PROYECTO DE FIN DE MÁSTER

Diseño e implementación de una Aplicación multiplataforma de comunicación en tiempo real mediante APIs, Frameworks y Tecnologías Web de última generación.

Autor: Juan Antonio Guerra Montenegro.
Tutor: Javier Jesús Sánchez Medina.
Fecha: Julio de 2017

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Esta hoja se ha dejado en blanco intencionadamente.

Tabla de contenido

Agradecimientos.

*En primer lugar, me gustaría agradecer a mi tutor **don Javier Jesús Sánchez Medina** el apoyo que me ha dado durante todas y cada una de las fases del proyecto, además de haber sido una inspiración para mí gracias a las clases que me impartió sobre emprendeduría e innovación durante el transcurso del Máster, y también a **Cayetano Guerra Artal**, que me ayudó a definir y a pulir el concepto inicial hasta llegar a la idea de la cual ha surgido el proyecto expuesto en esta memoria.*

*Sin duda alguna, también debo hacer una mención especial **hacia mis padres**, que han sido la piedra angular de mis estudios y que han estado a mi lado en todo momento. Sin su apoyo, nunca habría llegado al lugar en el que hoy me hallo, y es por ello por lo que parte de todo mi trabajo siempre les pertenecerá.*

Por último, pero no menos importante, debo agradecer a mis amigos y compañeros todo el apoyo brindado durante el desarrollo de este proyecto. Gracias por sacar mi cabeza de los libros de tanto en tanto, por todos esos buenos momentos y por inspirarme constantemente a la hora de superar todos los obstáculos que podían surgir.

Resumen.

El objetivo de este *Trabajo de Fin de Máster* es el de realizar el diseño e implementación de una aplicación multiplataforma de comunicación instantánea entre clientes mediante los últimos frameworks del mercado basados en tecnologías web. Dichas tecnologías han sido ***HTML5, CSS3/SASS, JavaScript, AngularJS, Ionic Framework*[1] *y Google Firebase***[2]. Tras el completo desarrollo de este *TFM*, se dispondrá de una App multiplataforma con las siguientes capacidades:

- Diseño de interfaz y de identidad basado en las últimas tendencias del mercado.
- Identificación y generación de perfiles de usuario mediante diferentes APIs (*Firebase, Facebook, Google y Twitter*).
- Envío y recepción de mensajes entre los usuarios de la aplicación.

No obstante, una vez alcanzados estos objetivos básicos, se decidió dotar al proyecto de un valor añadido mediante funcionalidades adicionales. Dichas funcionalidades son:

- Envío y recepción de imágenes.
- Sistema de amigos.
- Filtrado de elementos (chats, amigos).
- Envío y recepción de mensajes en chats de grupo.
- Cambio de imagen de perfil, creación y modificación de imagen de grupo.
- Servicio de notificaciones.

Esta Aplicación ha sido desarrollada para *Android*, ya que su cuota de mercado, su bajo coste y su gran capacidad de portabilidad lo convierten en un sistema operativo idóneo. No obstante, y gracias a la tecnología de *Ionic Framework*, si se dispone de un dispositivo *Apple* con *OSX* es posible realizar de manera instantánea una versión para *iOS* mediante una compilación en dicho dispositivo.

A la hora de realizar este proyecto, se han utilizado diferentes herramientas y tecnologías web, además de varios dispositivos de hardware. En el primer ámbito, se ha usado *Github Atom* como editor de código debido a su flexibilidad y a su perfecta integración con las tecnologías web que querían utilizarse. Asimismo, la Aplicación se ha implementado mediante el uso de *Ionic* y de *Google Firebase*. Respecto al hardware, se ha utilizado un dispositivo ***Huawei Mate 8*** con la *API 24 (Nougat 7.0)* de *Android*, además de un **LG *Nexus 5X*** con la *API 23 (Marshmallow 6.0)* virtualizado mediante *Android Development Kit (ADK)* y utilizando tecnología *Intel HAXM*[3] para permitir dicha virtualización en tiempo real. Todas las pruebas se han realizado en dichos dispositivos.

La motivación principal o *background* de este proyecto ha sido la de crear un sistema de comunicación para una aplicación que será desarrollada por varios estudiantes de este mismo posgrado, todo mediante la utilización de tecnologías web que actualmente forman parte del Estado del Arte. También ha tenido peso la idea de abordar el aprendizaje de un Framework multiplataforma como *Ionic*, el cual comienza a convertirse en un estándar de desarrollo para pequeñas empresas o *Start-Ups* debido a su adaptabilidad y facilidad de aprendizaje.

---

[1] *Ionic* es un *framework* de desarrollo que permite la realización simultánea de versiones web y móviles (Android e IOS) de una Aplicación sin tener que desarrollar código específico para cada versión.

[2] *Google Firebase* es un servicio provisto por Google Inc., que provee a los desarrolladores de un sistema de almacenamiento e intercambio de datos en la nube de manera no estructurada, así como de facilidades para implementar la identificación de usuarios mediante *Login Social*.

[3] *Intel HAXM* (**H**ardware **A**ccelerated e**X**ecution **M**anager) es una tecnología de motor de virtualización propietaria de INTEL que, combinada con las imágenes oficiales de AndroidOS y el Android SDK, permiten una ejecución mucho más rápida de una Máquina Virtual de Desarrollo Android.

También se debe mencionar una motivación que, pese a ser secundaria, posee una importancia notable y además actúa como aportación al entorno técnico en el que se desarrolla, y es el hecho de verificar como una sola persona, en cinco meses y con las tecnologías y *frameworks* más actuales, ha sido capaz de desarrollar el trabajo de un equipo multidisciplinar de profesionales. Hoy en día se tienden a reutilizar tecnologías que, pese a su validez, carecen de la modernidad y flexibilidad necesarias para propiciar un desarrollo sencillo y ágil. El uso de aplicaciones híbridas era desaconsejado debido a su poca optimización respecto a las herramientas nativas de cada sistema, pero hoy en día su potencia ha escalado en una medida tan notable que resulta no solo en la creación de aplicaciones de similar eficiencia con una utilización de recursos mucho menor, sino en la reducción de plantilla de un proyecto, siendo posible destinar a más ingenieros a otros proyectos de mayor importancia.

Es importante destacar el amplio espectro de desarrollo que cubre este trabajo, ya que se tratarán problemas de gran diversidad que irán desde el aprendizaje y dominio de *Ionic Framework* hasta la provisión de servicios de almacenamiento realizada por *Google Firebase*, sin olvidar la utilización de las diferentes APIs provistas por los más usuales proveedores de identificación social. Muchos de estos aspectos no han sido impartidos en la rama cursada del Máster, por lo que su estudio y posterior desarrollo me han permitido no solo aumentar mi nivel de conocimientos, sino aumentar mi comprensión referente a nuevos campos de desarrollo de los que era total desconocedor.

Respecto a las competencias designadas en este proyecto, debe hacerse mención en que han sido completamente desarrolladas tanto las generales como las específicas. Las competencias específicas son las siguientes:

- **C01:** Capacidad para proyectar, calcular y diseñar productos, procesos e instalaciones en todos los ámbitos de la ingeniería informática.
- **TI01:** Capacidad para modelar, diseñar, definir la arquitectura, implantar, gestionar, operar, administrar y mantener aplicaciones, redes, sistemas, servicios y contenidos informáticos.
- **TI02:** Capacidad de comprender y saber aplicar el funcionamiento y organización de Internet, las tecnologías y protocolos de redes de nueva generación, los modelos de componentes, software intermediario y servicios.
- **TI05:** Capacidad para analizar las necesidades de información que se plantean en un entorno y llevar a cabo en todas sus etapas el proceso de construcción de un sistema de información.
- **TI11:** Capacidad para conceptualizar, diseñar, desarrollar y evaluar la interacción persona–ordenador de productos, sistemas y servicios informáticos.

A la hora de crear este proyecto, se tuvieron en cuenta su origen y su objetivo (diseñar un chat multiplataforma para los miembros de una aplicación), además de las funcionalidades básicas que este producto iba a poseer (competencia ***C01***), las cuales han sido mencionadas previamente en esta memoria. También se desarrollaron la soltura y los conocimientos necesarios para cumplir la competencia ***TI01***, siendo de vital importancia la investigación y documentación del entorno de desarrollo y de las tecnologías utilizadas. Esto viene estrechamente ligado a la competencia ***TI02***, ya que muchas de las tecnologías pertenecen al ámbito de Internet y los protocolos de redes actuales. Una vez adquiridos los conocimientos necesarios, se analizaron las necesidades de información del entorno para construir una Base de Datos capaz de almacenar y gestionar de manera eficiente la información del sistema que se construyó, cumplimentando la competencia ***TI05***. Finalmente, la realización de todo el diseño, no solo de la interfaz de usuario, sino también de la identidad de la aplicación mediante un análisis detallado de la teoría del color y de la organización de elementos ha sido de gran importancia a la hora de desarrollar por completo la competencia ***TI11***.

Esta memoria se divide en varios apartados y subapartados. En primer lugar, se comienza con un *resumen* y una breve *introducción* en las cuales se proporciona una visión global del proyecto, además de proporcionar un preámbulo sobre los temas tratados en el mismo. Posteriormente se encuentra un capítulo dedicado al *Estado del Arte* sobre las tecnologías englobadas en este desarrollo.

Tras esta primera toma de contacto, se tratarán diversas cuestiones, comenzando por las *Metodologías* y el *Plan de Trabajo*, seguidos por el *Análisis* detallado del proyecto. A posteriori se verán los diferentes aspectos de *Diseño* de la Aplicación, en los que se pormenoriza la solución a los problemas planteados. A continuación, en el apartado de *Implementación*, se detallarán los diferentes aspectos de las herramientas utilizadas para llevar a cabo el proyecto. Por último, se expondrán las *Pruebas y Resultados* en los que se detalla la validación de la aplicación y, para finalizar, en el apartado de *Conclusiones y Futuro* se mostrarán las resoluciones a las que se ha llegado con la finalización del proyecto, así como sus posibles ampliaciones en un futuro.

Introducción.

Para abarcar la comprensión de todos los ámbitos reflejados en este proyecto, se procederá a la diferenciación y explicación de cada uno de los mismos y a como se han unido para así resultar en la creación de una completa aplicación móvil basada en su totalidad en las tecnologías web más modernas del mercado.

2.1. Aplicación Híbrida.

Según ***Ionic***[4], una *Aplicación Híbrida* es un conjunto de pequeños módulos web funcionando bajo un *Shell de Navegador*, dentro de una *app* que tiene acceso a la capa nativa de la plataforma bajo la que funciona. Actualmente, el desarrollo de aplicaciones híbridas está experimentando un *boom* respecto a su utilización, destacando en el uso de las *start-ups* y de los pequeños emprendedores o empresas debido a la sencillez a la hora de desarrollar aplicaciones complejas que mantengan unos ciertos estándares de calidad, pero en un tiempo sensiblemente inferior al normal.

El desarrollo de estas *Apps* se lleva a cabo mediante *frameworks*[5] diseñados para tal fin. Los *frameworks* más utilizados en la actualidad son *Ionic* (diseñado bajo *AngularJS* y *Apache Cordova*), *PhoneGap* (basado en *Apache Cordova*), *Mobile Angular UI* (basado en *AngularJS*), *o Sencha Touch* (basado en *HTML5)*. Dichos *frameworks* poseen funciones que se comunican con el *SO Objetivo* [6]para proveer de una apariencia y funcionamiento *nativos* a la *App* que ha sido desarrollada.

También es altamente recomendable la utilización de un dispositivo con el *SO Objetivo* para así poder testear de manera fiable la *App* que se está creando. En este caso, se han utilizado dos dispositivos (*Huawei Mate 8* y *LG Nexus 5X* emulado), ambos con imágenes oficiales del sistema operativo *Android*. No ha sido posible realizar el testeo de este TFM en un sistema *iOS*, debido a que se necesitan tanto un dispositivo físico como una plataforma de desarrollo pertenecientes al ecosistema *Apple,* pero es posible realizar una compilación de la *App* para ellos a posteriori, dado que el código desarrollado mediante *Ionic Framework* es multiplataforma.

Para desarrollar en *Android*, tal como se ha mencionado anteriormente, basta con conectar un dispositivo móvil mediante USB al equipo en el que se realiza el desarrollo. De no disponerse dicho dispositivo, puede realizarse una emulación mediante una máquina virtual de este *SO*, mediante la que podremos probar la *App*.

[4] *http://ionicframework.com/docs/guide/preface.html*

[5] *Estructura conceptual y tecnológica de soporte definido, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. (Wikipedia).*

[6] *Versión mínima del Sistema Operativo para la que la App desarrollada debe de ser compatible.*

2.2. Ionic Framework.

Ionic es un *framework* desarrollado por *Drifty Co*[7] cuyo lanzamiento oficial se produjo en 2013 y el cual posee licencia MIT[8]. Se ha diseñado sobre *AngularJS* y *Apache Cordova*, y proporciona un conjunto de herramientas y servicios que permiten desarrollar aplicaciones móviles mediante *Tecnologías Web* tales como *CSS3*, *HTML5* y *SASS*. Esas aplicaciones se compilan bajo estas tecnologías para, posteriormente, ser distribuidas de manera nativa en las *tiendas (stores)* de las plataformas seleccionadas e instaladas mediante la capa de *Apache Cordova*.

Este *framework* incorpora diversos mecanismos que simulan a la perfección la interacción nativa con el *SO Objetivo*, todo ello mediante un *front-end*[9] de programación muy parecido a *HTML5* el cual posee diversos elementos y animaciones de apariencia similar al de las plataformas más utilizadas (*Android* e *iOS*). Según la compilación que se realice, la apariencia de nuestra *aplicación* cambiará automáticamente para reflejar el *Sistema Operativo* bajo el que se utilice.

La mayor ventaja de *Ionic Framework* radica en su motor de compilación. Gracias a *Apache Cordova*, podemos realizar la programación de nuestra aplicación y luego compilarla para los dispositivos deseados independientemente del *SO Objetivo*, lo que nos permite desarrollar un proyecto en casi la mitad de tiempo al no tener que desarrollar diferentes versiones para cada plataforma deseada.

Ionic se sirve como un módulo de *Node.js*[10], siendo necesaria su previa instalación para poder comenzar a utilizar sus capacidades.

Cabe destacar que la versión utilizada en este proyecto ha sido *Ionic 1*. En el momento de la realización existía una versión *Ionic 2.0*, pero al encontrarse en un estado de BETA temprana y al cambiarse sus funciones de manera periódica, se optó por escoger la primera en base a su soporte a largo plazo.

A continuación, se definirán los componentes de *Ionic Framework* utilizados en este *Trabajo de Fin de Máster*.

[7] *Drifty Co.* es una compañía fundada por Ben Sperry y Max Lynch en 2012, centrada en la creación de productos y servicios orientados al desarrollo móvil.

[8] La licencia MIT es una licencia gratuita de software libre originada en el *Massachusetts Institute of Technology*, la cual permite la reutilización de software propietario siempre y cuando cada una de las copias de dicho software sea provista de dicha licencia.

[9] Capa de representación visual de un software.

[10] *Node.js* es un entorno de ejecución de *JavaScript*, *open-source* y *multiplataforma*, diseñado para desarrollar una gran variedad de aplicaciones y herramientas de servidor, el cual funciona como un *gestor de paquetes* para el proyecto que se esté desarrollando.

2.2.1. AngularJS.

AngularJS es un *framework front-end open-source* basado en *JavaScript* y principalmente mantenido por *Google*, lanzado el 20 de octubre de 2010. Su objetivo es el de simplificar el desarrollo y prueba de aplicaciones móviles multiplataforma mediante la adición de un *framework* para arquitecturas *Model-View-Controller (MVC)*[11] y *Model-View-Viewmodel (MVVM)*[12].

Su funcionamiento comienza tras realizar una primera lectura del fichero *HTML*, el cual posee *etiquetas (tags)* con atributos. Dichos atributos son interpretados como directivas que unen elementos de entrada o salida de datos a un modelo que es representado mediante una variable en *JavaScript*. El valor de esta variable puede ser modificado manualmente en el propio código, o mediante recursos dinámicos JSON. De esta manera, podemos incluir datos y estructuras de control en nuestro código HTML, los cuales se representan dinámicamente.

2.2.2. Apache Cordova.

Cordova es un *framework* de desarrollo de aplicaciones móviles, originalmente creado por *Nitobi*. Posteriormente, *Adobe Systems* lo compró y lo renombró *PhoneGap*. Finalmente, fue lanzada una versión *open-source* a la que se llamó *Apache Cordova*.

Este *framework* permite la creación de aplicaciones móviles mediante el uso de *CSS3*, *HTML5* y *JavaScript*, en lugar de la utilización de *APIs* específicas de la plataforma, tales como las de *Android*, *iOS* o *Windows Phone*. Esto se consigue mediante la compilación de las tecnologías anteriormente mencionadas en base a la plataforma seleccionada para el desarrollo. Las aplicaciones resultantes tras la compilación son *Híbridas.*

[11] Patrón de diseño de *Software* centrado en la implementación de interfaces de usuario en dispositivos informáticos. Divide una aplicación en tres partes interconectadas, separando la representación interna de información de aquella visualizada por el usuario.

[12] Patrón de diseño de *Software* centrado en la separación de la Interfaz Gráfica de Usuario y de la Lógica Interna *(Back-End)*. El *ViewMode*l es el responsable de convertir los objetos de datos de manera que sean sencillamente organizables y presentables.

2.2.3. SASS.

*SASS (**S**yntactically **A**wesome **S**tyle**S**heets)* es un lenguaje de Hojas de Estilo originalmente diseñado por Hampton Catlin y desarrollado por Natalie Weizenbaum. Es un lenguaje de *scripting* que es interpretado en *CSS*, cuyo nombre interno es *SassScript*.

La sintaxis original es similar a *HAML*[13], ya que utiliza indentación para separar bloques de código y caracteres *newline* para separar reglas. No obstante, la nueva sintaxis introducida, llamada *SCSS*, utiliza un formato de bloques similar al de CSS. Utiliza llaves **{ }** para reflejar bloques de código, y *semicolons* **;** para separar líneas. Las extensiones de fichero para *SASS* son ".sass" (para sintaxis *HAML*) y ".scss" (para sintaxis *SCSS)*.

Basándonos en que *CSS3* contiene una serie de selectores y pseudo-selectores que agrupan las reglas que se aplican a los mismos, *SASS* extiende *CSS* mediante diferentes mecanismos pertenecientes a lenguajes de programación más clásicos pero que no se encuentran disponibles per se en *CSS3*. Cuando *SassScript* es interpretado, crea bloques de reglas *CSS* para varios selectores definidos en el fichero *SASS*. Como valor añadido, *SASS* puede monitorizar los ficheros y traducirlos a *CSS* en el mismo momento en el que dichos ficheros sean guardados.

2.3. Google Firebase.

Firebase es un servicio para aplicaciones, el cual incluye unas herramientas e infraestructura diseñadas para ayudar a los desarrolladores a crear proyectos de gran calidad basados en la metodología *BaaS*. Comenzó como una *startup* llamada *Envolve*, creada por *James Tamplin* y *Andrew Lee* en 2011, la cual proveía de una *API* a los desarrolladores que facilitaba la integración de chats online en sitios web. Posteriormente se descubrió que los desarrolladores utilizaban *Envolve* para pasar datos complejos en lugar de mensajes, y se decidió crear una versión dedicada a este fin, a la que llamaron *Firebase*. El 21 de Octubre de 2014, *Firebase* fue adquirida por *Google*.

En la actualidad, *Firebase* ofrece diversos servicios al desarrollador de aplicaciones, que van desde *analíticas* hasta *almacenamiento* y *hosting*. En el caso de este *Proyecto de Fin de Máster*, se han utilizado los siguientes servicios:

SERVICIO	DESCRIPCIÓN
Firebase auth	Autenticación nativa y mediante *Social Login*.
Firebase realtime database	Base de Datos no estructurada, similar a *MongoDB*.
Firebase storage	Subida y descarga de contenidos en la Nube.

Tabla 1 - Componentes utilizados de Google Firebase

Respecto al almacenamiento, hay que destacar que *Firebase* utiliza un esquema de estructura de datos *no relacional*, también conocido como *noSQL*, que prescinde de las capacidades del modelo Entidad-Relación utilizado hasta ahora. Estas particularidades se tratarán en el apartado *5.3. Modelo de Base de Datos*, de la presente memoria.

[13] *HAML (**H**TML **A**bstraction **M**arkup **L**anguage)* es un sistema de maquetación concebido para crear documentos *HTML* limpios y sencillos, similar a *PHP*, *ASP* y *JSP*.

2.4. APIs de Terceros.

En este proyecto se han utilizado diferentes APIs de terceros para permitir el inicio de sesión mediante *Social Login* en la aplicación mediante Facebook, Google+ y Twitter. A continuación, se detallarán cada una de ellas.

2.4.1. Facebook.

Facebook es, en cuestión de *login social*, la primera de las APIs que debe de implementarse. Fundada por *Mark Zuckerberg* junto a *Eduardo Saverin*, *Chris Hughes* y *Dustin Moskovitz* en 2005, con cerca de los 2.000 millones usuarios (casi un tercio de la población mundial) y traducida a más de 140 idiomas, esta aplicación es una de las más extendidas en la industria de las aplicaciones web y móviles. El facilitar un *login social* que pueda realizarse con este proveedor aumenta la cuota de alcance de la aplicación que se desee desarrollar de una manera notable.

La API de Facebook nos permite recuperar información de interés sobre el usuario que inicie sesión mediante la misma en nuestra aplicación. Esta información en nuestro caso no pasa de la *imagen de perfil* y de su *dirección de correo electrónico*, pero podemos extender la misma hasta *creencias religiosas*, *amistades* y *gustos* de todo tipo. Obviamente se avisa al usuario de lo que la aplicación va a requerir al darle acceso, siendo totalmente elección del mismo el hecho de aceptar o no su utilización.

Ha de destacarse que es la más difícil de implementar a la hora de ofrecer un *login social,* ya que, en el momento de realización de este *Trabajo de Fin de Máster*, la *API* no ofrece las facilidades necesarias para realizar una conexión sencilla a nivel programático entre los diferentes servicios que deseen utilizarla.

2.4.2. Google / Google+

El 15 de junio de 2011, Google se aventuraba en el mundo de las redes sociales gracias a *Google+*, una red de usuarios basada en intereses. Curiosamente es una de las que menos usuarios posee (111 millones), aunque su utilización varía en función de los ajustes que *Google* va realizando sobre ella. Su uso ha fluctuado de manera notable desde su lanzamiento, siguiendo un crecimiento y decrecimiento inconstantes en su desarrollo. Unos años después, el inicio de sesión mediante *Google+* se fusionó con la propia cuenta de *Google* del usuario, permitiendo el acceso a todos los servicios de esta empresa mediante un solo *login*.

Esta *API* nos permite recabar información de la misma manera que la anterior, solicitando diferentes permisos al usuario, previo a su registro, para obtener datos acerca de su dirección de correo electrónico, edad, foto, círculos de amistad, etc.

2.4.3. Twitter.

Twitter es una red social fundada el 21 de marzo de 2006 por *Jack Dorsey, Noah Glass, Biz Stone y Evan Williams*, y su interacción está basada en la publicación de mensajes de 140 caracteres de longitud entre los usuarios del servicio. Actualmente es una de las redes sociales más utilizadas del mundo, con 319 millones de usuarios activos, y es utilizada por los usuarios para transmitir noticias, sucesos o eventos de manera corta y precisa. Ha experimentado un rápido crecimiento desde su fundación hasta 2015, donde dicho crecimiento ha decelerado de manera notable.

La *API* de *Twitter* nos permite acceder a prácticamente todos los datos del usuario, entre los que destacan sus publicaciones, su *timeline*, su dirección de correo electrónico y su foto de perfil.

2.5. Dispositivos Android.

En el desarrollo de este TFM se ha utilizado un dispositivo físico *Huawei Mate 8* con la versión 7.0 de *Android* (*API Level 24 – Nougat*) y un dispositivo virtualizado *LG Nexus 5X* con la versión anterior 6.0 (*API Level 23 – Marshmallow)*. De esta manera no sólo se ha comprobado que la *App* funciona en nuestro *SO Objetivo* (*Android 6.0*), sino también en una versión posterior.

Dentro de este apartado, debemos diferenciar el *Hardware* y el *Sistema Operativo*.

2.5.1. Hardware.

Un *smartphone* es un teléfono móvil con un avanzado sistema operativo que combina las capacidades de un PC con otras funcionalidades útiles para el uso móvil o táctil. El primer *smartphone*, llamado *Simon Personal Communicator*, fue creado por la empresa *IBM* en 1994, y combinaba las capacidades de un teléfono móvil con las de una *PDA*[14]. Poseía una CPU *Vadem* de 16 bits en arquitectura x86, a 16MHz, y era capaz de enviar y recibir *faxes*, *e-mails* y *mensajes de busca*.

Por suerte, en este proyecto contaremos con una tecnología un poco más reciente. Se utilizarán dos dispositivos móviles (uno de ellos *virtualizado*) para realizar las diferentes pruebas de funcionamiento en base a *dos sistemas operativos Android*, de los que hablaremos posteriormente. Dichos dispositivos son un *Huawei Mate 8* y un *LG Nexus 5X*. Las características de ambos dispositivos se describen con detalle en el apartado de ***Metodologías, Recursos y Plan de Trabajo.***

2.5.2. Sistema Operativo.

El sistema operativo Android es el sistema operativo móvil más extendido actualmente. Basado en Linux, fue diseñado para dispositivos con una pantalla táctil. Inicialmente fue desarrollado por Android Inc., hasta que Google decidió comprar la empresa y dirigir su camino. Android fue presentado en 2007, pero hasta el año siguiente no salió ningún dispositivo con este sistema operativo. Este dispositivo fue el HTC Dream, que fue lanzado en octubre de 2008.

Android está construido sobre el lenguaje C[15], con algunas librerías en C++[16], pero la inmensa mayoría de los programas, o aplicaciones, desarrollados para el mismo están escritos en Java[17], lo que implica que las aplicaciones trabajan sobre una máquina virtual, no sobre el propio dispositivo, es decir, que un desarrollador puede diseñar una aplicación para un dispositivo y utilizarla en otros dispositivos sin necesidad de recompilar o modificar el código fuente.

---

[14] Una *PDA*, o *Personal Digital Assistant*, es un pequeño ordenador de mano/dispositivo móvil originalmente diseñado como agenda electrónica. Poseían pantalla táctil y conexión por WiFi, y hoy en día han sido sustituidas por *smartphones* y/o *tablets*.

[15] C es un lenguaje de programación creado por Dennis Ritchie en 1972 en los Laboratorios Bell. Es un lenguaje imperativo, estructurado y poco tipado, muy orientado al desarrollo de Sistemas Operativos, entre ellos, Unix y Linux.

[16] C++ es un lenguaje de programación creado por Bjarne Stroustrup en los años 80. Inicialmente, fue una extensión del lenguaje C, incluyendo mecanismos de manipulación de objetos.

[17] Java es un lenguaje de programación creado por James Gosling en 1995 en Sun Microsystems. Es un lenguaje orientado a objetos y con una gran portabilidad, ya que trabaja sobre una máquina virtual, no sobre la propia máquina.

Además, se basan en la filosofía de software libre, lo que ha permitido que el ecosistema Android crezca exponencialmente, consiguiendo copar la tasa de mercado de teléfonos inteligentes o *smartphones*. Gracias a esta filosofía, cualquier desarrollador puede diseñar e implementar una aplicación para cualquier dispositivo, lo que facilita el aprendizaje e incrementa el número de aplicaciones diseñadas para el mismo.

Si bien Android empezó con desventaja frente a su competencia directa, Apple, su gran crecimiento ha conseguido que el número de dispositivos Android supere drásticamente a los dispositivos de Apple. Esto ha permitido que la comunidad de desarrolladores de Android sea extensa, facilitando, aún más si cabe, el aprendizaje y desarrollo de aplicaciones. Sabiendo esto, es lógico pensar la razón por la cual se decidió utilizar un dispositivo basado en Android para este proyecto:

- El desarrollo es gratuito, no requiere pagar ningún tipo de tasa.
- Hay gran cantidad de información y el desarrollo de las aplicaciones no es complejo.
- Al trabajar sobre una máquina virtual, una aplicación puede funcionar sobre múltiples dispositivos.

Estado actual del arte.

Hace algunos años, el desarrollo de aplicaciones híbridas se encontraba menospreciado debido a su falta de eficiencia y sencillez a la hora de programar. No obstante, también se debe destacar que la utilización de los Kits de Desarrollo nativos era, por aquel entonces, algo bastante complicado de manejar. Las instalaciones complejas y la escasa documentación hacían que convertirse en un buen desarrollador para un dispositivo no solo fuera difícil, sino que también consumiera una gran cantidad de tiempo.

Poco a poco, el desarrollo híbrido comenzó a ganar renombre. Las continuas mejoras en sus funcionalidades y en su rendimiento iban poco a poco atrayendo a cada vez más entusiastas que buscaban una manera más sencilla de crear sus aplicaciones, o bien lenguajes de programación alternativos a los ofrecidos de manera oficial. Comenzaron a surgir pruebas que dejaban entrever que la eficiencia de los *frameworks* híbridos comenzaba a estar a la par de la del desarrollo nativo, pero facilitando mucho más el mismo. Estos hechos, unidos a la capacidad de crear aplicaciones para un enorme abanico de dispositivos sin tener que programar para cada uno de ellos, propició el actual boom del desarrollo híbrido.

Hoy en día, *Ionic Framework* se posiciona como uno de los más importantes (si no el que más) *frameworks* de desarrollo híbrido multiplataforma del mercado. El uso de *Node.JS* para gestionar los *plugins* y la utilización de *Apache Cordova* para proveer de una interacción nativa a las aplicaciones, unido al hecho de que un mismo código pueda ser compilado para diferentes plataformas, hace del desarrollo híbrido una opción muy atractiva. Actualmente, *Ionic* es capaz de generar compilaciones nativas para web, *iOS*, *Android* y, como último añadido, la *Universal Windows Platform*, que permite generar versiones de aplicación tanto para dispositivos móviles con Windows Phone como para el propio sistema operativo Windows 10.

Asimismo, hay que hacer hincapié en el auge de la metodología *BaaS* (**B**ack-End **a**s **a** **S**ervice). En un ecosistema en el que el tiempo es crucial para obtener ventaja sobre los competidores, y en el que cualquier desarrollo adicional puede lastrar un proyecto, el utilizar servicios de empresas reputadas como *Amazon* o *Google* es una tendencia al alza. Es por ello que dicha metodología ha querido utilizarse en este proyecto.

Metodología, Recursos y Plan de Trabajo.

4.1. Definición del Modelo.

Antes de iniciar un proyecto, debe definirse un modelo de desarrollo que permita tener una idea del ciclo de vida que va a tener. Dicho modelo nos permite normalizar los procesos para obtener unas definiciones y estructuras más detalladas. También debe prestarse una gran atención a qué modelo se escoge, dado que una mala elección puede incluso conllevar que el proyecto vuelva a tener que iniciarse desde cero.

Para este proyecto, se decidió utilizar un Modelo de Desarrollo en Espiral debido a la gran cantidad de funcionalidades que se iban a integrar. Dado que se previó que los diferentes elementos del mismo iban a ser independientes, pero iban a entrelazarse, se decidió optar por este modelo para ir desarrollando el proyecto por partes, de manera controlada, hasta llegar a la versión final. Cada una de las nuevas funcionalidades se ha implementado una vez la anterior estaba completamente definida y operativa.

En cada una de las fases del modelo en espiral, se identifican los objetivos a conseguir en la fase actual, las posibles alternativas a la hora de alcanzarlos y las eventuales restricciones que puedan surgir. Tras esto, se inicia el desarrollo del primer prototipo y, tras completarlo, se plantea el prototipo para la siguiente fase. En cada uno de estos ciclos se finaliza una revisión del ciclo anterior y de los planes para el siguiente.

4.2. Aplicación del Modelo.

A la hora de aplicar el modelo, se definieron unas metas a alcanzar en cada una de las fases del proyecto. Al haber seleccionado el modelo de desarrollo en espiral, no se pasaría a la siguiente etapa hasta haber completado la anterior para, de manera incremental, alcanzar el objetivo final propuesto en el proyecto. Estas fases eran las siguientes:

1. Comprensión del funcionamiento de *Ionic Framework*.
2. Comprensión del funcionamiento de *Google Firebase*.
3. Desarrollo de la comunicación entre *Ionic y Firebase*.
4. Permitir la autenticación, registro y recuperación básica de un usuario.
5. Permitir autenticación social del usuario mediante Facebook, Google+ y Twitter.
6. Modificación del paradigma de enrutamiento por defecto de Ionic.
7. Cambio del motor de estilo de CSS a SCSS/SASS.
8. Almacenar datos de Usuarios en Firebase y mostrarlos en la aplicación mediante una vista de Perfil de Usuario.
9. Permitir la comunicación entre dos usuarios.
10. Creación de un sistema de amigos que solo permita la comunicación entre dos usuarios que se han aceptado previamente.
11. Refinamiento de todas las vistas previamente desarrolladas.
12. Permitir la comunicación en una sala entre más de dos usuarios de manera simultánea.
13. Creación de una vista que permita agregar o eliminar miembros de un chat de grupo tras haber sido creado.
14. Implementación del envío de imágenes.
15. Implementación del cambio de avatares de usuario.
16. Implementación de Notificaciones Locales.
17. Implementación inicial de un sistema de feedback de mensajes.
18. Ajustes finales, retoques de diseño y debugging.

Una vez planteados los objetivos de cada etapa, se comenzó el desarrollo de la aplicación.

4.3. Recursos necesarios.

En este apartado se desglosan los recursos que han sido utilizados a la hora de realizar este TFM, así como para realizar diferentes pruebas de funcionamiento.

4.3.1. Hardware.

- **Equipo de Desarrollo (Sobremesa):**
  - CPU: Intel Core i7 3930K @ 3.2GHz.
  - Placa Base: ASUS Rampage IV Extreme.
  - RAM: 32Gb DDR3 @ 1600MHz.
  - HDD: 2Tb Serial ATA @ 5400RPM.

- **Dispositivo físico de pruebas (Smartphone):**
  - Modelo: Huawei Mate 8.
  - CPU: Kirin 950 de 8 núcleos.
  - GPU: ARM Mali T-880 MP4.
  - Pantalla: 1080 x 1920 píxeles @368ppi.
  - OS: Android 7.0 (Nougat – API 24).

4.3.2. Software.

- **Sistema Operativo:**
  - Windows 10 Anniversary Update.

- **Entornos de Desarrollo:**
  - Android Studio.
  - Atom.

- **Editores de Texto:**
  - Microsoft Word 2016.

- **Dispositivo emulado de pruebas (Máquina Virtual):**
  - Modelo: LG Nexus 5X.
  - CPU: x2 ARM Cortex A-57 + x4 ARM Cortex A-52 (6 núcleos).
  - GPU: Adreno 418.
  - Pantalla: 1080 x 1920 pixels @ 423ppi.
  - OS: Android 6.0 (Marshmallow – API 23).

4.4. Plan de Trabajo.

Dado que la metodología a seguir es la del modelo en espiral, a la hora de crear un plan de trabajo se ha decidido alcanzar diferentes fases en las que las distintas funcionalidades iban a ser implementadas. El Plan de Trabajo escogido para el proyecto se detalla a continuación:

1. **Fase de Documentación de Tecnologías:**
    a. Ionic Framework.
    b. Google Firebase.
    c. Apache Cordova.
    d. Google AngularJS.
    e. JavaScript.
2. **Fase de Documentación de Entornos de Desarrollo:**
    a. Android Studio.
    b. Atom.
3. **Fase de Análisis:**
    a. Análisis de Requisitos.
    b. Diseño de la Aplicación.
    c. Implementación de la Aplicación.
4. **Fase de corrección:**
    a. Prueba y testeo en dispositivos.
    b. Búsqueda y corrección de bugs.
5. **Defensa del TFM:**
    a. Escritura de la memoria.
    b. Creación de diapositivas.
    c. Preparación para la exposición.

Análisis.

5.1. Modelo de Dominio.

Este modelo se utiliza para mostrar los tipos de objetos más importantes del sistema, y se compone de los elementos o los eventos del entorno en el que se ejecuta la aplicación.

En el caso de este *TFM*, el usuario podrá autenticarse de manera clásica o mediante login social, crear una cuenta o recuperar la contraseña de una. Una vez dentro de la aplicación, podrá enviar peticiones de amistad a diferentes miembros de la misma, y una vez aceptadas por estos, enviarles mensajes o imágenes, además de crear salas de chat para dos o más usuarios. El usuario también podrá cambiar su imagen de perfil y cerrar la sesión en su dispositivo. Todas estas acciones se llevarán a cabo mediante una comunicación entre *Google Firebase*, las *APIs* de autenticación social (en cada caso) y la aplicación.

El *modelo de Dominio* desarrollado incluye las siguientes entidades:

- ***AppWatchers:*** Es el sistema de comunicación entre *Google Firebase* y la Aplicación. Controla los cambios que se realizan en la base de datos.
- ***AppServices:*** Es el sistema encargado de servir (de ahí su nombre) la información a las diferentes vistas de la aplicación. Se comunica directamente con *Watchers* para actualizar los datos de la aplicación frente a los de *Firebase*.
- ***Notifications:*** Es el módulo dedicado al control de notificaciones locales. Se comunica directamente con *Services* para saber cuándo debe enviarse una notificación al dispositivo que ejecuta la Aplicación.
- ***Index:*** Estado inicial de la Aplicación desde el que se ofrecen todas sus funcionalidades.
- ***Authentication:*** Es el estado en el que el usuario puede introducir sus datos y acceder a la aplicación, además de recuperar su contraseña o de crear una nueva cuenta.
- ***Dashboards:*** Estado en el que el usuario puede ver los mensajes (individuales y de grupo).
- ***UserProfile:*** Estado en el que el usuario puede ver su información personal y modificar su imagen de perfil.
- ***GroupChat:*** Estado en el que el usuario puede enviar mensajes a una conversación de grupo.
- ***SingleChat:*** Es el estado en el que el usuario puede enviar mensajes a una conversación.
- ***Firebase:*** Es la entidad que gestiona todos los datos de nuestra Aplicación.

En la siguiente página, se muestra el diagrama del *modelo de Dominio* para la Aplicación:

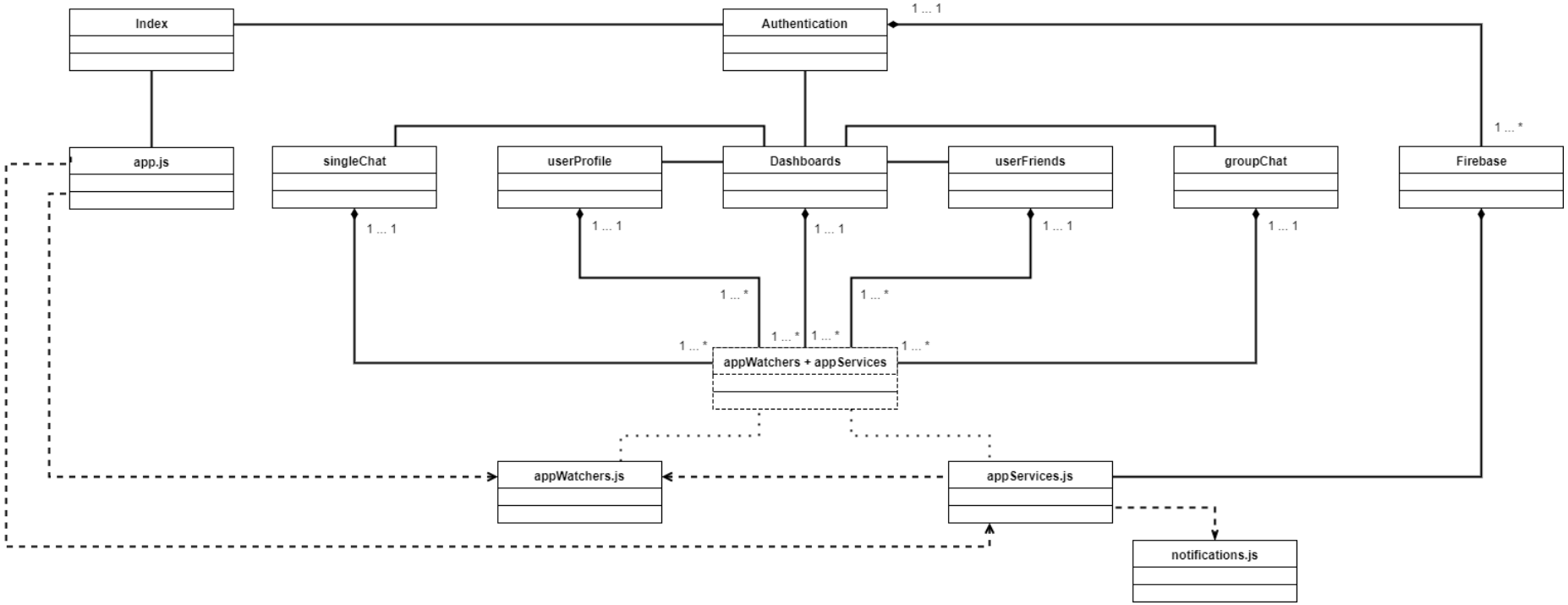

Ilustración 1 - Modelo de Dominio de la Aplicación.

Desde el estado inicial, el usuario debe autenticarse para acceder a las funcionalidades de la aplicación. Dicha autenticación requiere de una conexión a *Firebase* para ser verificada. Una vez hecho esto, el usuario accederá a los diferentes *dashboards* para chats individuales y de grupo. Dichos *dashboards* se conectan a *AppServices* y a *AppWatchers* para proporcionar información acerca de mensajes no leídos y de los chats que el usuario tiene activos. Desde aquí, el usuario puede acceder a su Perfil (*UserProfile*), y a las diferentes conversaciones (*UserProfile, GroupChat*). De nuevo, esto depende enteramente del módulo *AppServices*, ya que es el que extrae toda la información de la *Base de Datos* de *Google Firebase*. Por último, ha de mencionarse que el módulo *Notifications* se conecta al de *AppService* para enviar las notificaciones locales al dispositivo en el momento correcto.

5.2. Modelo de Casos de Uso.

El modelo de Casos de Uso tiene como objetivo la especificación de las acciones que un usuario puede llevar a cabo gracias a la aplicación que está utilizando. Siendo muy útiles para capturar los requisitos funcionales de un sistema, estos modelos se han convertido en una pieza esencial del diseño de software. A continuación, se definirá el modelo de Casos de Uso de la aplicación desarrollada en esta memoria.

5.2.1. Actores.

Los actores son los usuarios del proyecto que se representa, y están directamente relacionados con las capacidades que ofrece la aplicación que van a utilizar desempeñando un rol por cada caso de uso en el que interviene. En nuestro caso, solo es necesario un usuario ya que todas las acciones de la aplicación las realizará siempre el mismo tipo de entidad.

5.2.2. Casos de Uso.

Un caso de uso es una lista de acciones o eventos que definen las interacciones entre un actor y un sistema para alcanzar un propósito. En nuestro caso, el usuario utilizará nuestro sistema para autenticarse y enviar mensajes a otros usuarios autenticados en la plataforma, siempre y cuando hayan sido previamente agregados como amigos y estos aceptasen la propuesta. A continuación, se muestra el diagrama de *casos de uso* para la aplicación descrita en esta memoria:

Ilustración 3 - Diagrama de Casos de Uso

Tras la observación de este diagrama, procederemos a la descripción de cada uno de los *casos de uso* que en él se muestran:

Caso de uso	**Iniciar sesión**
Actores	Usuario.
Tipo	Básico.
Resumen	Autenticación del usuario.
Precondiciones	Ninguna.
Postcondiciones	Las credenciales del usuario se han verificado correctamente.
Flujo Principal	El usuario introduce su nombre de usuario y contraseña.
Flujos alternativos	El usuario se autentica mediante login social utilizando las credenciales de la plataforma escogida.
Excepciones	Un error en el inicio de sesión será notificado al usuario.

Caso de uso	**Recuperar contraseña**
Actores	Usuario
Tipo	Básico
Resumen	Recuperación de contraseña mediante el envío de un email al correo asociado a la cuenta.
Precondiciones	Debe de existir una cuenta registrada mediante el correo proporcionado.
Postcondiciones	Se ha enviado un correo electrónico de verificación al email proporcionado.
Flujo Principal	El usuario introduce su correo electrónico para restablecer su contraseña mediante un enlace.
Flujos alternativos	Ninguno.
Excepciones	Si el correo electrónico no coincide con los almacenados en la Base de Datos, se le notificará al usuario.

Caso de uso	**Registrar un usuario**
Actores	Usuario
Tipo	Básico
Resumen	Registro de una nueva cuenta/usuario.
Precondiciones	Ninguna
Postcondiciones	Los datos introducidos por el usuario han sido incluidos con éxito en la Base de Datos.
Flujo Principal	El usuario introduce los datos necesarios para garantizar una autenticación correcta y segura
Flujos alternativos	Ninguno.
Excepciones	Si existe alguna concordancia con una cuenta actual o faltan datos, se le notificará al usuario.

Caso de uso	Ver chats individuales
Actores	Usuario
Tipo	Básico
Resumen	La aplicación muestra un listado de conversaciones individuales activas.
Precondiciones	Usuario autenticado.
Postcondiciones	Ninguna.
Flujo Principal	Tras la autenticación, la aplicación entra por defecto a esta pantalla y permite al usuario seleccionar una conversación.
Flujos alternativos	Se accede a esta pantalla desde otra, mediante el *footer* de la aplicación.
Excepciones	Si no se puede realizar la comunicación con el servidor, un mensaje en pantalla se lo notificará al usuario.

Caso de uso	Crear chat individual
Actores	Usuario
Tipo	Básico
Resumen	Creación de una nueva sala de chat con otro usuario elegido.
Precondiciones	El usuario elegido debe de haber confirmado una petición de amistad del usuario origen.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón de nueva conversación y selecciona a otro usuario de la lista de contactos verificados de la aplicación.
Flujos alternativos	Desde la pantalla de Amigos, el usuario selecciona a un amigo previamente agregado.
Excepciones	Ninguna.

Caso de uso	Abrir chat individual
Actores	Usuario
Tipo	Básico
Resumen	Selección de una conversación y apertura de los mensajes enviados.
Precondiciones	La vista debe ser la de *Ver chats individuales (dashboard)*.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona una conversación del listado y la aplicación la abre para mostrar todos los mensajes de la misma.
Flujos alternativos	Ninguno.
Excepciones	Si no puede accederse a la conversación, se mostrará un mensaje de error en pantalla.

Caso de uso	Enviar mensaje
Actores	Usuario
Tipo	Básico
Resumen	Envío de un mensaje previamente especificado por el usuario.
Precondiciones	Vista de chat individual.
Postcondiciones	Ninguna.
Flujo Principal	El usuario introduce un mensaje en el campo de texto y lo envía a otro usuario.
Flujos alternativos	El usuario especifica una imagen escogida mediante un popup cuyo origen pueden ser la *Galería* o la *Cámara* del dispositivo, y la envía.
Excepciones	Si el mensaje no puede ser enviado, se mostrará un error en pantalla.

Caso de uso	Ver chats de grupo
Actores	Usuario
Tipo	Básico
Resumen	La aplicación muestra un listado de conversaciones de grupo activas.
Precondiciones	Usuario autenticado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona la vista de conversaciones de grupo mediante el icono en el *footer* y la aplicación muestra un listado de todas las conversaciones activas.
Flujos alternativos	Ninguno.
Excepciones	Si no se puede realizar la comunicación con el servidor, un mensaje en pantalla se lo notificará al usuario.

Caso de uso	Crear chat de grupo
Actores	Usuario
Tipo	Básico
Resumen	Creación de una nueva sala de chat con uno o más usuarios
Precondiciones	Los usuarios elegidos deben de haber confirmado una petición de amistad del usuario que crea el grupo.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón de nueva conversación y selecciona a uno o más usuarios de la lista de contactos verificados de la aplicación.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Abrir chat de grupo
Actores	Usuario
Tipo	Básico
Resumen	Selección de una conversación y apertura de los mensajes enviados.
Precondiciones	La vista debe ser la de *Ver chats de grupo (dashboard)*.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona una conversación del listado y la aplicación la abre para mostrar todos los mensajes de la misma.
Flujos alternativos	Ninguno.
Excepciones	Si no puede accederse a la conversación, se mostrará un mensaje de error en pantalla.

Caso de uso	Crear mensaje
Actores	Usuario
Tipo	Básico
Resumen	Envío de un mensaje previamente especificado por el usuario.
Precondiciones	Vista de chat de grupo.
Postcondiciones	Ninguna.
Flujo Principal	El usuario introduce un mensaje en el campo de texto y lo envía a otros usuarios.
Flujos alternativos	El usuario especifica una imagen escogida mediante un popup cuyo origen pueden ser la *Galería* o la *Cámara* del dispositivo, y la envía.
Excepciones	Si el mensaje no puede ser enviado, se mostrará un error en pantalla.

Caso de uso	Ver información del grupo
Actores	Usuario
Tipo	Básico
Resumen	Ver la imagen y los integrantes de un grupo.
Precondiciones	El usuario pertenece al grupo referenciado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón situado en el *header* de la aplicación y esta cambia a la vista pertinente.
Flujos alternativos	Ninguno.
Excepciones	Ninguno

Caso de uso	Añadir usuarios
Actores	Usuario
Tipo	Básico
Resumen	Agregar a un nuevo usuario a la conversación de grupo.
Precondiciones	Debe existir un grupo previo.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona un nuevo usuario del listado de contactos de la aplicación y lo agrega a la conversación.
Flujos alternativos	El usuario especifica una imagen escogida mediante un popup cuyo origen pueden ser la *Galería* o la *Cámara* del dispositivo, y la envía.
Excepciones	Si el mensaje no puede ser enviado, se mostrará un error en pantalla.

Caso de uso	Cambiar imagen del grupo
Actores	Usuario
Tipo	Básico
Resumen	Cambiar la foto representativa del grupo.
Precondiciones	El usuario debe de formar parte del grupo especificado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona una imagen de su galería o de la cámara de su dispositivo. Dicha imagen se sube a *Firebase* y se sustituye por la imagen anterior.
Flujos alternativos	Ninguno.
Excepciones	Si la imagen no puede ser enviada, se mostrará un error en pantalla.

Caso de uso	Dejar el grupo
Actores	Usuario
Tipo	Básico
Resumen	El usuario activo deja el grupo.
Precondiciones	El usuario debe de formar parte del grupo especificado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón indicado y la aplicación lo elimina del grupo en el que se encontraba.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Eliminar el grupo
Actores	Usuario
Tipo	Básico
Resumen	El usuario activo elimina los mensajes locales del grupo.
Precondiciones	El usuario debe de haber salido del grupo previamente.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón indicado y la aplicación eliminará la información y el acceso a la copia local de ese grupo.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Ver amistades
Actores	Usuario
Tipo	Básico
Resumen	Mostrar un listado de las amistades verificadas por otros usuarios.
Precondiciones	Usuario autenticado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el icono del header y la aplicación carga la vista en consecuencia.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Crear petición de amistad
Actores	Usuario
Tipo	Básico
Resumen	Envío de una petición de amistad a un usuario de la aplicación.
Precondiciones	Debe existir un grupo previo.
Postcondiciones	Ninguna.
Flujo Principal	El usuario selecciona una imagen de su galería o de la cámara de su dispositivo. Dicha imagen se sube a Firebase y se sustituye por la imagen anterior.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Aceptar petición de amistad
Actores	Usuario
Tipo	Básico
Resumen	Aceptar la petición de amistad de un usuario.
Precondiciones	Debe de haberse recibido una petición de amistad.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón de aceptar en la pantalla de nuevas peticiones de amistad.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Rechazar petición de amistad
Actores	Usuario
Tipo	Básico
Resumen	Denegar la petición de amistad de un usuario.
Precondiciones	Debe de haberse recibido una petición de amistad.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón de denegar en la pantalla de nuevas peticiones de amistad.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Ver perfil propio
Actores	Usuario
Tipo	Básico
Resumen	Acceder a la página de perfil del usuario activo.
Precondiciones	El usuario debe de estar autenticado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona el botón situado en el header de la aplicación y accede a su perfil.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Cambiar imagen de perfil
Actores	Usuario
Tipo	Básico
Resumen	Cambiar la imagen del perfil del usuario activo.
Precondiciones	Usuario autenticado.
Postcondiciones	Ninguna.
Flujo Principal	El usuario presiona la imagen de la foto de su perfil y puede escoger entre la Galería o la cámara del dispositivo. La imagen se sube a Firebase y se sustituye por la anterior.
Flujos alternativos	Ninguno.
Excepciones	Si la imagen no puede ser enviada, la aplicación mostrará un mensaje de error.

Caso de uso	Cerrar sesión
Actores	Usuario
Tipo	Básico
Resumen	Cierre de sesión de una cuenta activa.
Precondiciones	El usuario debe de estar previamente autenticado.
Postcondiciones	El usuario no verá el contenido de la aplicación hasta que vuelva a iniciar sesión con sus credenciales.
Flujo Principal	El usuario presiona el botón de cierre de sesión, y la aplicación ejecuta las funciones para eliminar sus credenciales activas en el dispositivo y cambiar la vista a la de Inicio de Sesión.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

Caso de uso	Recibir mensaje
Actores	Usuario
Tipo	Básico
Resumen	Recepción de un mensaje, sea de grupo o individual.
Precondiciones	Un usuario debe de haber enviado un mensaje a un chat en el que el usuario activo esté
Postcondiciones	Ninguna.
Flujo Principal	La aplicación muestra una notificación de mensaje recibido y también *badges* con el número de mensajes nuevos en ese chat.
Flujos alternativos	Ninguno.
Excepciones	Ninguna.

5.3. Modelo de Base de Datos.

Como se ha mencionado con anterioridad, para este proyecto se ha utilizado *Google Firebase*, un servicio que nos provee de una *Base de Datos No Relacional*, o *NoSQL*. Este tipo de estructura de datos prescinde del modelo clásico de Entidad-Relación a favor de una mayor flexibilidad a la hora de implementar y almacenar la información. Otro gran punto a su favor es que la escalabilidad del sistema es *lateral* en lugar de *incremental*, lo que hace que soporte estructuras distribuidas mediante *shards* o fragmentos de la Base de Datos sin ningún tipo de impacto en el rendimiento. Toda la información de la Base de Datos se almacena en un solo fichero en formato *JSON*

Debido a estos puntos, esta tecnología posee un desempeño superior al de su contraparte estructurada, lo que permite ejecutar un servicio de este tipo en equipos pequeños en lugar de en grandes instalaciones de servidores. Hoy en día, el uso del *noSQL* ha ganado bastantes partidarios debido a su rendimiento y a lo barato de su mantenimiento, tanto en términos de *software* como de *hardware*.

El modelo de Datos que representa la estructura de la información cambia notablemente en este tipo de sistema, ya que se pierden las estructuras relacionales. Una estructura arbórea es la manera más acertada a la hora de representar este tipo de Base de Datos. En el caso de este proyecto, la estructura de almacenamiento:

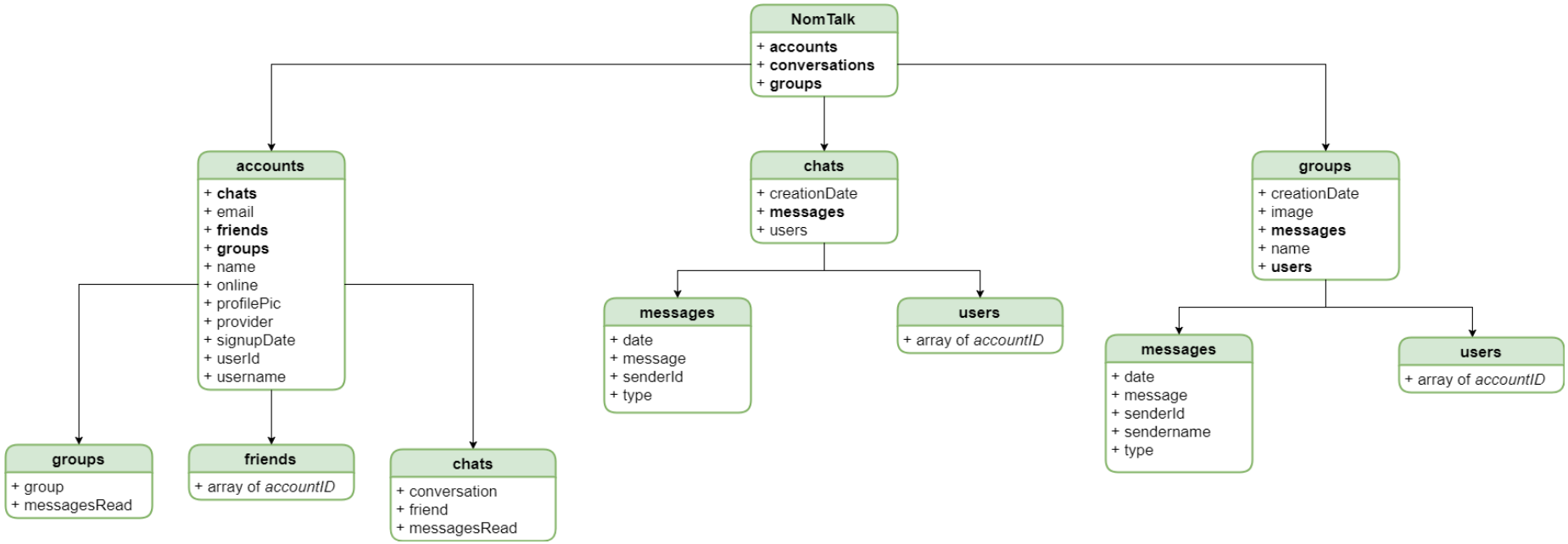

Ilustración 4 - Diseño de la Base de Datos noSQL

Todos los campos en **negrita** poseen un identificador para cada uno de sus elementos.

5.4. Requisitos.

A la hora de crear una aplicación, es imperativo que se definan unos requisitos que permitan saber qué estado debe de alcanzar el desarrollo para considerarse un *Proyecto Mínimo Viable*. A continuación, se definen dichos requisitos.

5.4.1. Requisitos de la Aplicación.

El conjunto de requisitos de funcionamiento de la aplicación detallada en este proyecto es el siguiente:

- Permitir la identificación clara de un usuario frente a otro usuario.
- Permitir la comunicación mediante mensajes entre dos o más usuarios.
- La aplicación debe de funcionar en diferentes plataformas.
- Dentro de la plataforma Android, la aplicación debe de funcionar para las versiones 16 (*Jelly Bean*) hasta la 24 (*Nougat*).

5.3.2. Requisitos de la Interfaz de Usuario.

En los tiempos que corren, el diseño de una interfaz de usuario puede significar la diferencia entre un gran éxito o un estrepitoso fracaso. Por esto, se decidió seguir el patrón de diseño estandarizado por aplicaciones de mensajería actuales como *Whatsapp* o *Telegram* para así reducir el tiempo de aprendizaje del usuario y aumentar la facilidad de uso.

Esta interfaz permite el uso intuitivo y sencillo de todas las funcionalidades de la aplicación mediante iconos y botones, posibilitando la interacción de manera totalmente gráfica con elementos visibles. Se han utilizado colores agradables para evitar el impacto visual, y un nivel de contraste adecuado para que la legibilidad de la interfaz sea lo mayor posible.

Diseño.

6.1. Interfaz de Usuario.

6.1.1. Pantalla de carga (Splash).

En la pantalla de carga se decidió colocar una imagen de fondo que resultara atractiva, estando difuminada para así resaltar la identidad de la aplicación y el *spinner* de carga, situado en la parte inferior.

Esta es la primera pantalla que un usuario ve, y desde ella accedemos o bien al estado de Inicio de Sesión, o bien al *Dashboard*, si el usuario ya se ha autenticado en el dispositivo.

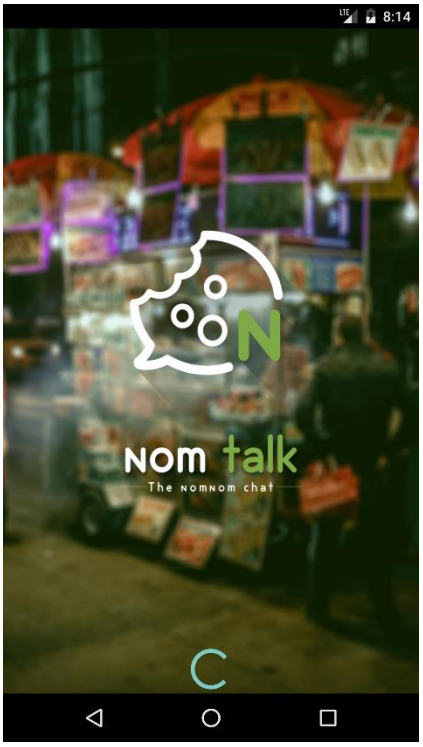

Ilustración 5 - Splash

6.1.2. Inicio de Sesión.

Tras la pantalla de carga, y si el usuario no se ha autenticado previamente, la aplicación mostrará la vista de *Inicio de Sesión*. En ella, el usuario puede introducir su dirección de correo electrónico y su contraseña, si ya se ha registrado previamente, o bien tratar de iniciar sesión mediante un proveedor de servicios de redes sociales. El botón de *Login* se encuentra desactivado hasta rellenar los campos con la información mínima requerida.

El *Social Login* se realiza mediante un proveedor externo (ya que es obligatorio para todos). Cuando el usuario seleccione uno de ellos, la aplicación cargará una vista web en la que aparecerá un cuadro de autenticación característico de cada proveedor. Una vez el usuario introduzca los datos y estos se verifiquen en el proveedor, este enviará un *token* de autenticación y la aplicación, recogiendo la información necesaria, creará una cuenta con la imagen de perfil de la red social escogida, el nombre utilizado y la dirección de correo electrónico de la misma.

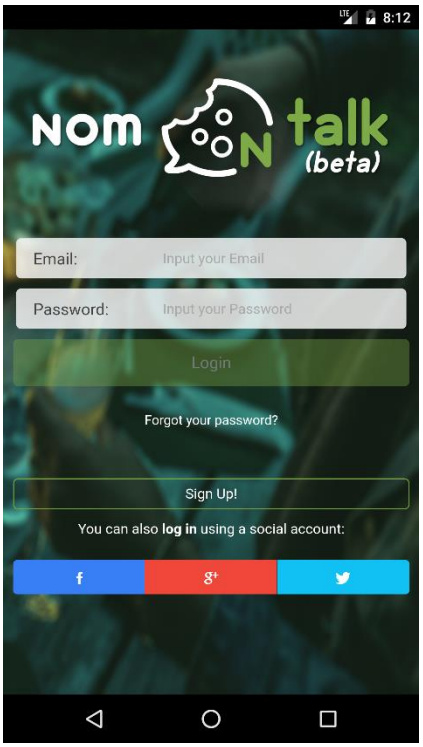

Ilustración 6 - Inicio de Sesión (login)

6.1.3. Creación de Cuentas.

Si el usuario no dispone de una cuenta y prefiere no utilizar *social login*, puede crearse una cuenta clásica en este apartado.

El registro de una cuenta requiere de la correcta formalización de los campos requeridos. Si no se rellenan, el botón de *Sign* Up permanecerá desactivado. Tras rellenar los datos adecuadamente, se creará la cuenta y se cargará directamente la vista de *Dashboard* para el nuevo usuario registrado.

En todo momento, el usuario puede volver a la vista anterior seleccionando el apartado inferior. Asimismo, a partir del momento del registro, el usuario podrá utilizar el *login* clásico desde la vista de *Inicio de Sesión*.

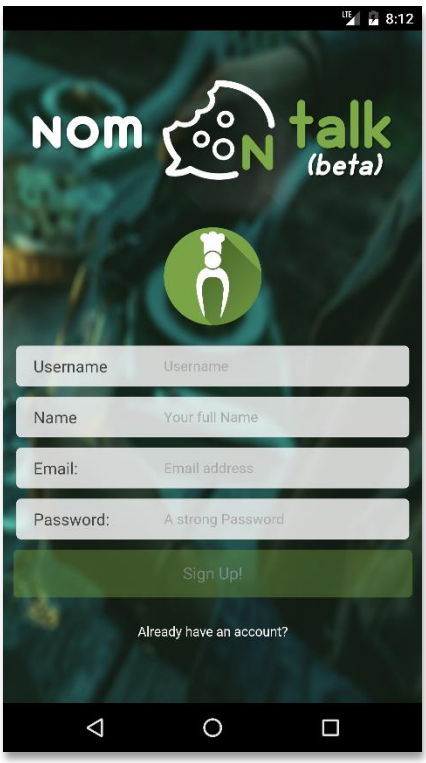

Ilustración 7 - Registro (Sign Up)

6.1.4. Recuperación de Contraseña.

En el caso de que un usuario no logre recordar su clave, la aplicación ofrece un mecanismo de recuperación mediante el envío de un correo electrónico a la dirección especificada en el campo activo. El botón de envío del correo electrónico se encuentra desactivado hasta que se rellene dicho campo de manera correcta.

Una vez rellenado correctamente y pulsado el botón, se enviará un correo de verificación a la dirección de correo electrónico proporcionada, siempre que esta se encuentre dentro de la Base de Datos de la Aplicación. De no estarlo, no sucederá nada.

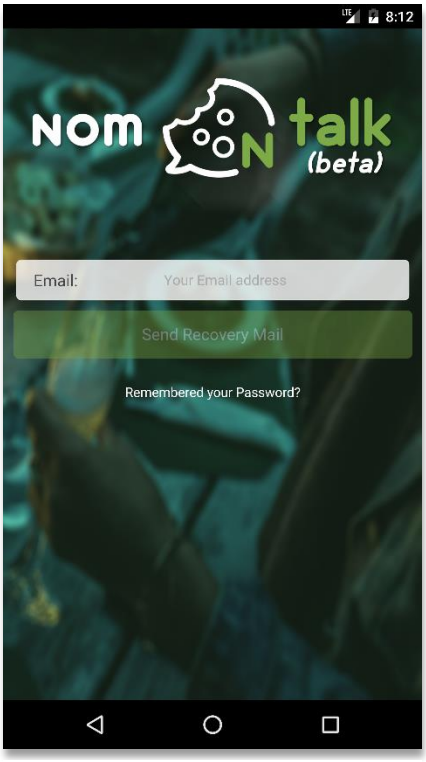

Ilustración 8 - Recuperación de Cuenta

6.1.5. Dashboard.

Una vez el usuario ha sido autenticado, la primera pantalla que la aplicación mostrará en su apertura será la del *Dashboard*, en la cual tendremos un listado de todos los chats individuales activos. Cada elemento de este listado incorpora la imagen de perfil de la persona con la que conversamos, su nombre, el último mensaje enviado, la hora a la que se envió y un *badge* que nos indica cuantos mensajes sin leer hay en esa conversación.

Asimismo, también tendremos *badges* junto a los iconos de *Amistades*, *Dashboard* y *Dashboard (Grupos)*, que nos indicaran el número de peticiones y mensajes pendientes en dichas zonas de la aplicación.

Por último, podremos iniciar una nueva conversación mediante el botón circular inferior derecho de la pantalla.

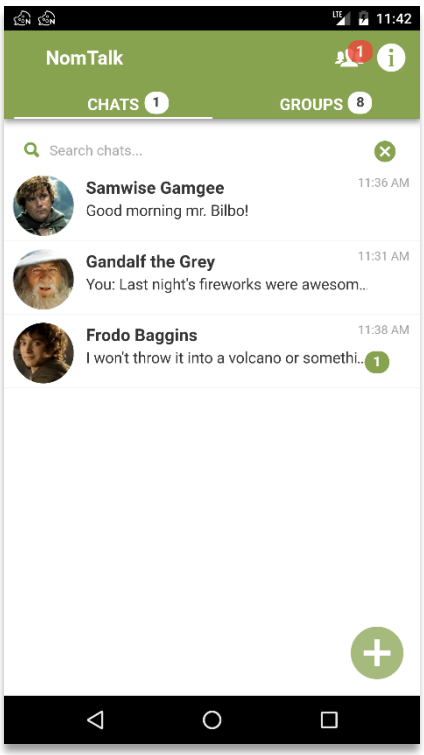

Ilustración 9 - Dashboard

6.1.6. Crear nueva conversación.

Accedemos a este apartado desde la pantalla del *Dashboard* individual. En esta vista, tenemos un listado de usuarios que viene provisto por las solicitudes de amistad aprobadas para el usuario autenticado.

A la hora de iniciar una conversación, basta con presionar sobre el usuario que deseemos y automáticamente se abrirá la pantalla de conversación pertinente.

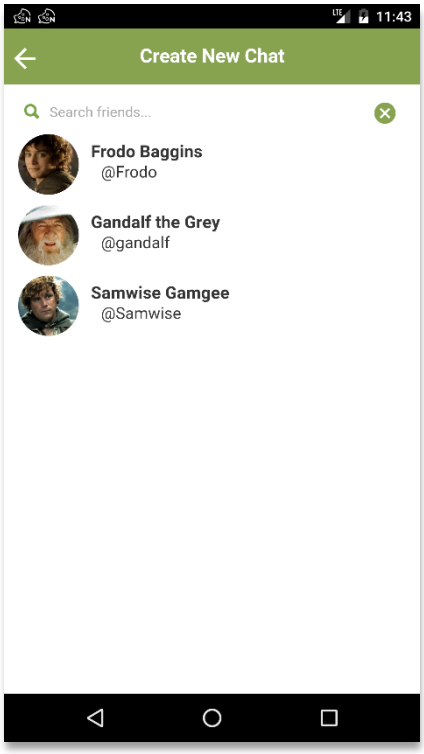

Ilustración 10 - Iniciar nuevo chat

6.1.7. Dashboard (Grupos).

Para esta vista, tenemos información similar a la mostrada en el *Dashboard* individual, pero en este caso orientada a los grupos de chat. De nuevo, se mostrará un listado de conversaciones previamente creadas. Cuando queramos añadir una nueva conversación, basta con presionar el botón inferior derecho.

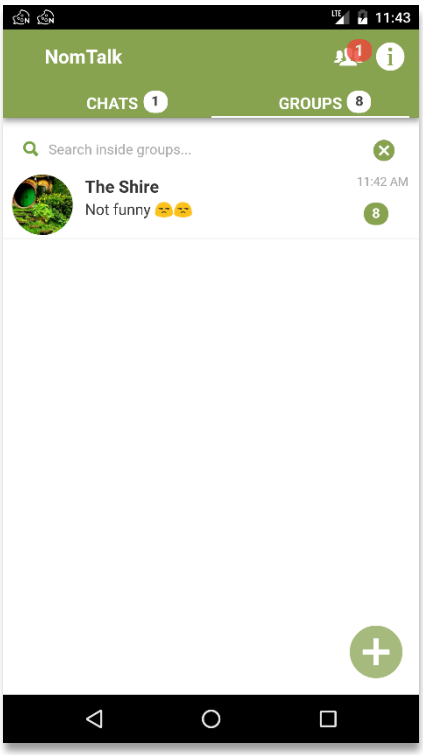

Ilustración 11 - Dashboard de grupo

6.1.8. Crear nuevo grupo.

Accedemos a esta pantalla desde el botón inferior derecho de la vista de *Dashboard de Grupos*. Una vez en ella, se deberá introducir un nombre para el grupo, y opcionalmente una imagen (de no hacerlo, se usará una imagen por defecto). Para añadir miembros al grupo, basta con seleccionarlos del listado inferior, y se irán agregando al chat. Para eliminarlos, solo hay que presionar sobre sus imágenes de nuevo.

Si los campos se han rellenado de manera adecuada, el botón de creación de la conversación de grupo quedará habilitado y, tras presionarlo, accederemos a la vista de chat para ese grupo.

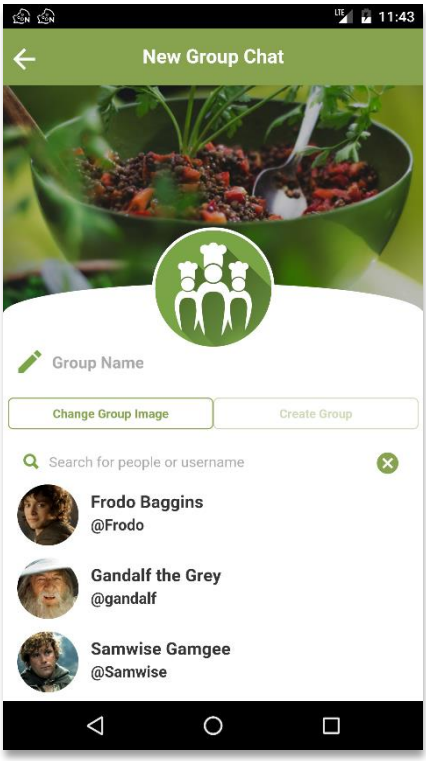

Ilustración 12 - Creación de Grupo

6.1.9. Conversación Individual.

El acceso a esta pantalla se realiza al presionar una de las conversaciones individuales mostradas en el *Dashboard* individual. Una vez dentro, podremos enviar mensajes escribiendo en el campo inferior de la pantalla, o bien enviar una imagen mediante el icono situado a la izquierda de dicho campo. La vista se actualiza automáticamente si el otro usuario ha agregado un mensaje a la Base de Datos mediante la aplicación.

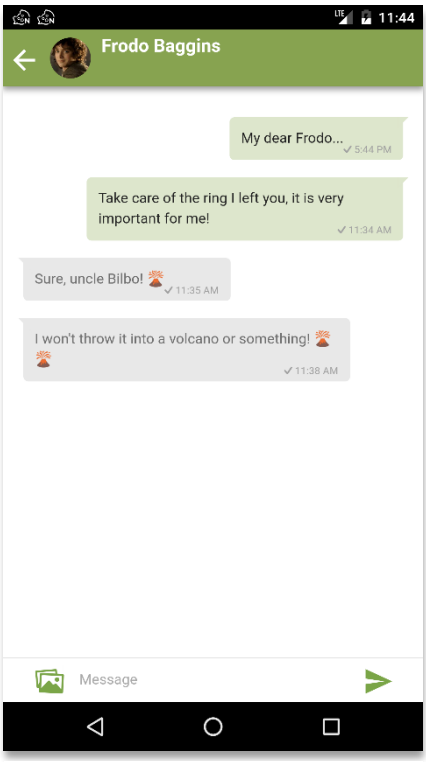

Ilustración 13 - Chat individual

6.1.10. Conversación de grupo.

Para acceder a esta pantalla basta presionar uno de los grupos listados en el *Dashboard de grupos*. Dentro, tendremos las mismas funcionalidades que en el chat individual: enviar mensajes escribiendo en el campo inferior de la pantalla, o bien enviar imágenes mediante el icono situado a la izquierda del campo de texto. En el *header* tenemos un icono adicional que nos lleva a la pantalla de *Perfil de Grupo*. La vista se actualiza automáticamente si cualquiera de los usuarios del grupo ha agregado un mensaje a la Base de Datos mediante la aplicación.

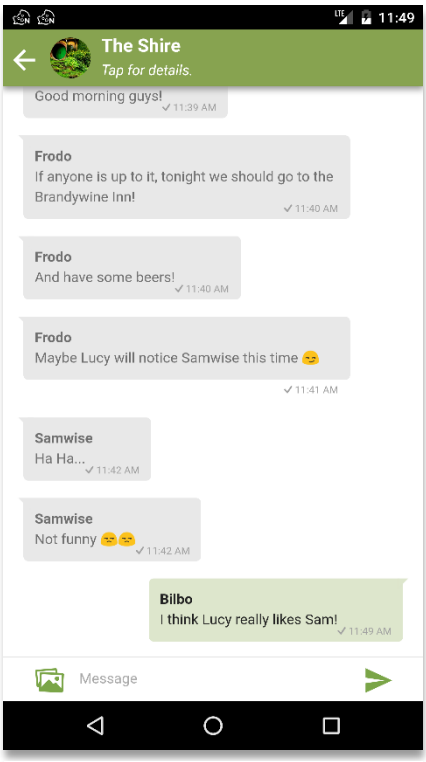

Ilustración 14 - Chat de grupo

6.1.11. Perfil de Grupo.

Accedemos a esta vista desde la pantalla de *Conversación de Grupo*. En ella podemos ver el nombre del grupo, los miembros actuales y tendremos la posibilidad de añadir nuevos integrantes a la conversación. También podemos abandonar el grupo y, tras ello, eliminarlo de nuestro dispositivo.

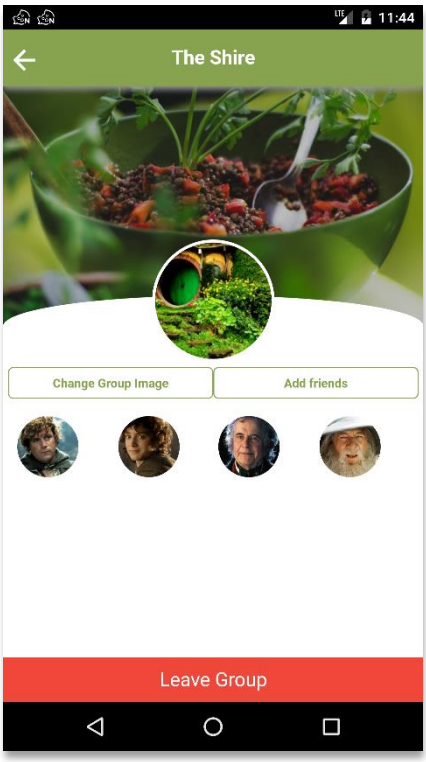

Ilustración 15 – Perfil del Grupo

6.1.12. Amigos y Peticiones de Amistad.

Accedemos a esta pantalla desde el botón inferior derecho de la vista de *Dashboard de Grupos*. Una vez en ella, se deberá introducir un nombre para el grupo, y opcionalmente una imagen (de no hacerlo, se usará una imagen por defecto). Para añadir miembros al grupo, basta con seleccionarlos del listado inferior, y se irán agregando al chat. Para eliminarlos, solo hay que presionar sobre sus imágenes de nuevo.

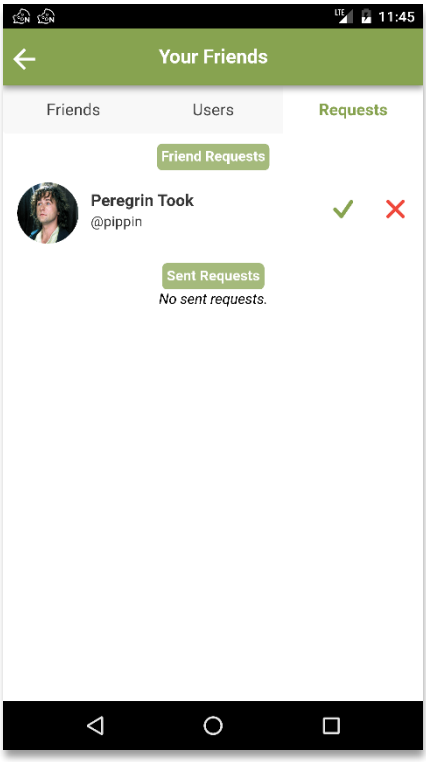

Ilustración 16 - Solicitudes de amistad

6.1.13. Perfil de Usuario.

Accedemos a esta pantalla desde el botón superior derecho de la pantalla del *Dashboard individual* o el de *Grupos*. Dentro, podemos ver nuestra imagen de perfil, el nombre, el nombre de usuario y la fecha de registro del mismo. También podemos cambiar nuestro avatar o cerrar sesión.

Ilustración 17 - Perfil del usuario

6.2. Identidad Digital.

Hoy en día, muchos olvidan que la apariencia de un proyecto puede suponer una grandísima diferencia respecto al éxito que puede tener en el mercado para el que ha sido creado. En el máster cursado se hizo hincapié en algunas asignaturas a este respecto, en especial en las relacionadas con emprendeduría. Debido a esto, se ha diseñado un *branding*[18] específico para el proyecto que dote a la aplicación de un look actual, con su propia seña de identidad y sus detalles específicos. En este apartado, se mostrarán los elementos utilizados en el proyecto para este fin.

En primer lugar, debe mencionarse que este proyecto viene condicionado por un proyecto anterior realizado en el propio máster por los alumnos de la rama de Tecnologías Web y Negocio Digital. Su nombre era **NomNom**, y estaba relacionada con la comida sana y natural, añadiendo el que cualquier persona pudiera disfrutar de ella con una buena relación calidad/precio. Uno de los componentes más importantes de esta aplicación era la creación de un chat que permitiera a los usuarios comunicarse entre sí a fin de intercambiar recetas y consejos culinarios. Así nació **NomTalk**, el chat de **NomNom**.

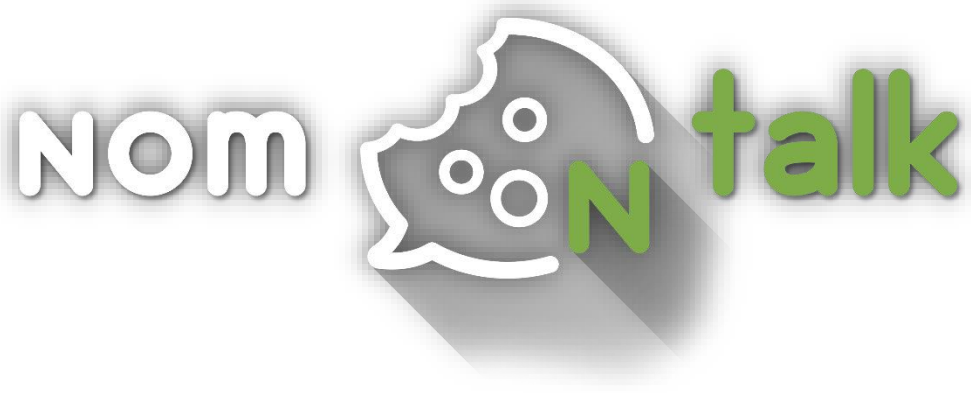

Ilustración 18 - Branding de la aplicación.

La etimología del logosímbolo es sencilla: En inglés, la onomatopeya "*Nom*" hace alusión al ruido que hace una persona al masticar o comer algo, y "*Talk*" hace referencia a hablar o comunicarse con alguien.

Cuando se realiza el *branding*, se deben de tener en cuenta varios factores para generar una buena estrategia. Dichos factores son:

- Resaltar en todo momento los valores de una marca.
- Generar credibilidad y confianza.
- Fortalecer la identidad de sus productos y servicios.
- Diferenciarse de la competencia.

[18] Anglicismo empleado en mercadotecnia, el cual hace referencia al proceso de crear y construir una marca mediante la administración estratégica del conjunto total de activos vinculados en forma directa o indirecta al nombre y/o logotipo que identifica a la marca (*https://es.wikipedia.org/wiki/Branding*)

Una de las partes más importantes del *branding* es, sin duda alguna, el logotipo. Es el encargado de transmitir la esencia de nuestra aplicación y de las filosofías que sigue la misma. Frente a los diseños complejos y sobrecargados (algo realmente frecuente), es necesario aportar un aire de sencillez, de colorido que resulte atractivo, pero no excesivo. Siguiendo estas normas, se creó el logotipo de **NomTalk**:

Ilustración 19 - Logotipo de NomTalk y estudio de color.

Se han utilizado líneas curvas, las cuales sirven para indicar cierto grado de informalidad y de sencillez, además de dotarlas de grosor para crear una sensación de impacto visual. Respecto a la forma del logotipo, es una burbuja de conversación con un mordisco, acompañada de unos círculos que dan la apariencia de ser una galleta o algo comestible, unidos a la *N* de la misma fuente de texto que posee el logosímbolo principal. Finalmente, la sombra da un aspecto dinámico al indicar movimiento, lo que da a entender el interés de la marca por innovar y seguir mejorando.

El color de estos elementos es blanco, ya que evoca una sensación de pureza o transparencia. El fondo es una gradación del color verde (**#87a350** en hexadecimal), y transmite la sensación de que algo es natural y agradable. Este mismo color es el que ha sido utilizado en la aplicación para resaltar todos los elementos importantes.

Todos estos conceptos, unidos en el logotipo y acompañados del logosímbolo final, muestran que NomTalk es una marca caracterizada por la innovación y por la cercanía al cliente, que representa el valor de la comida sana y natural y el hecho de que pueda compartirse y disfrutarse de ella entre todos.

Implementación.

A la hora de implementar esta aplicación, se ha utilizado *Ionic*, un framework que reúne diferentes características que permiten la realización de aplicaciones híbridas mediante diferentes tecnologías, lenguajes y *API*s. Los lenguajes utilizados a la hora de implementar la interfaz y las funcionalidades han sido *HTML5, CSS3 y JavaScript.*

Para el caso de *HTML5*, se ha usado el preprocesador de Ionic, el cual nos dota de un sistema nativo de *bootstrap*[19] y que contiene etiquetas personalizadas que facilitan la programación de las vistas de la aplicación. También se ha utilizado *AngularJS*, un sistema creado por *Google* que permite el paso de variables entre *JavaScript* y *HTML5*, lo que otorga la capacidad de mostrar información en las vistas en base a datos recogidos en el *back-end*.

Respecto a *CSS3*, se ha utilizado la variante de *SCSS/SASS*, la cual nos permite realizar una sobrecarga de las variables internas del *Bootstrap* de *Ionic* sin tener que modificar los archivos originales del *framework*, todo ello sin la utilización de ficheros ni lenguajes adicionales.

Finalmente, *JavaScript* se ha usado para realizar el *back-end* del proyecto, sirviendo de puente para la conexión entre *Firebase* y nuestra aplicación, para extraer y mostrar los datos necesarios para cumplir los requisitos funcionales previamente mencionados.

7.1. Herramientas Utilizadas.

7.1.1. Android Development Studio.

Pese a no haber sido utilizado de manera directa, sí que se ha hecho uso de algunas características del entorno de desarrollo de *Android* diseñado por Google. En este caso, hemos utilizado el sistema de emulación del *Android Development Studio* para probar la aplicación en un dispositivo virtualizado diferente al utilizado, con una *API* inferior.

También se ha utilizado el *Android SDK Manager* para descargar diferentes imágenes de sistema (todas ellas libres de aplicaciones de terceros) y organizarlas de manera eficiente.

Cabe destacar que ambos componentes se han actualizado para estar siempre en la versión estable, debido a que de esa manera podemos comprobar de manera más precisa el correcto funcionamiento de la aplicación en el dispositivo emulado, además de contar con las últimas mejoras y parches para la compilación de código *Ionic* en *Android.*

7.1.2. GitHub Atom.

Atom[20] es editor de código *open-source* desarrollado por *GitHub* para *Windows*, *Linux* y *macOS*, escrito en *Node.JS* y construido en base a tecnologías web. Su filosofía de paquetes hace que sea compatible con muchos de los lenguajes utilizados hoy en día, y la comunidad se encarga de su mantenimiento y actualización.

7.1.3. Draw.io.

Draw.io[21] es un sitio web que ofrece una herramienta online gratuita de creación de esquemas. En este proyecto, se ha utilizado para generar diagramas de los modelos de *Dominio, Casos de uso* y de la *Base de Datos.*

[19] *Bootstrap* es, en el ámbito del diseño de aplicaciones web, un término creado para definir un conjunto de reglas CSS que permiten la responsividad de todo el contenido desarrollado bajo dichas reglas.
[20] https://atom.io/
[21] https://www.draw.io/

7.2. Estructura del Proyecto.

A la hora de realizar este proyecto se ha tenido en cuenta la metodología de desarrollo *MVC*[22], además de perseguir un desarrollo con el mayor nivel de modularidad posible. En este apartado se explicarán las metodologías escogidas a la hora de diseñar dichos módulos y el motivo de dicho diseño.

7.2.1. Vistas.

Ionic Framework agrupa las vistas de una determinada manera para que la estructuración de la aplicación sea independiente. Todo se inicia a partir del archivo *index.html*, situado en la raíz del proyecto. En este fichero se definen mediante etiquetas *script* todos los controladores para las vistas, funcionalidades y *APIs* de terceros que la aplicación debe utilizar, así como su localización en la estructura del proyecto.

En nuestro caso, primero cargamos las *APIs* de *Ionic* y *Cordova*, luego los controladores de los servicios que ofrecemos, la configuración de la conexión con *Google Firebase* y, por último, los controladores de todas nuestras vistas. Finalmente, en el fichero cargamos la sección de la aplicación en la que definimos el contenido de vistas de *Ionic* junto a la declaración del estilo principal (*bar-balanced*, en este caso).

```
<!-- Script de conexión a Firebase y Variables de Conexión -->
<!-- Este script permite la conexión y el intercambio de datos con
Google Firebase, además de permitir la utilización de las diferentes
funciones de esta API. -->
<script src="https://www.gstatic.com/firebasejs/3.3.0/firebase.js"></script>
<script>
  var config = {
    apiKey: "AIzaSyAUq2oSSWq2GjbuP62KNIo4TFiQ46ysN18",
    authDomain: "nom-talk.firebaseapp.com",
    databaseURL: "https://nom-talk.firebaseio.com",
    storageBucket: "nom-talk.appspot.com",
    messagingSenderId: "709050415731"
  };
  firebase.initializeApp(config);
</script>
<!-- ////////////////////////////////////////////////////////////// -->
```

Ilustración 20 - Script de conexión a Google Firebase

El diseño de todas las vistas de la aplicación se encuentra definido en el apartado ***6.1. Interfaz de Usuario***. No obstante, sí que debe hacerse hincapié en que *angularJS* nos permite realizar algunas cosas que en HTML no son posibles, tales como la inclusión de bloques condicionados en base a una variable o la posibilidad de ejecutar una función de un controlador en base a la interacción del usuario con un elemento determinado.

Por último, tenemos que mencionar que las vistas han sido separadas en carpetas, dado que *Ionic Framework* las agrupa bajo el mismo directorio. De esta manera se ha podido llevar a cabo una separación mucho más modular de los controladores, los cuales serán mencionados en el siguiente apartado.

[22] MVC son las siglas de *Model-View-Controller*, o *Modelo-Vista-Controlador*. Es un patrón de arquitectura de software que aboga por la separación de los componentes arquitectónicos de la misma.

7.2.2. Controladores.

A la hora de trabajar con el *framework* utilizado en este proyecto, se ha tenido que realizar un cambio en la estructura de controladores que presenta por defecto. *Ionic* almacena todos los controladores bajo un mismo archivo, lo cual no es aconsejable si seguimos una metodología *MVC*. Es por ello por lo que, como se mencionó en el apartado anterior, se creó un sistema de carpetas para almacenar las vistas junto a sus controladores.

Fuera de este sistema se han almacenado los controladores de las funcionalidades en la ubicación por defecto (*/www/js*). A continuación, se realizará una descripción de los aspectos más importantes de cada uno de estos controladores.

7.2.2.1. Controlador: app.js

Este controlador es el encargado de definir los *estados (states)* de la aplicación para que el usuario pueda moverse por las vistas adecuadamente. Sin ellos, no sería posible realizar llamadas a los estados y, por consiguiente, la aplicación se quedaría bloqueada en una pantalla sin posibilidad de avanzar.

Quizá una de las cosas más importantes que se almacenan en el interior de este archivo sea la constante *Social*, la cual guarda en su interior todas las claves para conectarse a los servicios de autenticación social. Sin ella, solo sería posible iniciar sesión mediante el *login* básico de la dupla "usuario, contraseña", provisto por *Google Firebase*.

```
// Secretos y Claves del Login Social //
.constant('socialKeys', {
  facebookAppId: "1893191810909579",
  googleWebClientId:
  "709050415731-4864v9t300f85mpt2k0b6sblc5ab7o2k.apps.googleusercontent.com",
  twitterKey: "kx54UglDEETcmI8denqAOQc3G",
  twitterSecret: "mixDGAsKqW2fjz31WPoIHzhSb9WGANbAo26t6SF58mkZdq381L"
})
```

Ilustración 21 - Constante de claves de Servicios de Autenticación

En su interior también se ha definido la constante *errMessagesHandler*, que almacena una serie de ristras en un array, las cuales se utilizan para mostrar los diferentes mensajes de error que puedan surgir durante la ejecución de la aplicación.

Por último, en este fichero se encuentran almacenados los diferentes filtros que se utilizan para especificar las distintas búsquedas dentro de la aplicación, desde amigos a conversaciones.

7.2.2.2. Controlador: appWatchers.js

El objetivo de este módulo es el de recuperar información de *Google Firebase* para servírsela al módulo de servicios (*service.js*). El funcionamiento de los *watchers* es muy sencillo: se trata de pequeñas funciones que se conectan a *Firebase* y que, mediante distintos mecanismos, sirven la información de la Base de Datos a nuestra aplicación mediante otras funciones contenidas en el apartado de servicios. La conexión de los *watchers* es permanente, por lo que si se produce un cambio en los datos estos se actualizan en tiempo real en la aplicación.

```
// Watcher encargado del Perfil del Usuario autenticado.
watcherProfile: function(accountId) {
  var ref = firebase.database().ref('accounts/' + accountId);
  var callback = ref.on('value', function(account) {

    var profile = {
      date: $filter('date')(new Date(account.val().dateCreated),
      'MMM dd, yyyy'),
      id: account.key,
      name: account.val().name,
      provider: account.val().provider,
      profilePic: account.val().profilePic,
      username: account.val().username
    };
    $timeout(function() {
      AppServices.profileSet(profile);
      // Añadimos la ID a la lista de IDs excluidas para no mostrar nuestro
      // propio perfil en los filtros de búsqueda propios para
      // los usuarios de la App.
      AppServices.ExcludedIdAddition(profile.id);
    });
  });
  // Se añade el watcher a la lista de watchers para eliminarlo al salir
  // de la app.
  watchers.push({
    ref: ref,
    callback: callback,
    eventType: 'value'
  });
},
```

Ilustración 22 - Ejemplo de Watcher (Perfil de Usuario)

A la hora de realizar la conexión, es la propia API de *Firebase* la que abre los *websockets*[23] necesarios y establece una comunicación permanente, observando en todo momento cualquier *callback*[24] producido por la Base de Datos (de ahí el nombre de *watchers* - observadores).

[23] Tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

[24] Un *Callback* es, en programación, una función "A" que se usa como argumento de otra función "B". Cuando se llama a "B", ésta ejecuta "A". Para conseguirlo, usualmente lo que se pasa a "B" es el puntero a "A".

7.2.2.3. Controlador: appService.js

Una vez conectada a la Base de Datos, nuestra aplicación precisa de un conjunto de funciones capaces de manejar el contenido y los datos de la misma. Para esto, se ha creado el controlador de servicios (*service.js*), el cual posee una librería de funciones destinada al tratamiento de la información que se sirve de manera bidireccional con *Google Firebase*.

En primer lugar, se inicializan las variables que van a contener la información que se va a manejar. Tras esto, se van definiendo todas las funciones que la aplicación va a utilizar, como la preparación de un perfil o la adición de un ID de usuario a la lista de exclusiones del usuario autenticado.

```
// Getter para la imagen de perfil de usuario.
this.profileGetPic = function(id) {
  if ($localStorage.accountId == id) {
    return data.userProfile.profilePic;
  } else {
    for (var i = 0; i < data.userFriends.length; i++) {
      if (data.userFriends[i].id == id) {
        return data.userFriends[i].profilePic;
      }
    }
    for (var i = 0; i < data.usersList.length; i++) {
      if (data.usersList[i].id == id) {
        return data.usersList[i].profilePic;
      }
    }
  }
};
```

Ilustración 22 - Ejemplo de Service (Getter de Imagen de Perfil)

Para el ejemplo de la ilustración, primero se comprueba el ID del usuario actual y, en base al resultado, se recorren las diferentes estructuras de datos buscando la coincidencia con dicho identificador. Una vez encontrado, se devuelve el valor de la imagen que buscamos para que sea tratado de manera específica por el controlador de la vista que ha llamado a *Service.getProfilePic*.

7.2.2.4. Controlador: toolkit.js

Mientras se desarrollaba el proyecto, se descubrió que realizar debugging sin estar utilizando un IDE que no era nativo podía resultar bastante tedioso, al ser necesario tener que guiarse por los mensajes de los *logs* de la consola. Fue por ello que se creó un controlador para lanzar mensajes de error mediante el sistema de *IonicPopup* y de las *alerts* del sistema *Android*.

No obstante, con el paso del tiempo, el sistema de errores se adaptó para no solo dar *feedback* al programador, sino también al usuario. Fue en ese momento cuando se realizó una iteración en la que, mediante los códigos de error de *Google Firebase* y el paso de argumentos desde otros controladores, se desarrolló un sistema para que el usuario recibiese información adicional al producirse algún tipo de error no crítico en la aplicación.

Posteriormente, quedó patente que el sistema de *Ionic Popup* no era lo suficientemente flexible como para aplicar una personalización de su diseño, por lo que hubo que buscar una alternativa. Tras la lectura de la documentación de *Ionic*, se dio con *IonicLoading*, un sistema utilizado para mostrar mensajes de carga en forma de *Popup*. Al permitir el paso de diferentes parámetros, basta con crear funciones específicas que manejen la información a mostrar. El sistema de errores no solo se sustituyó, sino que se amplió para permitir la interactuación con el usuario. Esto propició la creación de diferentes *popups* que permitirían al usuario elegir el tipo de imagen que deseaba mandar o si deseaba salir de la aplicación.

```
// Cuando el usuario va a salir de la aplicación, se muestra este popup.
// En base a la selección que realice, se devuelve un valor que se
// utiliza para salir (o no) de la aplicación.
exit: function(message, text1, icon1, text2, icon2) {
  var dialog = $ionicPopup.confirm({
    cssClass: 'message-confirm',
    template: message,
    buttons: [{
      text: '<i class="icon ' + icon1 + '"></i><p>' + text1 + '</p>',
      type: 'button-confirm',
      onTap: function(e) {
        return true;
      }
    }, {
      text: '<i class="icon ' + icon2 + '"></i><p>' + text2 + '</p>',
      type: 'button-confirm',
      onTap: function(e) {
        return false;
      }
    }]
  });
  return dialog;
},
```

Ilustración 23 - Ejemplo de Popup (Salida de la aplicación)

Como puede observarse en el ejemplo, no hay ningún mensaje establecido, ya que se pasa como parámetro desde el controlador que invoca a la función *exit*. En esta sólo se gestiona la estructura del *popup* y lo que sucede al presionar cada uno de los botones que ofrece el mismo mediante la función *onTap*. Como mérito añadido, se decidió encapsular estas funcionalidades para dotar al proyecto de una mayor modularidad.

7.2.2.5. Controlador: notifications.js

Tras la completitud de los objetivos de este TFM, se fueron añadiendo funcionalidades adicionales para aumentar el valor de la propuesta. Una de las más interesantes fue la adición de un sistema de notificaciones para la aplicación, de modo que, cada vez que se produjera un evento de interés, el usuario pudiera ser notificado del mismo. Para ello, se utilizó el plugin de *Cordova Local Notifications*[25], que nos permite crear notificaciones de manera local para el dispositivo en el que se ejecute nuestra aplicación.

```
// addNew añade/sobreescribe una nueva notificación para chats individuales.
addNew: function(unreadMessages){
  if (unreadMessages == 1){
    alarmTime.setMinutes(alarmTime.getMinutes());
    $cordovaLocalNotification.schedule({
      id: "1234",
      date: alarmTime,
      title: "New message",
      message: "You have " + unreadMessages + " new message!",
      autoCancel: true,
      icon:'res://notification_msg',
      smallIcon: 'res://notificationicon_sm',
      // badge: 10, // Esto añade un badge a la derecha.
      sound: 'file://notifications/sound.mp3'
    });
  } else if (unreadMessages > 1){
    alarmTime.setMinutes(alarmTime.getMinutes());
    $cordovaLocalNotification.schedule({
      id: "1234",
      date: alarmTime,
      title: "New messages",
      message: "You have " + unreadMessages + " new messages!",
      autoCancel: true,
      icon:'res://notification_msg',
      smallIcon: 'res://notificationicon_sm',
      // badge: 10, // Esto añade un badge a la derecha.
      sound: 'file://notifications/sound.mp3'
    });
  }
},
```

Ilustración 24 - Función de Push de Notificación Local (Nuevo Mensaje)

Esta función se ejecuta con el paso de un parámetro que indica los mensajes sin leer. Mediante ristras definidas se especifica qué se quiere mostrar para lanzar la notificación. Se controla el número de mensajes para mostrar los adecuados.

[25] *https://github.com/katzer/cordova-plugin-local-notifications*

7.2.2.6. Controladores de las vistas.

Respecto a los controladores de las vistas, no hay mucho más que mencionar. En ellos se generan los modelos utilizados por las vistas, así como las llamadas a los métodos definidos en los controladores principales.

```
// Para utilizar el botón BACK de Android, debemos "desregistrarlo" para que
// otro controlador pueda utilizarlo. De no hacerlo, siempre se ejecutaría
// el primer controlador que lo utilice de manera exitosa.
var deregisterDashboard = $ionicPlatform.registerBackButtonAction(
  function() {
    var dialog = Toolkit.exit('Exit NomTalk?', 'Exit', 'ion-checkmark-round',
      'Cancel', 'ion-close-round');
    dialog.then(function(exit) {
      switch(exit){
        case true:
          ionic.Platform.exitApp();
          break;
        case false:
          Toolkit.hide();
          break;
        default:
          Toolkit.hide();
      }
    });
  }, 100
);
$scope.$on('$destroy', deregisterDashboard);
```

Ilustración 25 - Función para botón Back (dashboard.js)

Como puede observarse en la *ilustración 26*, se realiza una llamada al sistema de *toolkit.js* para mostrar un *popup* de confirmación. Se pasan como parámetros diferentes ristras y referencias a iconos de *ionic* para realizar correctamente la construcción. Una vez iniciada la función, se espera a que el usuario introduzca su decisión para ejecutar la salida de la aplicación o permanecer en ella. Finalmente, se establece la prioridad (100, para establecerse sobre mensajes de confirmación del sistema *Android*). Ante el evento *destroy* (salida de la pantalla actual), se inicia esta función.

7.2.3. Modelo.

Un modelo es la representación de la información con la cual el sistema opera. Por lo tanto, gestiona todos los accesos a esta información, sean consultas o actualizaciones, implementando también los privilegios de acceso definidos para la aplicación. Envía a la vista aquella parte de la información que se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan a este modelo a través de los controladores.

En el caso de este proyecto, *Google Firebase* se encarga de servir toda la información de nuestra aplicación a medida que el usuario la va solicitando o modificando mediante los controladores. Las vistas se cargan mediante el preprocesador de *Ionic*, el cual compila su código a *HTML* en base a los ficheros *CSS/SCSS*, mostrando la información del *modelo de datos* requerida.

La estructura del Modelo de la aplicación ha sido descrita previamente en la *Ilustración 4* del apartado ***5.3. Modelo de la Base de Datos***.

Pruebas y Resultados.

Durante el trascurso de este *TFM*, se tomó la determinación de no añadir nuevas funcionalidades hasta tener completadas aquellas que ya se habían introducido. Esto conlleva que, al incluir una nueva capacidad, se realizaran pruebas de todo tipo que garantizaran el correcto funcionamiento de dicha capacidad.

Para dichas pruebas se han utilizado dispositivos virtuales y físicos no conectados entre sí de manera directa[26]. Esto nos permite comprobar las funcionalidades entre dos clientes independientemente del método utilizado por ambos para enviar información hacia y desde la aplicación.

8.1. Pruebas básicas.

8.1.1. Autenticación.

Como una de las partes más importantes de la aplicación, la autenticación debe funcionar sin ningún tipo de fallo dado que, de no hacerlo, el usuario no podría entrar al sistema y utilizar todos los servicios que ofrece.

A la hora de comprobar la autenticación, se creó un sistema muy básico en el que, tras introducir los datos, se mostrara una vista simple con los datos almacenados en *Firebase* que el usuario había introducido cuando realizó el registro. No se puede acceder a esta vista si el proveedor de autenticación (*Facebook, Google+ o Firebase, y Twitter*) no envía una respuesta positiva a dicha autenticación.

8.1.2. Perfil de Usuario.

Tras la creación del sistema de autenticación, se determinó que el siguiente paso era la creación de una página de perfil en la que el usuario autenticado pudiera ver su información. Esta vista debía recoger información de *Firebase*, en concreto el Nombre y Apellidos, la Imagen de Perfil y el nombre de usuario. Para ello, se creó una vista con un controlador funcional que accediera a la Base de Datos y extrajera en ese momento la información necesaria.

8.1.3. Sistema de Avisos.

A la hora de realizar debugging, se pensó en utilizar la consola y el log de *ionic*, pero se descubrió que resultaba tedioso y especialmente difícil de seguir debido a toda la información que se muestra en el tiempo de compilación de la aplicación. Por ello, se tomó una aproximación mucho más práctica: La reutilización del sistema de Pop-Ups de *Ionic (ionicPopup)*.

Tras la implementación, no resultó especialmente difícil comprobar su funcionamiento. El sistema de la aplicación se creó para mostrar mensajes de error y realizar seguimientos de zonas de ejecución, por lo que bastaba con realizar una acción determinada para comprobar que en pantalla saltase el *popup* consecuente.

[26] No existe una conexión física entre los dispositivos. Ambos se conectan independientemente a Internet o a la red móvil, y envían y reciben sus resultados desde la misma.

8.1.4. Envío y recepción de mensajes.

Una vez la aplicación era capaz de almacenar y mostrar información de un usuario, se avanzó al siguiente paso, el poder enviar mensajes entre dos usuarios. Esta fue sin duda la etapa más compleja, dado que representaba el tronco central de todo el proyecto.

A la hora de comprobar que los mensajes se almacenaban en *la* base de datos, bastaba con examinar el panel de control de la misma, pero cuando se desarrolló el sistema de recepción, fueron necesarios dos dispositivos para comprobar que los mensajes se recibían correctamente entre los usuarios. Cabe destacar que, en un principio, solo existía una sala de conversaciones y todos los usuarios de la aplicación podían hablar entre sí. Fue a posteriori cuando se implementó un chat individual y la capacidad de hablar en privado en una sola sala.

Una vez modificadas las tablas de *Firebase*, los usuarios podían hablar en privado sin ningún tipo de restricción, siendo capaces de escoger al usuario que se deseara mediante un listado básico que contenía los nombres de todos los usuarios de la aplicación.

8.2. Pruebas Avanzadas.

Una vez comprobadas las funcionalidades básicas de la aplicación, se procedió a implementar sistemas adicionales. Esta etapa no comenzó hasta que el proyecto alcanzó un estado estable respecto a su funcionamiento básico. A continuación, se definen las pruebas realizadas para comprobar el estado operativo de las nuevas capacidades añadidas.

8.2.1. Envío y recepción de solicitudes de amistad.

Esta funcionalidad se implementó debido a la idea de pensar que podría resultar incómodo que cualquier persona pudiera hablar con el usuario autenticado. Esto podría generar un sinfín de mensajes al día y resultar en un funcionamiento inadecuado.

El sistema de solicitudes de amistad se entrelaza con el de la creación de una conversación de chat de cierta manera. Se crea un listado de amigos (usuarios que han aceptado la petición de amistad) y a la hora de seleccionar un usuario para iniciar una conversación, sólo se muestran dichos usuarios verificados. El resto es filtrado y descartado.

Para comprobar esta funcionalidad, bastó con generar dos listados (actualmente en uso en la aplicación final): uno para todos los usuarios de la aplicación, mediante el cual enviar las peticiones; y otro que contuviera los usuarios que aceptaran dichas peticiones. A la hora de iniciar un chat, se utiliza este último listado para que solo pueda realizarse una conversación con los usuarios verificados.

8.2.2. Envío de imágenes.

A la hora de crear el envío de imágenes, se pensó en un nuevo sistema independiente del anterior, pero dado que iba a compartir muchos de los componentes la idea quedó desechada.

En este caso, se creó una variante de la función del envío de mensajes que, en combinación con el sistema de avisos, nos permite enviar un objeto a *Firebase Storage* para guardarlo en formato *blob*[27] y después referenciarlo desde nuestras tablas.

Una vez implementado, bastó con enviar una imagen al chat individual y comprobar que esta se mostraba adecuadamente. También es importante señalar que, si accedemos a *Firebase Storage* dentro de la consola de la base de datos, podemos ver el objeto que se ha almacenado.

8.2.3. Crear un chat de grupo.

Respecto a la parte adicional del proyecto, este añadido supuso otro de los mayores retos debido a que se tuvo que abordar una estrategia diferente. Se cambió la estructura del almacenamiento en la base de datos para permitir diferenciar entre chats de grupo e individuales, además de crear una vista exclusiva para estas nuevas conversaciones de más de dos miembros.

Una vez realizados los cambios y la implementación de esta nueva funcionalidad, se crearon dos vistas especiales con el propósito de especificar que usuarios iban a ser incluidos en el chat, y de comprobar que dicha inclusión se había realizado correctamente en la base de datos. Una vez comprobado que esta funcionalidad operaba de manera adecuada, se procedió a implementar el chat de grupo.

8.2.4. Envío y recepción de mensajes de grupo.

Para este caso, se ha utilizado una versión de la vista de conversaciones individuales con muy pocos cambios gracias a la eficiente reestructuración de la base de datos a la hora de almacenar conversaciones de grupo.

A la hora de comprobar el funcionamiento, se pusieron en marcha dos dispositivos virtuales (además del *LG Nexus 5X*, se utilizó un *Nexus 6* con *Android 7.0*) y el dispositivo físico mencionado previamente (*Huawei Mate 8*).

Una vez los dispositivos estaban en funcionamiento, se inició sesión en ellos con tres cuentas diferentes para observar los mensajes enviados y su correcta recepción en cada uno de ellos.

[27] Un *blob* es un tipo especial de datos muy utilizado para almacenar imágenes. Consiste en la transformación de la información de dicha imagen en un conjunto codificado de datos, idóneo para el almacenamiento en una base de datos.

8.2.5. Cambio de imágenes de Perfil (Usuario y Grupo).

Como funcionalidad adicional, se añadió la posibilidad de cambiar la imagen de perfil del usuario y del grupo. Ambas opciones se engloban en un solo apartado ya que funcionan de la misma manera y utilizan el sistema de *popup* para permitir escoger al usuario el origen de la imagen (galería o cámara).

Para realizar la comprobación de que dicha imagen se establecía correctamente sobre la anterior, no sólo se utilizó la propia aplicación para ver los resultados, sino que también se accedió a la base de datos para verificar que la imagen original había sido borrada del sistema y en su lugar había sido añadida una nueva entrada en formato *blob*.

8.2.6. Sistema de notificaciones locales.

Como parte final del proyecto, se decidió añadir un sistema de notificaciones para avisar al usuario autenticado de la llegada de nuevos mensajes a su aplicación. Esto se realizó mediante la inclusión de un módulo de *Cordova* que permite el acceso al sistema de notificaciones del dispositivo y es capaz de enviar avisos en base a eventos locales de la aplicación.

Las pruebas a la hora de comprobar el correcto funcionamiento de las notificaciones fueron bastante sencillas, ya que es un servicio que de por sí resulta obvio cuando funciona de manera adecuada. No obstante, se introdujeron varios disparadores de eventos para que el sistema *popup* mostrara mensajes respecto a la situación de ejecución del código en determinados momentos, lo que permitía realizar una monitorización precisa de los fragmentos en los que podía haber un error. Una vez se verificó que las notificaciones estaban correctamente implementadas, se eliminaron estos fragmentos del código original.

Conclusiones y Añadidos Futuros.

Tras lo expuesto en la presente memoria, se ha creado una aplicación capaz de autenticar usuarios y de permitir la comunicación entre los mismos mediante las tecnologías previamente descritas en una plataforma móvil de gran actualidad, todo ello gracias a recursos, lenguajes y *frameworks* totalmente gratuitos.

Uno de los puntos más destacables es, sin duda, el nivel de dificultad a la hora de realizar un proyecto con un alcance tan amplio sin disponer de ningún trabajo realizado con anterioridad que ayudara a su desarrollo. Esto ha supuesto el tener que invertir una notable cantidad de tiempo en investigación para poder abordar este *TFM* de la manera más adecuada y óptima, además de realizar lecturas periódicas de la documentación disponible para saber cómo utilizar las tecnologías disponibles y qué funcionalidades era posible implementar dentro del margen de tiempo disponible.

El hecho de haber trabajado no solo con tecnologías de última generación, sino también fuera del ámbito de las tecnologías web vistas en la especialización del Máster en Ingeniería Informática por el alumno no solo ha hecho que descubra nuevos conocimientos, sino que también adquiera una amplia comprensión de los mismos, con especial énfasis en el área de la programación híbrida en dispositivos móviles. El haber adquirido conocimientos referentes a la especialización de Programación en Dispositivos Móviles ha supuesto un importante valor añadido para el estudiante, ya que ha aumentado sus capacidades a la hora de poder llevar al éxito proyectos de un mayor alcance.

También debe recalcarse que se han cumplido con creces los objetivos presentados en la propuesta de proyecto (formulario *TFT01*), los cuales se detallan a continuación:

- Diseño de una Identidad Digital característica para el proyecto.
- Implementación de una Interfaz de Usuario moderna e intuitiva.
- Aplicación completamente responsiva, adaptable a cualquier dispositivo móvil en el que se instale.
- Aplicación compilable multiplataforma.
- Capacidad de realizar un *login* social o, alternativamente, un registro clásico mediante dirección de correo electrónico y contraseña.
- Capacidad de realizar envío de mensajes entre usuarios de la propia aplicación.
- Posibilidad de utilizar el proyecto como módulo para otros proyectos de mayor envergadura.

Adicionalmente, el alumno ha ido un paso más allá y ha añadido funcionalidades que previamente no se habían definido en la propuesta inicial, dotando al proyecto realizado de un valor añadido. Dichas funcionalidades son las siguientes:

- El desarrollo se ha realizado de la manera más modular posible, pudiendo utilizar los módulos por separado para otras aplicaciones.
- La aplicación permite intercambiar no solo mensajes de texto, sino también imágenes.
- Cada usuario posee un perfil propio y la capacidad de cambiar su imagen o avatar.
- Sistema de solicitudes de amistad que impide que un usuario cree conversaciones aleatorias con otros miembros que no le han autorizado a ello.
- La aplicación, además de permitir la conversación entre dos usuarios, permite también la creación de grupos de conversaciones de dos o más usuarios.
- Cada grupo permite la posterior adición de nuevos usuarios, además de poder cambiar la imagen de perfil de dicho grupo.
- Envío de notificaciones locales al dispositivo cuando lleguen mensajes nuevos.

Finalmente, hay que hablar de las posibles mejoras y futuro del trabajo realizado en este TFM. Este proyecto ha sido creado inicialmente para formar parte de un proyecto mayor realizado entre los alumnos del *Máster de Ingeniería Informática*, ya que requería de un módulo de comunicación instantánea entre los usuarios. Este *TFM* es una parte de este proyecto. No obstante, han surgido nuevas ideas que podrían implementarse en un futuro para mejorar este proyecto en diferentes ámbitos. Los añadidos posibles pueden ser los siguientes:

- **Actualización y refactorización a *Ionic 3* y *Angular 4:*** Al comienzo de este proyecto, *Ionic Framework* se encontraba en su versión 1.0, existiendo una versión 2.0 en fase BETA. Se decidió no realizar el proyecto bajo dicho estándar no solo por las constantes actualizaciones, sino también porque la propia empresa creadora del *framework* lo desaconsejaba rotundamente. Actualmente el estándar de *Ionic* se encuentra en su versión 3.0, e incluye multitud de mejoras de rendimiento y soporte para *Angular 4*, entre otras ventajas.
- **Creación de una versión propia para *UWP:*** *Ionic 3* ha añadido en su última versión soporte para la *Universal Windows Platform*, siendo ahora posible realizar una compilación de la aplicación desarrollada para dispositivos de *Microsoft,* tales como *Surface* o *Windows Phone*.
- **Creación de una versión propia para *iOS*:** Como se ha mencionado previamente en esta memoria, el alumno no disponía de dispositivos *Apple* para realizar compilaciones de la aplicación en este entorno. Si se consigue un equipo *Mac* se podría realizar y probar una versión nativa para este tipo de dispositivos. La ventaja respecto al uso de *Ionic Framework* es que los cambios a realizar serían mínimos.
- **Cambio de Notificaciones Locales a Notificaciones Push:** Las notificaciones locales son funcionales siempre que la aplicación esté activa. El siguiente paso es añadir un servicio PUSH nativo mediante *Firebase Cloud Messaging* para así permitir la recepción de notificaciones estando la aplicación cerrada.

Bibliografía.

- Documentación oficial de *Ionic Framework*[28].
- Lista de iconos de Ionic: *Ionicons*[29].
- Documentación oficial de *Google Material Design*[30].
- Documentación oficial de *Google Firebase*[31].
- Documentación oficial de *Apache Cordova*[32].
- Documentación oficial del plugin *Cordova Local Notifications*[33].
- Documentación oficial de *Android Studio*[34].
- Documentación oficial de *SASS*[35].
- *Wikipedia* (para la información referente a compañías y fechas).

[28] https://ionicframework.com/docs/
[29] http://ionicons.com/cheatsheet.html
[30] https://material.io/
[31] https://firebase.google.com/docs/
[32] https://cordova.apache.org/docs/en/latest/
[33] https://github.com/katzer/cordova-plugin-local-notifications/wiki
[34] https://developer.android.com/index.html
[35] http://sass-lang.com/guide