industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

Universidad Politécnica de Cartagena

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Desarrollo de una aplicación web para compartir medio de transporte con AngularJS

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Fco Javier Avilés López**
Directores: Diego Alonso Cáceres
Francisco Sánchez Ledesma

Cartagena, 8 de Octubre de 2014

CONTENIDO

1 INTRODUCCIÓN

En este primer capítulo se proporciona una visión general del proyecto, explicando las motivaciones y objetivos que me han llevado a la realización del mismo. Además, se revisará la estructura general del documento.

1.1 MOTIVACIONES

A medida que la tecnología evoluciona, la sociedad la va adaptando en su día a día; actualmente, el dispositivo móvil es la tecnología que más cambio está experimentando (smartphones, tablets, laptops…), pero los diferentes sistemas operativos que utilizan obligan a los programadores a crear una aplicación para cada sistema operativo (IOS, Android, Windows phone, Windows, Linux…). Por este motivo, las aplicaciones web tienen una gran ventaja, y es que cualquier dispositivo con cualquier sistema operativo que posea un navegador web (todos hoy en día), es capaz de hacer uso de una aplicación web, en la cual el programador solo debe preocuparse de realizar un buen "responsive design" para adaptarse lo mejor posible al tamaño de pantalla del dispositivo que la está utilizando.

Esto, sumado a la creciente demanda de JavaScript como sustitución de PHP y otros lenguajes del lado servidor en aplicaciones web debido a su "liberación de trabajo" en el servidor, hace la idea de la creación de una aplicación web basada en JavaScript muy atractiva.

La aplicación hará uso de un patrón de desarrollo MVC (modelo-vista-controlador) implementado en JavaScript llamado AngularJS desarrollado por Google, muy ligero y potente, haciéndolo ideal para dispositivos móviles. También hará uso de CSS y HTML como es lógico ya que al fin y al cabo lo que se muestra en el navegador es una web. En cuanto a la base de datos, la aplicación funcionará sobre FireBase, una potente API de almacenamiento y sincronización de datos con un sistema de data binding, que combinado con el two-way-data-binding de Angular el usuario es capaz de ver lo que las variables contienen en directo, sin tener que actualizar la página, algo muy útil en aplicaciones web.

1.2 OBJETIVOS

Este proyecto de fin de grado se centra en el previo estudio de los principales lenguajes de programación web, así como la adquisición de un conocimiento más profundo en el framework AngularJS y la API FireBase, además, tiene como objetivo crear una aplicación Web donde los usuarios podrán encontrar a otras personas dispuestas a compartir coche a la hora de realizar un mismo trayecto, haciendo uso de las tecnologías ya mencionadas, con los beneficios que eso reporta: menos contaminación, gastos compartidos y un trayecto más ameno entre estudiantes.

La aplicación web deberá contar con un diseño totalmente adaptable a todo tipo de dispositivos y pantallas, así como ser compatible con los diferentes navegadores web del mercado: Safari, Mozilla Firefox, Internet Explorer y Google Chrome.

1.3 ESTRUCTURA DEL DOCUMENTO

En el capítulo “Estado de la técnica” realizaré una comparación de lenguajes del lado servidor como PHP o Java con lenguajes del lado cliente, como JavaScript con frameworks similares a AngularJS para aplicaciones web que se podrían haber usado para el proyecto, como son EmbedJS o BackboneJS. Posteriormente se realizará un repaso a las diferentes tecnologías que se necesitan conocer para el desarrollo de esta aplicación en el capítulo “Tecnologías utilizadas”, con un profundo análisis de AngularJS y FireBase.

Una vez realizado este análisis previo, se continuará el documento con un estudio de los posibles casos de uso de la aplicación, así como sus respectivos diagramas de secuencia para entender cómo trabajará la aplicación.

Una vez puestos en escena los posibles casos de uso, se dará pie al análisis de la aplicación web en el apartado “Implementación de la aplicación”, el cual consistirá en una explicación de las diferentes partes de las que se compone la aplicación, así como las técnicas de programación utilizadas, seguida de todo el contenido relativo a la Base de datos y su estructura.

Para finalizar se expondrán los resultados conseguidos, seguidos de una conclusión final y por último las referencias bibliográficas.

2 ESTADO DE LA TÉCNICA

Actualmente podemos distinguir entre dos tipos de tecnologías a la hora de desarrollar aplicaciones web, las que trabajan en el lado del servidor y las que trabajan en el lado del cliente. Actualmente, cada día son más usadas las del lado cliente debido a la liberación de carga de trabajo en los servidores, pudiendo crear aplicaciones muy ligeras del lado cliente.

Para este proyecto, se ha seleccionado una tecnología del lado cliente, un framework basado en JavaScript llamado AngularJS. No obstante, en este capítulo se van a mostrar algunas de las principales tecnologías que son utilizadas hoy día tanto en el lado servidor como en el lado cliente.

2.1 LADO SERVIDOR

2.1.1 PHP

PHP [15] (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

En lugar de usar muchos comandos para mostrar HTML (como en C o en Perl), las páginas de PHP contienen HTML con código incrustado que hace "algo". El código de PHP está encerrado entre las etiquetas especiales de comienzo y final <?php y ?> que permiten entrar y salir del "modo PHP".

Lo que distingue a PHP de algo como Javascript del lado del cliente, es que el código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el script, aunque no se sabría el código subyacente que era. El servidor web puede ser incluso configurado para que procese todos los ficheros HTML con PHP, por lo que no hay manera de que los usuarios puedan saber qué contienen los ficheros originales.

PHP está enfocado principalmente a la programación de scripts del lado del servidor, por lo que se puede hacer cualquier cosa que pueda hacer otro programa CGI, como recopilar datos de formularios, generar páginas con contenidos dinámicos, o enviar y recibir cookies. Aunque PHP puede hacer mucho más.

Existen principalmente tres campos principales donde se usan scripts de PHP.

Scripts del lado del servidor. Este es el campo más tradicional y el foco principal. Se necesitan tres cosas para que esto funcione. El analizador de PHP (módulo CGI o servidor), un servidor web y un navegador web. Es necesario ejecutar el servidor, con una instalación de PHP conectada. Se puede acceder al resultado del programa PHP con un navegador, viendo la página de PHP a través del servidor.

Scripts desde la línea de comandos. Se puede crear un script de PHP y ejecutarlo sin necesidad de un servidor o navegador. Solamente es necesario el analizador de PHP para utilizarlo de esta manera. Este tipo de uso es ideal para scripts ejecutados regularmente usando cron (en *nix o Linux) o el Planificador de tareas (en Windows). Estos scripts también pueden usarse para tareas simples de procesamiento de texto.

Escribir aplicaciones de escritorio. Probablemente PHP no sea el lenguaje más apropiado para crear aplicaciones de escritorio con una interfaz gráfica de usuario, pero si se conoce bien PHP, y se quisiera utilizar algunas características avanzadas de PHP en aplicaciones del lado del cliente, se puede utilizar PHP-GTK para escribir dichos programas. También es posible de esta manera escribir aplicaciones independientes de una plataforma. PHP-GTK es una extensión de PHP, no disponible en la distribución principal.

De modo que con PHP se tiene la libertad de elegir el sistema operativo, ya que puede usarse en todos los principales. Además, se tiene la posibilidad de utilizar programación por procedimientos o programación orientada a objetos (POO), o una mezcla de ambas.

Con PHP no se está limitado a generar HTML. Entre las capacidades de PHP se incluyen la creación de imágenes, ficheros PDF e incluso películas Flash (usando libswf y Ming). También se puede generar fácilmente cualquier tipo de texto, como XHTML y cualquier otro tipo de fichero XML. PHP puede autogenerar estos ficheros y guardarlos en el sistema de ficheros en vez de imprimirlos en pantalla, creando una caché en el lado del servidor para contenido dinámico.

Una de las características más potentes y destacables de PHP es su soporte para un amplio abanico de bases de datos. Escribir una página web con acceso a una base de datos es increíblemente simple utilizando una de las extensiones específicas de bases de datos (p.ej., para mysql), o utilizar una capa de abstracción como PDO, o conectarse a cualquier base de datos que admita el estándar de Conexión Abierta a Bases de Datos por medio de la extensión ODBC.

PHP también cuenta con soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros. También se pueden crear sockets de red puros e interactuar usando cualquier otro protocolo. PHP tiene soporte para el intercambio de datos complejos de WDDX entre virtualmente todos los lenguajes de programación web. Y hablando de interconexión, PHP posee soporte para la instalación de objetos Java y usarlos de forma transparente como objetos de PHP.

PHP tiene útiles características de procesamiento de texto, las cuales incluyen las expresiones regulares compatibles con Perl (PCRE), y muchas extensiones y herramientas para el acceso y análisis de documentos XML. PHP estandariza todas las extensiones XML sobre el fundamento sólido de libxml2, y amplía este conjunto de características añadiendo soporte para SimpleXML, XMLReader y XMLWriter.

2.1.2 JAVA JSF

JSF o JavaServer Faces [18], es una especificación de Java [16] para construir interfaces de usuario en aplicaciones web. JSF 2 hace uso de Facelets como sistema de plantillas por defecto, aunque otras tecnologías pueden ser empleadas para tal fin como por ejemplo XUL. Sin embargo, JSF 1.x hace uso de JavaServer Pages (JSP) como su sistema de plantillas por defecto.

Basándose en el diseño del modelo de un componente UI, JavaServer Faces usa archivos XML llamados vistas de plantilla o Facelets views. El proceso FacesServlet procesa las peticiones, carga las correspondientes vistas, construye un árbol de componentes, procesa los eventos y renderiza el DOM para el cliente. El estado de los componentes UI y otros objetos del scope son guardados al final de cada petición en un proceso llamado stateSaving, y recuperados en la siguiente creación de esa vista. Tanto el lado del cliente como el del servidor pueden guardar objetos y estados.

JSF se suele usar conjuntamente con Ajax, una importante tecnología. Ajax es una combinación de tecnologías que hacen posible crear interfaces web muy completas. Debido a que JSF soporta múltiples formatos de salida, los componentes de Ajax pueden ser fácilmente añadidos para enriquecer la interfaz web basada en JSF. La especificación de JSF 2.0 provee soporte “built-in” para Ajax, estandarizando el ciclo de vida de las peticiones de Ajax y permitiendo así el desarrollo de interfaces simples para manejar los eventos de Ajax, permitiendo a

cualquier evento disparado por el cliente tener una apropiada validación, conversión y finalmente invocación mediante un método antes de retornar el resultado al navegador via la actualización del DOM por parte de XML.

JSF 2 incluye soporte para minimizar la degradación de la aplicación cuando JavaScript se encuentra deshabilitado en el navegador.

2.2 LADO CLIENTE

2.2.1 EmbedJS

EmbedJS [21] es un framework JavaScript para dispositivos embebidos (móviles, televisiones, etc.) La gran ventaja que distingue a EmbedJS respecto a otros framework del lado cliente es que EmbedJS simplemente utiliza el código que necesita para cada dispositivo. Esto significa que habrá menos código que se debe cargar en el dispositivo y por tanto habrá menos código viajando en la ejecución de la aplicación y menos uso de memoria.

Embed es un proyecto de "Dojo Fundation. Es un software OpenSource y se encuentra disponible bajo las licencias BSD y MIT. Ambas garantizan el derecho al uso y construcción de aplicaciones con EmbedJS tanto para desarrollo como para la comercialización de la aplicación.

EmbedJS hace uso de un plugin para decidir que implementación debe ser usada a la hora de ejecutar las características que se requieren en cada proyecto; por tanto, a la hora de realizar un proyecto, simplemente se hará uso de las características que se requieran sin necesidad de hacer uso de código que nunca será utilizado.

2.2.2 BackboneJS

BackboneJS [22] es una librería JavaScript basada en model-view-presenter (MVP). Backbone conforma la estructura de las aplicaciones web abasteciéndolas de eventos personalizables, colecciones de APIs con numerosas funciones, vistas mediante el manejo de eventos, y todo lo conecta con la API existente mediante una interfaz "RESTful JSON". Se trata de ser una librería muy ligera, ya que su única dependencia es la librería JavaScript Underscore.js.

El proyecto se encuentra en GitHub, y el código fuente se encuentra disponible, así como una suit test online, un ejemplo de aplicación, numerosos tutoriales y una larga lista de proyectos reales que usan Backbone. Backbone se encuentra disponible para su uso bajo licencia MIT.

Cuando se trabaja en aplicaciones web que contemplan mucho JavaScript, se termina teniendo una aplicación llena de selectores jQuery y peticiones para mantener los datos sincronizados entre el HTML y la UI. Aquí nace la ventaja de Backbone, representando los datos como modelos, que pueden ser creados,

validados, destruidos y guardados en el servidor; siempre que una acción UI cause el cambio de un modelo, dicho modelo disparará un evento de cambio, y todas las vistas en las que se visualiza dicho modelo serán notificadas del cambio, siendo capaces de responder re-renderizándose ellas mismas con la nueva información.

2.2.3 jQuery Mobile

jQuery Mobile [19][20] es un framework JavaScript basado en HTML5 para el diseño de la interfaz de usuario, diseñado para hacer aplicaciones web "responsive" accesibles desde cualquier Smartphone, Tablet o dispositivo de escritorio. Se encuentra en continuo desarrollo por el equipo de jQuery. Es compatible con otros frameworks y plataformas como PhoneGap o Worklight.

Basa su ventaja sobre los demás framework en esta adaptabilidad a cualquier tipo de dispositivo así como plataformas móviles. Se construye sobre un núcleo jQuery por lo que es de fácil aprendizaje para los programadores familiarizados con la sintaxis jQuery.

Además, cuenta con la herramienta "builder" para generar un archivo JavaScript personalizado así como hojas de estilos también personalizadas para no usar más código del necesario.

3 TECNOLOGÍAS UTILIZADAS

3.1 HTML

El HTML [17], acrónimo inglés de HyperText Markup Language, es el código que posibilita la creación y edición de documentos web. HTML, es un metalenguaje heredado del SGML basado en etiquetas (tags) que tiene como virtud entre otras, la de poder ser implementado por código de otros lenguajes como JavaScript o Visual Basic Script que amplían y mejoran su capacidad original.

El código HTML utiliza el código ASCII puro que puede teclearse en cualquier editor básico para posteriormente ser interpretado secuencialmente por un objeto cliente denominado navegador (browser) que visualiza el resultado en pantalla.

La sintaxis de HTML está estructurada según el siguiente protocolo o sintaxis:

- Una etiqueta inicial que señala el comienzo de un elemento
- Un número determinado de atributos (y sus valores asociados)
- Una cierta cantidad de contenido (caracteres y otros elementos)
- Una etiqueta que marca el final.

Muchos componentes o elementos de HTML incluyen atributos específicos en las etiquetas de comienzo, que definen el comportamiento, o indican propiedades adicionales de dichos elementos. La etiqueta que delimita el final es opcional para algunos elementos. En muy pocos casos, un elemento vacío puede no contener o requerir la etiqueta de final.

De este modo, la página web contiene sólo texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final. Al ser un estándar, HTML busca ser un lenguaje que permita que cualquier página web escrita en una determinada versión, pueda ser interpretada de la misma forma (estándar) por cualquier navegador web actualizado.

Sin embargo, a lo largo de sus diferentes versiones, se han incorporado y suprimido diversas características, con el fin de hacerlo más eficiente y facilitar el desarrollo de páginas web compatibles con distintos navegadores y plataformas (PC de escritorio, portátiles, teléfonos inteligentes, tabletas, etc.). Sin embargo, para interpretar correctamente una nueva versión de HTML, los desarrolladores

de navegadores web deben incorporar estos cambios y el usuario debe ser capaz de usar la nueva versión del navegador con los cambios incorporados. Usualmente los cambios son aplicados mediante parches de actualización automática (Firefox, Chrome) u ofreciendo una nueva versión del navegador con todos los cambios incorporados, en un sitio web de descarga oficial (Internet Explorer). Un navegador no actualizado no será capaz de interpretar correctamente una página web escrita en una versión de HTML superior a la que pueda interpretar, lo que obliga muchas veces a los desarrolladores a aplicar técnicas y cambios que permitan corregir problemas de visualización e incluso de interpretación de código HTML. Así mismo, las páginas escritas en una versión anterior de HTML deberían ser actualizadas o reescritas, lo que no siempre se cumple. Es por ello que ciertos navegadores aún mantienen la capacidad de interpretar páginas web de versiones HTML anteriores. Por estas razones, aún existen diferencias entre distintos navegadores y versiones al interpretar una misma página web.

3.2 CSS

Hojas de Estilo en Cascada [17] (Cascading Style Sheets) es el lenguaje de hojas de estilo utilizado para describir el aspecto y el formato de un documento escrito en un lenguaje de marcas, esto incluye varios lenguajes basados en XML como son XHTML o SVG.

La información de estilo puede ser adjuntada como un documento separado o en el mismo documento HTML. En este último caso podrían definirse estilos generales en la cabecera del documento o en cada etiqueta particular mediante el atributo "<style>".

La filosofía de CSS se basa en intentar separar lo que es la estructura del documento HTML, de su presentación. Algunas opciones básicas del lenguaje CSS por ejemplo pueden ser el poder cambiar el color de algunas típicas etiquetas HTML como <H1> (h1 es una etiqueta en el lenguaje HTML destinada a mostrar un texto como encabezado, en tamaño grande). Pero también hay funciones algo más complejas, como introducir elementos propios con propiedades definidas por el programador <DIV> (div es una etiqueta HTML para identificar una determinada región o división de contenido dentro de una página web) o establecer imágenes de fondo.

El organismo encargado de la estandarización al respecto es el llamado W3C que definió la primera versión CSS1 en 1996. Posteriormente se han desarrollado las revisiones 2, 2.1 y 3 que es la más actual. El lenguaje CSS seguirá evolucionando, pero hoy día puede decirse que ha sido un éxito al simplificar el mantenimiento de páginas web. Antes, los contenidos y los estilos de presentación estaban mezclados. Con CSS, quedan separados y se hace más fácil el diseño y mantenimiento de páginas web.

El trabajo del diseñador web siempre está limitado por las posibilidades de los navegadores que utilizan los usuarios para acceder a sus páginas. Por este motivo es imprescindible conocer el soporte de CSS en cada uno de los navegadores más utilizados del mercado.

Internamente los navegadores están divididos en varios componentes. La parte del navegador que se encarga de interpretar el código HTML y CSS para mostrar las páginas se denomina motor. Desde el punto de vista del diseñador CSS, la versión de un motor es mucho más importante que la versión del propio navegador.

La siguiente tabla muestra el soporte de CSS 1, CSS 2.1 y CSS 3 de los cinco navegadores más utilizados por los usuarios:

Navegador	Motor	CSS 1	CSS 2.1	CSS 3
Google Chrome	WebKit	Completo desde la versión 85 del motor	Completo	Todos los selectores, pseudo-clases y muchas propiedades
Internet Explorer	Trident	Completo desde la versión 7.0 del navegador	Completo	Todos los selectores, pseudo-clases y muchas propiedades a partir de la versión 10.0 del navegador
Firefox	Gecko	Completo desde la versión 1.0 del navegador	Completo	Todos los selectores, pseudo-clases y muchas propiedades
Safari	WebKit	Completo desde la versión 85 del motor	Completo	Todos los selectores, pseudo-clases y muchas propiedades
Opera	Presto	Completo desde la versión 1.0 del navegador	Completo	Todos los selectores, pseudo-clases y muchas propiedades

Tabla 1 - Soporte de los navegadores web más usados actualmente para CSS

Como se puede observar, los navegadores Firefox, Chrome, Safari y Opera son los más avanzados en el soporte de CSS.

CSS define una serie de términos que permiten describir cada una de las partes que componen los estilos CSS. El siguiente esquema muestra las partes que forman un estilo CSS muy básico:

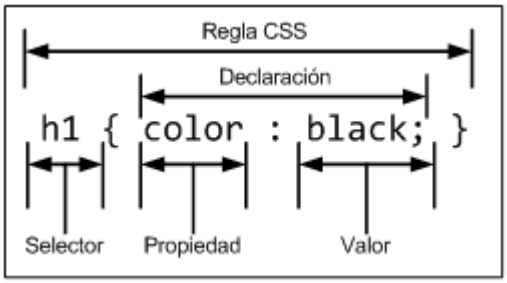


Ilustración 1 - Componentes de un estilo CSS básico

Los diferentes términos se definen a continuación:

Regla: cada uno de los estilos que componen una hoja de estilos CSS. Cada regla está compuesta de una parte de "selectores", un símbolo de "llave de apertura" ({), otra parte denominada "declaración" y por último, un símbolo de "llave de cierre" (}).

Selector: indica el elemento o elementos HTML a los que se aplica la regla CSS.

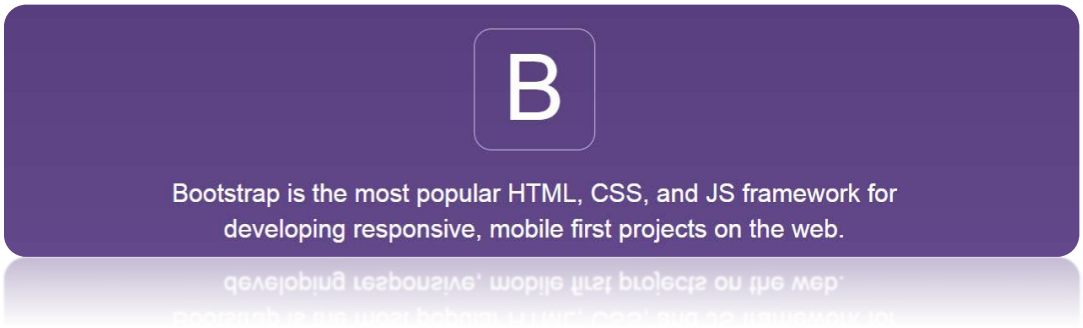
Declaración: especifica los estilos que se aplican a los elementos. Está compuesta por una o más propiedades CSS.

Propiedad: característica que se modifica en el elemento seleccionado, como por ejemplo su tamaño de letra, su color de fondo, etc.

Valor: establece el nuevo valor de la característica modificada en el elemento.

Un archivo CSS puede contener un número ilimitado de reglas CSS, cada regla se puede aplicar a varios selectores diferentes y cada declaración puede incluir tantos pares propiedad/valor como se desee.

3.2.1 Bootstrap



Hoy en día se habla mucho acerca del "Responsive design". El Responsive design hace referencia al diseño que se realiza de una web mediante CSS el cual es capaz de adaptarse a todo tipo de dispositivos o resoluciones de pantalla.

Bootstrap es una herramienta que proporciona de forma gratuita una librería de CSS para que directamente se añada en los proyectos web y se haga uso de las diferentes clases que tiene definidas en el código, un framework CSS especializado en diseños web para dispositivos móviles. Aunque algo compleja para usuarios recientemente iniciados en la programación de CSS, además de un enorme mundo de posibilidades en cuanto al formato se refiere, posee un potente sistema de grids o columnas. Dicho sistema divide la pantalla en doce partes, y las posibilidades de tamaño de pantalla en cuatro rangos de resoluciones, y de esta forma, si a cada elemento web de la aplicación se le asignan cuántas partes de esas doce partes se quiere que ocupe en cada uno de los posibles rangos de resoluciones, se podrá modelar a la perfección la forma en que la aplicación web se visualiza aunque tenga una considerable carga de trabajo.

3.3 JAVACRIPT

JavaScript [10] es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Se diseñó con una sintaxis similar al C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo Java y JavaScript no están relacionados y tienen semánticas y propósitos diferentes.

Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor, pero poco a poco está adquiriendo más responsabilidades.

3.3.1 Uso en páginas web

El uso más común de JavaScript es escribir funciones embebidas o incluidas en páginas HTML y que interactúan con el Document Object Model (DOM o Modelo de Objetos del Documento) de la página. Algunos ejemplos sencillos de este uso son:

- Cargar nuevo contenido para la página o enviar datos al servidor a través de AJAX sin necesidad de recargar la página (por ejemplo, una red social puede permitir al usuario enviar actualizaciones de estado sin salir de la página).

- Animación de los elementos de página, hacerlos desaparecer, cambiar su tamaño, moverlos, etc.

- Contenido interactivo, por ejemplo, juegos y reproducción de audio y vídeo.

- Validación de los valores de entrada de un formulario web para asegurarse de que son aceptables antes de ser enviado al servidor.

- Transmisión de información sobre los hábitos de lectura de los usuarios y las actividades de navegación a varios sitios web. Las páginas Web con frecuencia lo hacen para hacer análisis web, seguimiento de anuncios, la personalización o para otros fines.

Dado que el código JavaScript puede ejecutarse localmente en el navegador del usuario (en lugar de en un servidor remoto), el navegador puede responder a las acciones del usuario con rapidez, haciendo una aplicación más sensible. Por otra parte, el código JavaScript puede detectar acciones de los usuarios que HTML por sí sola no puede, como pulsaciones de teclado. Las aplicaciones como Gmail se aprovechan de esto: la mayor parte de la lógica de la interfaz de usuario está escrita en JavaScript, enviando peticiones al servidor (por ejemplo, el contenido de un mensaje de correo electrónico). La tendencia cada vez mayor por el uso de la programación Ajax explota de manera similar esta técnica.

Un motor de JavaScript (también conocido como intérprete de JavaScript o implementación JavaScript) es un intérprete que interpreta el código fuente de JavaScript y ejecuta la secuencia de comandos en consecuencia. El primer motor de JavaScript fue creado por Brendan Eich en Netscape Communications Corporation, para el navegador web Netscape Navigator. El motor, denominado SpiderMonkey, está implementado en C. Desde entonces, ha sido actualizado (en JavaScript 1.5) para cumplir con el ECMA-262 edición 3. El motor Rhino, creado principalmente por Norris Boyd (antes de Netscape, ahora en Google) es una implementación de JavaScript en Java. Rhino, como SpiderMonkey, es compatible con el ECMA-262 edición 3.

Un navegador web es, con mucho, el entorno de acogida más común para JavaScript. Los navegadores web suelen crear objetos no nativos, dependientes del entorno de ejecución, para representar el Document Object Model (DOM) en JavaScript. El servidor web es otro entorno común de servicios. Un servidor web JavaScript suele exponer sus propios objetos para representar objetos de petición y respuesta HTTP, que un programa JavaScript podría entonces interrogar y manipular para generar dinámicamente páginas web.

Debido a que JavaScript es el único lenguaje por el que los más populares navegadores comparten su apoyo, se ha convertido en un lenguaje al que muchos frameworks en otros lenguajes compilan, a pesar de que JavaScript no fue diseñado para tales propósitos. A pesar de las limitaciones de rendimiento inherentes a su naturaleza dinámica, el aumento de la velocidad de los motores

de JavaScript ha hecho de este lenguaje un entorno para la compilación sorprendentemente factible.

3.3.2 ¿Dónde puedo ver funcionando Javascript?

Entre los diferentes servicios que se encuentran realizados con Javascript en Internet se encuentran:

- Correo
- Chat
- Buscadores de Información

También se pueden encontrar o crear códigos para insertarlos en las páginas como:

- Reloj
- Contadores de visitas
- Fechas
- Calculadoras
- Validadores de formularios
- Detectores de navegadores e idiomas

3.4 ANGULARJS

AngularJS [1][2] es una tecnología del lado del cliente, un framework JavaScript opensource desarrollado por Google utilizado principalmente para crear aplicaciones web de una sola página; funciona con las tecnologías web más asentadas a lo largo del tiempo (HTML, CSS y JavaScript) para hacer el desarrollo de aplicaciones web más fácil y rápido que nunca. El código fuente de Angular está disponible gratuitamente en Github bajo la licencia MIT. Esto significa que cualquier persona puede contribuir y ayudar en su desarrollo.

Angular permite construir aplicaciones web modernas e interactivas mediante el aumento del nivel de abstracción entre el desarrollador y las tareas de desarrollo de aplicaciones web más comunes. Extendiendo el tradicional HTML con etiquetas propias (directivas), se podría decir que utiliza el patrón MVC (Model-View-Controller), aunque ellos mismos lo definen como un MVW (Model-View-Whatever (whatever works for you)).

Además, Angular es compatible con todos los navegadores de última generación (Chrome, Firefox, Safari, Opera, Webkits, IE9+) y se puede hacer compatible para Internet Explorer 8 o anterior mediante varios hacks.

También se encarga de las funciones avanzadas a las cuales los usuarios se han acostumbrado a tener en aplicaciones web modernas, tales como:

- Separación de la lógica de la aplicación, los modelos de datos y las vistas
- Servicios de Ajax
- Inyección de Dependencias
- Historial del navegador (botones atrás / adelante para la navegación en la aplicación)
- Testing
- Etc

En este ejemplo se pueden apreciar elementos nuevos como ng-app, ng-model y el nombre de una variable rodeada de dobles corchetes. Las dos primeras son lo que en Angular se llaman directivas, y la tercera es el modo de mostrar el valor de una variable del $Scope (algo así como el contexto de la aplicación) en pantalla, como si de un template se tratara.

```
<div ng-app>
        <input type='text' ng-model='miTexto' placeholder='Escribe algo aquí'/>
        <p>Esto es lo que estás escribiendo: {{miTexto}}</p>
</div>
```

3.4.1 Data Binding en AngularJS

Angular crea plantillas que se actualizan en todo momento, dando lugar a vistas dinámicas. Los componentes individuales de las vistas son dinámicamente actualizados en vivo. Esta característica de interpolación es sin duda uno de las características más importantes de Angular.

Esta característica funciona con solo incluir Angular.js en el HTML y establecer la ng app como atributo en un elemento en el DOM. El atributo ng app declara que todo dentro de la misma pertenece a esta aplicación Angular; de esta forma se puede anidar una aplicación Angular dentro de una aplicación web.

```
<input ng-model="name" type="text" placeholder="Your name">
<h1>Hello {{name}}</h1>
```

3.4.2 Modulos

La API module de Angular permite declarar un módulo usando el método Angular.module(). Cuando se declara un módulo, se necesitarán pasar dos parámetros al método; el primero es el nombre del módulo que se quiere crear, y el segundo es una lista de dependencias.

```
Angular.module('myApp', []);
```

Cuando se escriben aplicaciones extensas, se crearán bastantes módulos para hacer funcionar la lógica de la aplicación. Esta manera de llevar a cabo cada

funcionalidad por medio de un módulo distinto, proporciona la ventaja de aislar bloques funcionales para testear.

3.4.3 Scopes

Los Scopes son un núcleo fundamental de cualquier aplicación de Angular. Se utilizan en todo el framework, por lo que es importante conocerlos y saber cómo funcionan. El objeto $Scope es donde se define la funcionabilidad de la aplicación, los métodos en los controladores, y las propiedades en las vistas.

Los Scope sirven de nexo de unión entre el controlador y la vista. Justo antes de que la aplicación represente la vista para ser visualizada por el usuario, ésta se une al Scope, y la aplicación configura el DOM. Debido a esta "unión en directo", se puede confiar en la modificación del $Scope cuando la vista cambia, y de igual forma se puede confiar en la actualización de la vista cuando el $Scope sufre un cambio. Por lo tanto, en los scopes se guarda la información de los modelos que se representan en la vista y también atributos que se utilizan para manejar la lógica de la misma.

Los scopes se manejan principalmente desde los controladores y desde las directivas.

Cuando Angular arranca y genera una vista, creará la unión entre la directiva ng-app que se encuentra en el elemento del DOM que engloba la app, y el $rootScope. Este Scope, es una especie de padre de todos los $Scope de la aplicación, será lo más parecido que tiene Angular a una variable global, en este caso, un "contexto global", existiendo en toda la aplicación, y pudiendo ser visualizado en cualquier vista.

3.4.4 Controladores

Es el código con la lógica que comunica el modelo con la vista.

Los controladores son los encargados de inicializar y modificar la información que contienen los Scopes en función de las necesidades de la aplicación. Cuando se crea un nuevo controlador en una página, Angular le une un nuevo Scope, por lo que solo se tendrá que realizar la función del constructor.

Para crear acciones personalizadas que se puedan llamar desde las vistas, se puede crear funciones en el $Scope del controlador. Angular permite llamar a funciones del $Scope como si se estuviese referenciando una variable de datos.

Para unir un link o un botón a estas acciones que se han creado, se hará uso de una directiva denominada ng-click; esta directiva une el evento "click del ratón (en su flanco ascendente)", al "method handler", el cual llama al método especificado en el link, cuyo código se encuentra en los controladores.

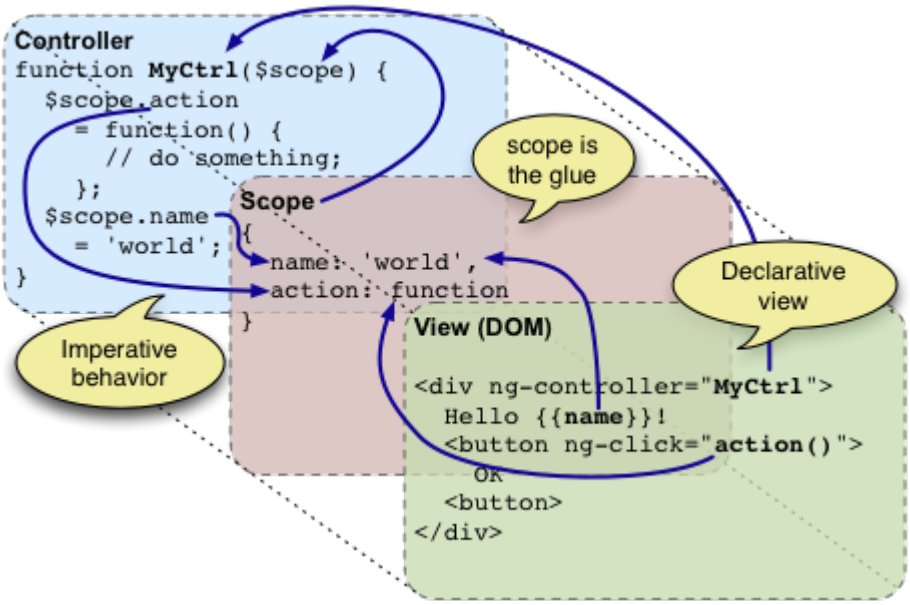


Ilustración 2 - Interacción entre el Controlador, la vista y el Scope en AngularJS

3.4.5 Directivas

Las directivas son el plato fuerte de Angular. Mediante el uso de las mismas se podrá extender la sintaxis de HTML y darle el comportamiento que se desee. Se pueden crear directivas a nivel de elemento, de atributo, de clase y de comentario. Un ejemplo sería el siguiente, mediante la directiva 'focusable' (una directiva a nivel de atributo) pudiendo modificar el comportamiento de los elementos input. En este caso cada vez que el input obtiene o pierde el foco cambia su color de fondo.

```
<div ng-app='MyApp'>
```

```
        <div>
                <input type='text' placeholder='Haz click aquí' focusable />
        </div>
</div>
```

Una de las cosas más interesantes de las directivas es la posibilidad de declararlas a nivel de elemento, lo que permite crear nuevas etiquetas de HTML que facilitan la creación de componentes reutilizables.

Por ejemplo, una nueva etiqueta HTML llamada Hello que será reemplazada por un botón que al hacer clic sobre él mostrará una alerta.

```
<div ng-app='MyApp'>
        <hello></hello>
</div>
```

Se podría distinguir entre dos tipos de directivas, las que incluye Angular, y las de creación propia.

Las primeras, ya incluidas en Angular, algunas sustituyen elementos del HTML como las etiquetas <form> y <a>, mejorando su funcionabilidad, como por ejemplo el comportamiento de <form> ante validaciones. Otras, inconfundibles debido a su prefijo "ng-", como por ejemplo ng-href, también son de gran utilidad. Ésta por ejemplo, no dispondrá del link disponible para el usuario hasta que la expresión que se ha puesto en ella sea evaluada y retorne un valor. Por último, otras no sustituirán o tendrán que ver con partes del HTML, como por ejemplo ng-controller; esta directiva puede usarse como atributo en cualquier etiqueta del HTML, para definir que en ella y todas las etiquetas anidadas, la aplicación trabajará con el controlador que se haya puesto en dicha directiva.

En cuanto a las directivas de creación propia, se definen usando el método directive() en uno de los módulos de Angular.

```
Angular.application('myApp', [])
        .directive('myDirective', function() {
        // A directive definition object
        return {
        // directive definition is defined via options
        // which we'll override here
        };
```

```
});
```

Como se puede observar en el ejemplo, el método directive() tiene dos argumentos:

- name (string)
  El nombre de la directiva, un string.
- factory_function (function)
  La función que retornará un objeto que definirá el comportamiento de la directiva.

También se podrá retornar una función en lugar de un objeto, pero será mejor práctica retornar un objeto. Cuando Angular arranca la aplicación, registrará los objetos que retornan como el nombre que se ha insertado en el primer argumento como string, por tanto el compilador de Angular buscará por el DOM elementos, atributos, comentarios o clases que usen el nombre de la directiva. Si encontrase alguno, utilizará la definición de la directiva para sustituir dicho elemento en el DOM.

```
<div my-directive></div>
```

Es buena práctica comenzar las directivas con 'my-' para no tener ninguna coincidencia con las tags de html.

3.4.6 Directivas y Scopes

Las directivas, las cuales se usan frecuentemente a lo largo del desarrollo de la aplicación, no crean $Scopes propios generalmente, pero hay casos en los que lo hacen.

Por ejemplo, la directiva ng-controller o ng-repeat crean su propio Scope y lo unen al elemento del DOM correspondiente.

3.4.7 Filtros

Los filtros permiten modificar el modo en el que se va a presentar la información al usuario. Angular cuenta con una gran cantidad de filtros ya creados para su uso directamente y una facilidad increíble para crear filtros propios.

La utilización de los mismos es similar a los Pipeline de Unix:

```
{{ expresion | filtro }}
```

Donde expresión puede ser cualquier tipo de expresión de Angular, como una variable del $Scope, y filtro el nombre del filtro que se quiera aplicar a la expresión.

En el siguiente ejemplo se hace uso de un filtro llamado uppercase, el cual transforma todo el texto recibido de la expresión en mayúsculas.

```
{{ name | uppercase }}
```

También se puede hacer uso de filtros desde Javascript con el servicio $filter; por ejemplo si se quiere hacer uso del filtro para transformar todo el texto en minúsculas:

```
myApp.controller('DemoController', ['$scope', '$filter', function($scope, $filter) {
        $scope.name = $filter('lowercase')('Ari');
}]);
```

Se puede pasar un argumento o varios al filtro en el HTML, añadiendo dos puntos después de cada argumento. Por ejemplo, el filtro number permite limitar el número de decimales que un número muestra:

```
<!—Mostrará: 123.46 -->
{{ 123.456789 | number:2 }}
```

También se podrán usar dos o más filtros a la misma vez añadiendo más “pipes”.

Angular trae varios filtros para monedas, fecha, hora, búsquedas, formato del texto, JSON, ordenar…

3.4.7.1 Crear filtros propios:

Para crear un filtro, la mejor práctica es crear un módulo propio para ello e incluirlo en la aplicación. Se va a crear un filtro que haga mayúscula la primera letra de un string.

```
Angular.module('myApp.filters', []).filter('capitalize', function() {
        return function(input) {
        // input será el string que se pasa a la función
        if (input)
                return input[0].toUpperCase() +
        input.slice(1);
        }
});
```

Si se quisiese hacer mayúscula la primera letra de una frase, se puede primero convertir todo el string a minúsculas y aplicar el nuevo filtro:

```
<!—Javier vive en el extranjero -->
{{ 'JAVIER VIVE EN EL EXTRANJERO' | lowercase | capitalize }}
```

3.4.8 Services

Los services son los encargados de comunicarse con el servidor para enviar y obtener información que después será tratada por los controllers para mostrarla en las vistas.

Hasta este momento, solo se ha mostrado cómo la vista se relaciona al $scope y como los controladores manejan los datos. Por motivos de rendimiento, los controladores son iniciados solo cuando se les necesita y el resto del tiempo son finalizados. Debido a esto, cada vez que se cambia de ruta o se recarga una vista, el controlador que está trabajando se limpia.

Con el fin de mantener datos permanentemente durante todo el ciclo de vida de la aplicación, como por ejemplo los datos de un usuario que inicia sesión, Angular hace uso de los servicios.

Son iniciados solo una vez por aplicación (por $injector), y son “lazyloaded” (solo se crean cuando se les necesita).

$http, es un ejemplo de un servicio de Angular. Provee acceso a bajo nivel al objeto XMLHttpReqest del navegador, en vez de ensuciar la aplicación con

llamadas continuas al objeto XMLHttpRequest, se podrá simplemente interactuar con la API $http.

```
// Ejemplo de un servicio que mantiene al usuario durante el tiempo de vida de la aplicación

Angular.module('myApp', []).factory('UserService', function($http) {
	var current_user;
	return {
		getCurrentUser: function() {
			return current_user;
		},
		setCurrentUser: function(user) {
			current_user = user;
		}
	}
});
```

Angular viene con muchos servicios con los que se suele interactuar constantemente a lo largo de una aplicación, además, es muy fácil crear servicios propios como se ha visto en el ejemplo. Una vez creado, se podrá referenciar y cargar como dependencia en todo el resto de la aplicación para tener acceso a los datos del mismo.

3.4.8.1 Como usar un servicio

Para usar un servicio, es necesario identificarlo como una dependencia en el componente donde se va a usar, bien sea un controlador, una directiva, un filtro u otro servicio. En el momento de correr la aplicación, Angular se hará cargo de iniciarlo y resolver las dependencias como normalmente.

Para inyectar el servicio en el controlador, se pasa el nombre como un argumento en la función del controlador.

Con la dependencia en el controlador, se podrá ejecutar cualquiera de los métodos que se han definido en el servicio.

```
Angular.module('myApp', ['myApp.services'])
.controller('ServiceController',function($scope, githubService) {
        $scope.events = githubService.events('auser');
});
```

3.4.8.2 Opciones al crear servicios

Aunque el método más común a la hora de crear servicios en la aplicación de Angular será a través del método factory(), hay otras APIs de las cuales se puede hacer uso en ciertas situaciones y de esta manera, escribir menos código.

Hay cinco métodos diferentes a la hora de crear servicios:

- factory()
- service()
- constant()
- value()
- provider()

3.4.8.2.1 factory()

Como se ha visto, este método es una manera rápida de crear y configurar un servicio. Necesita dos argumentos, name(string), el cual contiene el nombre del servicio que se quiere registrar, y getFn (function), la cual es iniciada cuando Angular crea el servicio.

3.4.8.2.2 service()

Si se quiere registrar el servicio desde un constructor, se podrá usar service(), ya que permite registrar una función constructora para el servicio. Recibe también dos argumentos, name(string), con el nombre del servicio, y constructor(function), donde se llama a la función para iniciar el servicio.

3.4.8.2.3 Provider()

Todos los servicios se crean a través del servicio $provide, los cuales son todos objetos con un método $get(). El $injector llama al método $get para crear el servicio. La base de todos los métodos para crear un servicio es el provider(),

responsable de registrar servicios en la $providerCache. Técnicamente, la función factory() es un método rápido de crear un servicio a través de provider(), donde se asume que el método $get() es la función que se pasa. Los dos métodos son funcionalmente equivalentes y crearán el mismo servicio.

3.4.8.2.4 constant()

Es posible registrar un valor existente como un servicio, el cual se podrá inyectar posteriormente en otras partes de la aplicación como un servicio. Por ejemplo, en el caso de que se quiera guardar una Key para un servicio del back-end; se podrá almacenar como un valor constarte usando constant(). Necesitará dos argumentos, name(string) como anteriormente se ha mencionado, en este caso para le nombre de la constante, y un value(constant value), donde se pasa el valor con el que se registra la constante.

3.4.8.2.5 value()

Si el valor que retorna el método $get en el servicio es una constante, no se necesita definir el servicio con un método más complejo; se puede simplemente usar la función value() para registrar el servicio. Esta función también aceptará dos métodos, name(string) y value(value) al igual que constant().

3.4.8.2.6 Value o constant

La mayor diferencia entre estos métodos es que puedes inyectar una constante en una función de configuración, mientras que no puedes inyectar un value. Sin embargo, con constant no se podrá registrar objetos en el servicio o funciones como en value. Básicamente, se usará value() para registrar un objeto o función como servicio, mientras que se hará uso de constant para los datos de configuración.

3.4.9 Eventos

Así como el navegador responde a ciertos eventos, como el click del ratón o el desplazamiento vertical, la aplicación de Angular responde a eventos de Angular. Esto permite comunicar ciertos elementos de la aplicación que no están diseñados para trabajar juntos.

Ya que el sistema de eventos de Angular no se comparte con el del navegador, solo se podrán recibir eventos de Angular, no del DOM.

3.4.10 Promesas

Una promesa es un método para resolver el valor de una variable de forma asíncrona. Las promesas son objetos que representan el valor de retorno o la excepción que retorna finalmente una función; son de gran utilidad cuando se trata con objetos remotos. Se podría ver desde el punto de vista de que fuesen la proxy de estos objetos remotos.

Tradicionalmente, Javascript hace uso de las llamadas "callbacks", pero no solo dificultan el debugging, si no que con las promesas, se podrán tratar con los datos de manera que ya hayan sido retornados, sin tener que depender de que el callback se dispare. Además de no disponer de una garantía a la hora de esperar la llamada.

Las promesas retornan un objeto:

```
User.get(fromId)
.then(function(user) {
        return user.friends.find(toId);
}, function(err) {
        // No existe el usuario
})
.then(function(friend) {
        return user.sendMessage(friend, message);
}, function(err) {
        // El amigo del usuario devuelve error
})
.then(function(success) {
        // El mensaje se envió al usuario
}, function(err) {
        // Ocurrió un error
});
```

Este código no sólo es mucho más comprensivo, sino que es más fácil de escribir que el de las callbacks. Además se garantiza el retorno de un solo valor, en lugar de tener que tratar con la interfaz de los callbacks.

Salir del "callback hell" es solo uno de los motivos por los que usar promesas. Realmente, el uso de promesas asegura poder hacer parecer funciones asíncronas a síncronas; pudiendo así obtener los valores de retorno y excepciones esperados.

3.4.11 Multiple Views and Routing

Un una aplicación de una sola página, navegar de una vista a otra es algo muy importante. Cuando la aplicación va creciendo y al a vez haciéndose más compleja, se necesitará una manera de manejar las vistas que el usuario verá conforme navega por la aplicación.

Se pueden manejar este tipo de aplicaciones incluyendo plantillas de código, pero esto no solo hará el código tedioso e inmanejable a largo plazo, si no que limitará la manera en que otros desarrolladores se unen al proyecto.

En vez de incluir plantillas de código en cada vista, se pueden incluir las vistas en un layout, e ir cambiando las vistas dentro de dicho layout(con la directiva ng-include) en base a la URL que acceda el usuario.

Angular permite hacer esto declarando rutas en el $routeProvider, mediante el cual se puede hacer uso del historial del navegador y permitir al usuario marcar e incluso compartir vistas específicas usando las URLs de acceso a las mismas.

Se necesitará referenciar Angular-route en el HTML tras referenciar a Angular.

```
<script src="lib/angular.js"></script>
<script src="lib/angular-route.js"></script>
```

La directiva ng-view en cuyo elemento del DOM se crearán las vistas dentro del HTML, sigue los siguientes pasos:

- En cualquier momento que el evento $routeChangeSuccess se dispare, la vista se actualizará en el elemento del DOM en el que se encuentre la directiva ng-view.

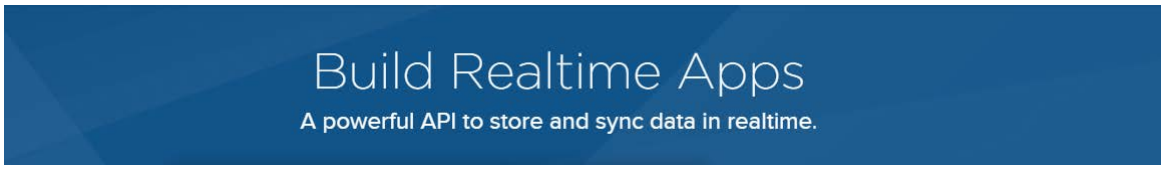
- Si hay alguna vista asociada a la actual ruta:
  - Crea un nuevo Scope

- Elimina la última vista, limpiando el anterior Scope
- Une el nuevo Scope y la nueva vista
- Une el controlador al Scope, si hay alguno especificado para dicha ruta
- Dispara el evento $viewContentLoaded

Para crear rutas en un módulo específico de la aplicación, usando la función de configuración config, y con el método when, el cual tendrá dos parámetros (url y la ruta de la vista), se podrá hacer el mapa de rutas:

```
Angular.module('myApp', []).
config(['$routeProvider', function($routeProvider) {
$routeProvider
.when('/', {
        templateUrl: 'views/home.html',
        controller: 'HomeController'
})
.when('/login', {
        templateUrl: 'views/login.html',
        controller: 'LoginController'
})
.when('/dashboard', {
        templateUrl: 'views/dashboard.html',
        controller: 'DashboardController',
resolve: {
        user: function(SessionService) {
        return SessionService.getCurrentUser();
}
}
})
.otherwise({
        redirectTo: '/'
});
}]);
```

3.5 FIREBASE



Firebase se presenta como una poderosa API para almacenar y sincronizar datos en tiempo real, una base de datos de tipo NoSQL. Pero Firebase es un servicio diferente a una simple base de datos NoSQL, un servicio dirigido tanto a aplicaciones web como aplicaciones móviles, el cual permite realizar, además del almacenamiento de datos, una sincronización en directo de los mismos en cada uno de los dispositivos que se encuentran conectados a la aplicación. Además, permite el funcionamiento offline y una posterior sincronización de los datos cuando el dispositivo vuelve a tener conexión a internet.

Además, cuenta con librerías para la mayoría de las plataformas web y móviles, y "bindings" para la mayoría de frameworks, entre los que se encuentra Angular, haciendo de Firebase un candidato ideal para este proyecto.

En cuanto a la seguridad de los datos, Firebase requiere encriptado SSL 2048-bit para toda la transferencia de datos.

3.5.1 Bases de datos NoSQL

Cuando se habla de bases de datos NoSQL [23][24], se incluye un variado rango de tecnologías, las cuales han sido desarrolladas para responder a la alta demanda de almacenamiento de información de usuarios, objetos y productos; así como la frecuencia en que estos datos son accedidos, dando éstas un mejor resultado en cuanto al rendimiento requerido. Las bases de datos relacionales, sin embargo, no fueron diseñadas para la alta demanda que requiere una aplicación moderna, ya que su rendimiento es peor en términos de precio de almacenamiento y poder de procesamiento del servidor.

En una base de datos NoSQL, se puede decir que todo es más flexible, ya que al contrario que las bases de datos SQL no se tiene que declarar todo el esquema de tablas antes de insertar los datos.

La clave a la hora de diseñar modelos de datos en estas bases de datos, se basa en la estructura de los documentos y en cómo la aplicación representa las relaciones de los datos.

3.5.1.1 Tipos de bases de datos NoSQL

Por un lado están las bases de datos basadas en documentos, que consisten en "keys" con una compleja estructura de datos asociada. Por otro lado las bases de datos de almacenamiento de "graphs", las cuales son utilizadas para almacenar información sobre redes, como Neo4J o HyperGraphDB.

Además, se deben tener en cuenta las del tipo "key-value", las más simples, en las que cada objeto que se almacena en ellas se almacenará como un "key" junto a un valor. Un ejemplo de estas bases de datos es Riak o Voldemort.
Por último, se encuentran las bases de datos como Cassandra o HBase, optimizadas para responder ante peticiones en grandes cantidades de datos, almacenados en forma de columnas de datos.

3.5.1.2 Comparativa bases de datos SQL y NoSQL

	SQL DATABASES	**NOSQL DATABASES**
Types	One type (SQL database) with minor variations	Many different types including key-value stores,document databases, wide-column stores, and graph databases
Examples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Data Storage Model	Individual records (e.g., "employees") are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., "manager," "date hired," etc.), much like a spreadsheet. Separate data types are stored in separate tables, and then	Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ("key" and "value"), with more complex information sometimes stored within the "value" columns. Document databases do away with the table-and-row model altogether, storing all relevant data

	joined together when more complex queries are executed.	together in single "document" in JSON, XML, or another format.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic. Records can add new information on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional requirements.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary
Development Model	Mix of open-source (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open-source
Data Manipulation	Specific language using Select, Insert, and Update statements.	Through object-oriented APIs
Consistency	Can be configured for strong consistency	Depends on product. Some provide strong consistency (e.g., MongoDB) whereas others offer eventual consistency (e.g., Cassandra)

Tabla 2 - Comparativa bases de datos SQL y NoSQL

3.5.2 Crear una cuenta

La primera cosa que necesitas hacer para comenzar a usar Firebase es crear una cuenta gratuita; una URL se asignará a tu base de datos con la dirección nombreDeCuenta.firebaseio.com.

Esta URL es la que se usará para almacenar y sincronizar datos. También se podrá acceder a dicha URL para ver la base de datos vía web y poder configurar algunas opciones de seguridad y login.

3.5.3 Instalar Firebase

Para incluir la librería de Firebase en la aplicación web, habrá que añadir una etiqueta de script en la sección <head> del HTML. Es recomendable usar el propio CDN de Firebase de la siguiente manera:

```
<script src="https://cdn.firebase.com/js/client/1.0.21/firebase.js"></script>
```

3.5.4 Leer y escribir desde Firebase

Para leer y escribir datos desde Firebase, lo primero que habrá que hacer es crear una referencia. Para crearla, habrá que pasar la URL de firebase como argumento en el constructor de Firebase:

```
var myFirebaseRef = new Firebase("https://<your-firebase>.firebaseio.com/");
```

3.5.4.1 Escribiendo datos

Una vez que se ha creado una variable con la referencia, se podrá escribir cualquier objeto con la nomenclatura JSON en la base de datos usando la instrucción set().

```
myFirebaseRef.set({
        title: "Hello World!",
        author: "Firebase",
        location: {
                city: "San Francisco",
                state: "California",
                zip: 94103
        }
});
```

Con la API específica para Angular que se verá a continuación, será algo diferente la manera en que se escriben los datos; una forma optimizada para Angular con muchas ventajas.

3.5.4.2 Leyendo datos

Leer datos desde la base de datos consistirá en realizar llamadas y resolver los eventos consecuentes. De igual forma que para escribir, se hará uso de la referencia creada; si por ejemplo se quisiera obtener el valor de "city" en la escritura anterior, haciendo uso del método on(), se haría de la siguiente manera:

```
myFirebaseRef.child("location/city").on("value", function(snapshot) {
        alert(snapshot.val());          //La alerta contendrá "San Francisco"
});
```

Con la API específica para Angular que se verá a continuación, será algo diferente la manera en que se escriben los datos; una forma optimizada para Angular con muchas ventajas.

3.5.5 Añadiendo autenticación

Una de las grandes ventajas de Firebase es la facilidad con que se pueden manejar la autenticación de usuarios. Firebase brinda diversas opciones de login como son los tradicionales usuario y contraseña, la posibilidad de realizar el login con una cuenta de Facebook o Twitter, etc…

Más adelante se verá el servicio SimpleLogin de Firebase para realizar el login en una aplicación web con usuario y contraseña. Esta es la forma que se usará para la aplicación web de este proyecto.

3.5.6 Seguridad en Firebase

En cuanto al control de acceso a los datos de la base de datos, se podrán establecer reglas de seguridad para los usuarios; por ejemplo permitir lectura a todos los usuarios, pero la escritura solo al admin:

```
{
        ".read": true,
        ".write": "auth.id === 'admin'"
}
```

3.5.7 Bindings

Firebase cuenta con APIs específicas para *AngularJS, Backbone, Ember y ReactJS*, haciendo más fácil y mejorando el uso de Firebase en estos frameworks.

Por supuesto se hará uso de la API para Angular, la cual se expondrá a continuación.

3.5.8 FirebaseSimpleLogin

Con este servicio de Firebase, se puede realizar un servicio de registro y autenticación de usuarios muy completo y sencillo en la aplicación mediante el uso del tradicional login con usuario/contraseña. Lo único que se debe hacer antes de proceder a hacer uso de todos los métodos es activar en la base de datos el uso de login mediante "password provider".

3.5.8.1 Métodos

new FirebaseSimpleLogin()
Crea un nuevo objeto de login para una referencia de Firebase dada.

```
new FirebaseSimpleLogin(ref, callback, [context])
```

login()
Realiza el login del usuario para el proveedor especificado (Usuario-contraseña, Facebook, Twitter…)

```
login(provider, [options])
```

logout()
Realiza el logout del usuario.

```
logout()
```

createUser()
Con este método se creará una nueva cuenta para la dirección email especificada. Es importante tener en cuenta que esta función NO realizará el login del usuario posteriormente de su creación; si se quisiese hacer funcionar de esa forma la aplicación, se deberá llamar posteriormente a la función de login() una vez el usuario haya sido creado.

```
createUser(email, password, [callback])
```

changePassword()
Este método sirve para cambiar la contraseña del usuario cuyo email se especifica como argumento.

```
changePassword(email, oldPassword, newPassword, [callback])
```

sendPasswordResetEmail()
Este método envía un email al usuario especificado para realizar el reset de la contraseña de su cuenta. Dicho email contendrá una contraseña temporal que el usuario deberá usar para autenticarse en la aplicación y una vez dentro poder cambiar la contraseña.

```
sendPasswordResetEmail(email, [callback])
```

removeUser()
Por último, este método se utilizará a la hora de eliminar una cuenta.

```
removeUser(email, password, [callback])
```

Si se quisiese hacer el login por medio de Facebook, para que los usuarios pudiesen entrar en la aplicación con su cuenta de Facebook, simplemente se debe activar en la base de datos el sistema de login por Facebook; a continuación, se debe crear una aplicación de Facebook, y en la configuración de la misma, en la sección de seguridad, habrá que añadir la URL:

https://auth.firebase.com/v2/<YOUR-FIREBASE>/auth/facebook/callback

En el apartado “Valid OAuth redirect URIs” y guardar los cambios. A continuación, habrá que introducir los credenciales de la aplicación App ID y App Secret en la configuración del login por método Facebook de Firebase. Una vez hecho esto, simplemente se deben usar los métodos correspondientes para realizar el login de los usuarios en la app por medio de Facebook.

3.5.9 FireBase y AngularJS: AngularFire

Citando el texto de la propia web de firebase:

AngularFire is the officially supported AngularJS binding for Firebase. The combination of Angular and Firebase provides a three-way data binding between HTML, JavaScript, and your Firebase backend.

Como se puede observar, AngularFire proporciona grandes ventajas para las aplicaciones realizadas con Angular

3.5.9.1 El papel de AngularFire

AngularFire es una librería de código abierto cuyo soporte corre a cuenta del equipo de Firebase y su comunidad de desarrolladores.

Algo a tener en cuenta, es que no por usar Angular en la aplicación, se tiene la obligación de hacer uso de AngularFire.

3.5.9.2 Comenzando con AngularFire

Tal y como se ha descrito en la sección anterior, se debe añadir los CDN en la sección <head>, pero con la diferencia de que para hacer uso de AngularFire también se debe añadir su correspondiente librería:

```
<script src="https://cdn.firebase.com/js/client/1.0.21/firebase.js"></script>
<script
src="https://cdn.firebase.com/libs/angularfire/0.8.2/angularfire.js"></script>
```

Una vez se han instalado las librerías, se puede proceder a incluir AngularFire en la aplicación de Angular, declarando “firebase” como dependencia en el modulo de la app:

```
var app = angular.module("sampleApp", ["firebase"]);
```

Por último, para finalizar la instalación de AngularFire en la app se debe inyectar el servicio $firebase en los controladores o servicios en los cuales se va a hacer uso de Firebase:

```
app.controller("SampleController", function($scope, $firebase) {
//Ya se puede usar $firebase para sincronizar datos entre los clientes y el servidor
        var ref = new Firebase("https://<your-firebase>.firebaseio.com/");
        var sync = $firebase(ref);
});
```

3.5.9.3 Resumen de la API

En la tabla se resumen las funciones más comunes para “hablar” con la base de

datos en AngularFire. Aquí podrás encontrar una lista completa y detallada de los todas las funciones.

$id	The key for each record (its path in Firebase as it would be returned by `ref.name()`)
$priority	The priority of each child node is stored here for reference. Changing this value and then calling`$save()` on the record will also change the priority on the server and move the record in the array.
$value	If the data in Firebase is a primitive (number, string, or boolean), `$asObject()` will still return an object. The primitive value will be stored under `$value` and can be changed and saved like any other child node.
$save()	Pushes local changes back to the Firebase server
$remove()	Removes this object from the Firebase server
$loaded()	Returns a promise which is resolved when the initial server data has been downloaded.
$bindTo()	Creates a 3-way data binding. Covered below under Three-Way data bindings

Tabla 3 - Funciones más comunes de AngularFire

3.5.9.4 3-way Data Bindings

Trabajar con objetos implicará estar haciendo uso continuamente de $save() cada vez que se realiza un cambio a nivel local en la aplicación y se quiera sincronizar en la base de datos. Para solventar dicho "problema", AngularFire cuenta con el sistema three-way-data-binding; con esta herramienta se podrán sincronizar objetos de forma que cualquier cambio en ellos a nivel local en el DOM será automáticamente sincronizado a Angular y de Angular a Firebase. Inversamente cualquier cambio en el servidor será realizado en Angular y en el DOM.

Para hacer uso del sistema three-way-data-binding, simplemente se debe llamar a **$bindTo()** en el objeto que se quiera que permanezca sincronizado. De esta forma no se necesitará seguir haciendo uso de $save(), ya que este sistema mantendrá el objeto sincronizado mediante el uso de $watch().

3.5.9.5 Sincronizar arrays con $asArray()

Este sistema deberá ser usado para cualquier lista de objetos con ids únicos que sea mostrada en la aplicación y se quiera mantener sincronizada. Simplemente se crearán arrays sincronizados con el método $asArray().

```
var messagesArray = sync.$asArray();

// place it in the scope for use in the DOM
$scope.messages = messagesArray;
```

Una vez realizada la sincronización, se podrá usar el array de forma normal en la aplicación:

```
<ul>
        <li ng-repeat="message in messages">{{message.user}}:
{{message.text}}</li>
</ul>
```

3.5.9.5.1 Summary of the API

En esta tabla se muestran los métodos más comunes para trabajar con los arrays sincronizados. Aquí puedes encontrar una lista completa de todos los métodos.

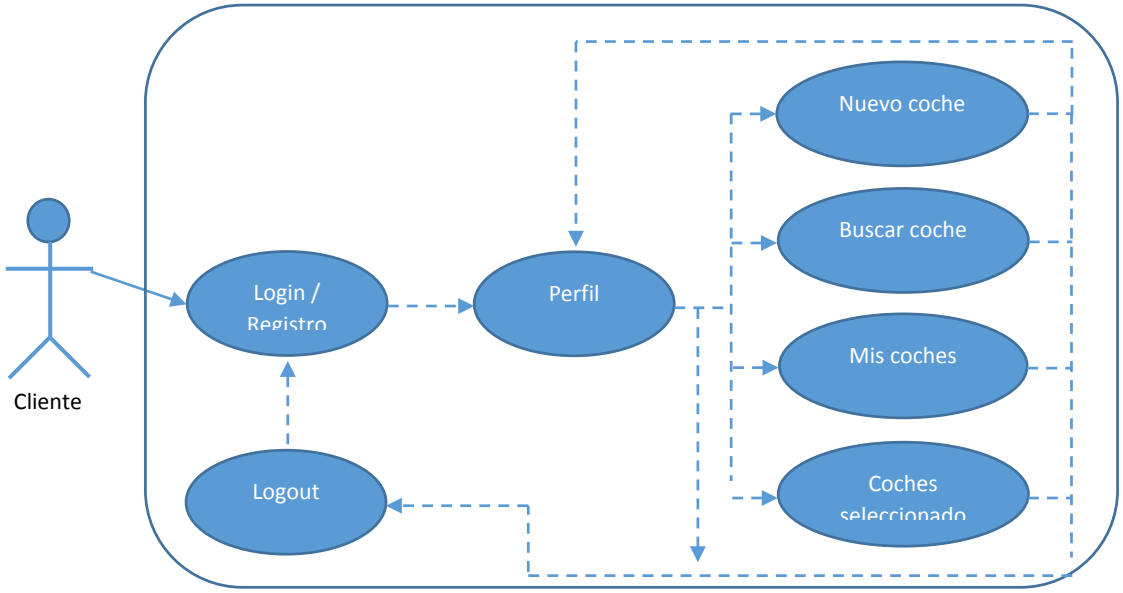
$add(data)	Creates a new record in the array. Should be used in place of `push()` or `splice()`
$save(itemOrIndex)	Saves an existing item in the array
$remove(itemOrIndex)	Removes an existing item from the array. Should be used in place of `pop()` or`splice()`
$getRecord(key)	Given the Firebase key, returns the item from the array. It is also possible to find the index with `$indexFor()`
$loaded()	Returns a promise which resolves after the initial records have been downloaded from Firebase.

3.5.9.5.2 Modificando los Arrays sincronizados

El contenido de éstos arrays se encontrará sincronizado con el servidor remoto, y por tanto AngularFire controlará las operaciones de adición, eliminación y ordenación de los elementos. Debido a esto, AngularFire utilizará los métodos **$add(), $remove() y $save()** para modificar los elementos de estos arrays.

4 CASOS DE USO

A continuación se exponen los diferentes casos de uso en los que consiste la aplicación que se ha realizado mediante el uso de diagramas UML [9]. Un diagrama de funcionamiento de la aplicación sería el siguiente:



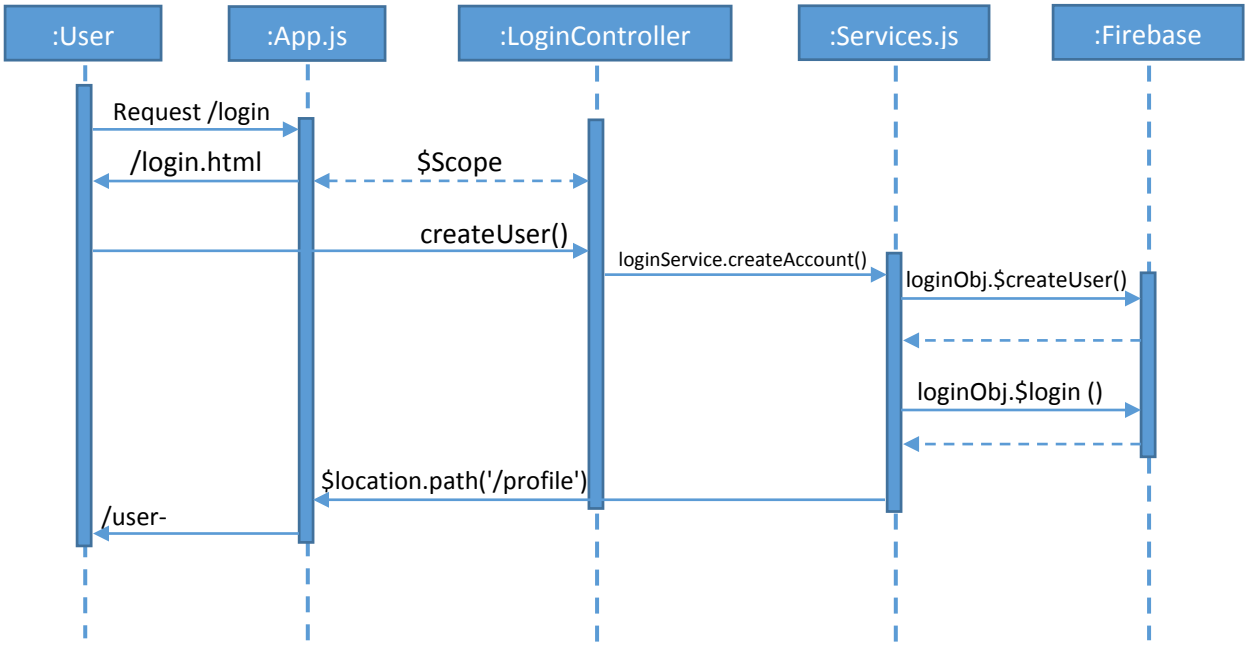
Por tanto se tendrán ocho diferentes casos de uso: Registro, Inicio/Login, Mis coches, Nuevo Coche, Buscar coche, Coches seleccionados, Perfil y Logout.

4.1 REGISTRO

En el primer acceso del usuario a la aplicación, se presenta el registro como la manera en que dicho usuario pretende conseguir acceso a la aplicación. En una misma vista de la aplicación, se contempla tanto el Login como el Registro, con la diferencia de que una vez el usuario inserte su correo y contraseña y haga click en registro, aparecerán otros campos a rellenar en la misma vista, en los que el usuario deberá seleccionar un nombre y un número de teléfono así como repetir de nuevo la contraseña para asegurarse de que la ha escrito correctamente.

Una vez presentado el formulario de registro, el usuario envía la información al servidor de la base de datos, pero antes de que esto se produzca, la aplicación comprobará de forma local si ambas contraseñas son iguales; si no lo son, arrojará un error explicando el motivo, pero si son iguales, se procede finalmente al envío de la información a la base de datos, la cual creará una nueva entrada en el listado privado de Firebase de usuarios registrados con email y contraseña, así como otra entrada en el listado de la aplicación de usuarios donde incluirá todos sus datos.

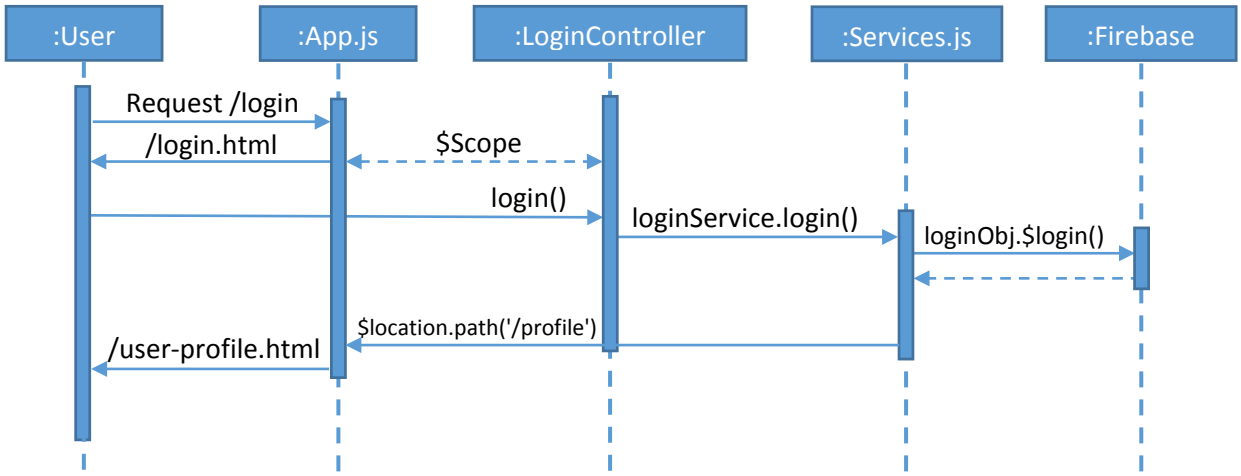
Tras el registro, la aplicación realizará el login automáticamente del usuario recientemente registrado, accediendo al menú principal de la aplicación. Al realizar el login, la aplicación carga en rootScope todos los datos del usuario desde un servicio; de esta forma, los datos del usuario que se ha autenticado se mantendrán todo el tiempo durante la aplicación hasta que el usuario realice el logout.



4.2 LOGIN

Los usuarios que ya se han registrado anteriormente, tendrán que autenticarse para acceder a la aplicación. Para ello, se presenta el formulario anteriormente descrito, pero en este caso solo aparecerán los campos login y contraseña. Una vez el usuario haya rellenado los datos y pulse el botón de login, la aplicación comprobará en la base de datos si el usuario y contraseña introducidos son correctos, concediendo acceso al menú principal de la aplicación en caso de serlo, y en caso contrario, denegando el acceso y arrojando un mensaje de error dependiendo el tipo de fallo en login, bien sea por usuario inexistente, o por contraseña no válida.

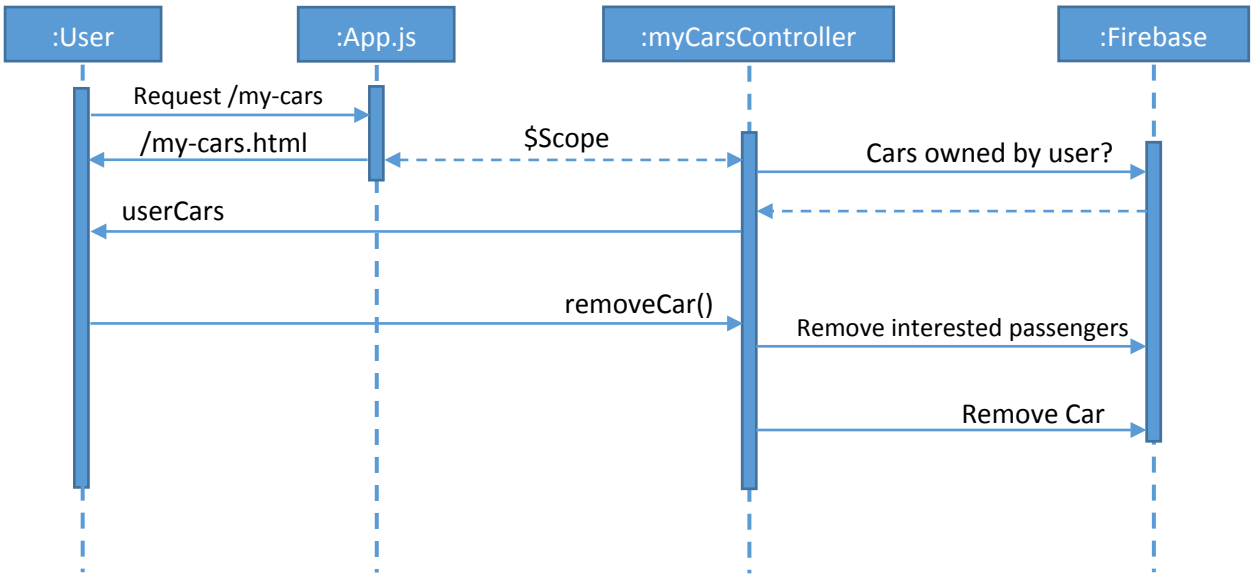
Al realizar el login, la aplicación carga en rootScope todos los datos del usuario desde un servicio; de esta forma, los datos del usuario que se ha autenticado en la aplicación se mantendrán todo el tiempo durante la aplicación hasta que el usuario realice el logout.



4.3 MIS COCHES

El usuario podrá seleccionar en el menú principal la opción de visualizar los coches que él mismo ha creado; al pulsar el botón correspondiente en el menú "Ver mis coches", la aplicación cargará la vista correspondiente, en la cual su controlador buscará entre todos los coches de la base de datos aquellos en los que el usuario que se encuentra autenticado en la aplicación sea el dueño, y los mostrará, así como los usuarios que se han interesado en cada uno de sus coches y sus números de teléfono correspondientes para poder contactar con ellos.

En esta misma página, por motivos de comodidad, se mostrará la opción de borrado de coche, la cual tras ser pulsado su botón correspondiente, procederá a la eliminación de dicho coche en la base de datos, así como de la eliminación de ese mismo coche como "coche seleccionado" en la base de datos en cada uno de los usuarios que habían mostrado interés por dicho coche.

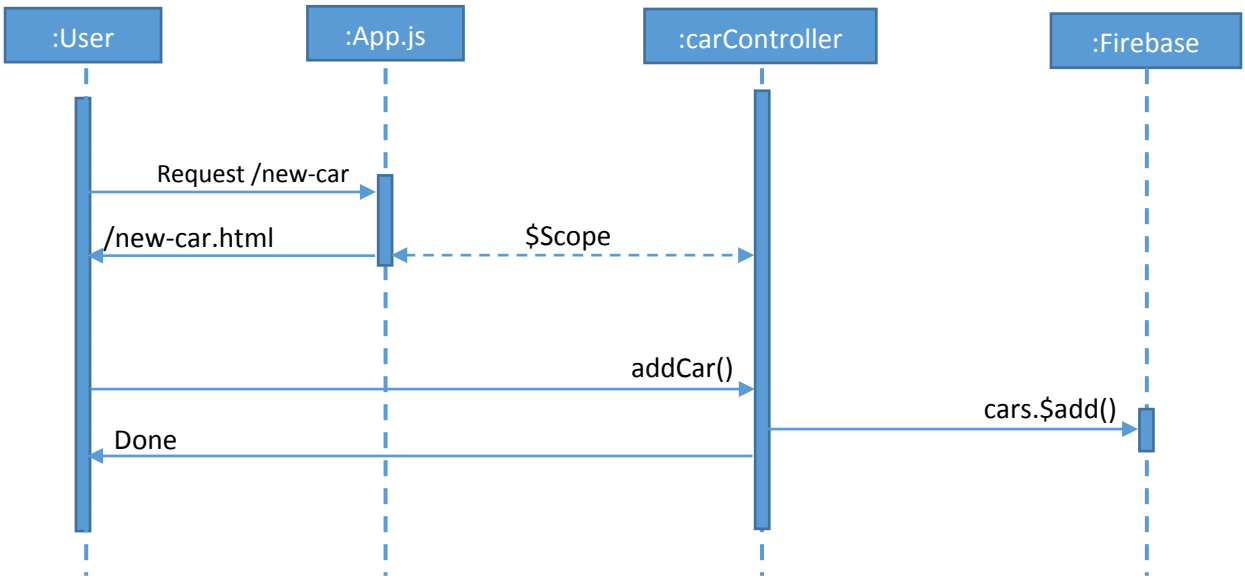


4.4 NUEVO COCHE

Cuando el usuario se dispone a crear un coche, tras pulsar el botón correspondiente y cargar su correspondiente vista y controlador, se mostrará en la aplicación el menú de creación de coche. Dicho menú consistirá en un formulario a completar por el usuario con diversos campos como son:

- Lugar de salida del coche
- Lugar de llegada del coche
- Fecha de salida
- Hora de salida
- Asientos disponibles en el vehículo
- Coste que tendrá cada pasajero por el viaje en coche

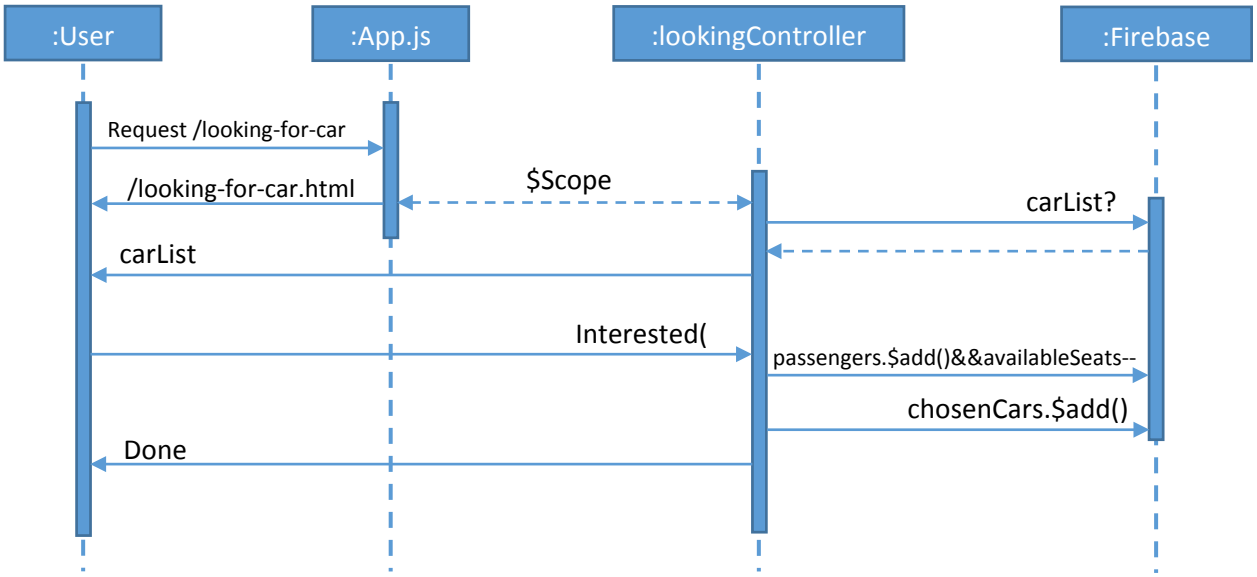
Una vez rellenos todos los campos, el usuario puede completar la creación mediante un botón. Una vez el usuario haga click en el mismo, se creará en los coches de la base de datos un nuevo coche gracias a una función destinada a ello en el controlador asociado, con todos estos datos así como el campo "dueño del coche", el cual se rellenará automáticamente con el nombre del usuario creador. A partir de este momento, este coche aparecerá en la búsqueda de coches y podrá ser seleccionado por tantas personas como asientos libres tenga.



4.5 BUSCAR COCHE

En este apartado se mostrarán todos los coches existentes en la base de datos, con todos sus datos, y junto a cada uno se mostrará el botón “Me interesa”. Dicho botón añadirá en la base de datos del usuario autenticado este coche como uno de sus coches seleccionados; además, restará uno al número de asientos disponibles y por último añadirá a dicho usuario como pasajero del coche en la base de datos.

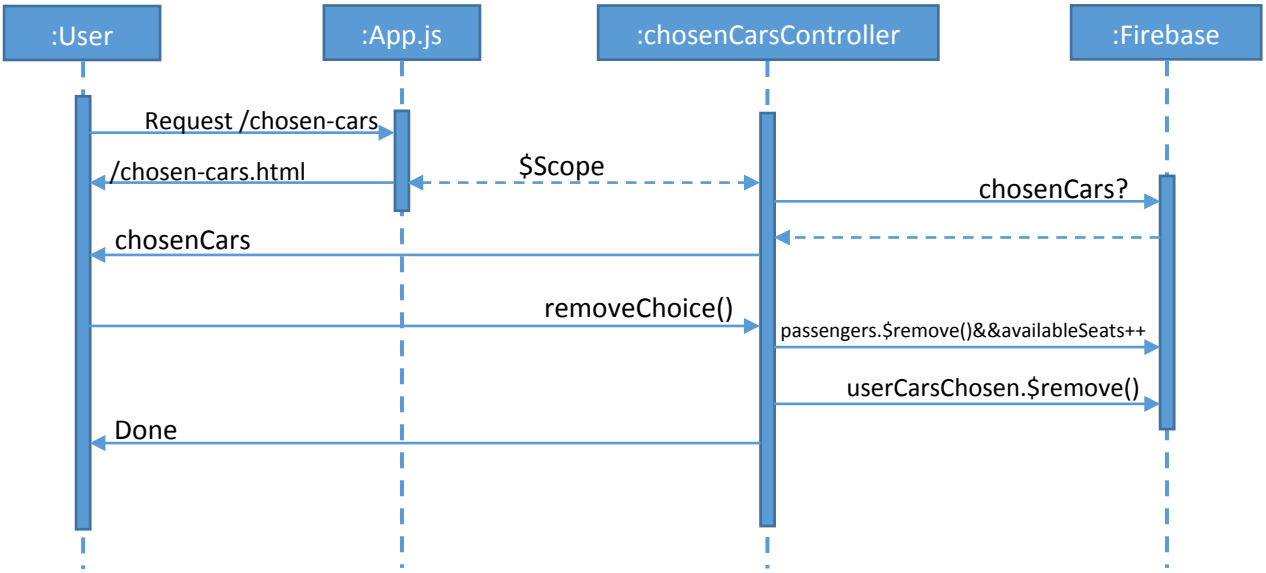
Si este coche ha sido creado por el usuario que se encuentra autenticado, el botón “Me interesa” no aparecerá. Por otro lado, si el usuario pulsa el botón y ya se había interesado anteriormente por ese mismo coche, aparecerá un mensaje de error indicándolo.



4.6 ADMINISTRAR COCHES SELECCIONADOS

Una vez que el usuario se ha interesado por uno o más coches, en el apartado "coches seleccionados" el usuario podrá visualizar gracias al controlador un listado de los coches por los que se ha interesado, con sus respectivos lugares de salida y destino, así como el número de teléfono de los dueños de dichos coches.

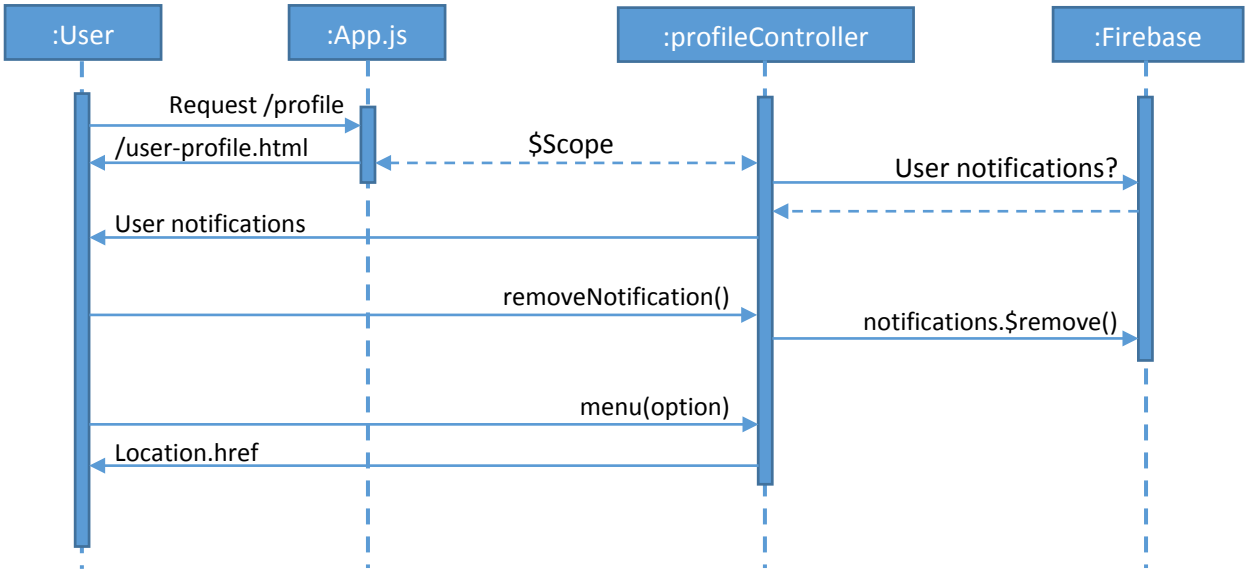
Además, el usuario contará con la posibilidad de eliminar su selección con un botón en cada coche. Al pulsarlo, el controlador asociado a esta vista ejecutará la función removeChoice, eliminando dicho coche de su listado de coches seleccionados así como su nombre de entre los pasajeros del coche.



4.7 PERFIL DEL USUARIO

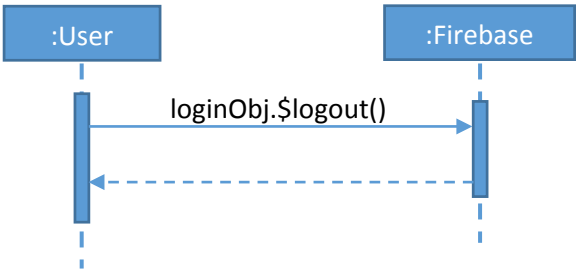
Este apartado se presenta como el menú principal de la aplicación, en el cual se mostrarán las opciones de menú del usuario, así como las notificaciones que pueda tener. El usuario contará con la posibilidad de borrar las notificaciones haciendo click en un botón de "entendido", el cual llamará a la función removeNotification() y se eliminarán de la base de datos.

Los botones de menú, en vez de ser links directos, al hacer click en ellos llaman a una función del controlador asociado que redirige al usuario a la url mediante el uso del servicio $location.



4.8 LOGOUT

En cualquier momento de la aplicación, el usuario podrá pulsar el botón “Logout”; al pulsarlo, el usuario hará una llamada al servicio $logout de Firebase, finalizando así sesión el usuario de la aplicación y borrando todas las variables de sesión como loginObj de rootScope y denegándole por tanto el acceso a cualquier apartado de la aplicación que no sea el de Login/Registro o Inicio.



5 IMPLEMENTACIÓN DE LA APLICACIÓN

A lo largo de todo el desarrollo de la aplicación, se ha hecho uso de diversos recursos bibliográficos, desde ejemplos básicos de aplicaciones [3] y paso de datos entre controladores [8], hasta estudio de técnicas utilizadas en aplicaciones más completas del servidor GitHub de angular [5] así como del servidor GitHub de AngularFire [13], con técnicas como el uso de servicios para almacenar funciones de autenticación. Además, se ha realizado un continuo uso tanto de la referencia de la API de Angular [4], como de la guía oficial de Angular [6]. También se debe mencionar el importante aporte formativo del canal de youtube de un programador de AngularJS junto con Firebase [7].

5.1 INDEX.HTML

La base de la aplicación se construirá en el index.html, ya que durante toda la aplicación el usuario se encuentra en él, y actuará a modo de frame, abriendo el resto de vistas dentro de unos de sus elementos del DOM.

Las partes a resaltar de dicho documento son las siguientes:

```
<html lang="es"  ng-app="tfgApp">
```

En esta línea se define con una directiva de Angular (ng-app) que la aplicación de Angular se encuentra dentro de dicho elemento del DOM, en este caso, se ha colocado en el html, el elemento que lo engloba todo, ya la aplicación se encontrará dentro de toda la página HTML.

```
<link href="http://getbootstrap.com/dist/css/bootstrap.min.css"
rel="stylesheet">
```

En esta línea de código se indica que para todo el documento, y por tanto para toda la aplicación, dicha URL es la que contiene el documento CSS en el cuál se basa todo el formato que se da a lo largo de todo el código. Cada atributo que se coloca en cada elemento del DOM, como class=”ejemplo”, lo buscará en dicha hoja CSS.

```
<script src="lib/angular/angular.js"></script>
```

```
<script src="lib/angular/angular-route.js"></script>
<script src="lib/angular/i18n/angular-locale_es-es.js"></script>
```

En estas tres líneas se indica dónde se encuentra el código de Angular necesario para hacer funcionar la aplicación, el básico (Angular.js), el documento para hacer funcionar el sistema ya mencionado de vista-ruta-controlador (Angular-route.js) y por último el encargado de mostrar correctamente el formato de horas, monedas y fechas en cada país (Angular-locale_es-es.js, para España en este caso).

```
<script src="js/controllers.js"></script>
<script src="js/app.js"></script>
<script src="js/services.js"></script>
```

En estas tres líneas se indica dónde se encuentran los principales scripts, en este caso el que contiene todo el código de los controladores, el de la base de la aplicación con el config y las rutas, y por último el que contiene el código de los servicios para inyectar datos en los diferentes controladores.

```
<script src="https://cdn.firebase.com/js/client/1.0.6/firebase.js"></script>
<script
src="https://cdn.firebase.com/libs/angularfire/0.7.1/angularfire.js"></script>
<script src="https://cdn.firebase.com/js/simple-login/1.3.0/firebase-simple-
login.js"></script>
```

Por último, en estas tres líneas se incluyen los scripts correspondientes a la base de datos. El básico, para hacer funcionar todos los comandos de la base de datos (firebase.js), en segundo lugar el encargado del funcionamiento de las funciones específicas de firebase para Angular (Angularfire.js) y el encargado del funcionamiento de todas las funciones de login, registro y logout en la base de datos, también de la mano de Firebase (firebase-simple-login.js). Cada uno en una versión específica para ser compatibles entre sí, ya que no todas las versiones son compatibles entre sí.

AngularFire Version	Firebase Version	Simple Login Version	Angular Version
0.3.0 - 0.5.0	v0	v0	1.1.2
0.6.0	1.0.2	1.2.3	1.1.2
0.7.0	1.0.5	1.2.5	1.2
0.7.1	1.0.6	1.3.0	>= 1.2, < 1.3.0
0.8.0	1.0.15+	1.3.0+	1.2.18+

Tabla 4 - Relación de compatibilidad entre las diferentes versiones de Firebase, AngularFire, Angular y SimpleLogin

```
<!-- NAVBAR================================================== -->
<div class="navbar-wrapper">
.
.
.
.
</div>
```

En esta parte del código se encuentra la barra superior de menú, la cual no cambiará su código durante toda la aplicación. Dicha barra contendrá algunos botones de navegación, como son el link durante todo momento al inicio de la aplicación, así como la posibilidad de login si no existe ningún usuario autenticado o la posibilidad de logout en caso de haberlo. Además, se mostrará el usuario autenticado en todo momento.

```
<div ng-view></div>
```

Esta simple línea es la más importante de todo el documento, ya que es en la que se indica con una directiva de Angular (ng-view) que dentro de este elemento del DOM es donde las vistas correspondientes a cada ruta se mostrarán.

```
<!-- FOOTER -->
```

```
<footer>
</footer>
```

Por último, para mantener siempre un pie de página, se ha colocado un footer que permanecerá constante al igual que la barra superior de navegación, lo cual es una buena práctica en HTML5.

Los scripts serán colocados posteriormente al final del documento, de manera que la aplicación web no tiene que esperar a leer todos los scripts antes de ser mostrada, lo que dará una "falsa impresión" de rapidez en caso de que los scripts puedan ralentizar la apertura de la aplicación.

Nota: Para restringir el acceso a las vistas cuando no se encuentra un usuario autenticado en la aplicación, se comprueba el estado de la variable user de rootScope; en caso de no existir, no se mostrará la vista.

```
<div ng-show='loginObj.user === null'>
   <p>No estás autenticado en la aplicación</p>
   <a href='#/login'><button>Login</button></a>
</div>
```

5.2 APP.JS

En este archivo se encuentra el núcleo de la aplicación.

```
var tfgApp = angular.module('tfgApp', ['ngRoute', 'tfgControllers', 'tfgServices']);
```

En la primera línea se crea tfgApp, que será el módulo principal de la aplicación, en el cual se incluirán como dependencias ngRoute para hacer uso de las vistas en función de la ruta, tfgControllers, archivo que contiene todos los controladores los cuales se van a referenciar en cada ruta, y tfgServices, archivo que contiene los servicios. Este módulo engloba toda la aplicación y es el núcleo de la misma.

A continuación se realiza el config, donde por medio del anteriormente explicado $routeProvider se realizará todo el sistema de rutas-vistas-controladores:

```
tfgApp.config(['$routeProvider',
function($routeProvider) {
$routeProvider.
when('/', {
templateUrl: 'partials/identification.html',
controller: 'loginController'
}).
when('/profile', {
templateUrl: 'partials/user-profile.html',
controller: 'profileController'
}).
when('/login', {
templateUrl: 'partials/login.html',
controller: 'loginController'
}).
when('/my-cars', {
templateUrl: 'partials/my-cars.html',
controller: 'myCarsController'
}).
when('/chosen-cars', {
templateUrl: 'partials/chosen-cars.html',
controller: 'chosenCarsController'
}).
when('/new-car', {
```

```
templateUrl: 'partials/new-car.html',
controller: 'carController'
}).
when('/looking-for-car', {
templateUrl: 'partials/looking-for-car.html',
controller: 'lookingController'
}).
otherwise({
redirectTo: '/'
});
}]);
```

Y por último, con .run se podrá configurar que suceda algo al arrancar la aplicación; en este caso, pasando como dependencia los datos que se inyectan desde el servicio loginService y rootScope, se indica que asigne a la variable loginObj de rootScope la variable que retornará la función init del servicio loginService, la cual será una referencia a la dirección de la base de datos, con el objetivo de mantenerla "viva" durante toda la aplicación, ya que rootScope se conservará durante todo el ciclo de vida de la misma. Además, ya que se hace continuamente uso de la referencia del listado de coches de Firebase, se guarda la misma en una variable de $rootScope para un cómodo acceso en los controladores.

```
tfgApp.run(['loginService', '$rootScope', '$firebase', function(loginService, $rootScope, $firebase) {
    $rootScope.loginObj = loginService.init('/login');
    var refCars = new Firebase ("https://tfg.firebaseio.com/cars");
    $rootScope.cars = $firebase(refCars);
}]);
```

5.3 CONTROLLERS.JS

Este archivo contendrá todo el código asociado los controladores de cada vista, aquí es donde se encuentra la lógica de la aplicación. Se puede resaltar la primera línea, donde se ha creado tfgControllers, un módulo cuya única dependencia será firebase. Añadiendo esta dependencia, se podrá trabajar con firebase en los controladores de la aplicación.

```
var tfgControllers = angular.module('tfgControllers', ["firebase"]);
```

Una vez creada tfgControllers, se podrá empezar a crear controladores, en los que se modificarán los Scopes y por tanto se encontrarán todas las funciones y lógicas de la aplicación:

```
tfgControllers.controller('nombreDelControlador', ['', function() {}];
```

Para esta aplicación se han creado seis controladores, uno para cada vista, como se ha visto en el archivo app.js. Pero hay un código común a todos para controlar que el usuario esté logueado al acceder a las vistas, y en caso de no estarlo, se le redirija:

```
if($rootScope.loginObj.user==null){
      $location.path('/login');
}
```

Por tanto, todos los controladores tendrán como dependencia el servicio $location.

5.3.1 'profileController'

```
tfgControllers.controller('profileController', ['$scope', '$rootScope', '$firebase',
'$location','loginService',function($scope,$rootScope,$firebase,$location,loginSer
vice){
```

Este controlador se encargará de la vista principal de la aplicación; una de las funciones de las que se encargará es de llamar al servicio loginService en caso de que el usuario quiera cambiar la contraseña de la cuenta, previa comprobación de que todos los datos introducidos son correctos. Por tanto necesitaremos añadir como dependencias loginService, $firebase y $rootScope:

```
$scope.changePassword = function (){
```

```
        $rootScope.loginError = null;
        if( !$scope.oldPass ) {
           $rootScope.loginError = 'Debe introducir la contraseña actual';
        }
        else if( !$scope.newPass ) {
           $rootScope.loginError = 'Debe introducir una nueva contraseña';
        }
        else if( $scope.newPass !== $scope.newPass2 ) {
           $rootScope.loginError = '¡Las contraseñas deben ser iguales!';
        }
        else{

loginService.changePassword($rootScope.user.email,$scope.oldPass,$scope.new
Pass);
            $scope.oldPass = null;
            $scope.newPass = null;
            $scope.newPass2 = null;
        }
   };
```

Además, este controlador será el encargado de manejar las notificaciones del usuario y mostrarlas, y una vez el usuario las haya visualizado podrá borrarlas llamando a la función correspondiente del controlador pasando la propia notificación como argumento:

```
var refNotifications = new Firebase
("https://tfg.firebaseio.com/users/"+$rootScope.userKey+"/notifications/");
$scope.notifications = $firebase(refNotifications);
$scope.keys = $scope.notifications.$getIndex();
```

```
$scope.removeNotification = function(notification){
      angular.forEach($scope.keys, function(key) {
         if($scope.notifications[key]===notification){
            $scope.notifications.$remove(key);
         }
      });
      $scope.keys = $scope.notifications.$getIndex();
```

Por último, el controlador dispone de una función para el control de las acciones de los botones:

```
$scope.menu = function (option){
      if(option==='new-car'){
         location.href = '#/new-car';
      }
      if(option==='my-cars'){
         location.href = '#/my-cars';
      }
      if(option==='chosen-cars'){
         location.href = '#/chosen-cars';
      }
      if(option==='looking-for-car'){
         location.href = '#/looking-for-car';
      }
}
```

5.3.2 'loginController'

```
tfgControllers.controller('loginController', ['$scope', '$firebase', 'loginService',
'$location', function($scope, $firebase, loginService,$location) {
```

Este controlador se encargará de realizar las funciones Login y createAccount, pero como se verá más detenidamente en el apartado Login de la base de datos, simplemente comprobará que no existen errores en el formulario rellenado por el usuario y hará uso del servicio loginService para realizar realmente dichas funciones. Se puede ver cómo además de $scope, se han añadido las dependencias $firebase y loginService para poder hacer uso de referencias a firebase y de las funciones del servicio loginService.

```
$scope.login = function() {
            $rootScope.loginError = null;
            if( !$scope.email ) {
              $rootScope.loginError = 'Porfavor introduzca una dirección email';
            }
            else if( !$scope.pass ) {
              $rootScope.loginError = 'Debe introducir una contraseña';
            }
            else {
              loginService.login($scope.email, $scope.pass);
            }
      };
```

```
    $scope.createUser = function() {
          $rootScope.loginError = null;
          if( !$scope.email ) {
            $rootScope.loginError = 'Porfavor introduzca una dirección email';
          }
          else if( !$scope.pass ) {
            $rootScope.loginError = 'Debe introducir una contraseña';
          }
          else if( $scope.pass !== $scope.confirm ) {
            $rootScope.loginError = '¡Las contraseñas deben ser iguales!';
          }
          else{
            loginService.createAccount($scope.email, $scope.pass, $scope.name,
$scope.phone);
          }
    };
```

5.3.3 'myCarsController'

```
tfgControllers.controller('myCarsController', ['$scope', '$rootScope',
'$firebase','$location', function($scope, $rootScope, $firebase,$location){
```

Este controlador en el cual se han incluido además de $scope, $rootScope y $firebase, ya que se utilizan variables de $rootScope y se hacen referencias a la base de datos, se encargará de realizar una búsqueda de los coches cuyo dueño es el usuario autenticado por medio de un bucle Angular.forEach que recorrerá todos los coches de la base de datos, para así mostrarlos al mismo en su correspondiente vista:

```
  var refCars = new Firebase ("https://tfg.firebaseio.com/cars");
  $scope.cars = $firebase(refCars);
  $scope.myCars= new Array();
  var keys = $scope.cars.$getIndex();
  var i=0;
  angular.forEach(keys, function(key) {
    if($scope.cars[key].owner===$rootScope.user.name){
      $scope.myCars[i]=$scope.cars[key];
      i++;
    }
```

```
});
```

Además, este controlador será el encargado de ejecutar la función de eliminación de un coche (removeCar), la cual podrá ser ejecutada por el usuario en esta misma vista:

```
$scope.removeCar = function (car,index){
    $scope.myCars.splice(index,1);
    $scope.passengerNames = new Array();
    keys = $scope.cars.$getIndex();
    angular.forEach(keys, function(key) {
      if($scope.cars[key]===car){
        if(angular.isDefined(car.passengers)){
          var refPassengers = new Firebase
("https://tfg.firebaseio.com/cars/"+key+"/passengers/");
          var passengers = $firebase(refPassengers);
          var pKeys = passengers.$getIndex();
          var j=0;
          angular.forEach(pKeys, function(pKey) {
            $scope.passengerNames[j]=passengers[pKey].name;
            j++;
          });
          var refUsers = new Firebase ("https://tfg.firebaseio.com/users");
          var users = $firebase(refUsers);
          var uKeys = users.$getIndex();
          angular.forEach(uKeys, function(uKey) {
            for(var l=0;l<$scope.passengerNames.length;l++){
              if(users[uKey].name===$scope.passengerNames[l]){
                var refChosenCars = new Firebase
("https://tfg.firebaseio.com/users/"+uKey+"/chosenCars/");
                var chosenCars = $firebase(refChosenCars);
                var cKeys = chosenCars.$getIndex();
                angular.forEach(cKeys, function(cKey) {
                  if(chosenCars[cKey].keyCar===key){
                    chosenCars.$remove(cKey);
                  }
                });
                var refNotifications = new Firebase
("https://tfg.firebaseio.com/users/"+uKey+"/notifications/");
                var notifications = $firebase(refNotifications);
```

```
                    notifications.$add({
                      "notificationBad":'El usuario "'+car.owner+'" ha borrado el
coche en el que ibas a viajar con destino "'+car.destination+'" el día '+car.date+' a
las '+car.time+'.'
                    });
                  }
                }

              });
            }

            $scope.cars.$remove(key);
          }
    });
    }
```

Esta función, además de borrar el coche, almacenará sus pasajeros en una variable temporal para posteriormente a su borrado, buscarlos y borrar el coche de entre los coches por los que se habían interesado. Por último añadirá una notificación a cada uno de los pasajeros informando de que el coche se ha borrado.

5.3.4 'chosenCarsController'

```
tfgControllers.controller('chosenCarsController', ['$scope', '$rootScope',
'$firebase' , '$location', function($scope, $rootScope, $firebase, $location){
```

Este controlador se utilizará en la vista “coches seleccionados”, por tanto será el encargado de buscar las referencias de los coches por los que el usuario se ha interesado, las cuales estarán en el campo chosenCars de los usuarios de la base de datos como se verá en el apartado correspondiente de base de datos, y mostrar todos los coches que corresponden con dichas referencias, así como los datos del dueño para poder contactar con él.

```
$scope.done = false;
    var refCars = new Firebase ("https://tfg.firebaseio.com/cars");
    $scope.cars = $firebase(refCars);
    var refUserCarsChosen = new Firebase
("https://tfg.firebaseio.com/users/"+$rootScope.userKey+"/chosenCars");
    var userCarsChosen = $firebase(refUserCarsChosen);
    $scope.chosenCars= new Array();
```

```
  $scope.chosenCarsKeys = new Array();
  var i=0;
  var keys = userCarsChosen.$getIndex();
  angular.forEach(keys, function(key) {
    $scope.chosenCarsKeys[i]=userCarsChosen[key].keyCar;
    i++;

  });
  var l=0;
  keys = $scope.cars.$getIndex();
  angular.forEach(keys, function(key) {
    for(var j=0;j<$scope.chosenCarsKeys.length;j++){
      if($scope.chosenCarsKeys[j]===key){
        $scope.chosenCars[l]=$scope.cars[key];
        l++;
      }
    }
  });
```

Además, este controlador será el encargado de realizar la función removeChoice(), la cual será ejecutada por el usuario cuando, en esta misma vista, pulse el botón para dejar de mostrar interés por este coche. Se trata de una función compleja, ya que no solo deberá borrar la referencia del coche de entre los coches por los que el usuario se ha interesado, si no que también deberá encontrar el coche en la base de datos, y borrar a este usuario de entre los pasajeros del coche (se puede ver con más detalle el documento coches en su apartado correspondiente en base de datos), así como añadirle un asiento disponible. Por último añadirá una notificación al dueño del coche avisando de que este pasajero se ha dado de baja.Se expone a continuación el código de dicha función:

```
  $scope.removeChoice = function (index) {
    $scope.done = true;
    $scope.chosenCars.splice(index,1);
    keys = userCarsChosen.$getIndex();
    angular.forEach(keys, function(key) {
      if(userCarsChosen[key].keyCar===$scope.chosenCarsKeys[index]){
        var refPassengers = new Firebase
("https://tfg.firebaseio.com/cars/"+userCarsChosen[key].keyCar+"/passengers/")
;
```

```
            var passengers = $firebase(refPassengers);
            var pKeys = passengers.$getIndex();
            angular.forEach(pKeys, function(pKey) {
               if(passengers[pKey].name===$rootScope.user.name){
                  passengers.$remove(pKey);
                  $scope.cars[userCarsChosen[key].keyCar].availableSeats ++;
                  $scope.cars.$save(userCarsChosen[key].keyCar);
               }
            });
            userCarsChosen.$remove(key);

         }
      });
            var refNotifications = new
Firebase("https://tfg.firebaseio.com/users/"+car.ownerKey+"/notifications/");
            var notifications = $firebase(refNotifications);
            notifications.$add({
               "notificationBad":'El usuario "'+$rootScope.user.name+'" HA DEJADO
DE MOSTRAR INTERÉS por tu coche con destino "'+car.destination+'" el día
'+car.date+' a las '+car.time+'.'
            });
   }
```

5.3.5 'carController'

```
tfgControllers.controller('carController',['$scope','$rootScope','$firebase','$locati
on', function($scope,$rootScope,$firebase,$location){
```

Este controlador será el encargado de la vista de creación de coches, por tanto su función será tener una referencia a los coches de la base de datos así como una función constructora de coches. Hará uso por tanto de las dependencias $scope, $rootScope y $firebase. Además, para que la aplicación guarde los coches ya creados por el usuario por si quiere volver a crear uno con los mismos datos, se tendrá una referencia a los “createdCars”, y se crearán cada vez que se cree un coche nuevo.

```
   var refCars = new Firebase ("https://tfg.firebaseio.com/cars");
   var cars = $firebase(refCars);
   $scope.done = false;
   $scope.reuse = false;
   if($rootScope.loginObj.user){
```

```
    $scope.owner = $rootScope.user.name;
    $scope.contactNumber = $rootScope.user.phoneNumber;
    $scope.ownerKey = $rootScope.userKey;
  }
```

Se puede resaltar que rellena automáticamente los campos dueño del coche y número del dueño del coche con los datos del usuario autenticado, ya que son datos que tendrá que escribir en la base de datos en el apartado de coches si el usuario finalmente crea algún coche.

Y por supuesto, contará con la función constructora de coches, la cual creará un coche o dos, dependiendo de si el usuario selecciona que también desea crear el coche de retorno al punto de origen, teniendo que seleccionar solo la fecha y la hora para la vuelta. Esta función también construirá el coche en el área de coches creados del usuario para poder reusar sus datos, siempre que el coche creado no sea ya reusando datos de uno creado anteriormente:

```
$scope.addCar = function () {
    $rootScope.cars.$add({
      "from": $scope.from,
      "destination": $scope.destination,
      "owner": $scope.owner,
      "ownerKey":$scope.ownerKey,
      "contactNumber":$scope.contactNumber,
      "seats": $scope.seats,
      "availableSeats": $scope.seats,
      "price": $scope.price,
      "date": $scope.date,
      "time":$scope.time
    });

    if($scope.return){
      $rootScope.cars.$add({
        "from": $scope.destination,
        "destination": $scope.from,
        "owner": $scope.owner,
        "ownerKey":$scope.ownerKey,
        "contactNumber":$scope.contactNumber,
        "seats": $scope.seats,
        "availableSeats": $scope.seats,
```

```
            "price": $scope.price,
            "date": $scope.date2,
            "time":$scope.time2
        });
    }
$scope.done=true;
    if($scope.reuse == false){
        $scope.createdCars.$add({
            "from": $scope.from,
            "destination": $scope.destination,
            "availableSeats": $scope.seats,
            "price": $scope.price
        });
    }   }
```

Finalmente, para poder hacer uso de los datos de los coches ya creado, el usuario dispone de una función para reusar un coche de entre el listado de los coches creados, la cual pondrá a “true” el flag para saber si se están reusando ya datos o no para no volver a crear el coche en “createdCars”:

```
  $scope.useCar = function (car){
        $scope.from = car.from;
        $scope.destination = car.destination;
        $scope.seats = car.availableSeats;
        $scope.price = car.price;
        $scope.reuse = true;
  }
```

Así como de una función para limpiar todos los coches de la lista “createdCars” del usuario. Esta función será activada desde un botón para ellos en la vista.

```
$scope.cleanCreatedCars = function(){
    $scope.createdCars.$remove();
}
```

5.3.6 'lookingController'

```
tfgControllers.controller('lookingController',['$scope','$rootScope','$firebase','$lo
cation', function($scope, $rootScope, $firebase,$location){
```

Por último, el controlador lookingController es el encargado de mostrar todos los coches de la base de datos en la vista “buscar coche”. Además, contiene una de

las funciones más importantes de la aplicación, la función “Me interesa”, mediante la cual un usuario se interesa por un coche. Por lo tanto, este controlador también hará uso de las dependencias $scope, $rootScope y $firebase.

El código del controlador sin la función interested() es muy sencillo, ya que salvo inicializar algunas variables lo único que debe hacer es referenciar los coches de la base de datos para mostrarlos en el DOM. Además contará con una función para el filtro de ordenamiento de los coches como se verá:

```
    $scope.done = false;
    $scope.error = null;
    var error1 = "!Ya estás interesado por este coche!";
    var error2="¡No quedan sitios libres en este coche!";
```

La función interested() será algo compleja, ya que no solo debe añadir al usuario autenticado como pasajero en el coche, si no que debe añadir dicho coche en el usuario en la base de datos como uno de los coches por los que se ha interesado. Por último, restará un asiento disponible al vehículo y añadirá una notificación al dueño del coche comunicándole que un nuevo usuario se ha unido al coche como pasajero:

```
    $scope.interested = function(carId){
      $scope.error = null;
      var refPassengers = new Firebase
("https://tfg.firebaseio.com/cars/"+carId+"/passengers/");
      var passengers = $firebase(refPassengers);
      var keys = passengers.$getIndex();
      angular.forEach(keys, function(key) {
        if(passengers[key].name===$rootScope.user.name)
          $scope.error = true;
      });
      if($scope.error){
        $scope.error = error1;
      }
      else{
        if($scope.cars[carId].availableSeats===0){
          $scope.error=error2;
        }
        else{
```

```
                console.log("else error!");
                $scope.done = true;
                $scope.cars[carId].availableSeats --;
                $scope.cars.$save();
                passengers.$add({
                    "name":$rootScope.user.name,
                    "phone":$rootScope.user.phoneNumber
                });
                var refChosen = new Firebase
("https://tfg.firebaseio.com/users/"+$rootScope.userKey+"/chosenCars/");
                var chosen = $firebase(refChosen);
                chosen.$add({
                    "keyCar":carId
                });
……….var refNotifications = new
Firebase("https://tfg.firebaseio.com/users/"+car.ownerKey+"/notifications/");
                var notifications = $firebase(refNotifications);
                notifications.$add({
                    "notificationGood":'El usuario "'+$rootScope.user.name+'" con
teléfono de contacto '+$rootScope.user.phoneNumber+' SE HA INTERESADO por
tu coche con destino "'+car.destination+'" el día '+car.date+' a las '+car.time+'.'
                });
            }
        }
    }
```

En esta función cabe destacar el control de errores, con el fin de evitar que el usuario se pueda interesar por un coche por el que ya se ha interesado anteriormente o por un coche cuyo dueño es el propio usuario. Los coches sin asientos disponibles no se mostrarán directamente, por lo que no supondrán un problema.

Por último, algo que también realiza este controlador es la obtención de la fecha actual para el posterior filtro de coches antiguos en la vista, como se explica en el apartado “filtros”:

```
    var todayDate =new Date();
    $scope.day=todayDate.getDate();
    $scope.month=todayDate.getMonth() + 1;
    $scope.year=todayDate.getFullYear();
```

5.4 SERVICES.JS

Con el fin de tratar toda la información relativa al usuario que ha iniciado sesión en la aplicación, se ha hecho uso de los servicios.

En primer lugar se define tfgServices como un módulo con la dependencia firebase como en el caso de los controladores.

```
var tfgServices = angular.module('tfgServices', ['firebase']);
```

Se quiere mantener viva la información del usuario autenticado durante todo el ciclo de vida de la aplicación, y poder inyectar esta información en los controladores que lo requieran, por tanto, se ha hecho uso de una fábrica denominada loginService, en la que se definen cuatro funciones: inicialización, login, logout y createAccount:

```
//Se crea la fábrica llamada loginService con todas las dependencias necesarias
tfgServices.factory('loginService', ['$rootScope',
'$firebase','$firebaseSimpleLogin','$location',
  function($rootScope, $firebase, $firebaseSimpleLogin, $location) {
//se inicializan ciertas variables que se usarán en la fábrica
    var loginObj = null;
    var refUsers = new Firebase ("https://tfg.firebaseio.com/users");
    var users = $firebase(refUsers);
    $rootScope.user = null;

     return {
//La función init se encargará de devolver una variable loginObj con la referencia
de la base de datos
       init: function() {
         var databaseRef = new Firebase('https://tfg.firebaseio.com');
         return loginObj = $firebaseSimpleLogin(databaseRef);
       },
```

La función login, la cual se verá más detenidamente más adelante en el apartado Login de base de datos, se encargará de recibir un email y una contraseña y realizar el login del usuario en el objeto loginObj de rootScope; además si se cumple la promesa de que el usuario se ha autenticado con éxito, y por lo tanto existe “user”

en rootScope, imprimirá por consola la uid del usuario que ha iniciado sesión; además se guardará la referencia del usuario que ha iniciado sesión en unas variables de rootScope para futura comodidad en los controladores y finalmente se redirigirá al usuario a su página de perfil (/profile). En caso de error en el login, se imprimirá el tipo de error por consola.

```
login: function(email, pass) {
  loginObj.$login('password', {
    email: email,
    password: pass
  }).then(function(user) {
      console.log('Logged in as: ', user.uid);


      var keys = users.$getIndex();
      angular.forEach(keys, function(key) {
         if(users[key].id===user.uid){
            $rootScope.user = users[key];
            $rootScope.userKey = key;
         }
      });
     $location.path('/profile');
   }, function(error) {
      console.error('Login failed: ', error);
   });


},
```

La función logout finalizará la sesión del actual usuario; además, pondrá a null la variable user de rootScope.

```
logout: function() {
  loginObj.$logout();
   $rootScope.user = null;
},
```

La función createAccount, recibirá todos los datos de registro y procederá a la creación del usuario en la base de datos si todo es correcto y finalmente realizará

el login. En caso contrario, arrojará un error. También se verá con más detenimiento en el apartado “Login” de base de datos.

```
        createAccount: function(email, pass, name, phone) {
          loginObj.$createUser(email, pass)
              .then(function(user) {
                 console.log('User created as: ', user.uid);
                 users.$add({
                    "name": name,
                    "email": email,
                    "id": user.uid,
                    "phoneNumber": phone,
                    "chosenCars": false
                 });

                 loginObj.$login('password', {
                   email: email,
                   password: pass
                 }).then(function(user) {
                    console.log('Logged in as: ', user.uid);

                    var keys = users.$getIndex();
                    angular.forEach(keys, function(key) {
                       if(users[key].id===user.uid){
                          $rootScope.user = users[key];
                       }
                    });
                   $location.path('/profile');
                 }, function(error) {
                    console.error('Login failed: ', error);
                 });
              }, function(error) {
                console.error('User creation failed: ', error);
            });

        }
      };
}]);
```

Por último, el servicio loginService contará con la función para cambiar la contraseña del usuario, la cual será llamada desde la vista del perfil del usuario. Previamente en dicha vista se comprobará que los datos son válidos en formato, por tanto la función changePassword cambiará la contraseña antigua por la nueva del usuario que se le indique en el campo email, que será el del usuario autenticado. Tanto en caso de éxito como en caso de error, se guardará en una variable de $rootScope para notificarlo al usuario.

```
changePassword: function(email, oldPass, newPass){
        $rootScope.loginError = null;
        loginObj.$changePassword(email, oldPass, newPass)
          .then(function(user){
            console.log('Password change succeded');
            $rootScope.change = true;
          }, function(error){
            console.error('Password change failed: ', error);
            $rootScope.loginError = error;
          });
      },
```

5.5 ROOTSCOPE

Uno de los pilares de la aplicación es $rootScope. Este contexto de trabajo es un $Scope global a toda la aplicación, cuyos datos se guardarán en todo momento mientras la aplicación se esté ejecutando. Por tanto, aquí es donde se guardará en la variable $rootScope.loginObj la referencia al usuario que se loguea como se verá más adelante en el apartado "Login" de base de datos. Además, se guardará también en una variable de $rootScope la referencia al listado de coches de la base de datos, ya que se hará uso de ella continuamente en todos los controladores. Estas referencias se crearán en el arranque de la aplicación como se ha visto en el apartado App.js:

```
tfgApp.run(['loginService', '$rootScope', function(loginService, $rootScope) {
    $rootScope.loginObj = loginService.init('/login');
    var refCars = new Firebase ("https://tfg.firebaseio.com/cars");
    $rootScope.cars = $firebase(refCars);
}]);
```

Donde la función init del servicio loginService realizará el binding de la variable de rootScope loginObj con la URL de la base de datos. Al realizar de esta manera el binding, en una función aparte desde un servicio, se consigue una aplicación mucho más limpia y versátil.

```
init: function() {
        var databaseRef = new Firebase('https://tfg.firebaseio.com');
        return loginObj = $firebaseSimpleLogin(databaseRef);
},
```

Otra de las variables que se utiliza es $rootScope.user, donde se almacena en todo momento el nombre del usuario autenticado, lo cual será muy útil a la hora de saber quién se encuentra autenticado y poder trabajar con ese usuario en cada controlador. Esto ocurrirá en la función login del servicio loginService que será tratada con detenimiento en el apartado Login de base de datos. También se hará uso continuo para el control de errores en el login, registro y cambio de contraseña, como también se verá en el mismo apartado.

```
login: function(email, pass) {
          loginObj.$login('password', {
            email: email,
            password: pass
```

```
}).then…….
```

5.6 DIRECTIVAS MÁS UTILIZADAS

A la hora de mostrar al usuario todos los datos con los cuales trabajan los controladores, lo cual es al fin y al cabo una parte muy importante de la aplicación, se ha hecho uso en cada una de las vistas de ciertas directivas de Angular:

5.6.1 Ng-view

Esta directiva se puede colocar en cualquier elemento del DOM que se quiera; dentro de dicho elemento, se cargarán las diferentes vistas de la aplicación desde el servicio $routeProvider. En index.html se encuentra el elemento de la aplicación donde se cargan las vistas:

```
<div ng-view></div>
```

5.6.2 Ng-controller

Una de las directivas más utilizadas; se colocará en cualquier elemento del DOM, y en ella se especificará el controlador que se encargará de trabajar con los datos que se quieran cargar en esa vista. En esta aplicación no se hace uso de dicha directiva ya que se trabaja con el sistema de routeProvider, el cual especifica el controlador para cada vista en el documento app.js, pero en caso de usarse sería de la siguiente forma:

```
<div ng-controller="loginController">
```

5.6.3 ng-show/ng-hide

Estas dos directivas son realmente útiles; al igual que las anteriores, se podrán escribir en cualquier elemento del DOM. En ellas se especificará una condición, y dependiendo de si se cumple o no dicha condición, y de si se usa *show* o *hide*, se le indicará al navegador que muestre o no todo el código que contiene dicho elemento del DOM. Es posible, en vez de indicar una condición en la directiva, simplemente poner una variable, y la condición se cumplirá si la variable no es *null* o *false*.

Resaltar que aunque esta directiva indique al navegador que no muestre dicho código, lo cual realizará añadiendo la clase hide al elemento del DOM, dicho código existirá en el DOM y será interpretado por el navegador.

Ambas se usan continuamente a lo largo de toda la aplicación, un ejemplo que se encuentra en casi todas las vistas es para no mostrar la vista en caso de que no se encuentre ningún usuario autenticado:

```
<div ng-show='loginObj.user != null'></div>
```

5.6.4 ng-if

La directiva ng-if realizará la misma función que la anterior directiva ng-show, mostrará el código del elemento del DOM en el que se encuentra si la condición de la directiva se cumple. Pero con una gran diferencia, ya que en caso de no cumplirse la condición, dicho código no se llegará a generar en el DOM que interpreta el navegador.

5.6.5 ng-repeat

Esta directiva es muy útil a la hora de mostrar listados de objetos. Cuando se tiene por ejemplo un listado de coches en una variable y se quiere mostrar por completo, simplemente se escribirá en la directiva ng-repeat dicha variable, el cual es importante que sea un nombre en plural. Al realizarlo, se repetirá el elemento del DOM en el que se encuentre la directiva tantas veces como elementos tiene la variable. Dentro del elemento del DOM en el que se encuentra la directiva, y por tanto el cual se repetirá, se podrá hacer referencia al objeto repetido como el singular de la variable, y realizar así la estructura que se quiere repetir así como los datos a mostrar de cada objeto del listado.

Además, angularFire incluye el filtro orderByPriority para esta directiva, el cual crea un $id para cada elemento con la id que cada elemento del listado tiene en la base de datos, muy útil a la hora de llamar funciones e indicar sobre qué elemento del listado debe trabajar la función que se ha llamado.

Además de $id, también se tendrán otras variables como $odd, $even, las cuales existirán para los elementos pares o impares del listado, pudiendo así identificar dichos elementos a la hora de mostrar el listado.

Un buen ejemplo se puede observar en la vista *looking-for-car* donde se utiliza esta directiva para repetir dentro de una listado todos los coches que hay en la base de datos; además, se da la posibilidad de mostrar interés por cada coche a

los usuarios con un botón que contiene una función a la cual se le pasa como argumento el anteriormente mencionado elemento $id:

```
<tr ng-repeat="car in cars | orderByPriority | filter:query">
        <td><div ng-show="(user.name != car.owner)"><button type="button"
class="btn btn-success btn-xs" ng-click="interested(car.$id)">Me
interesa!</button></div></td>
        <td>{{car.owner}}</td>
        <td>{{car.from | uppercase}}</td>
        <td>{{car.destination | uppercase}}</td>
        <td>{{car.date | date:'fullDate'}} a las {{car.time | date:'h:mma'}}</td>
        <td>{{car.price | currency}}</td>
        <td>{{car.availableSeats | number}} de {{car.seats | number}}</td>
</tr>
```

5.6.6 ng-click

Con la directiva ng-click se podrá asignar la llamada a una función del controlador que se encuentre en dicho momento activo. Solo se podrá incluir esta directiva en los elementos del DOM “clickables”, como botones o vínculos. Además de indicar la función que se quiere llamar en la directiva, también se incluirán los parámetros que se pasarán a la función, por ejemplo el id de dicho elemento (para una vez en el controlador saber sobre cuál objeto de todo el listado realizar la función).

En el ejemplo anterior se puede observar cómo se ha hecho uso de esta directiva para que el botón “me interesa” llamase a la función interested().

5.6.7 ng-model

Por último, esta directiva de obligatorio uso en cualquier aplicación de Angular, permitirá anidar a un elemento del DOM la variable del $Scope correspondiente que se escriba en ella. Por supuesto, el elemento del DOM deberá ser un input (para guardar en la variable lo que el usuario escriba en él), o algún tipo de checkbox (marcado – true en la variable, no marcado – false en la variable).

Esta directiva se utiliza frecuentemente en la aplicación, por ejemplo en el formulario de login/registro:

```
<form class="form-signin" role="form">
```

```
    <h2 ng-hide="createMode" class="form-signin-heading">Login:</h2>
    <h2 ng-show="createMode" class="form-signin-heading">Registro:</h2>
    <input type="email" ng-model="email" class="form-control"
placeholder="Dirección email" required="" autofocus="">
    <input type="password" ng-model="pass" class="form-control"
placeholder="Contraseña" required="">
    <input type="text" placeholder="Nombre" class="form-control" ng-
model="name" ng-show="createMode">
    <input type="tel" placeholder="Phone number" class="form-control" ng-
model="phone" ng-show="createMode">
    <input type="password" placeholder="Repetir contraseña" class="form-
control" ng-show="createMode" ng-model="confirm">
    <button class="btn btn-lg btn-success btn-block" ng-click="login()" ng-
hide="createMode">Login</button>
    <button class="btn btn-lg btn-success btn-block" ng-click="createMode =
true" ng-hide="createMode">Registro</button>
    <button class="btn btn-lg btn-success btn-block" ng-show="createMode" ng-
click="createUser()">Crear cuenta</button>
    <button class="btn btn-lg btn-success btn-block" ng-show="createMode" ng-
click="createMode = false">Cancelar</button>
    <div ng-show='loginError!=null' class="alert alert-danger"
role="alert">Ocurrió un error: {{loginError}}</div>
</form>
```

5.7 FILTROS UTILIZADOS

Todos los filtros utilizados en la aplicación se encentran en la vista *looking-for-car*:

```
<tr ng-repeat="car in cars | orderByPriority | filter:query |
orderBy:selectedOrder" ng-show="(car.availableSeats>0)&&((car.date |
date:'M')>=month)&&((car.date | date:'d')>=day)&&((car.date |
date:'yyyy')>=year)">
        <td><div ng-show="(user.name != car.owner)"><button type="button"
class="btn btn-success btn-xs" ng-click="interested(car.$id)">Me
interesa!</button></div></td>
        <td>{{car.owner}}</td>
        <td>{{car.from | uppercase}}</td>
        <td>{{car.destination | uppercase}}</td>
        <td>{{car.date | date:'fullDate'}} a las {{car.time | date:'h:mma'}}</td>
        <td>{{car.price | currency}}</td>
        <td>{{car.availableSeats | number}} de {{car.seats | number}}</td>
</tr>
```

Como se puede observar, aparte del filtro orderByPriority de la API de AngularFire ya comentado anteriormente, el primer filtro que se utiliza es “filter”, el cual filtrará todo el contenido del elemento DOM en el que se encuentre en función del contenido de la variable que se especifique en el filtro, en este caso la variable *query*. El contenido de esta variable se encontrará controlado por lo que escriba el usuario en un input, actuando en su conjunto como una búsqueda.

En segundo lugar, otro filtro muy importante es “orderBy:variable”, el cual ordenará la tabla según el campo que se indique en la variable. Para ello, se llama a una función del controlador asociado que dependiendo de dónde haga click el usuario, se pasa un argumento y ese argumento se guarda en la variable, haciendo por tanto que se ordene la tabla según el usuario lo prefiera.

A continuación se pueden observar los filtros “uppercase” para mostrar en mayúsculas el contenido de la variable, el filtro “date” para dar formato a las fechas (variante ‘fullDate’) y horas (variante ‘h:mma’), el filtro “currency” para dar formato a la moneda dependiendo de dónde se encuentre el usuario, y el filtro “number” para dar formato de número a la variable. Si este último filtro fuese para números decimales, se podría especificar cuántos decimales se quieren mostrar.

Por último, se puede observar como se ha utilizado la propiedad ng-show a modo de filtro para los coches cuyos asientos libres no son mayores de 0, o los coches cuya fecha es anterior a la actual (año, mes y día deben ser mayores que los actuales, que se obtienen en el controlador correspondiente).

5.8 BOOTSTRAP EN LA APLICACIÓN

Aunque ya se ha explicado qué es Bootstrap y para qué sirve, el papel que realiza Boostrap en la aplicación es muy importante.

En primer lugar, para hacer uso de esta herramienta, se deben añadir los CDN de Bootstrap, tanto del CSS como del JS, que servirá para ciertos efectos de despliegue de menú o paneles. Debido a que el documento bootstrap.js está basado en Jquery, también se debe añadir alguna versión de jquery para que funcione correctamente. Además, Bootstrap cuenta con algunas hojas de estilos adicionales para hacer uso de ciertos formatos, y en este caso se va a hacer de un documento CSS especial para formularios de Login o Registro (singing.css).

Teniendo en cuenta todo lo mencionado anteriormente, se añadirá el siguiente código en index.html para poder comenzar a trabajar con Bootstrap:

```
<link href="/css/bootstrap.css" rel="stylesheet">
<link href="/css/singin.css" rel="stylesheet">
<script src="https://ajax.googleapis.com/.../jquery.min.js"></script>
<script src="http://getbootstrap.com/dist/js/bootstrap.min.js"></script>
```

Las clases más utilizadas en la aplicación de bootstrap.css son:

5.8.1 Container

El primer elemento del que se ha hecho uso es la clase “container”; el elemento del DOM en el que se coloque esta clase será el que englobará todas las vistas, todos los demás elementos del DOM. La clase “container” consiste en que el elemento del DOM en el que se aplica dicha clase tenga un tamaño proporcional a la pantalla del dispositivo que está haciendo uso de la aplicación web. La clase “container” distingue cuatro rangos de screen mediante el uso de las reglas @import de CSS, menos de 768px (donde el tamaño del “container” será del 100% del ancho de la pantalla), entre 769 y 991 px (donde el tamaño será de 769px), entre 992 y 1199px (donde el tamaño será de 992px) y más de 1200px (donde el tamaño será de 1200px).

Otra posible clase de la que se podría haber hecho uso es “container-fluid”, la cual siempre ocupa el 100% de la pantalla, pero para el sistema de columnas y filas de

Bootstrap la clase “container” funciona mucho mejor como se verá a continuación.

5.8.2 Col y row

La clase “col” es la base del sistema de columnas de Bootstrap utilizado para optimizar una página web a la hora de realizar un “responsive design”. Este sistema básicamente divide el elemento del DOM en el que se esté trabajando en doce partes, y además tiene en cuenta los rangos de pantalla en los que se encuentra al igual que la clase “container”, nombrándolos de la siguiente forma:

- <768px rango xs
- 768px-991px rango sm
- 992px-1199px rango md
- >1200px rango lg

Por tanto, si se pretende que un elemento del DOM ocupe la mitad de la pantalla del dispositivo, sea cual sea el tamaño de pantalla del mismo, se le tendrán que aplicar las siguientes clases “col”:

```
class=”col-xs-6 col-sm-6 col-md-6 col-lg-6”
```

Sin embargo, si se pretende que el elemento solo ocupe media pantalla en pantallas grandes, pero en resoluciones pequeñas como dispositivos móviles ocupe toda la pantalla, se aplicaría de la siguiente forma:

```
class=”col-xs-12 col-sm-12 col-md-6 col-lg-6”
```

Si a continuación de este elemento, se pusiese otro igual, ambos se juntarían en una misma fila para pantallas grandes, pero ocuparían una fila completa cada uno en pantallas pequeñas.

Además, siempre que se haga uso de “col”, se deben englobar los elementos que son candidatos a estar en una misma fila en algún momento con la clase “row”, ya que “col” añade un margen al elemento que utiliza la clase, y “row” restará 15px de margen para que cuando varios elementos con “col” se unan en una misma fila no se acumulen todos sus márgenes, si no que solo se utilice el de uno.

Una variante de las “col” es las off-col, que servirán para crear un offset en el elemento, de manera que si a un elemento se le coloca un offset de 6 columnas en sm, ese elemento será adelantado un espacio equivalente a 6 columnas en la escala sm.

5.8.3 Hidden/visible

Las clases “hidden” y “visible” se utilizan para hacer que un elemento sea mostrado o no en la vista que se especifique (xs, sm, md, lg) de la forma:

```
<div class=”hidden-xs” o class=”visible-md”></div>
```

5.8.4 Btn

Esta clase solo es aplicable a los botones (buttons en HTML), y se encarga de darles un formato con borde redondeado y propiedades para “active” y “hover” de manera que cambien al pasar el ratón o cuando el usuario haga click en ellos. Además, esta clase se combinará con otras dos clases, una para la combinación de colores del botón, btn-estilo (btn-danger, btn-warning, btn-info, btn-success, btn-primary…), y otra para el tamaño del botón, btn-tamaño (btn-xs, btn-sm, btn-md o btn-lg). Por tanto un botón de tamaño medio y color verde quedaría de la siguiente forma:

```
<button class=”btn btn-success btn-md”>Texto botón</button>
```

5.8.5 Panel

Esta clase sirve para transformar un div normal en un panel que se adaptará a todo el ancho del elemento que lo contiene forma proporcional y además contendrá dos elementos con las clases “panel-heading” y “panel-body” que conformarán la estructura completa del panel. Por último, se podrán seleccionar los colores del panel acompañando la primera etiqueta de clase con la etiqueta del color de la misma forma que los botones. Por tanto una estructura básica de un panel rojo quedaría de la siguiente forma:

```
<div class=”panel panel-danger”>
        <div class=”panel-heading”>
                Contenido cabecera del panel
        </div>
        <div class=”panel-body”>
                Contenido cuerpo del panel
        </div>
</div>
```

5.8.6 Text-center, pull-left y pull-right

Como se puede observar, estas clases sirven para alinear elementos en la pantalla; con la clase “text-center” se podrá centrar un elemento, con la clase “pull-left” se podrá alinear a la izquierda, y con la clase “pull-right” se podrá alinear a la derecha.

5.8.7 Icons

Bootstrap cuenta con una amplia gama de iconos, los cuales se podrán escalar como si de un texto se tratase. Para colocarlos, siempre se hará dentro de un indicando que se trata de un icono, y otra clase para indicar qué icono se quiere colocar; un ejemplo de un icono de bandera quedaría de la siguiente forma:

```
<span class=” glyphicon glyphicon-flag”></span>
```

5.8.8 Img-responsive

Esta clase solo se aplicará a elementos <img>, y sirve para que las imágenes se adapten al tamaño del elemento que las contiene en vez de tener un tamaño prefijado:

```
<img class=”img-responsive” src=”…”>
```

Un buen ejemplo que hace uso de todas las clases mencionadas anteriormente se encuentra en la vista *my-cars* de la aplicación, donde cada coche que se muestra estará contenido en un panel, y cada elemento dentro del panel hace uso del sistema de columnas para adaptarse perfectamente a cada tipo de resolución así como las clases hidden y visible. Por otro lado la imagen del coche hará uso de “img-rsponsive” y se hace uso de dos iconos para los símbolos de origen y destino del coche. Además, se hace uso en ciertos elementos del centrado de elementos y los estilos para los botones. Por último, toda la vista está contenida en un div con la clase “container”.

Código de un coche en la vista my-cars:

```
<div class=”container”>
        <div class="panel panel-success">
            <div class="panel-heading">
                <h3 class="panel-title">Coche {{$index+1}}</h3>
            </div>
```

```
            <div class="panel-body" style="color:black;">
                <div class="row">
                  <div id="car" class="col-xs-12 col-sm-12 col-md-7 col-lg-7">
                    <div class="row">
                      <div id="origin" class="col-xs-6 col-sm-3 col-md-3 col-lg-
3">
                        <h4 class="text-center">{{myCar.from}}</h4>
                        <p class="text-center"><span class="glyphicon
glyphicon-remove" style="font-size:20px;"></span></p>
                      </div>
                      <div id="path" class="hidden-xs col-sm-6 col-md-6 col-lg-
6">
                        <div class="row">
                          <div id="carImg" class="col-sm-8 col-md-6 col-lg-6">
                            <img src="/img/blueCar.png" class="img-responsive
center-block" alt="blue car"/>
                          </div>
                          <div id="seats" class="col-sm-4 col-md-6 col-lg-6">
                            <p class="text-center">Asientos</p>
                            <p class="text-center">disponibles</p>
                            <p class="text-
center">{{myCar.availableSeats}}/{{myCar.seats}}</p>
                          </div>
                        </div>
                      </div>
                      <div id="destination" class="col-xs-6 col-sm-3 col-md-3 col-
lg-3">
                        <h4 class="text-center">{{myCar.destination}}</h4>
                        <p class="text-center"><span class="glyphicon
glyphicon-flag" style="font-size:20px;"></span></p>
                      </div>
                      <div id="path-xs" class="visible-xs col-xs-12">
                        <div class="row">
                          <div id="carImg" class="col-xs-6">
                            <img src="/img/blueCar.png" class="img-responsive
center-block" alt="blue car"/>
                          </div>
                          <div id="seats" class="col-xs-6">
                            <p class="text-center">Asientos</p>
                            <p class="text-center">disponibles</p>
```

```
                                    <p class="text-
center">{{myCar.availableSeats}}/{{myCar.seats}}</p>
                                </div>
                            </div>
                        </div>
                    </div>

                    <div id="dateAndPhone" class="hidden-xs col-sm-8 col-md-3
col-lg-3">
                        <h4 class="text-center"><span class="glyphicon glyphicon-
calendar"></span> {{myCar.date}} - {{myCar.time}}</h4>
                        <h4 class="text-center"><span class="glyphicon glyphicon-
earphone"></span> {{myCar.contactNumber}}</h4>
                    </div>
                    <div id="removeButton" class="hidden-xs col-sm-4 col-md-2
col-lg-2">
                        <h4 class="text-center">Borrar coche</h4>
                        <p class="text-center"><button class="btn btn-danger btn-
xs" ng-click="removeCar(myCar,$index)">Borrar coche</button></p>
                    </div>

                </div>
                <div class="visible-xs row ">
                    <div class="col-xs-12 well well-sm">
                        <div id="dateAndPhone-xs" class="col-xs-8">
                            <h4 class="text-center"><span class="glyphicon glyphicon-
calendar"></span> {{myCar.date}} - {{myCar.time}}</h4>
                            <h4 class="text-center"><span class="glyphicon glyphicon-
earphone"></span> {{myCar.contactNumber}}</h4>
                        </div>

                        <div id="removeButton-xs" class="col-xs-4">
                            <h4 class="text-center">Borrar coche</h4>
                            <p class="text-center"><button class="btn btn-danger btn-
xs" ng-click="removeCar(myCar,$index)">Borrar</button></p>
                        </div>
                    </div>
                </div>
                <div class="row">
```

```
                    <div id="passengers" class="col-xs-12 col-sm-12 col-md-12 col-
lg-12">
                        <h4 style="background-color:#5cb85c; color:white;
padding:5px 8px 5px 8px;">Pasajeros interesados</h4>
                            <ul class="list-inline">
                                <il ng-repeat="passenger in myCar.passengers">
                                    <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2 well
well-sm">
                                        <h4 class="text-center">{{passenger.name}}</h4>
                                        <h4 class="text-center"><span class="glyphicon
glyphicon-earphone"></span>{{passenger.phone}}</h4>
                                    </div>
                                </il>
                            </ul>
                    </div>
                </div>
            </div>
</div>
```

5.8.9 Alert

Aunque pueda parecerlo, esta clase no tiene nada que ver con las alertas de JavaScript, ya que es simplemente un estilo que aplicado a un elemento del DOM le otorga un estilo con un borde, un color de texto y un fondo de un color concreto a modo de alerta. Por supuesto el color será seleccionado con una clase asociada como en los botones y los paneles. Esta clase se utiliza en la aplicación por ejemplo para avisar de errores en el login o registro de usuarios:

```
<div ng-show='loginError!=null' class="alert alert-danger" role="alert">Ocurrió
un error: {{loginError}}</div>
```

5.8.10 Form

Esta clase se aplicará a los elementos HTML del tipo formulario <form></form> para hacer que se adapten perfectamente, así como para que tenga un estilo redondeado.

```
<form class="form"></form>
```

5.8.11 Table

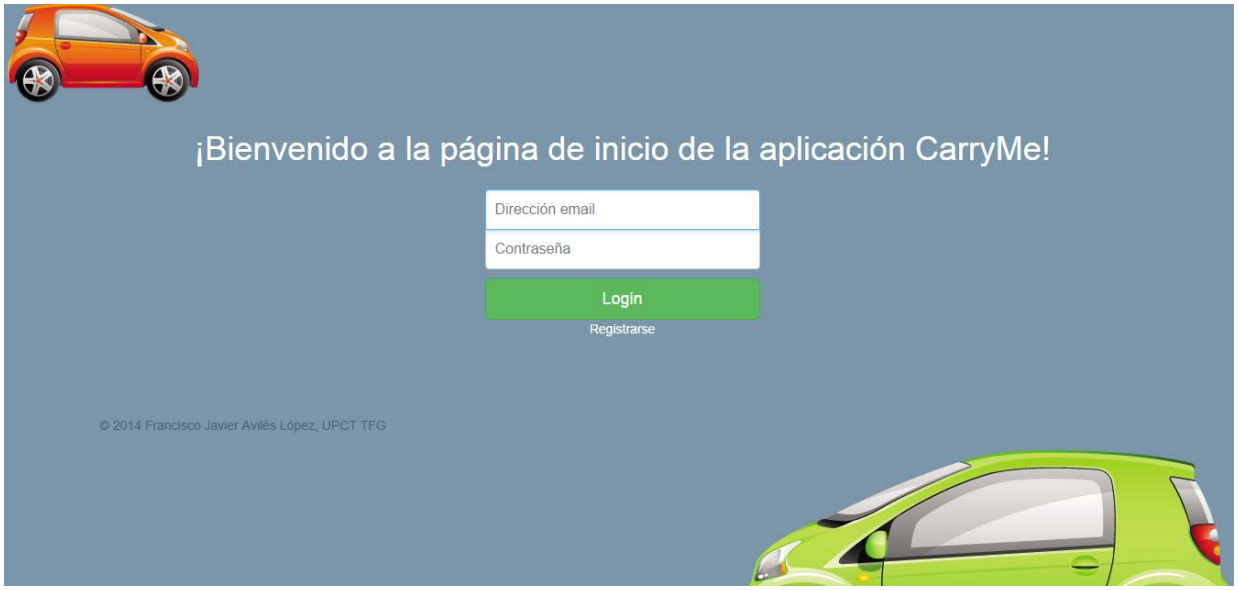
Esta clase tiene infinidad de variantes en Bootstrap y muchas clases posibles en ellas para obtener diferentes estilos y colores, pero básicamente, asignando la clase “table” a un elemento <table></table> y conteniendo dicha tabla en un elemento con la clase “table-responsive”, se obtendrá una tabla acorde a todo el diseño adaptativo que se está realizando para la aplicación. Un buen ejemplo de cómo se ha hecho uso de esta clase es en la búsqueda de coches, donde gracias a la clase “table-responsive” se podrá ver la tabla completa en cualquier dispositivo:

```
<div class="table-responsive">
    <table class="table">
        <thead>
        </thead>
        <tbody>
        </tbody>
    </table>
</div>
```

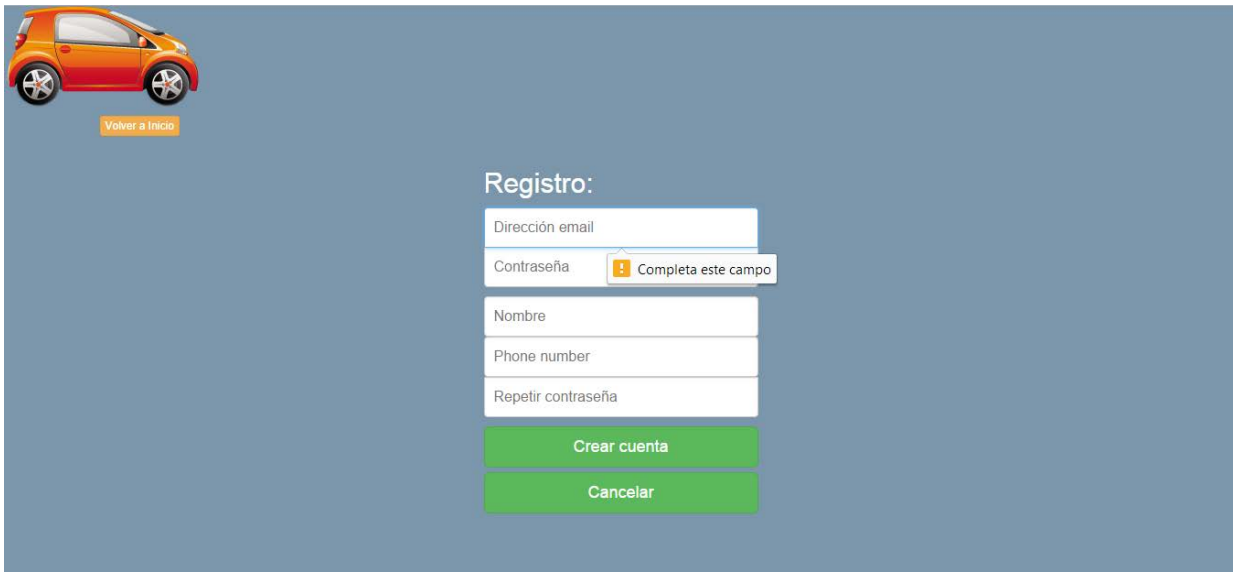
5.9 CAPTURAS

En este apartado se va a mostrar cómo se presenta la aplicación una vez terminada al ser ejecutada en un navegador web desde un ordenador con una resolución de más de 1200px de ancho, así como en un móvil en posición vertical, con una resolución de ancho de menos de 768px.

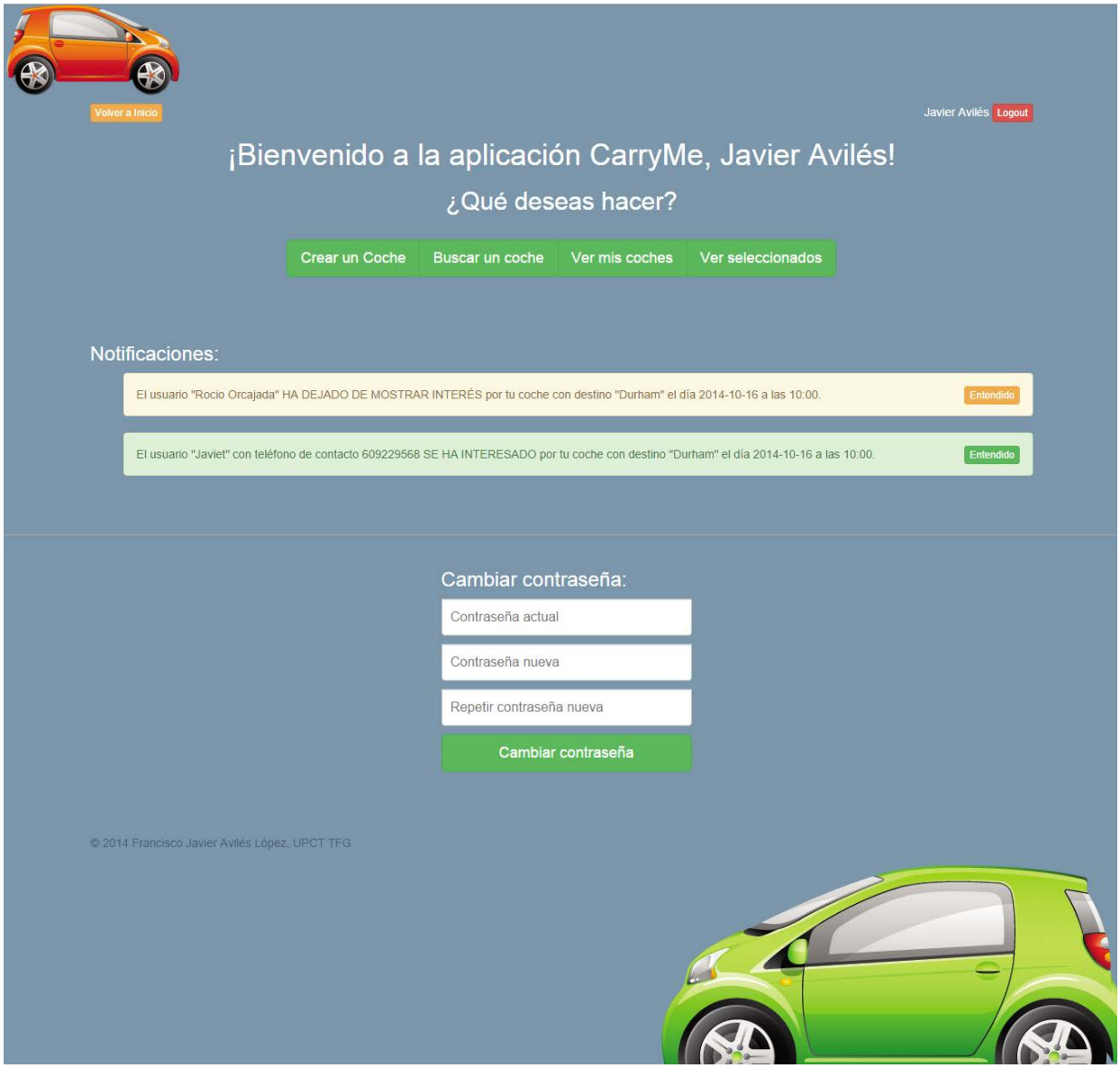
Vista de inicio de la aplicación



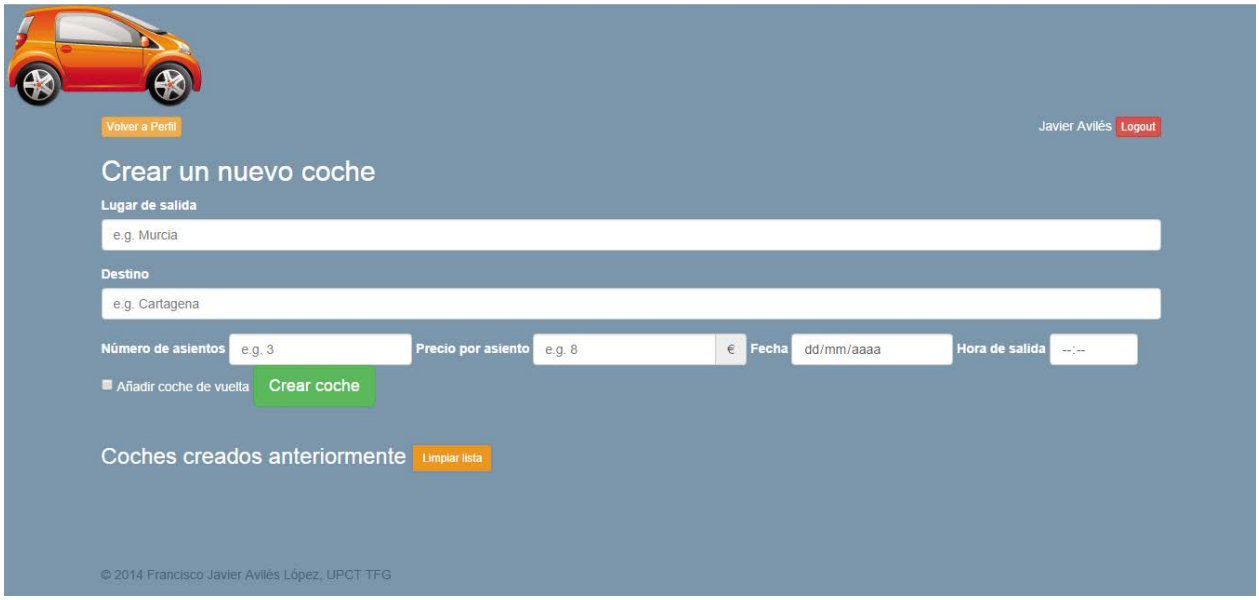
Vista de Registro de la aplicación



Vista de perfil de usuario de la aplicación, menú principal

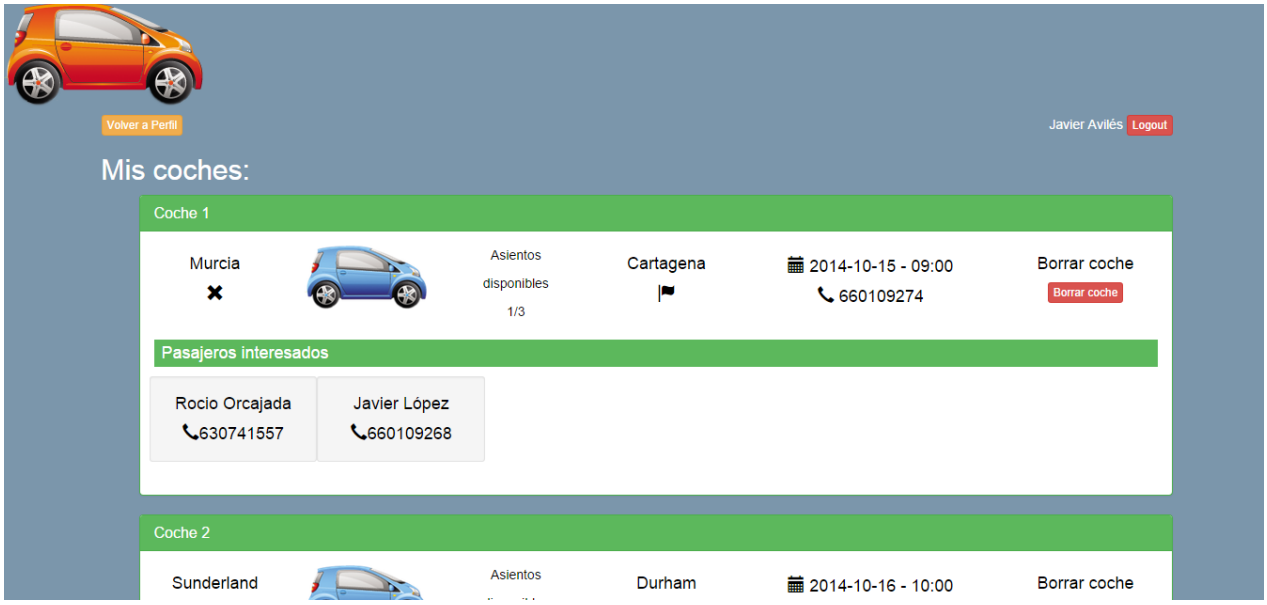


Vista para la creación de un nuevo coche de la aplicación

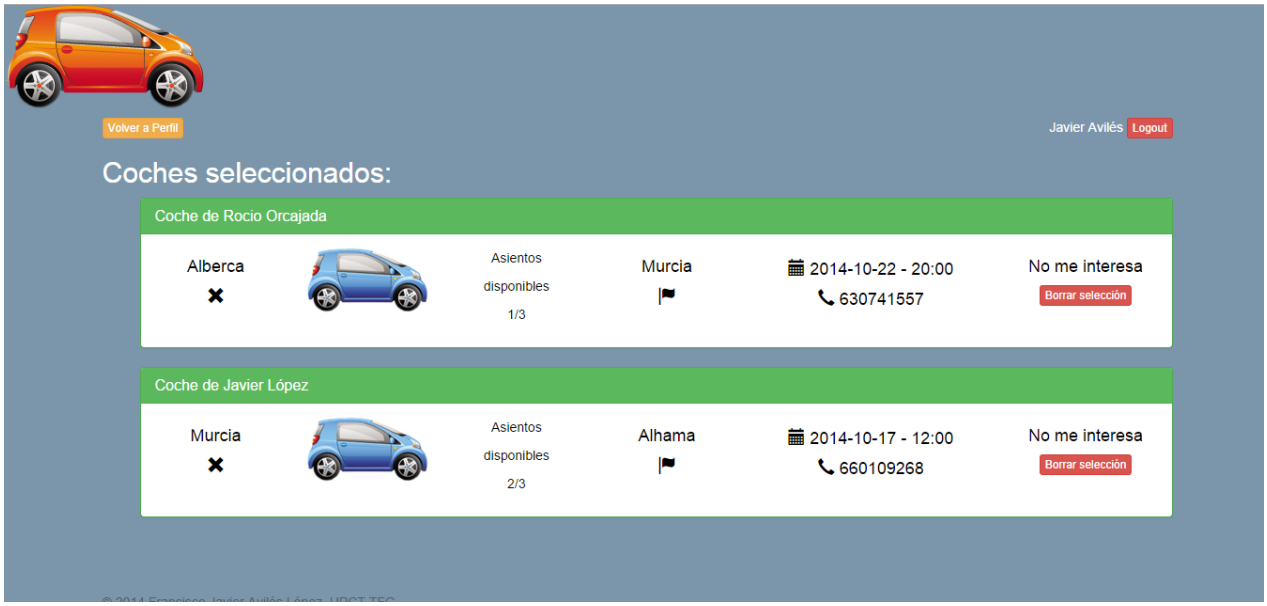


Vista de búsqueda de coches de la aplicación

Vista de coches creados por el usuario de la aplicación



Vista de coches seleccionados por el usuario de la aplicación

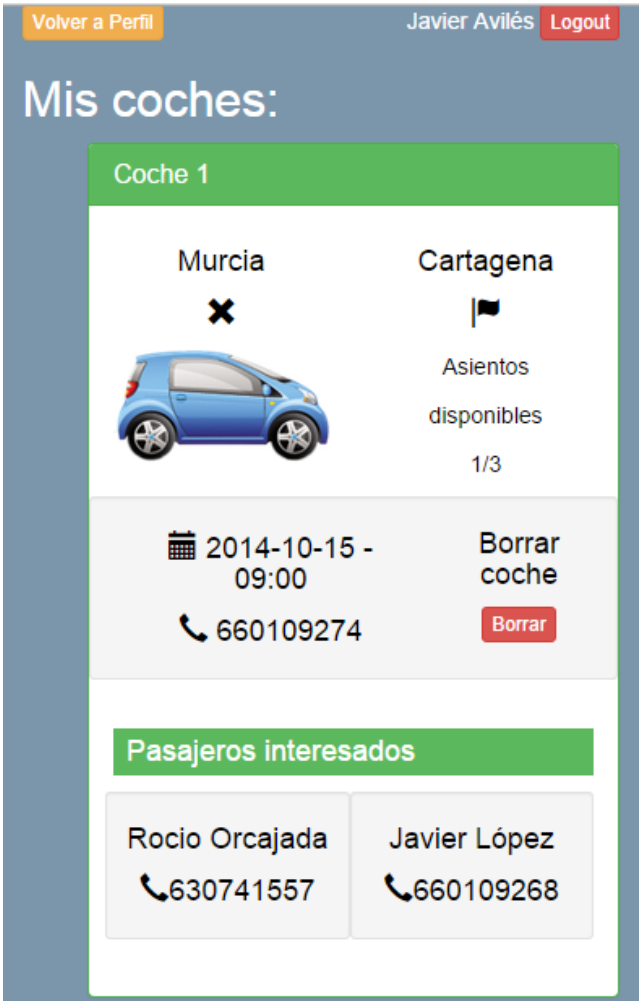


Vista en pantalla pequeña del perfil de usuario de la aplicación, menú principal

Vista en pantalla pequeña de la creación de un nuevo coche de la aplicación

Vista en pantalla pequeña de búsqueda de coches de la aplicación

Vista en pantalla pequeña de coches creados por el usuario de la aplicación



Vista en pantalla pequeña de coches seleccionados por el usuario de la aplicación

6 BASE DE DATOS

En el apartado “Tecnologías utilizadas” se ha tratado con profundidad Firebase. Para esta aplicación, se ha construido una base de datos de Firebase cuya URL es http://tfg.firebaseio.com

Por supuesto, se ha hecho uso de la API AngularFire [11][12], también tratada anteriormente; a continuación se abordará la estructura que se ha utilizado para la base de datos así como el método para el registro y autenticación de usuarios.

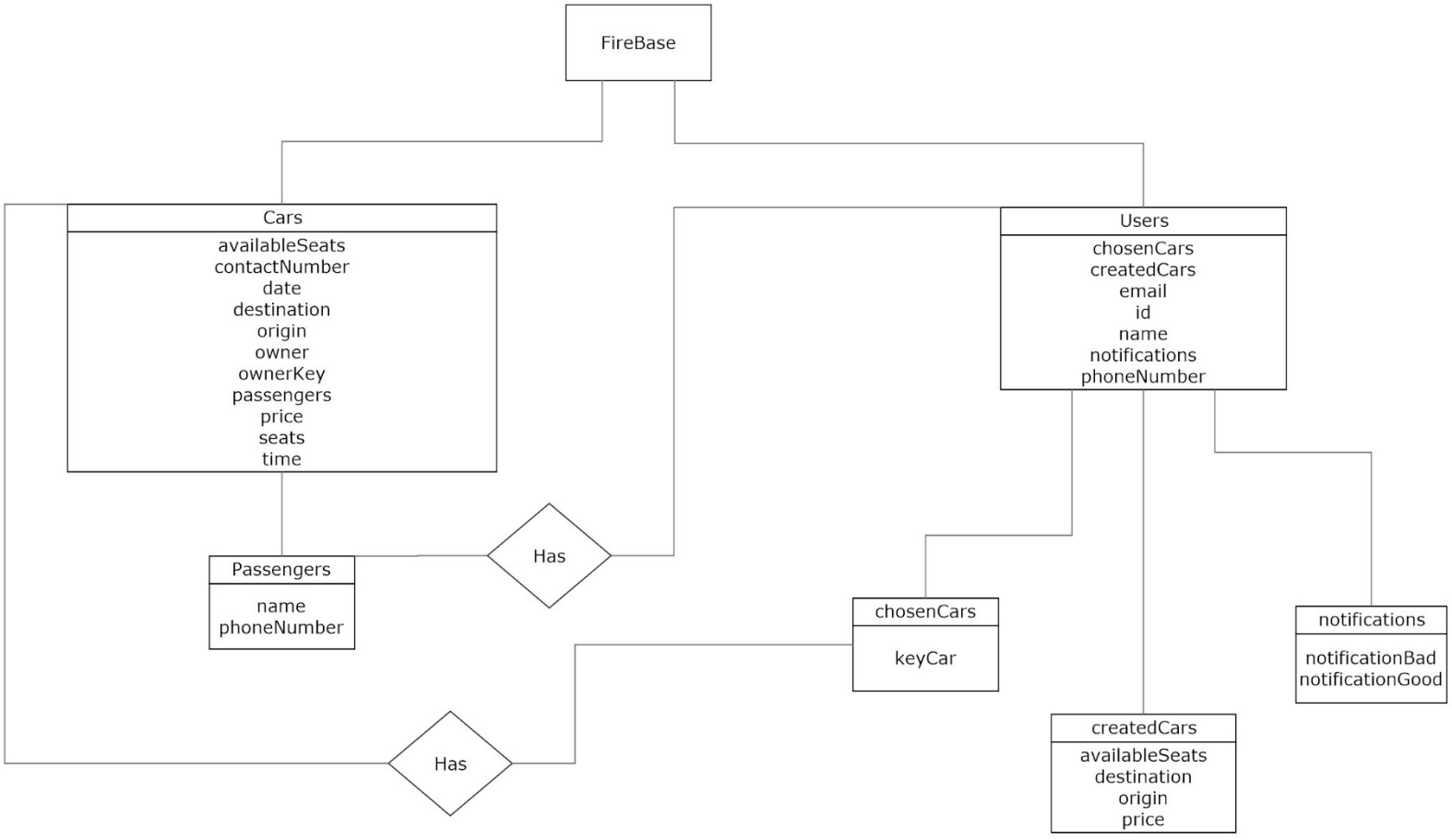


Ilustración 3-Diagrama entidad relación de la base de datos

6.1 LOGIN

Para el login se ha utilizado el servicio **FirebaseSimpleLogin** [14] de Firebase, con el sistema de **usuario/password**. Tras ser habilitado en la base de datos, se han implementado las funciones de login, registro y logout en el controlador loginController asociado a la vista login.html. Estas funciones simplemente llaman a las del servicio creado loginService tras hacer una pequeña comprobación de que los datos introducidos en el formulario por parte del usuario son válidos.

Login

Se comprueba que se ha introducido una dirección de email y una contraseña, y entonces es cuando se procede a llamar a la función login del loginService, la cual realmente hace el login.

```
$scope.login = function() {
            $scope.err = null;
            if( !$scope.email ) {
              $scope.err = 'Please enter an email address';
            }
            else if( !$scope.pass ) {
              $scope.err = 'Please enter a password';
            }
            else {
              loginService.login($scope.email, $scope.pass);
            }
      };
```

Registro

En primer lugar se comprueba igual que antes que se han introducido un email y una contraseña, pero ahora también se comprueba que se ha confirmado la contraseña y que ambas son iguales; una vez realizadas todas las comprobaciones se llama a la función de loginService, la cual realmente realiza la creación de la cuenta.

```
      $scope.createUser = function() {
            $scope.err = null;
            if( !$scope.email ) {
              $scope.err = 'Please enter an email address';
            }
```

```
            else if( !$scope.pass ) {
              $scope.err = 'Please enter a password';
            }
            else if( $scope.pass !== $scope.confirm ) {
              $scope.err = 'Passwords do not match';
            }
            else{
              loginService.createAccount($scope.email, $scope.pass, $scope.name,
$scope.phone);
            }
        };
```

En cada una de sus respectivas funciones, el controlador llama a las del servicio **loginService**, y éste es el encargado de realizar el login, el logout o la creación de usuario, así como del cambio de contraseña del usuario con las funciones de FirebaseSimpleLogin:

Login

Una vez se realiza el $login con éxito en la referencia anteriormente creada en rootScope loginObj, se imprimirá por consola "Logged in as {{idUsuario}}". Después se guardarán en las variables de rootScope user y userkey el nombre de usuario y key para comodidad en cuanto al trato con los controladores. Por último, el usuario es redirigido a la vista /profile.html.

Si el login fallase por cualquier motivo, por ejemplo debido a una contraseña errónea o un usuario inexistente, se imprimirá el error concreto por consola.

```
        login: function(email, pass) {
          loginObj.$login('password', {
            email: email,
            password: pass
          }).then(function(user) {
              console.log('Logged in as: ', user.uid);


              var keys = users.$getIndex();
              angular.forEach(keys, function(key) {
                 if(users[key].id===user.uid){
                    $rootScope.user = users[key];
                    $rootScope.userKey = key;
```

```
            }
          });
          $location.path('/profile');
        }, function(error) {
          console.error('Login failed: ', error);
        });

      },
```

Logout

Para el logout simplemente se realizará el $logout de la referencia al usuario de rootScope y se pondrá a null la variable user de rootScope.

```
      logout: function() {
        loginObj.$logout();
        $rootScope.user = null;
      },
```

Registro

En cuanto al registro de un nuevo usuario, se realiza la creación del usuario en la referencia loginObj de rootScope. Una vez ha sido creado, se imprimirá por consola "User created as {{idUsuario}}". Además, se añadirá a los usuarios de la base de datos un nuevo usuario, con los campos nombre, email, id, número de teléfono, y coches seleccionados (inicialmente vacío, puesto a false).

Una vez se ha realizado la creación de un nuevo usuario, se procede a realizar el login de igual forma que el paso anterior.

```
      createAccount: function(email, pass, name, phone) {
        loginObj.$createUser(email, pass)
          .then(function(user) {
            console.log('User created as: ', user.uid);
            users.$add({
              "name": name,
              "email": email,
              "id": user.uid,
              "phoneNumber": phone,
              "chosenCars": false
            });

            loginObj.$login('password', {
```

```
                email: email,
                password: pass
              }).then(function(user) {
                  console.log('Logged in as: ', user.uid);


                  var keys = users.$getIndex();
                  angular.forEach(keys, function(key) {
                     if(users[key].id===user.uid){
                        $rootScope.user = users[key];
                     }
                  });
                 $location.path('/profile');
               }, function(error) {
                  console.error('Login failed: ', error);
               });



          }, function(error) {
           console.error('User creation failed: ', error);
       });

    }
```

6.2 APARTADO COCHES

Uno de los pilares en la estructura de la base de datos de esta aplicación es el documento coches. En este apartado se almacenan cada uno de los coches que los usuarios crean.

En la vista new-car.html, cuyo controlador asociado es carController, se le piden al usuario una serie de campos para crear un nuevo coche. Estos campos son lugar de salida del coche, lugar de destino, número de teléfono de contacto, asientos del vehículo, precio por pasajero, fecha de salida y hora de salida. Una vez el usuario pulsa el botón de creación del coche, ésta será la función que ejecute el controlador carController:

```
$scope.addCar = function () {
    $rootScope.cars.$add({
      "from": $scope.from,
      "destination": $scope.destination,
      "owner": $scope.owner,
      "ownerKey":$scope.ownerKey,
      "contactNumber":$scope.contactNumber,
      "seats": $scope.seats,
      "availableSeats": $scope.seats,
      "price": $scope.price,
      "date": $scope.date,
      "time":$scope.time
    });

    if($scope.return){
      $rootScope.cars.$add({
        "from": $scope.destination,
        "destination": $scope.from,
        "owner": $scope.owner,
        "ownerKey":$scope.ownerKey,
        "contactNumber":$scope.contactNumber,
        "seats": $scope.seats,
        "availableSeats": $scope.seats,
        "price": $scope.price,
        "date": $scope.date2,
        "time":$scope.time2
      });
```

```
    }
    $scope.done=true;

    $scope.createdCars.$add({
       "from": $scope.from,
       "destination": $scope.destination,
       "availableSeats": $scope.seats,
       "price": $scope.price
    });
  }
```

Por tanto, se añade un nuevo coche a la base de datos, con todos esos campos, cuya información ha introducido el usuario, salvo el campo “owner”, “ownerKey” y “contactNumber” que serán rellenados automáticamente con los datos del usuario creador.

Además como se puede observar, se añadirá un nuevo campo al apartado “createdCars” del usuario, en donde también se añadirán algunos datos del coche para poder ser reusados a la hora de crear más coches.

Por último, hay un campo que se añadirá a cada coche posteriormente, y es el campo “passengers”. Este campo se creará automáticamente cuando un usuario se interese por este coche y se añada como pasajero. Esto ocurrirá en la vista /looking-for-car.html cuyo controlador asociado es lookingController. Cuando un usuario esté buscando un coche y pulse el botón “Me interesa”, se ejecutará la función interested() del controlador lookingController ya comentada en el apartado correspondiente de controllers.js; la única parte importante en cuanto al documento coches de la base de datos es la cual en la que ese usuario interesado se añade como pasajero:

```
$scope.interested = function(carId){
        var refPassengers = new Firebase
("https://tfg.firebaseio.com/cars/"+carId+"/passengers/");
        var passengers = $firebase(refPassengers);
             passengers.$add({
                "name":$rootScope.user.name,
                "phone":$rootScope.user.phoneNumber
             });
}
```

Por supuesto, previamente en esta misma función se comprueba que el usuario no sea ya pasajero del coche así como una comprobación de asientos disponibles y otras funciones que se verán más adelante.

6.3 APARTADO USUARIOS

El otro pilar en la estructura de la base de datos de esta aplicación es el documento usuarios. En este apartadi se almacenan cada uno de los usuarios que se registran en la aplicación, así como todos sus datos: email, id, número de teléfono, nombre y los coches por los que se ha interesado, así como una relación de coches que ha creado para hacer uso de ellos de nuevo.

En la vista /login.html cuyo controlador es loginController es el momento en el que el cliente puede realizar el registro de un nuevo usuario. Este proceso ya se ha explicado en el apartado "Login" de Base de datos, y es precisamente aquí donde se crean los usuarios de la base de datos. En el momento en que la función createAccount se ejecuta en el controlador, y tras la comprobación de errores ésta llama a la función createAccount del servicio loginService, éste crea los usuarios:

```
users.$add({
                    "name": name,
                    "email": email,
                    "id": user.uid,
                    "phoneNumber": phone,
                    "chosenCars": false
});
```

En primera instancia, el campo chosenCars se pone a false, y se irá rellenando con los coches por los que el usuario se interesa. Esto ocurrirá en la misma función en que se rellenan los pasajeros de un coche, la función interested() del controlador lookingController asociado a la vista /looking-for-car.html.

```
$scope.interested = function(carId){
            var refChosen = new Firebase
("https://tfg.firebaseio.com/users/"+$rootScope.userKey+"/chosenCars/");
            var chosen = $firebase(refChosen);
            chosen.$add({
               "keyCar":carId
            });
}
```

Se puede observar cómo se añadirá a los “chosenCars” del usuario el coche por el cual se ha interesado al llamar a esta función. De esta forma se añadirá el campo keyCar, que contendrá la referencia del coche en el apartado coches de la base de datos.

Además, como ya se ha comentado, se añadirán al apartado usuarios los coches que ha creado cada uno, lo cual se realizará en la función de añadir nuevos coches de la vista new-car:

```
var refCreatedCars = new Firebase
("https://tfg.firebaseio.com/users/"+$rootScope.userKey+"/createdCars/");
$scope.createdCars = $firebase(refCreatedCars);
    $scope.createdCars.$add({
        "from": $scope.from,
        "destination": $scope.destination,
        "availableSeats": $scope.seats,
        "price": $scope.price
    });
```

De esta forma, se irán conformando los documentos de la base de datos de una manera clara. Se pueden consultar directamente los datos de la base de datos o incluso modificarlos en la URL de firebase de la base de datos:

http://tfg.firebaseio.com

7 CONCLUSIONES

Para la elaboración de este Trabajo Fin de Grado se ha realizado un estudio previo de los principales lenguajes de programación web como son HTML, CSS y JavaScript. A continuación, se ha realizado un profundo estudio de la tecnología AngularJS y de cómo desarrollar aplicaciones web con un diseño adaptable a todo tipo de dispositivos mediante el uso del framework Bootstrap, así como del uso de la API Firebase para almacenar y sincronizar datos desde AngularJS a modo de base de datos y un servicio completo de autenticación de usuarios en aplicaciones web mediante FirebaseSimpleLogin.

Se ha conseguido obtener una solución completa, a partir de la idea de compartir el medio de transporte con otros estudiantes para acudir a la Universidad diariamente. Una aplicación adaptable a cualquier dispositivo y que solo requiere un navegador web para su utilización, de fácil y cómodo uso pero con muchas funcionalidades, entre las que cabe destacar:

- No solo cuenta con una fácil autenticación de usuarios, si no que el registro de un nuevo usuario es también muy rápido.

- La posibilidad de dar de alta un coche en segundos, pudiendo con un solo click programar el coche para la vuelta, en el horario deseado.

- La posibilidad de reusar los datos de un coche ya creado para crear uno nuevo aún más rápido.

- Visualización de todos los coches disponibles en directo, con la posibilidad de mostrar interés por uno de ellos con un solo botón.

- Visualización de todos tus coches y los pasajeros interesados de un solo vistazo, con el número de contacto de todos ellos en la misma vista.

- De igual forma, visualización de todos los coches por los que el usuario se ha interesado de un solo vistazo, con el contacto del dueño del coche en la misma vista.

- Notificaciones en tu perfil, informándote del estado de tus coches y los coches por los que te has interesado.

Si hay algo que destacar de la aplicación además de su adaptabilidad a cualquier tamaño de pantalla, es su fácil manejo, no requiriendo ningún tipo de aprendizaje previo incluso en los usuarios menos experimentados en aplicaciones de este tipo.

Pese a la plena funcionalidad de la aplicación, siempre se va pensando en más funcionalidades a medida que se realiza el desarrollo de la aplicación, por lo tanto una serie de trabajos futuros que se plantean para la aplicación son:

- Posibilidad de tener una imagen de perfil en cada usuario, dando así más confianza a los usuarios de tu vehículo, así como imágenes del vehículo.

- Posibilidad de certificar tu usuario/número de teléfono como estudiante universitario.

- Posibilidad de votar un usuario una vez que has viajado con él, otorgando "medallas" como pueden ser conductor prudente, puntual, vehículo de calidad o buena compañía; así como la posibilidad de otorgar malas impresiones como pueden ser exceso de velocidad o indicar que el usuario falto a la cita.

- La difusión de la aplicación para su uso con una estrategia de marketing en las redes sociales, así como el uso del servicio Google Adwords para su promoción y un trabajo SEO específico en el código para mejorar su posicionamiento en los principales buscadores web. Afiliación con la UPCT.

A modo personal, la principal conclusión que puedo destacar es la motivación que supone ver que he podido llevar a cabo un proyecto de esta magnitud partiendo de cero y sin tener ningún conocimiento previo de programación web. A lo largo de las fases de aprendizaje y desarrollo he ido adquiriendo seguridad a la hora de hacer uso de tecnologías totalmente nuevas para mí, así como facilidad a la hora de programar y documentar mis trabajos.

Por último, debido al gran auge que están teniendo los dispositivos y aplicaciones móviles, esta última formación que he obtenido en mi carrera universitaria me será de gran ayuda en mi futuro profesional.

8 BIBLIOGRAFÍA

[1] "¿Qué Es AngularJS? Una Breve Introducción." http://pablolazarodev.blogspot.com.es/2013/05/que-es-angularjs-una-breve-introduccion.html.

[2] Lerner, Ari. *Ng-Book - the Complete Book on Angularjs*. Fullstack.io, 2013.

[3] "AngularJS Tutorial: Learn to Build Modern Web Apps with MEAN." *Thinkster*. https://thinkster.io/angulartutorial/mean-stack-tutorial.

[4] "AngularJS API Docs," n.d. https://docs.angularjs.org/api.

[5] "Angular/angular.js." *GitHub*. https://github.com/angular/angular.js.

[6] "Guide to AngularJS Documentation," n.d. https://docs.angularjs.org/guide.

[7] "Israel Guzmán - YouTube." https://www.youtube.com/user/angularjstutoriales/videos?flow=grid&view=57.

[8] "Mejorando El Paso de Parámetros Entre Controladores En AngularJS | Koalite." http://blog.koalite.com/2013/11/mejorando-el-paso-de-parametros-entre-controladores-en-angularjs/.

[9] "UML Tutorial," n.d. http://www.tutorialspoint.com/uml/uml_tutorial.pdf.

[10] Crockford, Douglas. *JavaScript: The Good Parts: The Good Parts*. O'Reilly Media, Inc., 2008.

[11] "AngularFire." https://www.firebase.com/docs/web/libraries/angular/api.html.

[12] "AngularFire Development Guide," n.d. https://www.firebase.com/docs/web/libraries/angular/guide.html.

[13] "Firebase/angularfire-Seed." *GitHub*. https://github.com/firebase/angularfire-seed.

[14] "User Authentication," n.d. https://www.firebase.com/docs/web/guide/user-auth.html.

[15] Philip Olson. "Manual de PHP." *PHP*. http://www.php.net/manual/es/.

[16] "Fundamentos de Programación En Java," n.d. http://pendientedemigracion.ucm.es/info/tecnomovil/documentos/fjava.pdf.

[17] Goldstein, Alexis, Estelle Weyl, and Louis Lazaris. *HTML5 & CSS3 for the Real World*. SitePoint, 2011.

[18] "JavaServer Faces." *Wikipedia, the Free Encyclopedia*, http://en.wikipedia.org/w/index.php?title=JavaServer_Faces&oldid=628437441.

[19] "jQuery Mobile API Documentation." http://api.jquerymobile.com/.

[20] "jQuery UI API Documentation." http://api.jqueryui.com/.

[21] "EmbedJS," http://uxebu.github.io/embedjs/.

[22] "BackboneJS," http://backbonejs.org/.

[23] "NoSQL Databases Explained." http://www.mongodb.com/nosql-explained.

[24] "Data Modeling Introduction — MongoDB Manual 2.6.4." *Https://github.com/mongodb/docs/blob/master/source/core/data-Modeling-Introduction.txt*. http://docs.mongodb.org/manual/core/data-modeling-introduction/.