece en sesión datos como el identificador de usuario y los roles, así como un token de seguridad para evitar un ataque por Cross Site Request Forgery.

5.6.5 DBLAYER

La capa DbLayer se encarga de realizar los accesos a la base de datos. Para ello busca el tipo de operación en los ficheros XML que contienen las consultas. Una vez que la encuentra sustituye los parámetros de la petición en la consulta y la envía a la base de datos. Si todo ha ido bien devuelve el resultado de la consulta en un objeto JSON, si no, lo registra en el Log y redirige al usuario a una página de error 500 (Error interno del servidor).

5.6.6 CONEXIÓN ENTRE CAPAS

La Figura 14 muestra un ejemplo de los componentes de una capa, como se conectan entre ellas y los patrones de diseño empleados. Todas las capas del sistema tienen una única interfaz de la que dependen, lo que permite abstraer las capas y poder modificar o reordenar cualquiera de ellas sin mayor esfuerzo.

Una capa, en nuestro ejemplo BackEnd, está formado por módulos. Cada uno de los módulos tiene un conjunto de operaciones. Un ejemplo de módulo podría ser el de autentificación que contiene operaciones como Login, Logout…

Cada capa y sus módulos, se conectan con la siguiente capa a través del patrón cadena de responsabilidades (Chain of responsbility). Utilizando este patrón se permite conectar módulo con módulo, capa con capa, capa con módulo y módulo con capa. La idea detrás de este diseño es desacoplar el emisor y el receptor dándole a varias capa la posibilidad de tratar la petición, que pasa a través de una cadena de capas hasta que es procesada por alguna de ellas.

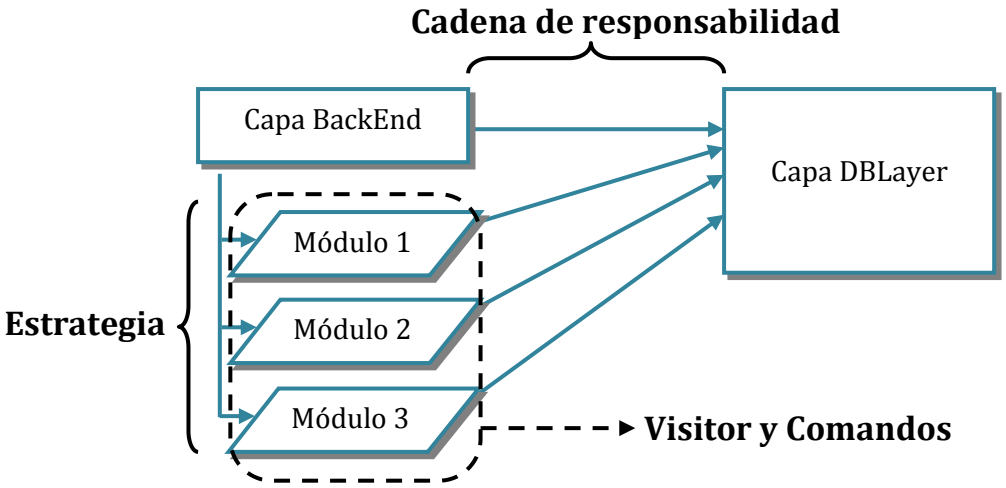

Figura 14- Conexión entre capas en el servidor

Otro patrón de diseño importante utilizado es Estrategia (Strategy). Este patrón de comportamiento se utiliza para elegir el módulo en función de la operación a realizar. Para cada operación se busca su correspondiente implementación a partir del identificador que proveen ambos.

Comandos (Command) es otro de los patrones de diseño utilizados para la parte del servidor o Back-End. Cuando se recibe el comando de una operación no conocemos cuál será su funcionalidad. Comandos elige de la interfaz del visitor cuál es el método que se encarga de ejecutar su lógica. Este patrón nos permite abstráenos de la tarea a ejecutar.

El último patrón de diseño empleado ha sido vistor que tiene como finalidad extraer del comando la lógica de su implementación. Cada comando se representa dentro del visitor como un método. Visitor permite tener varias implementaciones para la misma operación. En muchos casos con ayuda de este patrón se implementan interceptores de la operación cuya funcionalidad es específica, por ejemplo el cifrado de contraseña, o la normalización del login que se utiliza para pasar a minúsculas el nombre de usuario.

En conclusión tenemos una estructura robusta, escalable y transparente al programador. La abstracción de las capas permite que éstas se puedan intercambiar o eliminar, incluso se podrían añadir más capas sin que le afecte al desarrollador. Además los patrones utilizados nos proporcionan una flexibilidad muy grande a la hora de incorporar más módulo u operaciones.

5.7 APIS ADICIONALES

En esta sección vamos a hablar de las librerías adicionales que se han usado en nuestro sistema para cumplir ciertos requisitos.

5.7.1 GOOGLE MAPS

Nuestro aplicación requería en muchas ocasiones hacer uso de un mapa para localizar por ejemplo centros. En un primer lugar se opto por una librería que integraba AngularJS con GoogleMaps implementada por un usuario de la comunidad GitHub. Sin embargo debido a que su funcionamiento no era el esperado se decidió cambiar directamente a la API que proporciona Google Maps[8].

[8]https://developers.google.com/maps/?hl=es

Antes de añadir la API de GoogleMaps a nuestra aplicación es necesario conocer las limitaciones que ésta impone en la versión gratuita. Sólo se permiten 2500 peticiones diarias y nos ofrece una resolución de 640x640. En caso de que ese número de peticiones se quedase corto se puede optar por la versión Business.

En nuestro sistema se ha utilizado la versión gratis para usuarios registrados. Para ello es necesaria una clave de autenticación. En el ANEXO A se muestra como utilizar GoogleMaps para una aplicación web.

5.7.2 ANGULAR FILE UPLOAD

Ésta es otra librería externa utilizada en el proyecto. Surge de la necesidad de subir imágenes a la aplicación. La librería Angular-File-Upload proporciona mecanismo para enviar una imagen al servlet que se encarga de almacenarlas.

5.7.3 ANGULAR-STRAP

Esta librería ofrece la funcionalidad JavaScript de BootStrap integrada con Angular y algunos componentes adicionales. Una de los motivos por la que se ha utilizado es porque permite eliminar la dependencia con JQuery y así disminuir el tamaño de la carga inicial.

Otro motivo importante por el que se ha decido usar esta librería es porque nos proporciona ciertos componentes muy interesantes para nuestra página. Algunos de los componentes utilizados han sido selectores de fecha y hora.

6 PRUEBAS

Durante el desarrollo del proyecto hubo dos tipos de pruebas sobre las que se hizo especial hincapié, las pruebas de validación por parte del cliente y la compatibilidad con los navegadores.

Al tratarse de un proyecto con un ciclo de vida de prototipado evolutivo las pruebas era una de las formas de comprobar que los requisitos estaban bien definidos. Además en muchas ocasiones las pruebas daban lugar a nuevos requisitos que se incluían en el catalogo.

6.1 PRUEBAS DE VALIDADCIÓN

En esta sección se van a exponer algunos ejemplos de las pruebas realizadas por los clientes para comprobar que su funcionalidad es la correcta.

Buceador añade una nueva inmersión

La prueba consiste en hacer login en la aplicación con un usuario de tipo buceador. Una vez a accedido a la aplicación tendrá que poder crear una inmersión y además ver un listado con todas las creadas. En ésta prueba se valorarán dos conceptos.

En primer lugar se validará que el funcionamiento es el esperado, para ello no sólo debe poder crear la inmersión, además durante el proceso de creación se debe informar al usuario de los campos requeridos o si un el tipo de entrada de un campo es incorrecto.

Una vez el usuario ha llegado correctamente a la página que permite crear inmersiones se dispone a rellenar el cuestionario.

Prueba 1-Añadir inmersión vacía: El usuario pulsa en guardar inmersión sin rellenar ninguno de los campos. El **resultado** es un mensaje de error indicando que debe rellenar los campos requeridos, además en el formulario aparece una X en esos campos.

Prueba 2-Añadir inmersión sin campos requeridos: El usuario rellena todos los campos excepto uno requerido y vuelve a pulsar el botón de guardar. El **resultado** es el mismo que en la prueba anterior.

Prueba 3 -Añadir inmersión cambiando tipos de datos: El usuario introduce todos los datos requeridos, sin embargo en un campo donde se espera una fecha introduce un texto. El **resultado** es que la operación falla y el usuario es redirigido a una página de error.

- **Primera solución:** Para todos los campos que requieran un tipo de dato específico se crear una expresión regular que lo haga cumplir. Esta fue la primera solución implementada.
- **Segunda solución:** Para facilitar aun más la introducción de tipos de datos específicos se añaden selectores de fecha y hora, en el resto se seguirá usando la expresión regular. Sobre la primera solución se implementa ésta para los campos que se pueda.

Envío de mensajes privados

La prueba consiste en el envío de mensajes privados entre usuarios. No sólo se va a comprobar que un usuario pueda enviar un mensaje a otro, también se comprobará el funcionamiento de las respuestas a los mensajes y algún caso especial como cuando un usuario bloquea a otro.

Prueba 1- Envío de mensaje: Un usuario entra en la página de mensajes. Introduce el nombre del usuario al que quiere mandar el mensaje, como esta búsqueda utiliza un liveSearch no es necesario que escriba el nombre completo. Una vez localizado lo añade como destinatario, por último escribe el mensaje y pulsa enviar. El **resultado** es un dialogo que informa que se ha enviado correctamente. Además tras desaparecer el diálogo la lista de mensajes se recarga apareciéndole en primera posición el que acaba de enviar.

Prueba 2 – Recepción de mensaje y contestación: Un usuario accede a la página de mensajes. El primer mensaje de la lista aparece como no leído. Al pulsar sobre él se

marca como leído y se expande el contenido. El usuario tiene un cuadro de texto en el que escribe la contestación y pulsa enviar. El **resultado** es de nuevo un diálogo que le informa del correcto envío del mensaje, después se recarga la lista de mensajes.

Prueba 3 –Envío de mensajes a usuarios bloqueados: Un usuario accede a la página de mensajes. Busca el nombre de un usuario que le tiene bloqueado, le añade como destinatario escribe el mensaje y pulsa enviar. Esta prueba consiste en un caso de error por sí misma ya que no debería permitirse el envío de mensajes entre usuarios bloqueados.

- **Primera solución:** En el liveSearch se busca sobre todos los usuario excepto los bloqueados.

Prueba 4- Respuesta a mensajes de usuarios bloqueados: El usuario accede a los mensajes. Busca en su bandeja un usuario al que tiene bloqueado, expande el contenido, escribe una respuesta y pulsa en enviar. De nuevo estamos en un caso de error, la aplicación no debe permitir responder mensajes a usuarios bloqueados o bloqueantes.

- **Primera solución:** Al enviar la respuesta se comprueba si el usuario destino puede recibir el mensaje. Si no puede se muestra un mensaje que informa de la situación. Ésta fue la primera solución que se implementó.
- **Segunda solución:** Cuando se realiza la petición de los mensajes de un usuario, se trae además de los mensajes la relación con los usuarios de dichos mensajes. Si su relación es de bloqueado o bloqueante se oculta el campo de la respuesta. Ya que la primera solución quedaba poco intuitiva se implementó la segunda solución.

Agregar amigos

La prueba tendrá dos partes, en la primera se comprobará cómo se comporta la aplicación al enviar peticiones de amistad, en la otra parte se verá el comportamiento en el receptor de la petición. También haremos pruebas para la funcionalidad de bloquear y eliminar usuario.

La página de añadir amigo tendrá tres secciones, una que muestra los amigos, otra que muestra las peticiones pendientes y por último una sección que mostrará el listado de los usuarios bloqueados.

Prueba 1 - Añadir nuevo amigo de Scube: El usuario accede a la aplicación y accede a través del menú a amigos. Dentro de amigos se sitúa en la sección amigos, pulsa

en añadir nuevo amigo y le aparece un diálogo. En el diálogo le aparece un buscador, busca el usuario y pulsa en añadir. El **resultado** es que se oculta el diálogo y se actualiza la lista de amigos con el nuevo añadido. Este último amigo tendrá la etiqueta "pendiente de confirmación".

Prueba 2- Añadir amigo repetido: El usuario accede a la aplicación y entra en amigos. En el diálogo busca un usuario que ya tiene agregado como amigo y pulsa en añadir. El **resultado** es un error en la base de datos, el servidor redirige al usuario a una página de error.

- **Primera solución :**Cuando el usuario añade un amigo en el servidor se comprueba si ya está añadido antes de insertar la relación. Si ya existe se devuelve un código de error con el que se mostrará un mensaje informativo al usuario. Esta fue la primera solución implementada.
- **Segunda solución:** Cuando el usuario busca un nuevo usuario, además nos traemos la relación con el usuario actual, si la tiene. Dependiendo si tiene relación se mostrará el botón de añadir o una etiqueta con su relación, por ejemplo amigo. Ya que la primera solución requería una petición al servidor innecesaria se acabó por implementar la segunda.

Prueba 3- Aceptar petición de amistad: El usuario accede a amigos, dentro se sitúa en la sección peticiones pendientes. Ahí se muestran los usuarios que esperan la confirmación de amistad. Por último el usuario acepta la petición. El **resultado** es que la petición desaparece de la sección de peticiones pendientes, además la lista de amigos se actualiza con el nuevo amigo.

Prueba 4- Bloqueo de la petición de amistad: Un usuario accede a la peticiones de amistad pendiente. y pulsa en bloquear sobre la petición. El **resultado** es un mensaje de confirmación, si pulsa en aceptar se elimina de la lista de peticiones y se actualiza a la lista de bloqueados.

Prueba 5- Añadir amigo bloqueado: El usuario accede a la sección de amigos y busca un usuario por el que está bloqueado. Pulsa en añadir sobre el usuario. El **resultado** es un error en la operación y el servidor redirige a una página de error.

- **Solución 1:**Como ya se hizo para otras secciones, se trunca la búsqueda de forma que nunca aparezcan los usuarios que tienen relación de bloqueo.

6.2 PRUEBAS DE COMPATIBILIDAD EN NAVEGADORES

Las pruebas de compatibilidad en navegadores nos proporcionan una comprobación de que el sistema se verá correctamente en la mayoría de los navegadores existentes. Antes de exponer estas pruebas hay que tener en cuenta que en el proyecto no se da soporte de Internet Explorer 8 o inferiores, entre otras cosas por el uso global de IE8 que sólo representa el 3,82% según W3Counter[9] y por la cantidad de trabajo extra que supone adaptar el diseño a estas versiones. La herramienta utilizada para realizar estas pruebas es BrowserStack[10]

Safari

En esta prueba se comprobará las versiones de safari que tienen una correcta visualización del contenido tanto en OS X como en Windows.

En primer lugar se realiza la prueba en Windows sobre la versión 5.0 de Safari. Esta versión del navegador no carga el contenido, únicamente muestra el fondo de la página. A continuación probamos para la versión más reciente la 5.1. La carga de la página se muestra correctamente.

Ahora vamos a comprobar si el contenido se muestra correctamente en Safari para OS X. La primera versión que probaremos es la 7.0 sobre OS X Mavericks, en esta versión todo se ver correctamente y las transiciones entre las páginas son fluidas. La siguiente versión que probaremos es Safari 6.1 sobre Mountain Lion. El contenido para ésta versión se muestra correctamente. Por último probamos sobre la versión 5.1 sobre OS X Lion. De nuevo el navegador sale bien parado y muestra el contenido perfectamente.

Firefox

En esta prueba se comprobará la correcta visualización de la aplicación en las versiones más actuales de Firefox tanto para la plataforma Windows como OS X. Como Firefox está configurado para actualizarse automáticamente, por ello es suficiente con comprobar que se muestra correctamente en la última versión.

[9] http://www.w3counter.com/trends
[10]http://www.browserstack.com/

La primera prueba la hacemos sobre Firefox 30.0 en OS X Mavericks. El resultado de la prueba es satisfactorio y muestra el contenido perfectamente. Las transiciones entre páginas responden de manera fluida.

Ahora haremos probaremos Firefox 30.0 sobre Windows XP. La página se muestra correctamente si ningún problema para esta versión.

Chrome

Igual que el resto, se van a realizar pruebas tanto sobre Windows como OS X. Al igual que Firefox, Chrome también tiene las actualizaciones automáticas por defecto por lo que sólo se probará la última versión.

La última versión actualmente de Chrome es la 35.0. La prueba de éste navegador tanto en Windows como en OS X da un resultado satisfactorio. La carga de la página es un poco más rápida y fluida que en el caso de Firefox.

Internet Explorer

De todos los navegadores que se han probado hasta ahora Internet Explorer es el más problemático. Como se ha comentado de este navegador se soportará a partir de la versión 9.0.

Según W3Counter estos son los datos de uso de las diferentes versiones de IE:

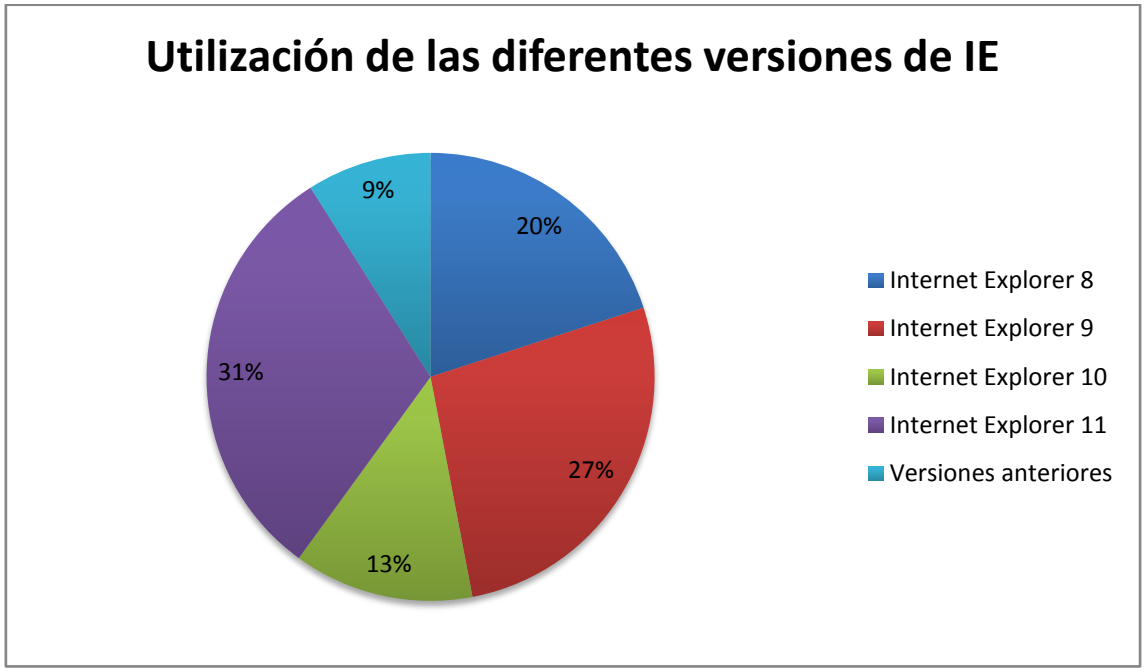

Figura 15-Uso de las diferentes versiones de Internet Explorer

A pesar de que la versión 8 de Internet Explorer representa el 20%, con los datos del fin de soporte de Google para esta versión y con el anuncio de que Microsoft deja de dar soporte a Windows XP, se espera que este porcentaje se reduzca bastante durante este año. Todas las pruebas para este navegador se han realizado en Windows 7.

Internet Explorer 9

Las pruebas con Internet Explorer 9 muestran algunas diferencias con respecto a los otros navegadores. Los colores no son exactamente los mismos, es difícil de apreciar pero se nota un matiz diferente. Por otra parte, los textos que se muestran en los campos de entrada están un poco descentrados.

A parte de los detalles mencionados Scube se ve perfectamente en esta versión, con una experiencia muy parecida al resto de los navegadores.

Internet Explorer 10 y 11

Al realizar las pruebas en ambas versiones sobre Windows 7 comprobamos que muestran correctamente todos los elementos. Además la interacción con la web es correcta como en el resto de los navegadores.

7 CONCLUSIONES

A lo largo del TFG se ha llevado a cabo la implementación de una red social real a partir de un prototipo. Actualmente el proyecto está en producción en fase de captación de usurarios, se espera que en pocos meses albergue una cantidad considerable de ellos.

El objetivo principal de la creación de la red social está cumplido, sin embargo este trabajo también tenía otros hitos que se debían cumplir. Uno de ellos era aportar una documentación base para que cualquier persona pueda incorporarse al proyecto y conocer la arquitectura completa del sistema, además podrá ver cómo ha ido evolucionando el proyecto a lo largo de las diferentes fases.

Tras la implementación del proyecto se puede hacer un balance de los resultados obtenidos. Se han utilizado muchas herramientas para la creación del sistema completo, las cuáles han supuesto una gran ayuda y con ellas se ha obtenido un buen resultado.

El paradigma Single Page Application ha proporcionado una mejora en cuanto a la optimización de los contenidos que se muestra, ha sido una buena elección a pesar de que aún se podría refinar más para conseguir un sistema más rápido. Este tipo de aplicaciones aún están en expansión por lo que con el tiempo habrá mayor soporte y ayuda para ellas.

Otro apartado importante de este proyecto era la introducción de Frameworks para facilitar el desarrollo del Font-End. No cabe duda de que ha sido un gran acierto utilizar tanto BootStrap como AngularJS ya que han proporcionado una estructura que sigue el patrón MVC y una interfaz uniforme para todo el sistema. Al comienzo del proyecto parecía que estos Framework iban a aportar poco, sin embargo con el gran ritmo

al que ha crecido, han resultado ser fundamentales en todos los sentidos. Además su aprendizaje no ha llevado demasiado tiempo ya que no se hace un uso intensivo de ellos.

En el documento se ha hecho especial hincapié en el desarrollo del Back-End. Este apartado era fundamental y se ha conseguido implementar con éxito, la utilización de capas bien definidas ha permitido controlar la seguridad al detalle. y con la posibilidad de modificarlo fácilmente en el futuro.

Por último cabe destacar también la utilización de APIs externas. Estas APIs han aportado un gran valor al sistema, por ejemplo GoogleMaps a pesar de algunos detalles menores, ha resultado ser una elección perfecta para la implementación de mapas.

En definitiva el trabajo realizado es muy satisfactorio, con todos los elementos bien implementados, desde la interfaz hasta la iteración con la base de datos pasando por el Back-End.

7.1 TRABAJO FUTURO

Una vez concluido el proyecto se tiene la arquitectura completa para la red social con la mayoría de requisitos que se exigían implementados, sin embargo el trabajo aún no ha terminado. A medida que el cliente va haciendo revisiones de la aplicación se van aumentando los requisitos y las bifurcaciones que pueden tomar cada uno de ellos. La prioridad del trabajo futuro se centra en los siguientes puntos:

- **Nuevos usuarios:** Actualmente soporta usuarios de tipo buceador, centro de buceo y administrador, lo siguiente sería crear usuarios para agencias de viaje, centros de alquiler de equipo, etc.
- **Integración con redes sociales:** Para que la red social tenga una mayor repercusión en el futuro se realizará una integración con redes como Facebook o Twitter para que cuando se publique algún contenido en Scube aparezca también en estas redes sociales.
- **Añadir funcionalidad:** La aplicación cuenta con un gran número de funcionalidades implementados, sin embargo aun se podrían añadir algunas, como un calendario para todos los miembros de la red que informe de eventos, fechas en las que se realizaron inmersiones, cursos, etc.
- **Separación de componentes:** A pesar de que el sistema se encuentra separado en módulos admitiría una división más, en concreto en el futuro se pretende separar el Back-End del Front-End.

Bibliografía

1. **Mikowski, Michael S and Powell, Josh C.** *Single Page Web Application.* New York 11964 : Manning Publications Co., 2014.

2. **Ruiz, Alba Calvo.** *Gestión de Trabajos en Movilidad(TFG).* España : s.n., 2013.

3. **Freeman, Adam.** *Pro AngularJS.* New York, 233 Spring Street, 6th Floor : Apress, 2014. p. 651. 978-1-4302-6449-1.

4. **Green, Brad and Seshadri, Shyam.** *AngularJS.* 1005 Gravenstein Highway North, Sebastopol : O'REILLY, 2013.

5. **Shaw, Peter.** *Twitter Bootstrap.* Morrisville, NC 27560 : Syncfusion Inc., 2014.

6. **Spurlock, Jake.** *Bootstrap.* 1005 Gravenstein Highway North, Sebastopol : O'REILLY, 2013.

7. **Group, Global Development.** Ventajas de PostgreSQL. *Sitio web de PostgreSQL.* [Online] [Cited: 2014 йил 01-06.] http://www.postgresql.org/about/advantages/.

8. **Obe, Regina and Hsu, Leo.** *PostgreSQL: Up and Running.* 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, 2012.

9. **Pérez, Reisel González.** *Inroducción al Sistema de Gestión de Base de DAtos PostrgreSQL.* s.l. : Universidad de las Ciencias Informáticas, 2008.

10. **Laboratorio Nacional de Calidad del Software, INTECO.** *Ingeniería del Software: Metodologías y clicos de vida.* 2009.

11. **Juzgado, J.** *Procesos de contrucción del software y cliclos de vida.* Universidad Politécnica de Madrid. Madrid : s.n., 1996.

12. **Christopher, Alexander.** *A Pattern Language.* Oxford University Press, New York : s.n., 1977. Vol. 1.

13. **Gamma, Erich, et al.** *Design Patterns.* s.l. : Addison-Wesley, 1994. p. 395.

14. *Técnicas de programación.* [pdf] Madrid : Universidad Carlos III, 2008 йил.

15. **Douglas, Korry and Douglas, Susan.** *PostgreSQL.* [ed.] Sams Publishing. 2. 2005.

Anexos

Anexo A. Utilización de mapas con GoogleMaps

El objetivo de este anexo es mostrar cómo se utiliza la API de GoogleMaps. Una vez hemos obtenemos la key de Google añadimos la dependencia de la siguiente forma.

```
var googleMaps = {
 url:
https://maps.googleapis.com/maps/api/js?key=KEY&sensor=false&callback=cal
lbackGoogleMapsReady',onReadyFunctionName: 'callbackGoogleMapsReady'
}
```

Añadida KEY a GoogleMaps

A continuación para todas las páginas que hagan uso de GoogleMaps se le añadirá el tag **requiere: [googleMaps]** en la referencia de la página (app.js). Este ejemplo muestra como quedaría la inclusión de esta etiqueta en nuestra página de localizaciones:

```
when('/profile', {baseName: 'profile',
require: [googleMaps]
}).
```

Añadida dependencia de GoogleMaps a las páginas necesarias.

Una vez añadida la dependencia vamos a añadir el mapa a la página profile. En el HTML quedará un código como el siguiente:

```
<div id="map_canvas"style="height: 350px; top: 10px;"></div>
```

Tabla 8-HTML para mostrar mapa GoogleMaps

El siguiente paso es añadirlo al controlador de la página, para ello necesitamos incluir el siguiente código:

```
Var map;
Var infowindow = newgoogle.maps.InfoWindow({content: ''});

$scope.initMap = function() {
    $timeout(function() {
        $scope.createMap();
    });
};
$scope.createMap = function() {
if (map) {
   return;
 }
varmapOptions = {
        zoom: 3,
        center: newgoogle.maps.LatLng(39, -4),
mapTypeId: google.maps.MapTypeId.SATELLITE
    };

map=newgoogle.maps.Map(document.getElementById('map_canvas'),
mapOptions);
/*Aquí podemos incluir la inicialización de los marcadores */

};
```

Tabla 9-Código para controlar el mapa de GoogleMaps.

Este código inicializa el mapa, establece el zoom a 3, centra la visualización en las coordenadas 39,-4 y coloca el tipo de vista satélite.

Ya se ha creado el mapa tanto en HTML como en el controlador, ahora se va a suponer que se quiere colocar un marcador en el mapa en la posición (3,20). Para ello en primer lugar se crea una variable que se llamará marker a la que le se le asignará el mapa que se ha creado de la siguiente forma **marker.setMap(map);**. Si lo que se quiere es un listado de marcadores se procederá de la siguiente forma:

```
var markers=[];
angular.forEach(markers, function(marker) {
     marker.setMap(map);
});
```

Tabla 10-Código para establecer un listado de marcadores en el mapa.

Este código puede ir incluido en la inicialización del mapa al final del código anterior. El siguiente paso es dar valor a los marcadores para situarlos en el mapa, para ello nos se puede crear una función que se llamará showInMap que recibe una lista con las localizaciones, cada localización debe llevar el atributo longitud y latitud.

```
$scope.showInMap = function(locations){
	angular.forEach(locations, function(location) {

	var latLng = newgoogle.maps.LatLng(
	location.latitude,
	location.longitude
		);
	Var marker = newgoogle.maps.Marker({
		position: latLng,
	map: map,
	title: location.name
	    });
	markers.push(marker);
	});
}
```

Tabla 11-Código que añade posición a los marcadores.

Como se ve en el código se va recorriendo el listado de localizaciones para en primer lugar crear un objeto LatLng con las coordenadas del marcador. Una vez se tiene el objeto lo siguiente es asignárselo al marcador. Por último introducimos el marcador en el listado.

Anexo B. Prototipo vs versión final

En este anexo se van a mostrar dos ejemplos del rediseño que se llevo a cabo durante este proyecto. El prototipo forma parte del trabajo realizado durante las practicas en empresa, el rediseño se realizó como parte de éste proyecto.

Figura 16-Prototipo muro.

Figura 17-Versión actual muro.

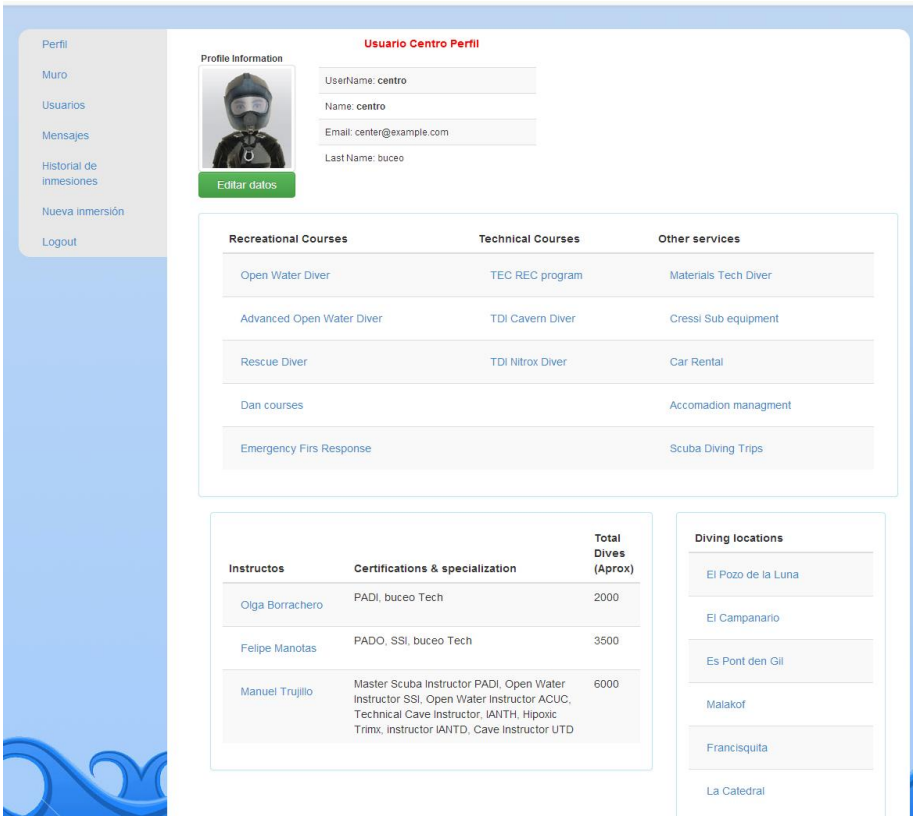

Figura 18-Prototipo perfil de centro de buceo

Figura 19- Versión actual del perfil de centro