MEMORIES CLOUD

IMPLEMENTACIÓN DE UNA PILA DE RECUERDOS PARA SMARTPHONES

Autores:

David de los Santos Gil

José Serrano Álvarez

TRABAJO FIN DE GRADO EN INGENIERÍA DEL SOFTWARE, FACULTAD DE INFORMÁTICA, UNIVERSIDAD COMPLUTENSE DE MADRID

Madrid, junio de 2018

DIRECTORA:

Dra. Victoria López

Índice general

Índice de figuras

Fuente: Elaboración propia

Figura 6.10. Encuesta aplicación: Pregunta 4

Fuente: Elaboración propia

Índice de tablas

Resumen

Este trabajo consiste en el desarrollo de una aplicación para dispositivos móviles inteligentes. La aplicación tiene como objetivo la gestión de cadenas cortas de texto que simbolizan objetos de la memoria a corto plazo que las personas utilizamos para recordar algunos detalles en momentos puntuales pero que no se mantienen en el tiempo, como sí ocurre con los recuerdos de la memoria a largo plazo. Un ejemplo sencillo es el número de la plaza de parking donde tenemos el coche (lo recordaremos durante unas horas o incluso un par de días pero luego se olvidará definitivamente); por el contrario, una clase de biología o un método de resolución de ecuaciones a veces se recuerda toda la vida (es información que interesa almacenar durante más tiempo en nuestra memoria). Hoy en día la información que recibimos es enorme y una aplicación que ayude a gestionar estos pequeños objetos de memoria se hace imprescindible. Además, cada día es más habitual que las personas hagan uso de diversos dispositivos conectados a internet. Por ello, hemos implementado MemoriesCloud como una aplicación accesible desde cualquiera de los dispositivos con los que trabajamos habitualmente compartiendo la misma información.

En este trabajo se ha desarrollado una aplicación sencilla y ligera para que pueda ser utilizada sin producir sobrecarga perceptible en los dispositivos donde se instala, de forma que su uso sea agradable para el usuario. La implementación se ha hecho siguiendo una estructura de pila de recuerdos donde el usuario puede ocasionalmente organizar la información de acuerdo con sus intereses. La forma principal de interactuar con la pila es el portapapeles de los dispositivos, desde el que se pueden obtener y añadir recuerdos a la pila. La aplicación se ha implementado para que la pila pueda ser compartida desde diferentes dispositivos, aunque el prototipo que se presenta solo está habilitado para dispositivos Android (Smartphones y Tabletas). Para la implementación de la app se ha utilizado el framework React Native, que permite crear aplicaciones nativas con JavaScript reutilizando código entre plataformas.

Abstract

This Final Degree Project of the Software Engineering Degree at Complutense University of Madrid consists of the development of an application for smart mobile devices. The aim of the app is to manage short strings of text that symbolize objects in the short-term memory that people use to remember some specific details but that do not hold up over time, contrary to long-term memories. A simple example is the number of the parking place where we have the car (we will remember it for a few hours or even a couple of days but then it will definitely be forgotten); on the contrary, a class of biology or a method of solving equations is sometimes remembered all of life (it is more information that is interesting to store for a longer time in our memory). Nowadays the information we receive is enormous and an application that helps to manage these small memory objects is essential. In addition, every day people use more different devices connected to the internet. For this reason, we have implemented MemoriesCloud as an application that can be accessed from any of the devices with which we usually work, sharing the same information.

In this work a simple and light application has been developed so that it can be used without producing noticeable overload in the devices where it is installed, therefore its use is pleasant for the user. The implementation has been done with a memory stack structure where the user can occasionally organize the information according to their interests. The main way to interact with the stack is the clipboard of the devices, from which you can obtain and add memories to the stack. The application has been implemented so that the stack can be shared from different devices, although the prototype presented is only enabled for Android devices (Smartphones and Tablets). For the implementation of the app the React Native framework has been used, which allows to create native applications with JavaScript by reusing code between platforms.

1a. Introducción general

Este Trabajo Fin de Grado de Ingeniería Informática de la Facultad de Informática de la Universidad Complutense de Madrid, trata sobre la implementación de una herramienta ligera que resuelva un problema muy habitual entre los usuarios de dispositivos conectados a internet. La herramienta desarrollada así como la redacción de esta memoria han sido posibles gracias al estudio y al entrenamiento realizado durante los años en la Universidad en materias de Ingeniería Informática. En este capítulo se explica con más detalle el problema que tratamos de resolver y su motivación.

En la actualidad la gran mayoría de las personas tienen acceso a internet y cada vez es más habitual que una persona haga uso de varios dispositivos diariamente. Utilizamos ordenadores, smartphones, tabletas, etc., y combinamos su uso dependiendo de diversos factores. En sectores poblacionales desarrollados, y especialmente en el desarrollo de la sociedad digital, la transformación digital de las empresas y el desarrollo de las smartcities y el Internet de las Cosas, el número de dispositivos por persona es de dos o tres, muchas veces incluso mayor. La tendencia es además creciente (por ejemplo, según los estudios de Gartner [1]).

La conexión a internet nos permite compartir información con otras personas o guardar un link donde acceder posteriormente para leer alguna noticia que nos interesó sin necesidad de descargar la información completa. Es esta última acción la que motiva el trabajo que aquí se presenta. Hemos observado que en muchas ocasiones las personas desean guardar links o cualquier otra referencia (título de un libro, nombre de un programa, etc.) para utilizarlo más adelante. También hemos observado que para ello solemos realizar alguna acción ad hoc como enviarnos un link a nuestro correo, copiar el dato en un block de notas, o marcar un link en la sección favoritos de nuestro navegador. Además de esto, hemos visto que no hay un modo organizado de gestionar esa información y como consecuencia el interesado debe invertir algo más de tiempo en localizar la información guardada cuando desea recuperarla, a veces sin éxito.

El objetivo de este trabajo es desarrollar una aplicación que permita gestionar esos pequeños objetos de información combinando el uso de varios dispositivos. Esta información puede generarse de diversas maneras. Por ejemplo, mientras navegamos, encontramos enlaces web o sitios de interés que no guardamos porque a priori no está clara su reutilización inmediata o porque podremos acceder de nuevo fácilmente mediante el uso de los sistemas de búsqueda actualmente instalados en nuestros dispositivos. A pesar de que los propios dispositivos disponen de sistemas de recuperación de información tales como cookies, las cuales están orientadas a recoger información para aplicaciones o clientes externos al dispositivo, muchas veces resultan insuficientes, principalmente si no estamos haciendo uso del mismo dispositivo, si es de un fabricante diferente o si no estábamos identificados en una red. La rapidez en recordar dónde rescatar la información de interés muchas veces es un elemento clave, puesto que, en la mayoría de las ocasiones, si no podemos recordarlo inmediatamente, simplemente no lo usamos y procedemos a ejecutar nuevos procesos de búsqueda con la consecuente repetición de acciones y consumo de tiempo. Para todos esos casos en los que deseamos volver a visitar una página o recordar un dato al que hemos accedido,

pero no recordamos dónde o qué información contenía, se nos ocurre como solución implementar una sencilla pila de registros de acceso rápido donde cualquier usuario pueda recuperar la información (dato, enlace, etc.) que le permita el acceso de una forma inmediata.

Actualmente tenemos algunas soluciones, como marcar sitios web como favoritos, añadirlos a marcadores o, incluso, enviar enlaces a nuestro propio correo, lo cual es una práctica muy habitual. Pero estas soluciones no son todo lo eficaces que nos gustaría: es posible que a pesar de nuestras intenciones a priori, luego no volvamos a querer ver la información, así que habremos marcado páginas como favoritas sin serlo realmente o habremos inundado nuestro correo con basura. Otro ejemplo de uso es cuando nos gustaría enviar a alguien un enlace a algún artículo o sitio al que hemos accedido anteriormente. O cuando necesitamos un dato, como un código, recordamos que hace un tiempo lo estuvimos viendo en nuestro dispositivo, pero no recordamos el enlace donde se encuentra.

La solución a este tipo de problemas viene de la mano del desarrollo de una aplicación informática que es la motivación para nuestro Trabajo de Fin de Grado, ya que no sólo podemos poner en práctica los conocimientos adquiridos durante nuestros estudios universitarios, sino que además tenemos la oportunidad de experimentar la creación de un nuevo servicio y analizar los resultados de su puesta en práctica. Para confirmar el interés y la utilidad del desarrollo de esta aplicación hemos realizado encuestas mediante un formulario de google y se han enviado a usuarios potenciales de la aplicación. En total han contestado la encuesta 74 personas entre las que se encuentran estudiantes universitarios de la Facultad de Informática, amigos y conocidos habituados a utilizar varios dispositivos electrónicos (ordenadores, smartphones, tabletas, etc.) con frecuencia. Los resultados de esta encuesta podemos verlos en el punto 6.2 de esta memoria. En general podemos afirmar que el interés en una aplicación que resuelva el problema planteado es grande, por lo que se justifica su implementación.

1b. General introduction

This Final Degree Project of the Software Engineering Degree at Complutense University of Madrid deals with the implementation of a lightweight tool that solves a very common problem among users of devices connected to the Internet. The tool developed, as well as the writing of this memory, have been possible thanks to the studies taken for years in the University in Computer Engineering. In this chapter, we explain in detail the problem we are trying to solve and its motivation.

Nowadays most of people have access to the internet and it is becoming more common for a person to use several devices daily. We use computers, smartphones, tablets, etc., and we combine their use depending on different factors. In developed population sectors, and especially in the development of the digital society, smartcities and Internet of Things, the number of devices per person is, at least, two or three, many times even greater. The trend is also growing (for example, according to Gartner's studies [1]).

The Internet connection allows us to share information with other people or save a link to access later, for example, to read some news that interested us without having to download the complete information. It is this last action that motivates the work presented here. We have observed that in many occasions people want to save links or any other reference (title of a book, name of a program, etc.) to use it later. We have also observed that for this purpose we usually perform some ad hoc action such as sending a link to our mail, copying the data in a notepad, or marking a link in the favorites section of our browser. In addition, we have seen that there is no organized way to manage this information and as a consequence the interested part should invest some more time in locating the information stored when he wants to recover it, sometimes without success.

The objective of this work is to develop an application that allows managing these small information objects combining the use of several devices. This information can be generated in some ways. For example, while browsing, we find web links or sites of interest that we do not save because, its immediate reuse is not clear or because we can access it again easily by using the search systems currently installed on our devices. Although the devices themselves have information retrieval systems such as cookies, which are aimed at collecting information for applications or clients external to the device, they are often insufficient, mainly if we are not using the same device, if it is from a different manufacturer or if we were not identified in a network. The speed in remembering where to rescue the information of interest is often a key element, since, in most cases, if we can not remember it immediately, we simply do not use it and proceed to execute new search processes with the consequent repetition of actions and consumption of time. For all those cases in which we want to visit again a page or remember a data that we have already accessed, but we do not remember where or what information it contained, we think of a solution to implement a simple stack of quick access records where any user can recover the information (data, link, etc.) that allows access in an immediate way.

We currently have some solutions, such as bookmarking websites, adding them to favorites or even sending links to our own mail, which is a very common practice. But these solutions are not as effective as we would like: it is possible that despite our intentions a priori, then we will not want to see the information again, so we will have marked pages as favorites without really being or we will have flooded our mail with garbage. Another example of use is when we would like to send someone a link to an article or site to which we have previously accessed. Or when we need a data, like a code, we remember that a while ago we were seeing it on our device, but we do not remember where the link is.

The solution to this kind of problems comes hand in hand with the development of a computer application that is the motivation for our Final Degree Project, since not just can we put into practice the knowledge acquired during our university studies, but we also have the opportunity to experience the creation of a new service and analyze the results of its implementation. To confirm the interest and usefulness of the development of this application, we have conducted surveys using a Google form and have been sent to users of the application. In total, the survey has been answered by 74 people, among whom are university students of the Computing Faculty, friends and acquaintances who are used to using some electronic devices (computers, smartphones, tablets, etc.) frequently. The results of this survey can be seen in point 6.2 of this report. In general we can affirm that the interest in an application that solves the problem posed is great, for what its implementation is justified.

2. Introducción a *MemoriesCloud*

El objetivo de este Trabajo de Fin de Grado es diseñar e implementar una herramienta para gestionar pequeñas cadenas de caracteres que interesa recordar en el futuro (memoriesCloud), las cuales estarán almacenadas en una pila. Esta pila es modificada mediante sencillas acciones desde cada uno de los dispositivos habituales para las personas. Si el usuario desea utilizar la aplicación en varios dispositivos y en todos ellos acceder a la misma pila de recuerdos, bastaría con usar las mismas credenciales. Las cadenas de caracteres son enviadas a la nube, junto con el resto de la lista de recuerdos que ya tengamos almacenados para su acceso desde todos los dispositivos compatibles.

Con respecto al funcionamiento de la aplicación es necesaria una identificación, ya que contendrá datos personales del usuario, pero es una identificación simple y rápida para conservar la ligereza de la aplicación. Tras esta identificación no es necesario ningún paso más para acceder a la lista de recuerdos. Para agregar un recuerdo, se puede efectuar mediante dos toques de pantalla, si la información que queremos guardar proviene de una aplicación externa como una aplicación de mensajería o un navegador, o bien, mediante una funcionalidad que permite al usuario poder guardar una cadena de texto que el mismo crea conveniente, como, por ejemplo, la plaza de garaje que anteriormente hemos usado como escenario de uso. Esto hace que el usuario tenga libertad y flexibilidad a la hora de guardar sus recuerdos.

Hemos intentado que la manera de ejecutar las acciones sea la más eficaz e intuitiva posible, ya que nuestra prioridad es que sea una aplicación ligera y sencilla. La gestión del tiempo es clave, y más cuando necesitamos tener la seguridad de poder almacenar un recuerdo para el uso a corto plazo en un tiempo suficientemente pequeño para que se gestione de forma cómoda.

A continuación, podemos ver una imagen que explica de manera general el flujo de nuestra aplicación desde varios dispositivos, todos accediendo a la misma pila de recuerdos que funcionaría como una nube.

Figura 2.1. Ejemplo de funcionamiento de la aplicación
Fuente: Elaboración propia

En la siguiente imagen mostramos un esquema en el que vemos que la misma pila de enlaces puede ser visualizada y actualizada en tiempo real por distintos dispositivos, siempre autentificados con la misma cuenta de usuario. Con esto intentamos solucionar uno de los principales problemas que nos planteábamos al principio, que era la pérdida de información y de tiempo por no poder recuperar datos que nos han interesado en algún momento, ya que en principio los buscábamos desde un dispositivo (por ejemplo, nuestro Smartphone) y al querer visualizarlos de nuevo en otro dispositivo (por ejemplo, un ordenador), no podríamos hacerlo de una manera rápida y cómoda. Nuestra aplicación intenta solucionar este problema.

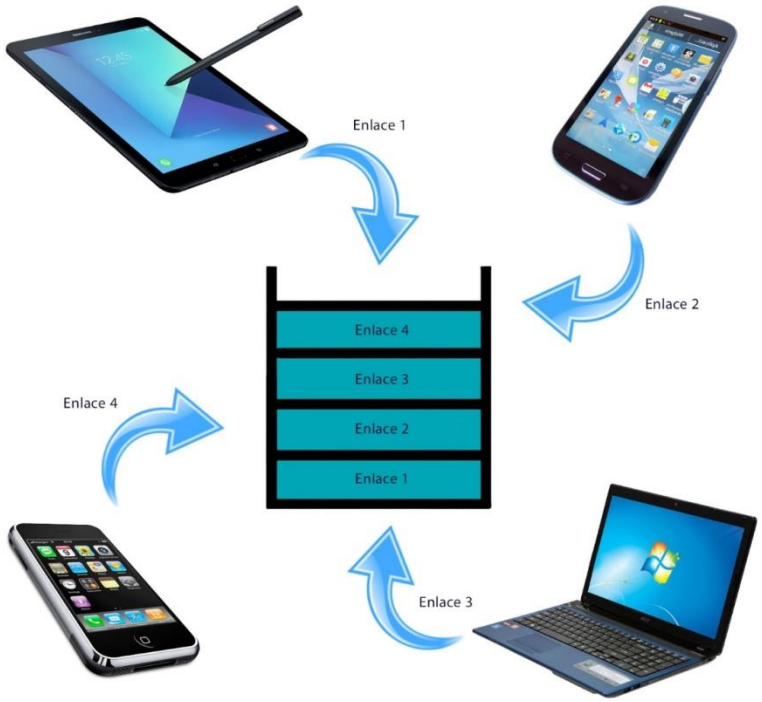


Figura 2.2. Conexión compartida entre dispositivos
Fuente: Elaboración propia

La realización de este Trabajo de Fin de Grado nos ha hecho aprender a utilizar lenguajes que hasta ahora desconocíamos, cosa que al principio nos resultó dura ya que estos lenguajes nos parecieron bastante diferentes a los vistos en la carrera, sobre todo en lo referido a la forma de utilizar parámetros, variables, etc. También tuvimos que aprender a conectar una aplicación con la nube y un servidor reales, hecho que contrasta con lo practicado en la carrera, ya que prácticamente todas las conexiones, como por ejemplo con la base de datos, se daban en el equipo local.

2.1 Encuesta sobre utilidad de la aplicación

Para comprobar la usabilidad real de nuestra aplicación y su futuro éxito en el mercado, hemos creado un cuestionario de Google Forms en el que se definen 7 preguntas con las que recolectar una valiosa información acerca de lo que pensaban personas habituadas al uso de dispositivos inteligentes, sobre nuestra futura aplicación.

Para filtrar y saber qué personas podrían realmente utilizarla, la primera pregunta era sobre la cantidad de dispositivos con conexión a internet que utilizaba normalmente la persona que realizaba la encuesta. Esto nos servía de pista para saber si la gente que utilizaba varios dispositivos, que es uno de nuestros principales objetivos de mercado, realmente echaba de menos una aplicación así. Por esta razón, estas preguntas se enviaron tanto a alumnos de la facultad de informática como a amigos y familiares que sabemos que están acostumbrados a manejar varios dispositivos. Los resultados fueron más o menos los esperados: prácticamente todos los usuarios preguntados utilizaban al menos dos dispositivos con conexión a internet, como podemos ver en el siguiente gráfico.

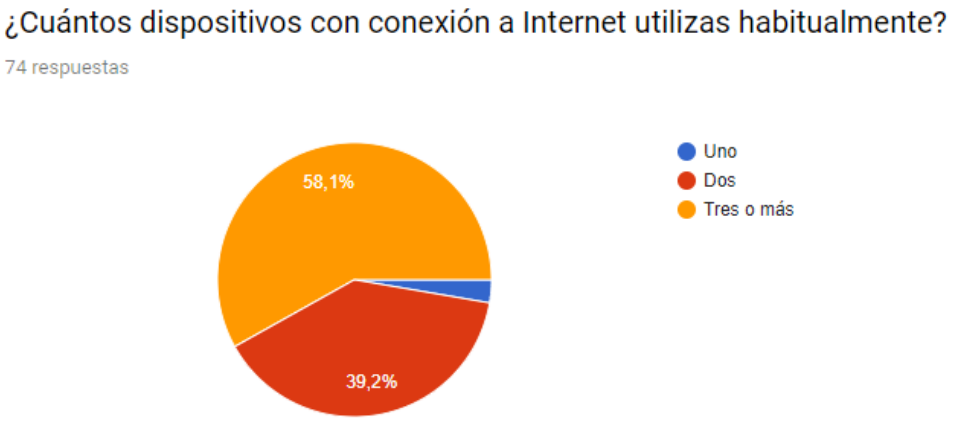


Figura 2.3. Encuesta utilidad de aplicación: Pregunta 1
Fuente: Elaboración propia

Las siguientes preguntas estaban orientadas a ver si el problema que estábamos tratando de resolver, el de poder almacenar pequeños recuerdos de una manera rápida y sencilla, realmente estaba presente en la sociedad. Nos sorprendió gratamente el alto grado de necesidad de una aplicación así, cosa que vimos reflejada en las respuestas de los encuestados. Esto lo notamos, por ejemplo, en las respuestas a una pregunta que manifestaba algo que nos da la impresión que todos hacemos, pero no estamos seguros de si el resto también lo hace, que es escribirnos un correo a nosotros mismos con la intención de recordar algo después. Principalmente nos sorprendió por lo poco eficiente que resulta escribir un correo y mandarlo a nuestra dirección, ya que el tiempo que perdemos entre abrir el correo, escribir destinatario y el cuerpo del mensaje es alto. Los resultados de esta pregunta nos animaron aún más a continuar con la aplicación, ya que nos dimos cuenta de que el almacenamiento de pequeños recuerdos era un problema muy presente.

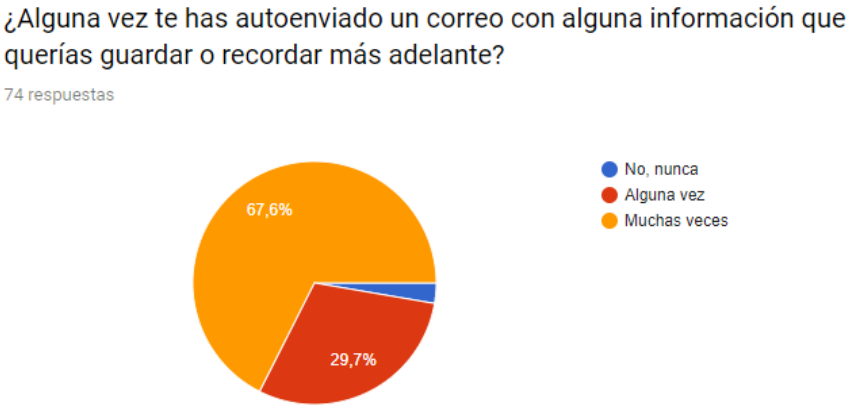


Figura 2.4. Encuesta utilidad de aplicación: Pregunta 2
Fuente: Elaboración propia

El resto de preguntas eran preguntas más genéricas, con las que tanteábamos a los posibles futuros usuarios de nuestra aplicación si esta les podría resultar útil. Las respuestas fueron, en general, muy positivas.

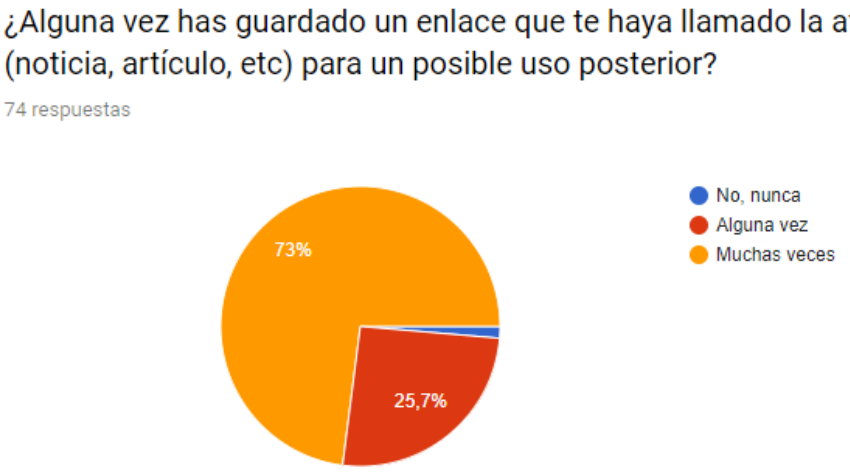


Figura 2.5. Encuesta utilidad de aplicación: Pregunta 3
Fuente: Elaboración propia

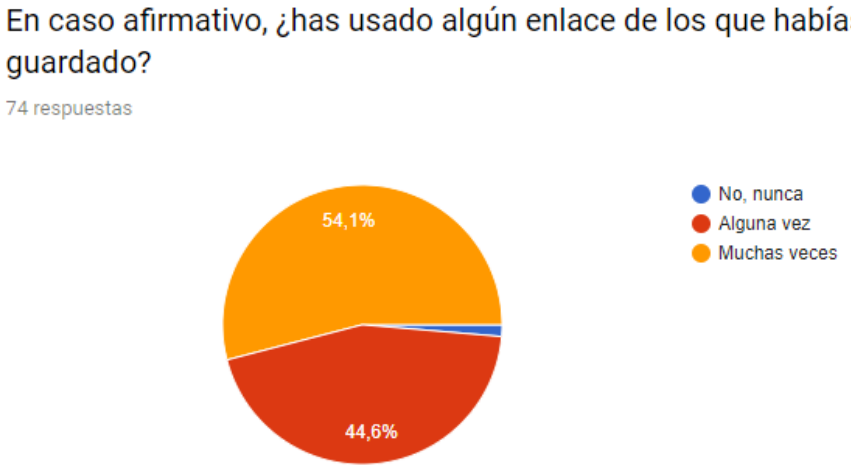


Figura 2.6. Encuesta utilidad de aplicación: Pregunta 4
Fuente: Elaboración propia

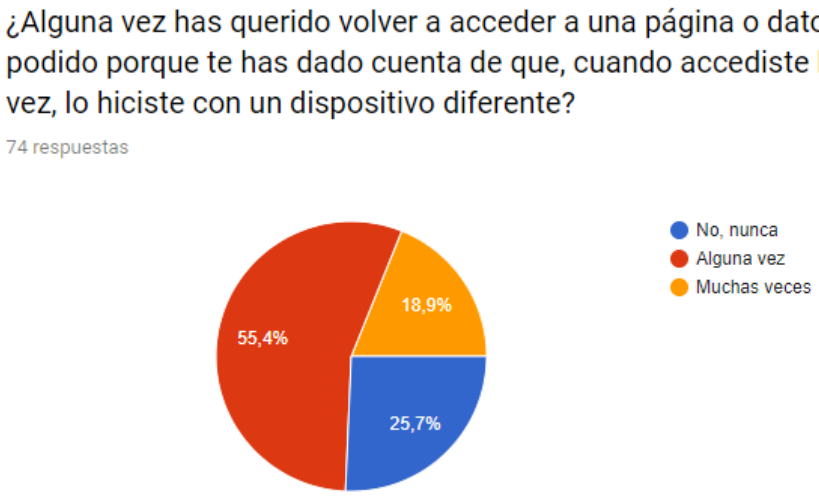


Figura 2.7. Encuesta utilidad de aplicación: Pregunta 5
Fuente: Elaboración propia

Figura 2.8. Encuesta utilidad de aplicación: Pregunta 6
Fuente: Elaboración propia

Por último, le pedimos al usuario que nos cuente un ejemplo en el que haya querido recuperar una información que tenía anteriormente y que, por olvidarla, no haya podido hacerlo. Sabíamos que esta pregunta era la que más iba a costar responder y, en pro de que los encuestados no desistieran de realizar la encuesta, pusimos esta última pregunta como opcional. A pesar de ello, pudimos recabar alrededor de 20 respuestas, de las cuales la mayoría fueron de utilidad. Algunas de las respuestas más destacables fueron:

"Cuando en el autobús me pongo a hacer algún trabajo, busco ejemplos de cómo hacer las cosas en el móvil porque es donde tengo conexión a internet. Cuando luego en casa busco las mismas en el ordenador me acaban saliendo resultados diferentes, por lo que la mayoría de las veces acabo perdiendo las páginas que me habían gustado en un principio."

"Alguna vez que he estado viendo páginas web donde se describía como tratar enfermedades de plantas, pero luego al cabo del tiempo no he encontrado el sitio donde lo estuve viendo."

"*Yo uso Telegram. Justo puedes enviarte mensajes a tu propio usuario. Al ser multiplataforma (existe Telegram de escritorio y Telegram web) no tengo ningún problema para recuperar información.*"

Estas respuestas estaban divididas principalmente en dos grupos. Por un lado, el grupo de la gente que perdía la información o le costaba recordarla, y por otro el grupo que se había buscado una solución alternativa, pero solía ser una solución que solo mitigaba en parte el problema, como el caso de Telegram que mencionamos arriba.

2.2 Análisis DAFO

Antes de comenzar el desarrollo, decidimos realizar un análisis DAFO, que nos permita ver la magnitud del problema, así como sus limitaciones. Este análisis nos ayuda a analizar las Debilidades, Amenazas, Fortalezas y Oportunidades de nuestra aplicación, en la fase previa a su desarrollo. El DAFO nos sirve para definir los objetivos del producto a desarrollar y nos proporciona una mejora en el desarrollo posterior.

Como vemos en la tabla, esta aplicación tiene una gran cantidad de amenazas. Una buena parte de ellas se relacionan con el último punto de la sección de amenazas, en el que hablamos de problemas de gestión de la aplicación. Estos están causados por el hecho de que no estamos acostumbrados a tratar con usuarios reales, que ya sea de forma maliciosa o inintencionada, puedan causar problemas [2].

Otro punto a destacar de la tabla es que podemos encontrar el hecho de que exista una aplicación similar tanto en debilidades como en fortalezas. Por un lado, creemos que esta aplicación, de la que se hablará más adelante, puede hacer que la gente ya esté utilizándola no esté interesada en la nuestra al tener ya el problema resuelto. Por el otro lado, la observación y utilización por nuestra parte nos ha servido para compararla con la aplicación que teníamos en mente y valorar cosas que no se nos habían ocurrido.

A pesar de que el número de amenazas es notable, creemos que el valor de las fortalezas y utilidades es muy alto, por lo cual consideramos que merecía la pena intentar poner en marcha una aplicación así.

Tabla 2.1. Análisis DAFO

Análisis DAFO	**Análisis Interno**	**Análisis Externo**
Negativos	**Debilidades** - La sobrecarga del sistema - La existencia de una aplicación similar	**Amenazas** - La resolución del problema por parte de las grandes empresas - Que aparezcan competidores - Que el tiempo de desarrollo disponible sea insuficiente - Que no haya suficientes personas en las que probar la herramienta - Que tengamos problemas al gestionar la aplicación, al tener que enfrentarnos a usuarios reales.
Positivos	**Fortalezas** - Es una aplicación sencilla de utilizar, muy ligera y rápida - La existencia de una aplicación similar	**Oportunidades** - Hay gente que echa de menos esta utilidad

2.3 Memoria a corto plazo

Para diseñar esta aplicación nos inspiramos en el funcionamiento de la memoria real de un ser humano [3]. El siguiente esquema nos muestra de una manera general el funcionamiento de la memoria. En él podemos observar que, cuando recibimos un evento desde el exterior, se activa nuestra memoria sensorial. En el caso de prestar atención a este evento que ha activado nuestros sentidos, se activa nuestra memoria a corto plazo, la cual puede avanzar en dos direcciones: hacia la memoria a largo plazo, lo cual requerirá un trabajo por nuestra parte al tener que esforzarnos en memorizarla y recuperarla más adelante, o hacia el olvido.

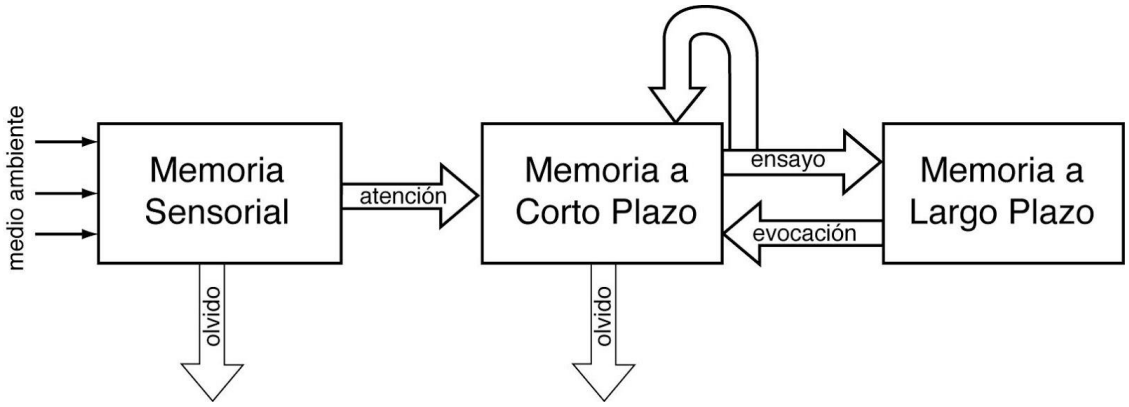


Figura 2.9. Explicación general del funcionamiento de la memoria

Un ejemplo de la memoria a corto plazo podría ser cuando aparcamos nuestro coche en una plaza de garaje. En este caso, tenemos que hacer un esfuerzo por almacenar en nuestra memoria el número o letra de la plaza en la que hemos dejado el coche para, pasados unas horas, recordarla. Es un dato que solo necesitamos hasta que volvamos a por el coche y que olvidaremos tras recogerlo, por lo tanto, es una información que no necesitamos almacenar a largo plazo, es un recuerdo muy concreto que queremos almacenar de forma rápida en cuanto hemos aparcado y recuperar solo en el momento exacto en el que estamos en el parking buscando nuestro coche.

Hay una analogía bastante clara entre la memoria a corto plazo y la memoria RAM de un ordenador [4]. En ambos casos la memoria está presente para recordar datos que queremos recordar durante una cantidad de tiempo breve. Tras esta recogida de datos, es en el estado de reposo cuando esta memoria a corto plazo pasa a transferirse a la memoria a largo plazo. Este estado de reposo en el caso de los humanos es cuando dormimos, y en el caso de los ordenadores cuando están apagados. Como el hipocampo humano, que es donde está almacenada nuestra memoria a corto plazo, el almacenamiento en la memoria RAM es muy volátil, razón por la cual se creó la memoria a largo plazo de los ordenadores: los discos duros, que equivaldría a nuestra memoria a largo plazo, la cual está localizada en el córtex cerebral. Nuestra aplicación quiere simular el funcionamiento de una memoria RAM, o de nuestra memoria a corto plazo.

El intento de simular nuestra memoria a corto plazo tiene como objetivo el no querer malgastar nuestras energías ni nuestro tiempo en intentar recordar y recuperar algo de manera inútil, ya que estamos poniendo a prueba de manera innecesaria nuestra memoria a corto plazo, la cual, como ya hemos explicado, está muy limitada

tanto por su duración como por la capacidad de almacenamiento, sobre todo cuando se están realizando actividades de mayor importancia o que tienen la mente ocupada.

Con esta aplicación intentamos evitar el esfuerzo de retener en nuestra memoria a largo plazo el evento que queremos recordar por unos días u horas, pero evitando que caiga en el olvido. A pesar de una codificación correcta de la información, en una gran parte de los recuerdos tenemos un riesgo importante de sufrir un deterioro en el almacenamiento de esa información, debido a los llamados siete pecados de la memoria [5] enumerados por el investigador Daniel Schacter [6]: distracción, transitoriedad, bloqueo, confusión, sugestión, sesgo y persistencia, los cuales nos limitan a la hora de intentar recuperar esa información.

2.4 Metodología de trabajo y organización de esta memoria

El desarrollo de este trabajo se ha basado en el estudio previo de las necesidades reales de los usuarios potenciales y en metodologías ágiles de desarrollo de software.

En primer lugar, hemos realizado un planteamiento del problema que se ha visto reforzado por la consulta mediante encuesta a un número relevante de personas. Se han analizado escenarios de uso y mediante el análisis DAFO se han determinado las funcionalidades de necesidad real previa a la implementación de la herramienta.

Una vez decidida la implementación del proyecto, el mismo se ha definido mediante la metodología SCRUM [7] para su desarrollo efectivo. Las metodologías ágiles, como SCRUM, se basan en la toma de decisiones, la transparencia y la comunicación entre los miembros del equipo. SCRUM es una metodología de desarrollo de software en la cual se aplican un conjunto de buenas prácticas para trabajar, colaborativamente, en un proyecto. Se realizan entregas parciales y regulares del producto final. SCRUM está recomendada para proyectos complejos, donde los requisitos son cambiantes o poco definidos y donde la innovación, competitividad, flexibilidad y productividad son fundamentales. La Figura 1.4 muestra un pequeño esquema de funcionamiento de esta metodología.

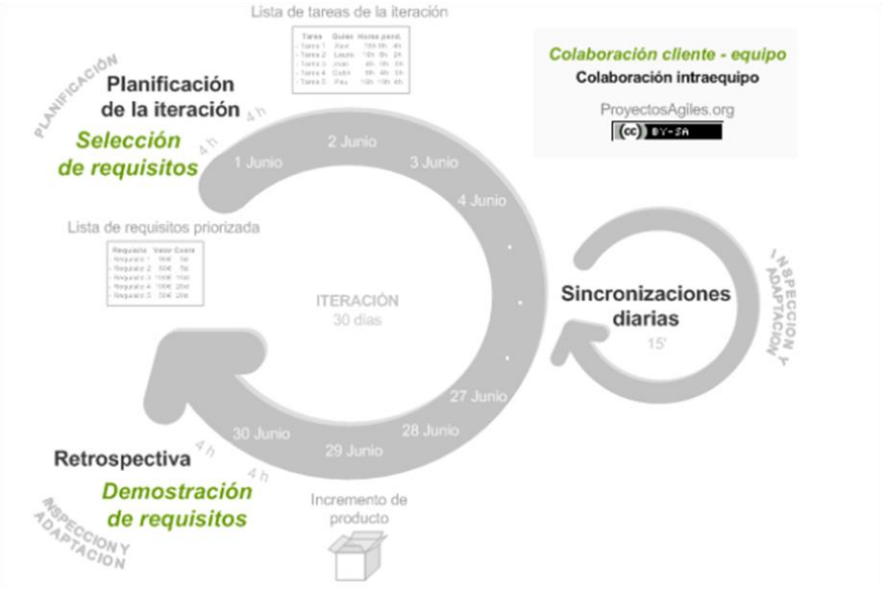


Figura 2.10. Metodología SCRUM

Una de las características de la metodología SCRUM, utilizada para el desarrollo y seguimiento de la aplicación y sus distintas tareas, desde la fase inicial de documentación y aprendizaje hasta la fase final de pruebas, ha sido Kanban [8]. Kanban es un método evolutivo e incremental para gestionar el trabajo, con énfasis en la entrega justo a tiempo, mientras no se sobrecarguen los miembros del equipo. Esta metodología sirve para que todos los participantes en el proyecto tengan una visión global de éste y puedan comprobar las tareas realizadas y que faltan por realizar, así como tener siempre presente el progreso del proyecto de una manera visual. La Figura 1.5 muestra un ejemplo de pizarra Kanban.

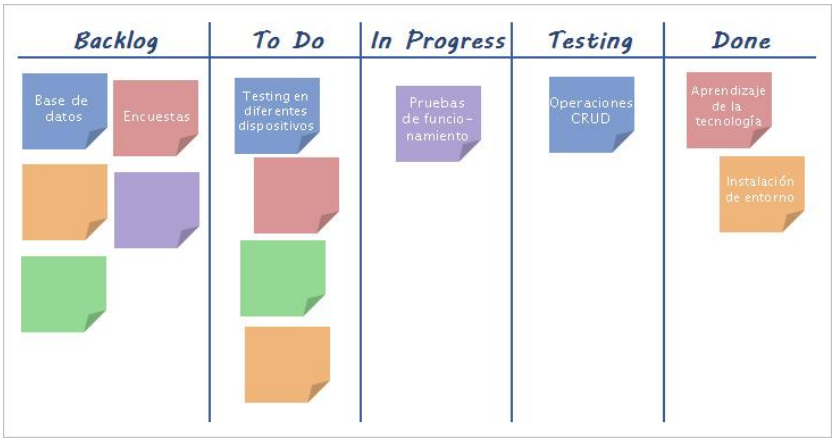


Figura 2.11. Modelo de pizarra Kanban
Fuente: Elaboración propia

SCRUM es una metodología basada en sprints [9] continuos, después de los cuales se realiza la entrega de una parte del producto o de una versión nueva con más funcionalidades. En nuestro caso esta entrega del producto era una nueva entrega realizada a la tutora del trabajo de fin de grado. Estas reuniones, además de para la entrega de ciertas partes del trabajo, han servido para resolver dudas puntuales, normalmente acerca de los requisitos, e ir comunicando los avances y problemas que nos hemos ido encontrando.

Por último, hemos realizado pruebas con un primer prototipo para 10 usuarios durante dos meses. Se han realizado encuestas de satisfacción a estos usuarios con el fin de mejorar el prototipo y de anotar sus ventajas. Los resultados se explican con más detalle en el capítulo 5 de esta memoria.

Además de este capítulo introductorio, este documento consta de los siguientes capítulos. El capítulo 2 describe el estado del arte y las tecnologías relacionadas que hemos analizado como paso previo al desarrollo del proyecto. El capítulo 3 está dedicado a la descripción de la herramienta React Native como elemento clave para la implementación. En el capítulo 4 se describe el desarrollo de la aplicación a partir del análisis de requisitos. En el capítulo 5 se muestran los resultados de la aplicación finalizada así como unas encuestas a usuarios de la misma. Por último, el capítulo 6 contiene las conclusiones y el trabajo futuro.

Esta memoria se complementa con una descripción del trabajo individual realizado por cada uno de los autores, la bibliografía utilizada de interés para el proyecto y un anexo sobre React Native que hemos redactado para nuestro propio uso durante el proyecto pero que consideramos puede ser de interés para la comprensión de esta memoria y también para personas interesadas en trabajos relacionados.

3. Estado del arte y tecnologías relacionadas

En este capítulo se explicará el estado del arte, destacando el funcionamiento de una aplicación real que se asemejaba a nuestra aplicación, así como diferentes alternativas para resolver el problema, especialmente en los navegadores. También en este capítulo hablaremos de las tecnologías y herramientas utilizadas, tanto para el desarrollo de la aplicación como para el control de versiones.

3.1 Estado del arte

Con el fin de mejorar la comprensión del problema y desarrollar una solución lo más útil posible, hemos investigado las alternativas y sistemas relacionados.

La principal similitud que nos hemos encontrado con la aplicación realizada es una aplicación que encontramos en la Play Store de Android llamada “Pocket” [10]. Esta aplicación consiste en una lista de enlaces guardados tanto desde smartphone como desde ordenador. Para utilizarla es necesario descargarla desde la Play Store en el caso de querer utilizarla en un teléfono móvil, y descargar una extensión para navegador en caso de querer utilizarla en nuestro ordenador. No es posible descargarla para ordenadores MAC o teléfonos con sistema operativo iOS. Tras la descarga e instalación de la aplicación, es necesaria la creación de una cuenta, para lo cual nos deja dos opciones: la entrada a la aplicación mediante una cuenta de Google, en cuyo caso tendríamos que seleccionar la cuenta de Google que queremos vincular a la aplicación y una contraseña particular de la aplicación, y un registro normal, introduciendo nombre, apellidos, correo y contraseña. Su funcionamiento es rápido y completo, incluyendo funcionalidad tanto en enlaces como incluso en tweets de la red social Twitter, lo que nos da una idea de las posibilidades que tiene una aplicación así.

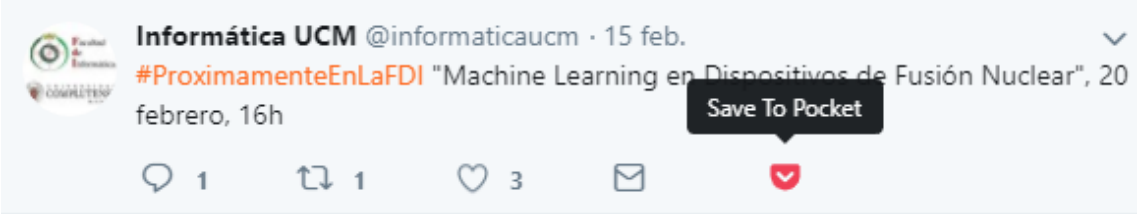


Figura 3.1. Ejemplo de uso de la aplicación “Pocket” en Twitter
Fuente: Perfil de la Facultad de Informática de la UCM en Twitter

Lo factores diferenciales de nuestra aplicación respecto a “Pocket” es la ligereza. Al abrir “Pocket” nos encontramos con el título de lo que hemos guardado, una descripción y, en la mayoría de casos, una foto o vídeo en miniatura, lo que nada más entrar en la aplicación resulta agobiante. Además de esto, la aplicación contiene 3 botones en la parte superior y 5 en la inferior, lo que en nuestro caso consideramos excesivo para una aplicación que buscamos que sea sencilla.

Por un lado, la existencia de la aplicación “Pocket” nos ha ayudado a coger ideas y a ser más críticos con nuestra aplicación ya que tenemos con qué comparar. Por otro lado, tenemos un importante competidor y le quita el factor innovador al proyecto, lo que podemos considerar un reto.

Figura 3.2. Icono aplicación “Pocket”

Además de la aplicación “Pocket”, a continuación explicaremos las herramientas actuales más utilizadas de las que disponemos actualmente para solucionar el problema que estamos tratando.

Una de las más utilizadas son los marcadores del navegador. Esto nos permite almacenar un enlace de manera que podamos acceder rápidamente a él. Esta solución tiene principalmente dos problemas: por un lado, el añadir un enlace a favoritos o a marcadores tiene una intención de uso más prolongada que simplemente un enlace al que queramos acceder dentro de un rato. Las barras de marcadores están ideadas para páginas que visitemos habitualmente, tales como periódicos, redes sociales, correos, etc. Además de esto, estas características no se comparten entre dispositivos, con lo cual, si vemos algo de interés en un dispositivo no podemos recuperarlo fácilmente en otro. Otra de las herramientas, o más bien una de las maneras que tenemos muchas personas de recordar visitar un enlace más adelante, es escribirnos a nosotros mismos un correo, con el consiguiente gasto de tiempo y la poca eficiencia que ello conlleva: abrir la aplicación del correo, autentificarnos si no lo estamos, y escribir un nuevo correo pegando en enlace en el cuerpo y añadiendo nuestro propio correo a los destinatarios. Hay otras maneras de hacer esto, como una que descubrimos gracias a la encuesta, que realizamos sobre la aplicación (punto 6.2 de esta memoria), que es enviarnos a nosotros mismos un mensaje con Telegram, funcionalidad que desconocíamos, ya que en aplicaciones como Whatsapp esto no es posible.

Como conclusiones de este punto podemos deducir que hay opciones que nos pueden servir de apoyo en ciertos momentos a la hora de recuperar enlaces, pero es cierto que son opciones en parte desconocidas o poco eficientes de utilizar.

3.2 Tecnologías y herramientas usadas

En este capítulo detallaremos qué tecnologías y herramientas hemos determinado usar para nuestro proyecto. Una de nuestras mayores motivaciones ha sido el encontrarnos con una gran cantidad de modernas y novedosas tecnologías, que nos sirven para estar concienciados de las tecnologías que ahora mismo tienen mayor proyección en el mercado, algunas de las cuales hemos podido aprender en el desarrollo de la aplicación.

Aprender de cero un sistema completo ha supuesto un reto, con la recompensa de que todas las herramientas usadas eran muy avanzadas tecnológicamente e iban a ser un gran complemento para nuestra formación académica, ya que prácticamente

ninguna de estas se estudia en el grado, pero tienen un valor muy alto en el mundo empresarial.

Aun habiendo una gran diferencia entre algunas de las tecnologías usadas para desarrollar este Trabajo de Fin de Grado, es cierto que la carrera va adaptando nuestro pensamiento a la manera en la que trabajan las máquinas, y esto nos ha sido bastante útil a la hora de aprender estas nuevas tecnologías. Sin embargo, no todas las tecnologías han sido desconocidas para nosotros, ya que hemos utilizado conocimientos de Bases de Datos y Ampliación de Bases de Datos para crear la nuestra propia. También hemos utilizado lenguajes aprendidos en la asignatura Aplicaciones Web, como Javascript y CSS.

Hemos intentado seleccionar el justo número de herramientas, ya que la mayoría son bastantes versátiles y servían para más de un cometido. Casi todas son completamente gratuitas, y las que no, teníamos licencia para estudiantes otorgada por la propia universidad, lo que nos ha facilitado bastante la labor del desarrollo.

Primero detallaremos la tecnología principal, la que más hemos investigado, ya que había muchas posibilidades y todas muy buenas como explicaremos en el siguiente capítulo. Luego, explicaremos las herramientas de apoyo que hemos usado para poder trabajar en equipo paralelamente y poder llevar un control de versiones de la aplicación por si necesitábamos revertir algún cambio o hacer alguna consulta de una versión anterior. También hemos incluido las herramientas usadas para crear este documento. A pesar de ser un documento sencillo, hemos también optado por un control de versiones y un uso paralelo del mismo para agilizar la creación de la memoria.

3.2.1 Tecnología principal: React Native

En el mundo del desarrollo de aplicaciones para móvil, los desarrolladores siempre han buscado intentar dirigir mejor los esfuerzos, ya que ante tanta variedad de dispositivos, programar para cada uno siempre ha sido una tarea demasiado complicada, tanto por la variedad de lenguajes como de sistemas. Si se tiene un equipo lo suficientemente grande es más factible, ya que se pueden dividir recursos, pero ante equipos unipersonales o de pocos miembros es más rentable buscar otro tipo de soluciones para dirigir mejor los recursos.

La redirección de estos recursos es posible gracias a la llegada de distintos frameworks que ayudan a que el mismo código sea funcional para cualquier dispositivo. El problema se ha limitado a la elección de qué framework usar, ya que al haber una gran cantidad que solucionan los problemas de forma efectiva, elegir uno u otro es ya simplemente cuestión de enfoque. En la actualidad, varios comparten el podio entre los más usados y mejor valorados [11]. Nosotros también hemos tenido que pasar por esa decisión al encontrar entre todos los que hay, dos framework que se adaptan muy bien a lo que queríamos: Apache Cordova [12] y React Native [13], ambos multiplataforma, con sus ventajas e inconvenientes. React Native parte de la base de que aprendiéndolo a usar se puede escribir código para cualquier plataforma, a pesar de que ciertos componentes habría que reescribirlos dependiendo de la plataforma en la que estuviéramos, mientras que Apache Cordova instaba a crear una aplicación web que luego serviría en cualquier dispositivo sin casi ningún cambio.

Apache Cordova es muy flexible, porque permite usar cualquier tecnología web que esté al alcance, y al ser un ecosistema muy variado, tenemos casi cualquier posibilidad con él. Sin embargo, esto crea un retardo en la interfaz hace que sea menos

atractivo para nuestra aplicación como explicaremos posteriormente. Estos problemas realmente son solucionables, pero sería gastar recursos en algo que otro framework lo tiene solucionado de base.

React Native por su parte, se escribe con Javascript, más concretamente JSX [14]. Se compila por cada sistema mediante el framework JavaScriptCore [15]. Con él podemos acceder a todos los controles del dispositivo, y mediante su sistema de etiquetas para los elementos de la aplicación, cada sistema lo traduce a componentes que él mismo entiende, por lo que se personaliza dependiendo del dispositivo. La curva de aprendizaje de este lenguaje es mayor, ya que es muy moderno y aunque tiene muchísima documentación, hay que aprender de nuevo muchos conceptos y formas de crear cosas.

A la hora del desarrollo, ambos frameworks ofrecen muchas opciones y facilidades, teniendo entre una de sus grandes diferencias la depuración de código. Apache Cordova, al estar basado en tecnología web, nos permite depurar en un navegador, que luego será adaptado a móvil. Esto tiene la desventaja de que, con algunos cambios, podríamos perder el estado de la aplicación, ya que estamos sustituyendo elementos por completo con cada actualización. Con React Native tenemos una emulación de un dispositivo en el que cuando queramos probar un cambio, no hace falta reiniciar por completo, ya que ofrece una actualización de módulos que hace que no perdamos el estado y podamos seguir depurando con los mismos datos. A su vez, la simulación de la aplicación nos sirve para tener datos precisos ante cualquier problema o fallo, lo cual nos ayuda a poder repararlo de una manera más eficiente, con una referencia directa al código desde el emulador.

La elección del framework a utilizar no ha sido del todo sencilla. Por una parte, Cordova tiene la ventaja de usar un entorno web que iba a ser más fácil de desarrollar y menos laborioso de adaptar a todo sistema, ya que con unos pocos ajustes se puede adaptar para cualquier dispositivo, mientras que por otra parte React Native proporciona unas aplicaciones más fluidas y con mayor usabilidad, con el coste de tener que redefinir ciertas partes de código para adaptarlo a distintas plataformas a pesar de ser multiplataforma.

A pesar de haber muchas más diferencias, una de las claves para elegir uno u otro fue la mayor usabilidad de React Native. Como su propio nombre indica, crea aplicaciones en las que el usuario tiene la sensación de que cada pulsación o tarea que haga fluye como si fuera una aplicación creada en el código nativo del sistema en el que la esté desarrollando. Apache Cordova, al ser realmente una aplicación web traducida a aplicación, se comporta prácticamente como si fuera una web, ya que realmente se ejecuta en el navegador nativo del sistema. Esto conlleva los inconvenientes de los típicos retrasos entre presionar un link y la redirección a la nueva página, que, como hemos dicho, son factibles de arreglar mediante programación avanzada. Pero nuestro enfoque y objetivo hacían que esos recursos que habría que gastar para solucionarlo, podemos dedicarlos a mejorar más aún otros detalles de la aplicación.

Por todos estos detalles hemos decidido elegir React Native. Además, tiene una gran comunidad de desarrolladores a los que les gusta aportar información sobre cualquier asunto que surja con React Native, ya que al estar basado en React, avala mucho su fuerza en las nuevas tecnologías, y hay mucho material y componentes creados en la web que nos ayudarán en el desarrollo. También el hecho de que tuviera más fluidez nos ha gustado, porque nuestro objetivo era tener una aplicación sencilla pero que funcionara muy bien, así que el hecho de tener que trabajar e investigar algo

más por el mejor funcionamiento de la aplicación era algo que nos compensaba frente a la calidad final. Otro detalle era el poder usar las entradas de interfaz al completo del dispositivo, por si en alguna iteración del proyecto incluyamos alguna funcionalidad que fuera necesitar tales datos de entrada. Cordova no permite esto ya que, al ser realmente una página web, tiene más limitado el acceso a datos del dispositivo. Había componentes que lo solucionaban, pero era el mismo problema que con los retardos, era investigar algo que React Native ya lo dejaba solucionado.

Después de probar y manejar algo de React Native, llegamos a la conclusión de que habíamos tomado la decisión acertada. Tanto el lenguaje usado, como las herramientas que nos proporcionaba y la facilidad de depurar la aplicación en un simulador de Android dentro del propio computador nos encantó nada más usarlo. Tiene una simplicidad que ayuda mucho al aprendizaje, y, aunque sean conceptos nuevos, van asentándose fácilmente con la mera práctica. Los primeros desarrollos de prueba eran muy sencillos pero bonitos y funcionales, y era lo que queríamos para nuestra aplicación, ya que buscábamos que el usuario tuviese la sensación de estar usando una aplicación simple, pero muy robusta y funcional.

Figura 3.3. Icono "React Native"

Como esta herramienta es la herramienta fundamental en la que está basada la aplicación, se le dedica un capítulo especial (Capítulo 4).

En los siguientes capítulos detallaremos el desarrollo de aplicación y explicaremos los componentes que hemos usado para crear la aplicación, que son únicamente una pequeña muestra de unas de las muchísimas funcionalidades que nos ofrece este potente framework.

3.2.2 Herramienta principal: Visual Studio Code

Para facilitarnos el desarrollo general del código se ha utilizado un editor de código ligero (Visual Studio Code [16]). Esta herramienta cuenta como navegación con un sencillo árbol de directorios y con un reconocedor de sintaxis de una gran cantidad de lenguajes de programación. Una vez modificado el código, se compila por consola.

Elegimos esta aplicación por la confianza que ofrece Microsoft en todos sus productos, además de que es una herramienta con la que estábamos familiarizados al haberla utilizado en nuestros respectivos trabajos fuera de la universidad. Destacamos ante todo su rapidez y sencillez a la hora de desarrollar, ya que gracias a la simplicidad de la aplicación no necesitábamos un entorno de desarrollo complejo.

Figura 3.4. Icono Visual Studio Code

3.2.3 Android Studio

Como herramienta secundaria para el desarrollo de esta aplicación hemos utilizado Android Studio [17], que es el entorno de desarrollo oficial para las aplicaciones en Android. En el 2013, reemplazo a Eclipse [18] para servir como entorno para los programadores de aplicaciones de Android. Con esta herramienta podemos tener el directorio de carpetas del proyecto y todos los archivos perfectamente ubicados para poder modificar cualquiera que necesitemos, además de ofrecernos funciones de autocompletar y ayuda para el programador. Así mismo, también incluye un sistema muy cómodo para poder emular directamente la aplicación en una máquina virtual de Android, que podrá recargarse automáticamente mientras depuramos la aplicación, teniendo unas funcionalidades muy potentes para poder probar la aplicación y cambiar código a nuestro antojo durante la depuración.

Android Studio es gratuito y cómodo de instalar, y cualquier actualización o paquete nuevo que queramos instalar está disponible para descargar automáticamente y también gratuitamente. Consideramos que eran los mejores entornos que podíamos tener para el desarrollo, y como la primera iteración del proyecto estará basada en Android, nos pareció la mejor opción para empezar a trabajar con calidad cuanto antes.

Figura 3.5. Icono "Android Studio"

3.2.4 GitHub y GitHub Desktop

El control de versiones lo hemos realizado con el sistema de GitHub [19]. Es una forja que nos permite colaborar para crear el código de la aplicación paralelamente. Sus repositorios son gratuitos y públicos, aunque con una cuenta de pago se pueden crear repositorios privados. Hemos elegido esta opción porque la hemos visto en el pasado y funciona realmente bien.

Gracias a la aplicación "GitHub for Desktop 0.8.0.", podremos tener a unos pocos pasos el repositorio actualizado y el historial de cambios sin necesidad de acceder al explorador. Es realmente cómodo y facilita mucho la actualización del proyecto. Combina una potente interfaz gráfica en la que podremos aplicar nuestros cambios y

tener un control de las distintas versiones de nuestra aplicación. Con ella, podemos trabajar en paralelo en el mismo proyecto en cualquier dispositivo que instalemos esta herramienta, y también poder visualizar los cambios en la página web de GitHub.

Esta aplicación es relativamente sencilla de manejar. Simplemente hay que instalarla, registrarse o introducir las credenciales personales de GitHub y clonar el repositorio. Una vez que tenemos el repositorio descargado en nuestro equipo, ya podemos abrir el proyecto con el Android Studio para trabajar con él y desarrollar la aplicación entre los integrantes del equipo.

El resto de la aplicación es muy intuitivo. Tenemos un historial de cambios en el que podremos ver las diferentes versiones de la aplicación y revertir cambios en caso de ser necesario. También se puede dividir el proyecto en distintas ramas de desarrollo, aunque en nuestro proyecto no hará falta por el momento, ya que al trabajar únicamente en una iteración a la vez, hacía que cerraremos cada una al finalizar y empezaremos la nueva en la misma rama.

Figura 3.6. Icono "Github"

3.2.5 IBM RSA

IBM Rational Software Architect [20] (Rational Software, 2006) es una herramienta CASE en la que se crean los diagramas UML de la aplicación, que posteriormente serán convertidos en código. Está optimizado para C++, Java y Servicios Web. Está basado en el framework de Eclipse.

Esta herramienta fue usada y proporcionada con una licencia de estudiantes en la asignatura de "Modelado de Software". Fue de las pocas herramientas que ya habíamos tenido un acercamiento previo. Los modelos desarrollados se ajustan a la arquitectura multicapa (Alur et al., 2003; Fowler, 2002).

La hemos usado para crear los modelos de la aplicación y poder tener antes de la creación de código, una arquitectura bien definida para no perder recursos en cambios o riesgos ocasionados por una mala especificación de requisitos.

Figura 3.7. Icono "IBM RSA"

3.2.6 Google Drive

Google Drive [21] es un sistema de alojamiento de ficheros creado por Google el 24 de abril de 2012. Anteriormente, era conocido por Google Docs, pero se amplió su funcionalidad hasta ser la nube de archivos que conocemos ahora. Puede ser usado online, y también tiene un cliente para Windows para poder usar una carpeta del equipo como carpeta en la nube.

Para nosotros, Google Drive ha tenido una importancia muy grande, ya que todo este texto ha sido escrito paralelamente por ambos autores gracias a la funcionalidad de Google Drive para editar documentos en la red. Podíamos tener acceso a este documento de forma privada desde cualquier dispositivo, y eso ha hecho que fuera más fácil de crear. También, aprovechando que podíamos tener cualquier tipo de fichero en la nube, hemos compartido imágenes y material necesario para el desarrollo del proyecto.

Google Drive también nos ha ayudado a tener una vía que sirviera para mostrar en las sesiones con la tutora el contenido actualizado hasta la fecha en ese momento y poder tener una referencia sus indicaciones en tiempo real. Su descarga es gratuita en la misma página de Google. Únicamente hacía falta sincronizar las carpetas entre los participantes mediante invitaciones por email.

Figura 3.8. Icono "Google Drive"

3.2.7 Microsoft Word

Microsoft Word [22] es un software procesador de textos de Microsoft de la suite de ofimática de Microsoft Office. Es de los más importantes y el que más opciones tiene para Windows.

A pesar de crear la mayoría del contenido de la memoria con Google Drive por motivos de trabajar en paralelo, necesitábamos este procesador para el resultado final, ya que ofrecía mejores opciones para dejar estéticamente mejor el documento, y así poder tener una memoria más atractiva para el lector.

Hemos usado la versión para estudiantes de este software, cuya licencia ha sido ofrecida por la universidad.

Figura 3.9. Icono "Microsoft Word"

3.2.8 Skype

Skype [23], de Microsoft, es una herramienta que permite mensajería instantánea y video llamadas entre cualquier dispositivo a través de Internet. También permite llamadas telefónicas mediante un sistema de pago (VoIP).

Para nosotros, la mayor ventaja de usar Skype, era poder comunicarnos mediante mensajería instantánea y llamadas a la vez que estábamos desarrollando el proyecto. Si aparecía cualquier duda, únicamente una llamada bastaba para desarrollar a la distancia el tema a hablar y ahorrar tiempo, ya que esperar a la siguiente reunión o hacerlo mediante la mensajería podía habernos hecho perder valioso tiempo clave.

Esta herramienta viene instalada de base en la mayoría de equipos con Windows, y si no, se puede descargar gratuitamente en su página web.

Figura 3.10. Icono "Skype"

3.2.9 Dropbox

Dropbox [24] es una nube de archivos multiplataforma, que pertenece a la compañía con el mismo nombre. Se puede acceder tanto por la web como por cliente para Windows. Sus ventajas es la rapidez de sincronización y su simple interfaz. También tiene un gestor de cambios para llevar el control de versiones.

Nosotros ya estábamos usando Google Drive como nube para nuestros archivos, ya que al tener la opción de poder modificar los documentos en línea paralelamente, nos era más útil que ir modificando el documento cada uno en momentos distintos.

Sin embargo, necesitábamos algún medio para tener copias de seguridad en la nube de nuestro proyecto. Podíamos haber hecho estas copias de seguridad en nuestros equipos, o en algún disco físico, pero no hubiéramos podido acceder a ellos sin tenerlos delante, así que la mejor opción que teníamos era tener la copia de seguridad en otra nube.

De esta forma, programamos al final de cada sesión de trabajo, hacer una copia del documento en esta nueva nube, y de los archivos del proyecto por si en algún momento, surgieran problemas con la nube principal o cualquier problema técnico.

Figura 3.11. Icono "Dropbox"

3.2.10 Lucidchart

Lucidchart [25] es una herramienta gratuita y online para la creación de diagramas UML. No es necesario descargarse ningún programa, y vale con enlazar la cuenta de google para el registro. La hemos utilizado como alternativa al IBM RSA para poder crear tanto el modelo de dominio como los diagramas de flujo de cada caso de uso.

Figura 3.12. Icono "Lucidchart"

4. React Native

En esta sección de la memoria se tratará la información necesaria para la instalación, el control de versiones, la creación y ejecución de un proyecto en el framework React Native y algunas pinceladas para entender mejor el desarrollo de la aplicación y sus componentes clave. Toda la información contenida en este punto está ampliada en el Anexo 1 – React Native.

La mayoría de la información necesaria para el aprendizaje de la utilización de React Native, el framework utilizado, ha sido obtenida mediante cursos online, en mayor medida de Youtube, de varios autores tales como los ofrecidos por Mario Díez en su canal de Youtube [26].

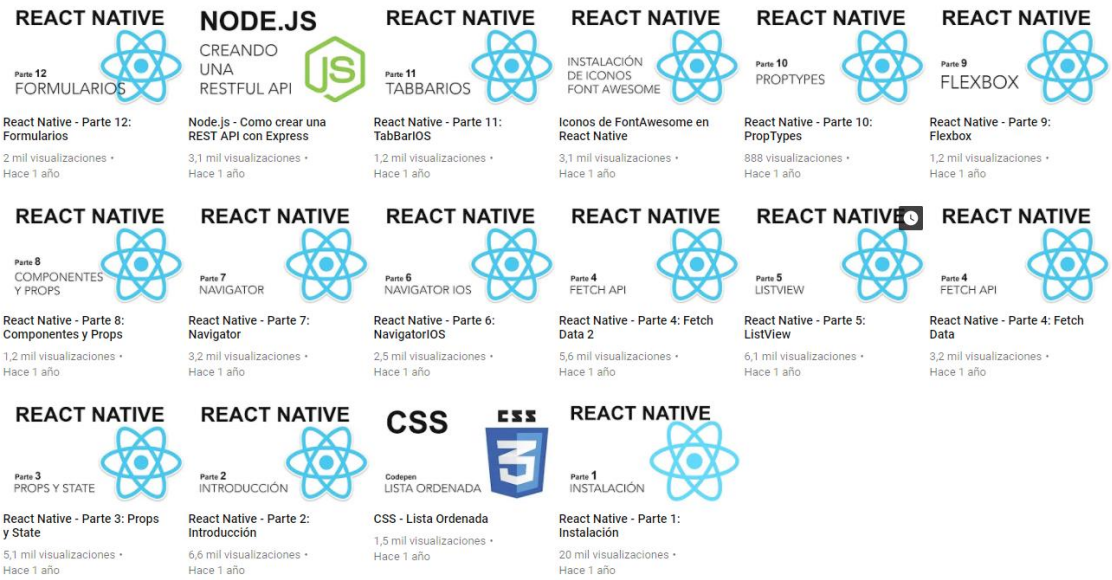


Figura 4.1. Ejemplo de videotutoriales de los que hemos aprendido
Fuente: Canal de Mario Díez en Youtube

4.1 Instalación de la aplicación

Para poder empezar a desarrollar código, necesitamos instalar algunas dependencias del framework Chocolatey [27], al cual se puede acceder mediante la ejecución de comandos en la consola de administrador. Junto a él, instalamos las siguientes dependencias:

- Node 4
- Python2
- JDK 8

Para terminar, necesitamos instalar React Native Command Line Interface (CLI), también desde la consola de administrador. Opcionalmente podemos instalar Yarn, que nos permitirá organizar e instalar mejor los paquetes y componentes que descarguemos.

Por último, necesitaremos el entorno de desarrollo para Android, Android Studio. Se debe instalar y, una vez instalado, ejecutarlo y elegir un Custom Setup y comprobar que se selecciona lo siguiente siempre:

- Android SDK
- Android SDK Platform
- Performance (Intel ® HAXM)
- Android Virtual Device

Una vez instalado debemos, en la configuracion del Launcher, entrar en SDK Manager (o, en el caso de no verla, al abrir el programa doble shift y buscar “SDK Manager”) e instalar Android 6.0 (Marshmallow), eligiendo con “Show Package Details” las siguientes opciones:

- Google APIs
- Android SDK Platform 23
- Intel x86 Atom_64 System Image
- Google APIs Intel x86 Atom_64 System Image

También debemos configurar el Android SDK Build-Tools para seleccionar la versión 23.0.1. También seleccionaremos el Android Emulator.

Para terminar con la configuración de Android Studio, debemos agregar la variable de entorno siguiente:

Nombre: `ANDROID_HOME`
Valor: `c:\Users\YOUR USERNAME\AppData\Local\Android\Sdk`

Con esto último, ya tendríamos configurado todo para empezar a crear un proyecto.

4.2 Control de versiones con GitHub

El control de versiones lo haremos con el sistema de GitHub, gracias a la aplicación GitHub for Desktop [28]. Esta aplicación combina una potente interfaz gráfica en la que podremos aplicar nuestros cambios y tener un control de las distintas versiones de nuestra aplicación. Con ella, podemos trabajar en paralelo en el mismo proyecto en cualquier dispositivo que instalemos esta herramienta, y también poder visualizar los cambios en la página web de GitHub.

Únicamente hay que instalarla, registrarse o introducir las credenciales personales de GitHub y clonar el repositorio. La dirección del repositorio de nuestro proyecto es:

```
ingeniero92/memoriesCloud
```

Una vez que tenemos el repositorio descargado en nuestro equipo, ya podemos abrir el proyecto con el Android Studio para trabajar con él y desarrollar la aplicación entre los integrantes del equipo. Es obligatorio hacer una actualización de los cambios antes de trabajar para evitar los conflictos de archivos en la mayoría de lo posible.

Para descargar/actualizar los cambios, hay que presionar en el botón “Fetch origin”. Esto descargará los nuevos cambios y, en caso de haber también cambios en nuestro equipo, nos aparecerán en la lista los archivos disponibles para subir al repositorio. Pondremos el título de la subida en inglés, y en la descripción un breve texto explicando el cambio. Hay que presionar el botón de “Commit to master” para terminar con la subida de archivos. Si hubiera algún conflicto, GitHub for Desktop muestra un mensaje de error, indicando que debes resolver los conflictos después de hacer la subida de nuestros cambios.

Nuestros cambios estarán en HEAD, y los que ya estaban en el servidor abajo. Debemos borrar las líneas sobrantes en un editor externo y quedarnos con el cambio que más nos interese. Una vez eso, hacemos la subida de estos arreglos y ya estará solucionado el error. Esto suele pasar cuando se ha hecho una subida de unos archivos en un equipo, y a la vez en los mismos en otro. Se puede evitar intentando no tocar los mismos archivos si se trabaja en el mismo momento. Para ello la comunicación en el equipo es lo más importante.

El resto de opciones de la aplicación es muy intuitivo. Tenemos un historial de cambios en el que podremos ver las diferentes versiones de la aplicación y revertir cambios en caso de ser necesario. También se puede dividir el proyecto en distintas ramas de desarrollo. Hemos creado una rama una vez hemos terminado el desarrollo base, para hacer una rama para la versión BETA y otra para la versión 1.0.

4.3 Creación y ejecución de un proyecto en React Native

Para crear un proyecto nuevo de cero, hay que ejecutar en consola la siguiente orden:

```
react-native init NOMBRE PROYECTO
```

Se creará en la ruta en la que estemos situados en la consola. Si ya está creado/bajado de GitHub, únicamente hay que importar el proyecto desde archivos ya existentes seleccionando la carpeta raíz del proyecto.

Hay que aceptar dentro del Android Studio la solicitud de la ventana que nos salga de “Framework Android detected”, para poder acceder a todas las opciones de Android Studio.

4.3.1 Ejecutar aplicación con emulador

Posteriormente, tenemos que configurar el emulador con el AVD Manager. Está en la barra de herramientas de arriba (Tools->Android->AVD Manager). Ahí dentro podemos crear una nueva máquina virtual. Hay que instalar HAXM si no está instalado, y seleccionar como imagen de sistema la versión Marshmallow 23 de Google API’s de x86_64. Si no está instalada, la descargamos. Luego solo queda configurar el emulador para que acepte entrada de teclado, aunque suele estar activado. Esto nos servirá para poder recargar la página.

El emulador se puede ejecutar manualmente mediante un comando de consola, o por el contrario desde la aplicación de Android Studio. En Tools->Android->AVD Manager y pulsando en “Ejecutar”. Para vincular la aplicación con el emulador,

ejecutamos este código en la consola de la aplicación (parte inferior) situándonos en la carpeta del proyecto:

```
react-native run-android
```

Y ya podemos programar y depurar. Para abrir el menú desarrollador, usamos “Control + M”.

4.3.2 Ejecutar aplicación en dispositivo físico Android

Para depurar la aplicación en un dispositivo físico Android, debemos conectarlo vía USB y, en el teléfono, activar la depuración por USB en las opciones de desarrollador. Si al conectarlo nos pregunta por algún método de conexión, elegiremos MTP, que es transferencia de archivos. Para abrir el menú desarrollador en el dispositivo, tenemos que agitar el teléfono y ya instalará la aplicación y la ejecutará. Para poder refrescar los cambios que hagamos en el ordenador hay que abrir el menú desarrollador en el teléfono.

4.4 Sintaxis React Native

Una de las características que más nos ha llamado la atención del framework React Native es que seguía un flujo y una sintaxis que nos resultaban bastante extrañas comparadas con las que hemos estado tratando durante el grado. Por ello, hemos creído conveniente reunir toda esta información que hemos ido recabando, y de la cual aún no hay una documentación clara ni extensa en la red, en un anexo adjunto a la entrega de esta memoria. En este anexo se tratará de manera más detallada todo lo concerniente al framework utilizado.

4.5 Firebase

Para gestionar los usuarios y los recuerdos en nube de nuestra aplicación, vamos a usar Firebase [29], una plataforma que nos permitirá crear la autentificación de usuarios en la aplicación y poder guardar sus datos en una base de datos NoSQL, que presenta diversas diferencias respecto al estándar [30].

Tiene una potente gestión de usuarios, que permite crearlos a partir de redes sociales o mediante un email y contraseña. También proporciona verificación de email y posibilidad de recuperar la contraseña.

La ventaja de usar Firebase es que, aparte de la autentificación de usuarios, que era la motivación principal para usar este servicio, es que podíamos, en la misma plataforma, tener una base de datos para guardar los recuerdos de los usuarios y distintas variables generales de la aplicación, tales como la versión actual, haciendo que podamos elegir tener una versión mínima de la aplicación por si hay que forzar al usuario a una actualización. En un principio se iba a usar MongoDB como base de datos, pero al haber elegido Firebase para la autentificación de usuarios era posible usar una única plataforma para todo, además de que era más difícil para nosotros por logística tener un servidor propio dedicado para MongoDB y las peticiones de la aplicación.

Todo se puede administrar desde cualquier parte desde la web de Firebase. Unificando el sistema de autentificación, base de datos y una administración remota,

hizo que nos decantáramos por este servicio externo, que, en un futuro, podría ser implementado por nosotros, aunque para la primera etapa de la aplicación, nos estaría proporcionando herramientas muy potentes.

Con una cuenta de google, se pueden acceder a todas las funciones de forma gratuita. Únicamente hay que importar en el proyecto las dependencias de Firebase, e incluir en un fichero los datos de configuración de nuestra cuenta de Firebase. Si la aplicación crece, siempre se puede migrar a un plan de pago que permita sin problemas todo el flujo de datos o, como previamente se ha comentado, configurar un servidor y tener nuestra propia API.

4.6 Redux

La aplicación, tiene una serie de distintas pantallas que están usando componentes que hemos creado de React Native. Estos se enlazan para navegar entre ellos mediante un componente externo (React Navigation). Sin embargo, el estado de la aplicación y las variables compartidas necesitan de un componente externo que lo maneje.

Este es Redux [31], que es a grandes rasgos un contenedor del estado, es decir, de los datos de la aplicación. Junto a React Navigation, son los componentes clave para toda la aplicación. Cuando crece el código, se vuelve más complicado añadir funciones y tratar con datos. Redux nos ayudara con eso, ya que carga en el archivo inicial de la aplicación, donde cargamos todas las dependencias, el navegador y cualquier elemento que debe inicializarse antes de llamar a la pantalla de carga. Está directamente enlazado con el navegador de la aplicación [32], y es capaz de mantener las variables que, mediante llamadas, ha almacenado en él para un uso posterior desde cualquier componente que incluya a Redux [33].

El funcionamiento se puede ver en la siguiente figura:

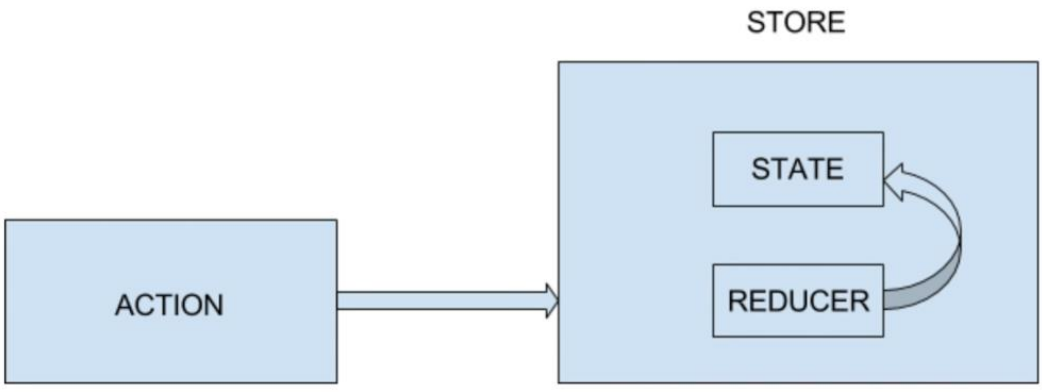


Figura 4.2. Diagrama de Redux
Fuente: Canal Youtube de Mario Díez

Como se puede ver, Action es el encargado de enviar una solicitud a Reducer. Entonces Reducer recibe una Action y, dependiendo del tipo, obtiene o modifica el Estado. Reducer y State conforman la Store, que encapsula a estos dos para funcionar de contenedor de estado.

Para usar Redux con la aplicación en React Native, hay que englobar el componente principal de la aplicación en un Provider, que es un componente que nos proporcionará la Store que contendrá los datos que depositemos ahí con nuestras Action y Reducer. Para combinar Redux con alguno de nuestros componentes deberemos usar Connect, que hará que ese componente pueda acceder a la Store. Este Connect tiene dos parámetros, el primero es un método que conecte los Reducer que necesitemos, y el segundo, los Action que usaremos.

Una vez conectado, únicamente deberemos acceder a los Prop del componente, para poder obtener los datos de la Store o ejecutar las Action para modificarlo.

5. Desarrollo de la aplicación

En el siguiente capítulo, daremos detalles adicionales de todo el proceso de desarrollo de la aplicación, el aprendizaje de las tecnologías, el análisis de requisitos y la funcionalidad necesaria en el sistema.

Gracias a los conocimientos adquiridos en el Grado de Ingeniería de Software, hemos sido capaces de poder aprender un lenguaje nuevo, y poder realizar un proyecto nuevo en un entorno que era totalmente desconocido por nosotros. El auto-aprendizaje y la base de programación que se adquiere durante el grado ha sido nuestro mayor aliado durante este Trabajo de Fin de Grado.

Como ya se ha descrito en la introducción, nuestra aplicación busca ser utilizada en los casos en los que el usuario requiera guardar un enlace o una información para ser utilizada posteriormente, desde el mismo o desde otro dispositivo.

La aplicación va a ser utilizada principalmente en dispositivos móviles. Para el desarrollo de esta parte de la aplicación hemos utilizado como entorno de desarrollo Visual Studio Code y Android Studio, con el que podemos crearnos una máquina virtual en la que ejecutar nuestra aplicación, la cual es un simulador que nos crea una ventana que recrea perfectamente la apariencia y el comportamiento de un dispositivo móvil, lo que nos facilita ver cómo será su funcionamiento y apariencia sin tener la necesidad de ejecutar la aplicación en un dispositivo móvil real.

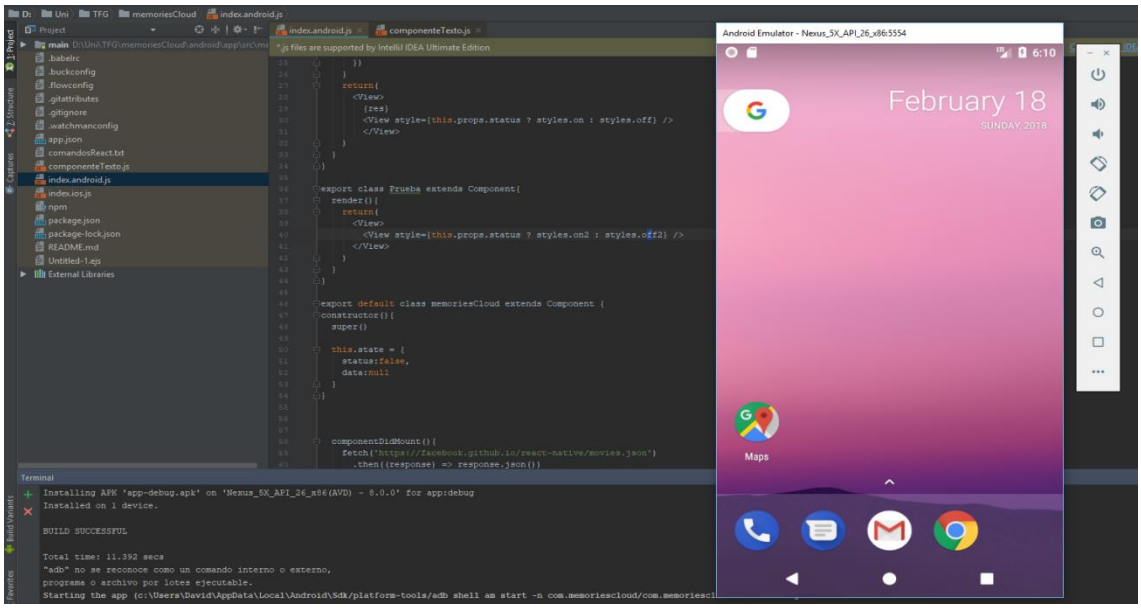


Figura 5.1. Ejemplo de ejecución del emulador de dispositivo Android
Fuente: Elaboración propia

Para realizar el análisis de requisitos se han tenido en cuenta los escenarios de uso de la aplicación. Hemos analizado los usos que queríamos que la aplicación tuviera para resolver los problemas explicados anteriormente de la memoria a corto y largo plazo.

Durante todo el desarrollo de la aplicación se han producido leves modificaciones que nos han hecho variar algunos diagramas y casos de uso. También hemos corregido ciertas funcionalidades o implementado nuevas a partir de los comentarios y sugerencias de los usuarios, que al final son los que finalmente utilizaran la aplicación en su día a día como herramienta para ayudar a guardar importantes recuerdos que, de otra forma, podrían olvidar o no recuperar en el momento justo que los necesitaran.

La aplicación está disponible en el Github del Trabajo de Fin de Grado, desde el cual se puede visualizar y descargar todo el código para su inspección, en la siguiente URL de repositorio de Github:

```
ingeniero92/memoriesCloud
```

A continuación, describiremos los requisitos funcionales de la aplicación, mediante casos de uso y diagramas de flujo siguientes:

- Validación de usuarios (Tabla 4.1 y Figura 4.3)
- Contraseña olvidada (Tabla 4.2 y Figura 4.4)
- Añadir recuerdo (Tabla 4.3 y Figura 4.5)
- Modificar recuerdo (Tabla 4.4 y Figura 4.6)
- Eliminar recuerdo (Tabla 4.5 y Figura 4.7)
- Compartir recuerdo (Tabla 4.6 y Figura 4.8)

Figura 5.2. Logo de MemoriesCloud
Fuente: Elaboración propia

Tabla 5.1. Caso de uso de validación de usuarios

Función:	Validación de usuarios
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se hace el registro o la autentificación del usuario
Entrada:	Correo y contraseña del usuario
Salida:	Pila de recuerdos del usuario
Origen:	Pantalla principal tras entrar a la aplicación
Destino:	Sistema o base de datos
Necesita:	Conexión a internet
Precondición:	Aplicación instalada
Postcondición:	Se puede añadir un usuario a la base de datos

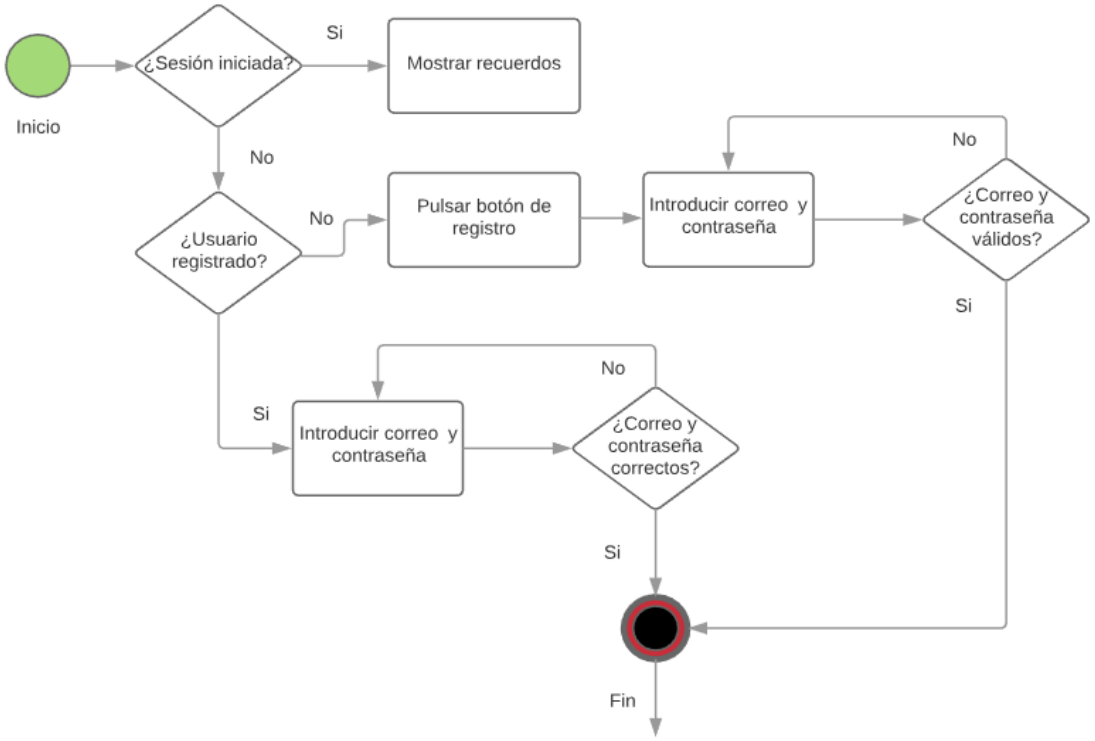


Figura 5.3. Diagrama de flujo de la validación de usuarios
Fuente: Elaboración propia

Tabla 5.2. Caso de uso de contraseña olvidada

Función:	Contraseña olvidada
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se recupera la contraseña olvidada del usuario
Entrada:	Correo del usuario
Salida:	Nueva contraseña del usuario
Origen:	Pantalla de autentificación
Destino:	Base de datos
Necesita:	Conexión a internet
Precondición:	Aplicación instalada
Postcondición:	Se modifica la contraseña

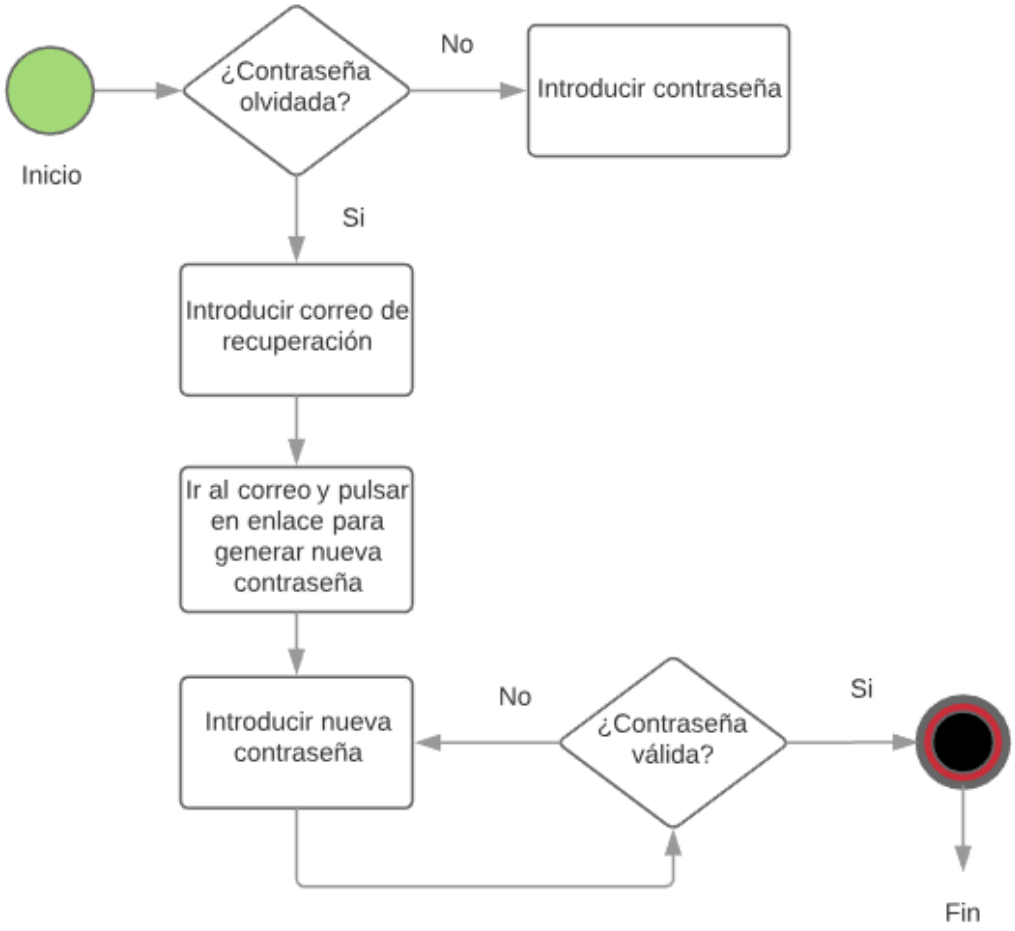


Figura 5.4. Diagrama de flujo de contraseña olvidada
Fuente: Elaboración propia

Tabla 5.3. Caso de uso de añadir recuerdo

Función:	Añadir recuerdo
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se añade un recuerdo a la pila de recuerdos del usuario
Entrada:	Título y descripción del recuerdo
Salida:	Nuevo recuerdo
Origen:	Pantalla de añadir recuerdo
Destino:	Base de datos
Necesita:	Conexión a internet
Precondición:	Aplicación instalada y usuario autentificado
Postcondición:	Se añade un recuerdo a la base de datos

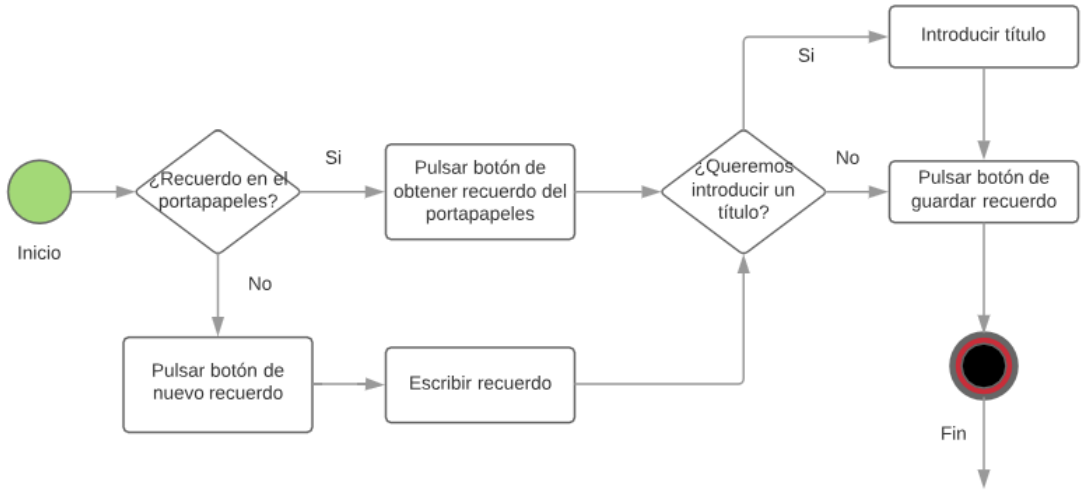


Figura 5.5. Diagrama de flujo de añadir recuerdo
Fuente: Elaboración propia

Tabla 5.4. Caso de uso de modificar recuerdo

Función:	Modificar recuerdo
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se modifica el título o el recuerdo
Entrada:	Recuerdo original
Salida:	Recuerdo modificado
Origen:	Pantalla principal de la aplicación
Destino:	Base de datos
Necesita:	Conexión a internet
Precondición:	Aplicación instalada, usuario autentificado y recuerdo existente
Postcondición:	Se modifica el recuerdo

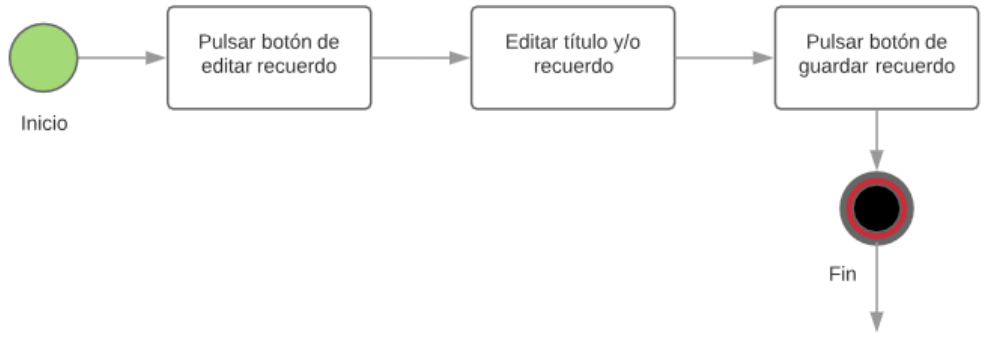


Figura 5.6. Diagrama de flujo de modificar recuerdo
Fuente: Elaboración propia

Tabla 5.5. Caso de uso de eliminar recuerdo

Función:	Eliminar recuerdo
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se elimina un recuerdo de la pila de recuerdos del usuario
Entrada:	Recuerdo a eliminar
Salida:	-
Origen:	Pantalla principal de la aplicación
Destino:	Base de datos
Necesita:	Conexión a internet
Precondición:	Aplicación instalada, usuario autentificado y recuerdo existente
Postcondición:	Se elimina el recuerdo

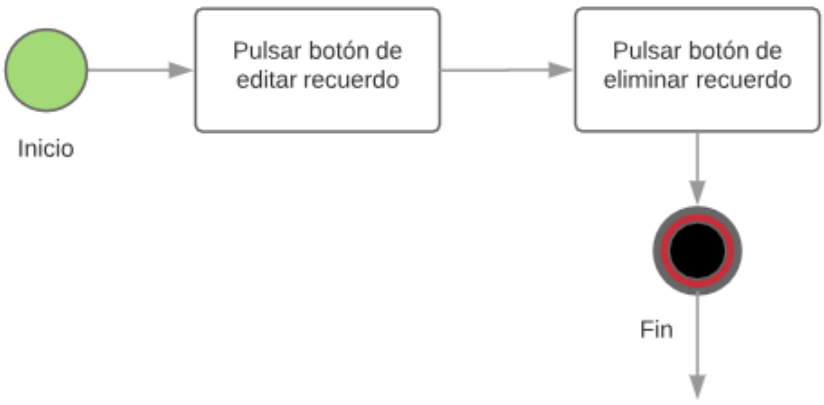


Figura 5.7. Diagrama de flujo de eliminar recuerdo
Fuente: Elaboración propia

Tabla 5.6. Caso de uso de compartir recuerdo

Función:	Compartir recuerdo
Prioridad:	Alta
Estabilidad:	Alta
Descripción:	Se comparte un recuerdo mediante contactos o redes sociales
Entrada:	Recuerdo a compartir
Salida:	-
Origen:	Pantalla principal de la aplicación
Destino:	Contactos o redes sociales
Necesita:	Conexión a internet
Precondición:	Aplicación instalada, usuario autentificado y recuerdo existente
Postcondición:	Se comparte el recuerdo

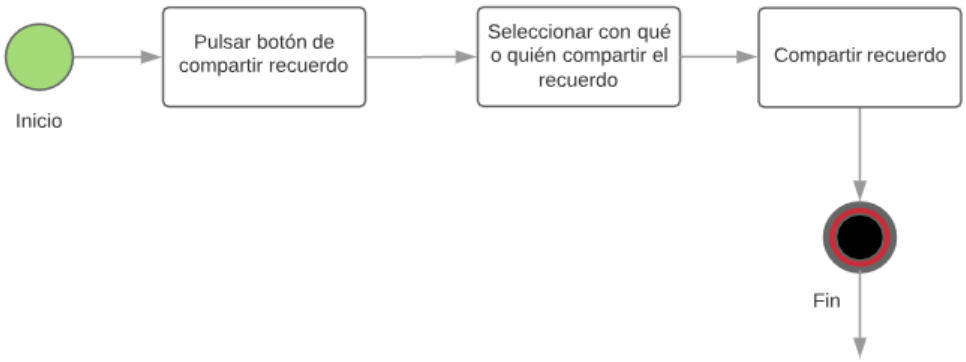


Figura 5.8. Diagrama de flujo de compartir recuerdo
Fuente: Elaboración propia

Por último, mostramos el diagrama de clases simplificado de la aplicación, en el que podemos hacernos una idea de cómo interactúan las clases en la aplicación.

La aplicación inicia en el Index, que, dependiendo del flujo, navegará a la pantalla de compartir recuerdo, o a la pantalla principal de la aplicación. Esta última, cargara todos los componentes necesarios para que la aplicación pueda funcionar. Entre ellos, Redux para almacenar el estado y React Navigation para las transiciones entre pantallas. En Routes, tenemos todas las pantallas de las distintas funcionalidades para poder acceder a ellas. En estas pantallas, si se necesita obtener o modificar algún tipo de información, tendrán que ejecutar una Action, la cual, gracias a Firebase, nuestra base de datos y gestión de control de usuarios, realizara la tarea pedida. La pantalla principal, App, cargara los tres elementos básicos, el Header, el Menu Slide del lateral y la lista de recuerdos con sus botones. Todas las pantallas se conectan con Redux, que mantendrán los datos en la Store con los Reducers.

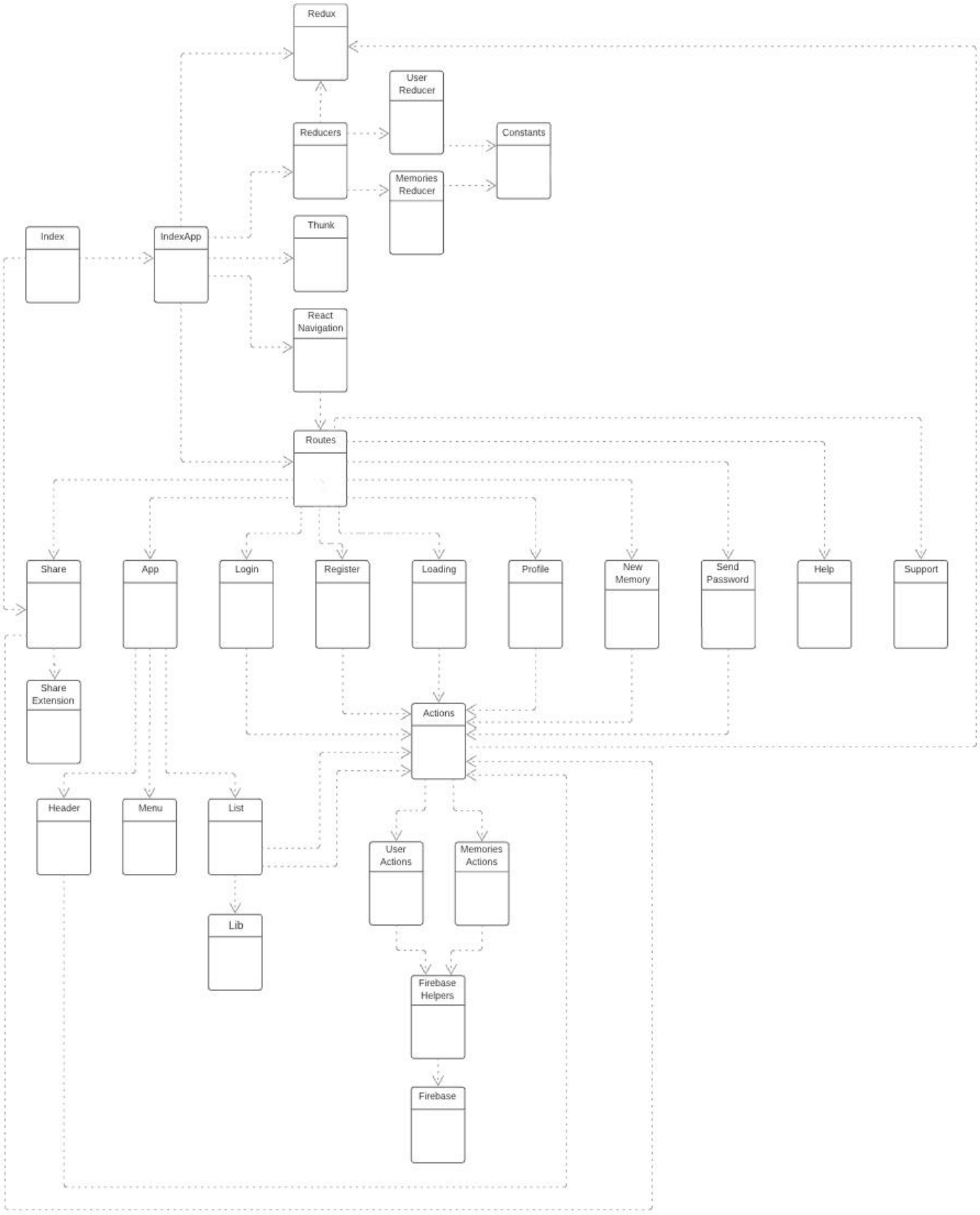


Figura 5.9. Diagrama de clases
Fuente: Elaboración propia

6. Resultados

En este apartado vamos a mostrar los resultados de la aplicación una vez finalizada, así como también, los resultados de la encuesta realizada sobre la aplicación en uso en una primera versión.

6.1 La aplicación MemoriesCloud

Nuestra aplicación ha tenido un tiempo de desarrollo de unos 8 meses, desde el aprendizaje del framework que hemos utilizado hasta el lanzamiento y la subida a la Play Store de MemoriesCloud [34].

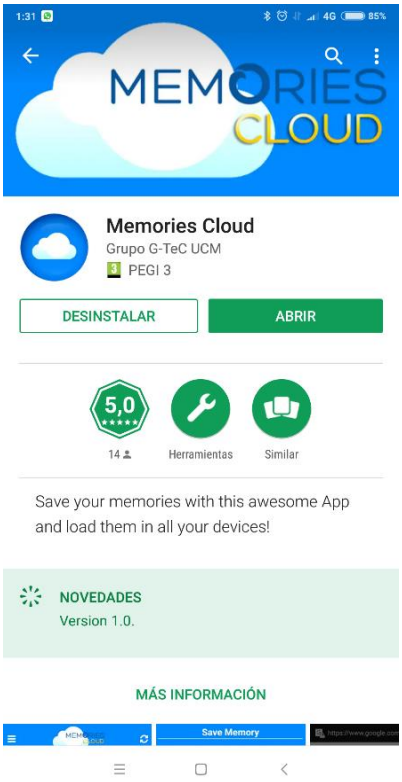


Figura 6.1. Aplicación en la Play Store
Fuente: Elaboración propia

Esta aplicación comienza con una pantalla de carga, en la que se cargan los componentes principales de la aplicación y se comprueba si el usuario tenía abierta la aplicación y estaba autentificado, en cuyo caso apareceré directamente la que será la pantalla principal de la aplicación, la pila de recuerdos, o la pantalla de autentificación/registro, en caso contrario. Estos casos se contemplan en la siguiente figura (Figura 6.2).

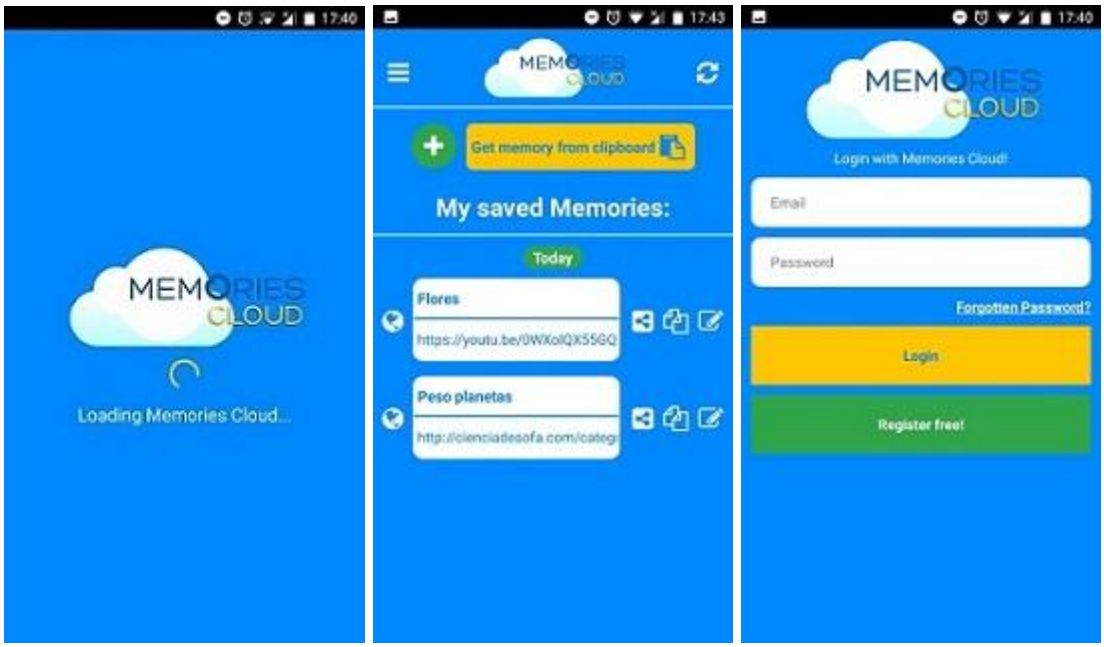


Figura 6.2. Entrada a la aplicación
Fuente: Elaboración propia

En el caso en el que el usuario no esté autentificado, pasaremos por la opción de registrarse, donde introduciremos nuestro correo y nuestra contraseña por duplicado, para comprobar que la hayamos introducido correctamente (Figura 6.3).

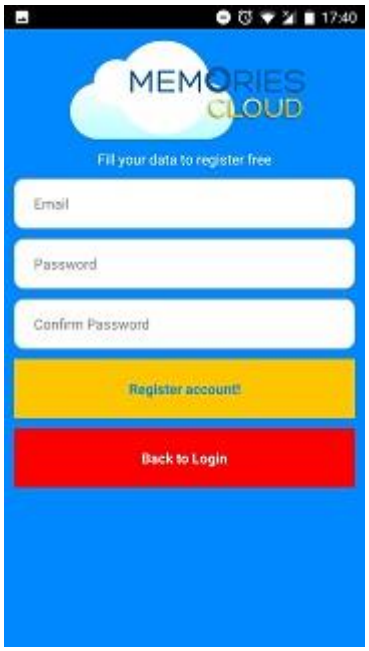


Figura 6.3. Registro
Fuente: Elaboración propia

En la pantalla de autentificación también tenemos la opción de contraseña olvidada. En caso de que estemos registrados, pero no recordemos nuestra contraseña, podemos usar esta opción, donde nos pedirá que introduzcamos nuestro email para mandarnos un correo de recuperación de contraseña. Tras recibir este email,

deberemos entrar en el enlace del correo y escribir nuestra nueva contraseña (Figura 6.4)

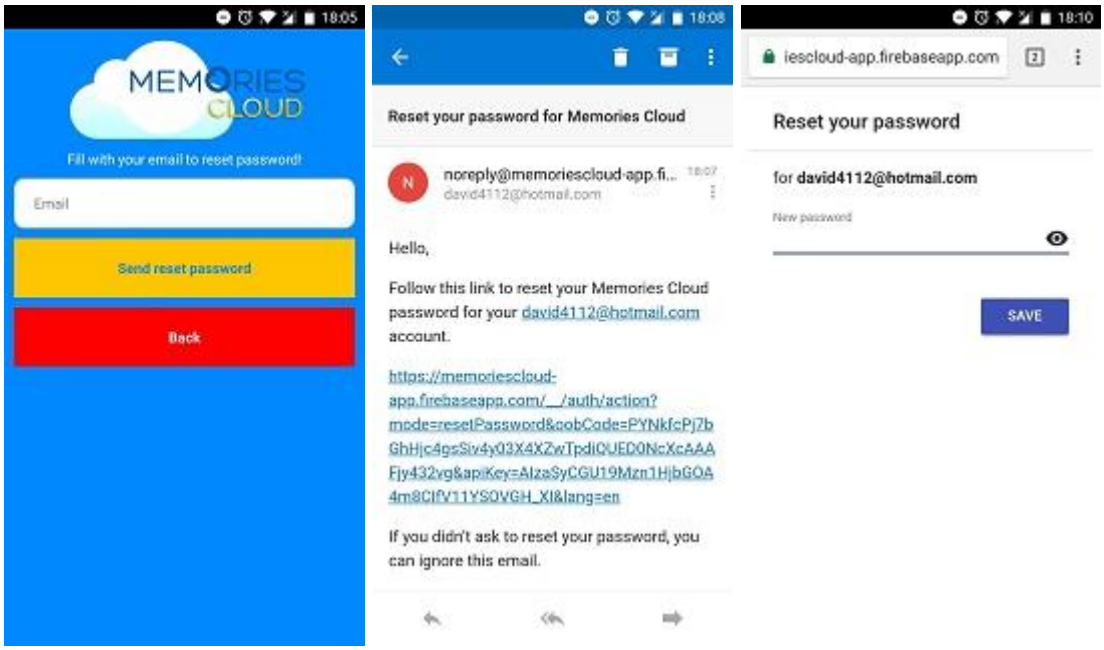


Figura 6.4. Recuperación de contraseña
Fuente: Elaboración propia

Desde la pila de recuerdos tenemos a la izquierda una imagen de un globo terráqueo, que es la opción de acceder al enlace del recuerdo, y a la derecha las opciones de compartir, copiar en el portapapeles y editar recuerdo (Figura 6.5).

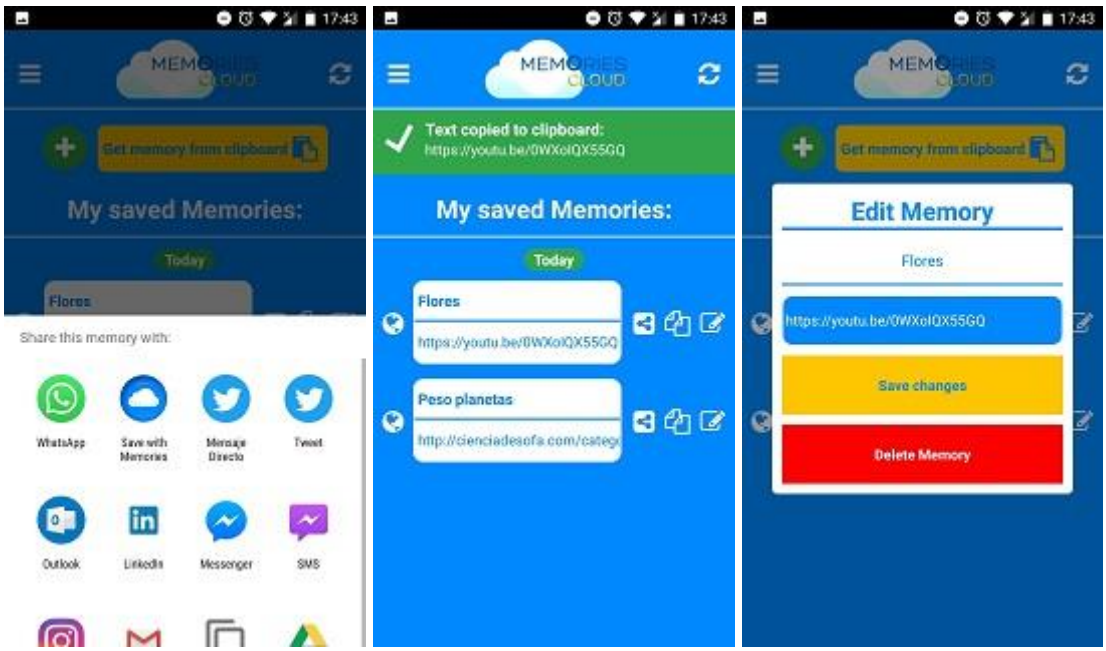


Figura 6.5. Opciones de recuerdo
Fuente: Elaboración propia

Por otro lado, tenemos un menú desplegable arriba a la izquierda (Figura 6.6) que consta de las opciones de “User settings”, “Help” y “Support” (Figura 6.7) donde el

usuario podrá cambiar su nombre, ver un pequeño tutorial y ponerse en contacto con el equipo de soporte, respectivamente.

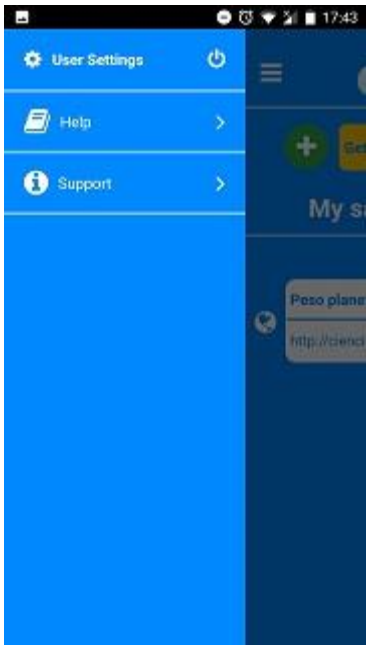


Figura 6.6 Menú desplegable
Fuente: Elaboración propia

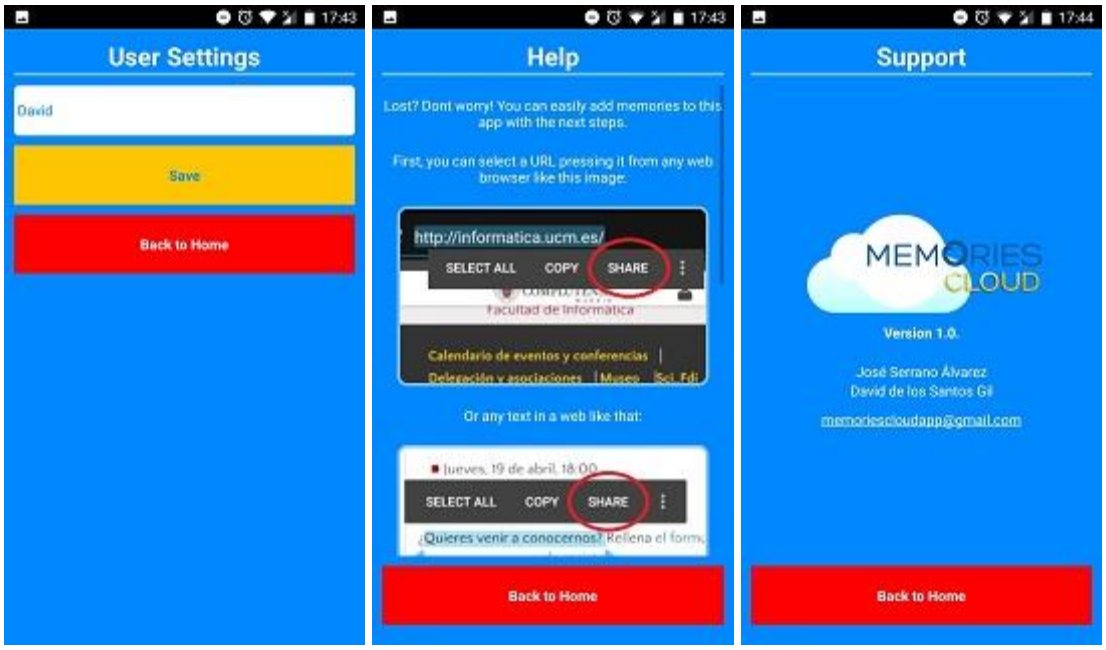


Figura 6.7. Opciones del menú desplegable
Fuente: Elaboración propia

Esta es la funcionalidad de la que consta nuestra aplicación en la versión 1.0, que es la versión de la aplicación a la entrega de este Trabajo de Fin de Grado.

6.2 Encuesta sobre la aplicación

Una vez terminada la primera versión de la aplicación, se ha procedido a la creación de una encuesta para saber la satisfacción de los usuarios con la misma. Conseguimos realizar esta encuesta sobre 10 personas, las cuales eran personas cercanas a nosotros, con lo que podíamos estar seguros de que eran personas que habían puesto interés en el uso de la aplicación, y que íbamos a recabar datos reales en la encuesta.

En esta encuesta se preguntaban aspectos que estaban relacionados con los que considerábamos que eran los puntos fuertes de la aplicación, empezando por su ligereza. En el siguiente gráfico podemos ver que de las 10 personas de las que pudimos reunir respuestas, el 90% consideraba que la aplicación resulta rápida y ligera.

Figura 6.8. Encuesta aplicación: Preguntas 1 y 2
Fuente: Elaboración propia

Además de eso, nos interesaba conocer qué pensaban los usuarios sobre el diseño de la aplicación, ya que, a pesar de gustarnos, considerábamos que este era uno de los puntos a mejorar, aunque quizá para ello necesitásemos otro tipo de conocimientos. A la mayoría de la gente el diseño le gustaba bastante (60%), frente al resto (40%), que respondió “Más o menos”.

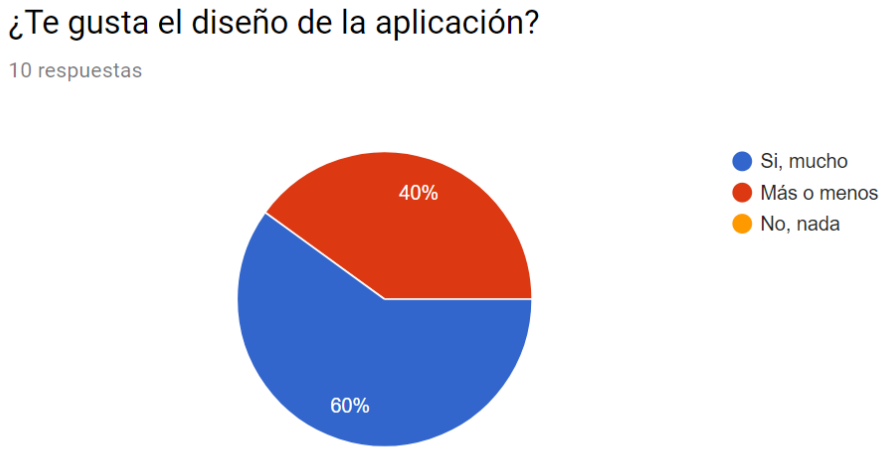


Figura 6.9. Encuesta aplicación: Pregunta 3
Fuente: Elaboración propia

En la siguiente pregunta, mencionábamos uno de los puntos clave de la aplicación, que era su funcionamiento ante la falta de internet del dispositivo. Si utilizamos la aplicación de forma recurrente, es inevitable que tarde o temprano nos quedemos sin internet por múltiples razones: falta de cobertura, mal funcionamiento de nuestro dispositivo, etc. Por esa razón, hemos visto que hay un porcentaje relativamente alto de gente que ha tenido algún problema con la conexión a internet. Esto se resuelve en la sección de trabajo futuro, en la que mencionamos una pila interna de la aplicación que guardará estos recuerdos en caso de que no haya internet para, cuando lo haya, subirlos a la base de datos.

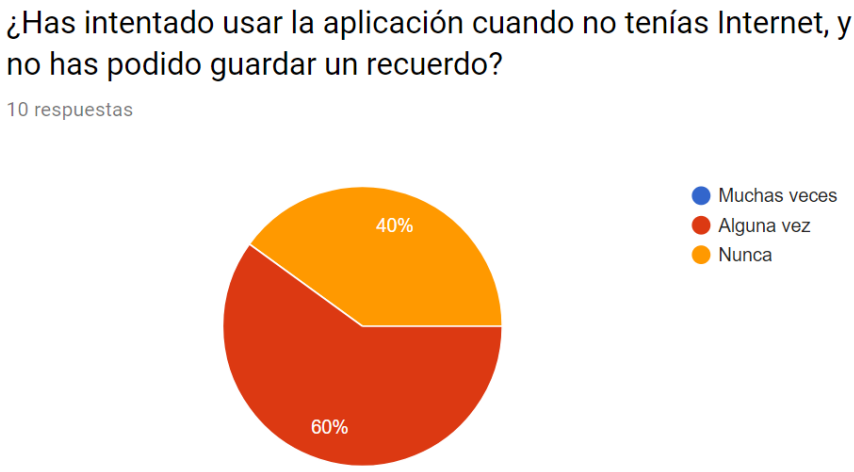


Figura 6.10. Encuesta aplicación: Pregunta 4
Fuente: Elaboración propia

Por último, formulábamos en esta encuesta dos preguntas orientadas a mejorar la usabilidad y resolver los problemas de una manera más general, dejando al encuestado que se expresase libremente. Esto nos sirve, sobre todo, para poder conocer problemas que han experimentado los usuarios que nosotros no hayamos tenido en cuenta. Las dos preguntas eran: "¿Te has encontrado con algún problema al utilizar la aplicación?" y "Cuéntanos qué mejorarías de la aplicación".

Para la pregunta acerca de problemas en la aplicación, en general los usuarios no han tenido ningún problema con ella. Un par de respuestas que merece la pena mencionar fueron el que la aplicación no se encontraba aún en el market de Android, cosa que a día de hoy está resuelta, y otra que nos comunicaba que no había podido recuperar notas que había eliminado sin querer. Sobre esta última respuesta, consideramos la opción de meter un cuadro de diálogo que te preguntase si estabas seguro de querer eliminar un recuerdo de la pila iba a sobrecargar una aplicación que queremos que sea lo más simple posible, con lo que de momento hemos decidido no introducir esta funcionalidad.

En cuanto a la opción que preguntaba sobre posibles mejoras de la aplicación, se mencionaban mejoras como poder elegir el idioma, cosa que de momento y para el público que pensamos tener (y por el poco texto y lo intuitiva que consideramos que resulta la aplicación) no consideramos introducir. Un par de personas mencionaban que se podría mejorar el diseño, poder reordenar los elementos mediante deslizamiento de los mismos y una papelera de notas. Estas respuestas nos parecen bastante interesantes, con lo que las tenemos consideradas para trabajo futuro.

7a. Conclusiones y trabajo futuro

Tras el fin del desarrollo de la primera versión de la aplicación, creamos un fichero APK para que un pequeño número de personas pudiesen probarla, y así contar con la opinión de personas externas al desarrollo del proyecto.

Nos encontramos ante una aplicación con una importante proyección hacia el futuro. A pesar de que consideramos que la ligereza y sencillez es uno de los pilares de la aplicación, esta puede contar con una gran cantidad de ampliaciones y mejoras, como las que mencionamos a continuación:

- La posibilidad de hacer que los recuerdos tengan la funcionalidad de añadir recordatorios y priorizarlos mediante un sistema de fijación al principio de la pila.
- Hacer que los recuerdos, si provienen de orígenes conocidos como “Google” o “Youtube”, importen el icono de la web y el título de la URL.
- Una papelera de recuerdos eliminados, para dar al usuario una segunda oportunidad a recuerdos que pensaba que no iba a usar.
- La opción de migrar la aplicación a sistemas operativos iOS, para lo cual teóricamente apenas habría que hacer cambios, pero es algo que no hemos podido comprobar ya que no disponíamos de ningún dispositivo iOS para probar la aplicación.
- También migrar a Windows. Para ello, hay múltiples opciones a estudiar. Ya que React usa de base Javascript, se podría migrar el código para que funcionara en web, o incrustar la aplicación en un emulador ligero de Android para escritorio. Según recientes noticias [35], se podrán usar aplicaciones de Android en Windows 10 en un futuro, aunque la alternativa de migrarlo a web haría que funcionara para cualquier dispositivo en el que se pueda acceder simplemente a internet.
- Cambiar la funcionalidad de compartir por la de mandar mensajes dentro de la aplicación, transformando la aplicación en una especie de red social de compartición de enlaces.
- Otra de las opciones que dejamos para trabajo futuro es la de poder utilizar la aplicación sin internet. En este caso, la aplicación tendría una base de datos interna que guardaría los recuerdos para poder enlazarlos con la base de datos en cuanto el dispositivo tenga conexión, en vez de funcionar solo cuando tengamos internet, que es lo que ocurre ahora.
- Desde el comienzo del proyecto hemos tenido en mente hasta dónde podríamos dejar al usuario personalizar la pila de la aplicación. Esto se podría hacer ampliando el número máximo de recuerdos, personalizando el aspecto de la pila, añadir etiquetas, categorías y prioridades a los recuerdos, etc.
- Mejora en el aspecto e interfaz de la aplicación, añadiendo animaciones, transiciones, y haciéndola al usuario más agradable visualmente.

7b. Conclusions and future work

After the development of the first version of the application, we created an APK file so a small number of people could test it, and thus have some opinions of people that is not related with the development of the project.

We are facing an application with an important projection towards the future. Although we consider that agility and simplicity are the main requirements of the application, we can see a big number of improvements, such as the following:

- The chance of making memories have the functionality of adding reminders and prioritizing them through a fixation system at the beginning of the stack.
- Make the memories, if coming from known origins as "Google" or "Youtube", import the web icon and the title of the URL.
- A trash of deleted memories, to give the user a second chance if he changes her mind about the using of past memories.
- The option of migrating the app to iOS operating systems, for which theoretically it would not be necessary to make changes, but it is something that we couldn't verify since we did not have any iOS device to test the application.
- The option of migrating the app to Windows. For this case, there are multiple options to study. Since React uses JavaScript base, you could be able to migrate the code to the web, or embed the application in a simple Android desktop emulator. According to recent news [35], Android applications may be used in Windows 10 in the future, although the alternative of migrating it to the web would make it work for any device that can simply access the Internet.
- Change the sharing functionality by sending messages within the application, transforming the application into a kind of social network sharing links.
- Another option that we leave for future work is to be able to use the application without internet. In this case, the application would have an internal database that would store the memories in order to link them to the database as soon as the device has a connection, instead of only working when we have Internet, which is what happens now.
- Since the beginning of the project we had in mind how far we could let the user customize the application's stack. This could be done by expanding the maximum number of memories, personalizing the appearance of the stack, adding labels, categories and priorities to the memories, etc.
- Improve the appearance and interface of the application, add animations, transitions, and make it more visually pleasing to the user.

Trabajo individual

José Serrano Álvarez

Ha realizado la investigación inicial para considerar qué tecnologías utilizar en el proyecto para la realización de la aplicación, así como los programas y aplicaciones necesarios para el desarrollo del proyecto.

Exploró las soluciones disponibles y escogió la que parecía más viable y con futuro para complementar la formación académica de cara a un entorno profesional, mediante la lectura de comparativas sobre características de distinto software.

Desarrollador principal de la aplicación una vez escogido el framework a utilizar, incluyendo el desarrollo de código, utilización de la base de datos, concurrencia de la aplicación, solución de problemas, etc. Así mismo, ha realizado parte de los apartados técnicos de la memoria y colaborando en el desarrollo activo de esta. También ha documentado durante el transcurso de este Trabajo de Fin de Grado los conocimientos técnicos adquiridos, especialmente sobre React Native, los cuales están reflejados en el Anexo.

David de los Santos Gil

Ha estado orientado principalmente a la parte comercial y documental del proyecto. Empezó este Trabajo de Fin de Grado aprendiendo el framework React Native, y desarrollando una pequeña aplicación para poner a prueba los conocimientos adquiridos. Tras haber aprendido los conceptos fundamentales, comenzó a desarrollar junto con su compañero José, el desarrollador principal, la aplicación.

Principalmente se ha encargado de desarrollar la memoria, con continua comunicación con el otro miembro del proyecto para mantenerla siempre actualizada y correcta, cosa que era fundamental por los continuos cambios que tenían la aplicación y la memoria, ya que una de ellas repercutía en la otra. Ha realizado la comparación de nuestra aplicación con otras aplicaciones similares del mercado, así como la investigación de funciones del navegador y otras posibles soluciones al problema que nos atañe, para ver cuál era el estado del arte y hacia dónde debíamos dirigirnos.

También ha llevado a cabo las dos encuestas, preparándolas y distribuyéndolas intentado que llegue a la máxima gente posible, mediante contactos de redes sociales, correos, etc. para de esta manera conseguir unos resultados fiables que nos guiaran en el desarrollo de este Trabajo de Fin de Grado.

Bibliografía

[1] Internet de las cosas. https://es.wikipedia.org/wiki/Internet_de_las_cosas. Consultado 20/03/2018.

[2] Amenazas desarrollo de aplicaciones. https://www.welivesecurity.com/la-es/2014/05/22/amenazas-comunes-desarrollo-aplicaciones/. Consultado 01/05/2018.

[3] Memoria. https://bluesmarteurope.wordpress.com/2013/06/09/los-diferentes-tipos-de-memoria-la-memoria-a-corto-plazo/.Consultado 15/01/2018. Libro: Psicología. 9ª Edición. Editorial Médica Panamericana. Autor: David G Meyers. (Cap 8, Pags 327-365)

[4] Memoria humana vs memoria computador. http://www.matlabtips.com/computer-vs-human-memor/. Consultado 23/1/2017.

[5] Siete pecados de la memoria. https://es.wikipedia.org/wiki/Los_siete_pecados_de_la_memorias. Consultado 20/12/2017.

[6] Daniel Schacter. https://en.wikipedia.org/wiki/Daniel_Schacter. Consultado 20/12/2017.

[7] SCRUM. https://proyectosagiles.org/que-es-scrum/. Consultado 25/03/2018.

[8] Definición Kanban. https://es.wikipedia.org/wiki/Kanban_(desarrollo). Consultado 13/03/2018.

[9] Sprints. https://proyectosagiles.org/ejecucion-iteracion-sprint/. Consultado 01/05/2018.

[10] Aplicación Pocket. https://getpocket.com/.

[11] Pódium mejores herramientas de desarrollo móvil. https://stackshare.io/stackups/apache-cordova-vs-ionic-vs-react-native. Consultado 20/11/2018.

[12] Apache Cordova. https://cordova.apache.org/.

[13] React Native. http://www.reactnative.com/.

[14] JSX. https://reactjs.org/docs/introducing-jsx.html.

[15] JavaScriptCore. https://developer.apple.com/documentation/javascriptcore.

[16] Visual Studio Code. https://code.visualstudio.com/

[17] Android Studio. https://developer.android.com/studio/index.html?hl=es-419.

[18] Eclipse. https://www.eclipse.org/.

[19] Github. https://github.com/.

[20] IBM RSA. https://www.ibm.com/developerworks/downloads/r/architect/index.html.

[21] Google Drive. https://www.google.com/intl/es-419_ALL/drive/.

[22] Microsoft Word. https://www.microsoft.com/es-es.

[23] Skype. https://www.skype.com/es/.

[24] Dropbox. https://www.dropbox.com/.

[25] Lucidchart. https://www.lucidchart.com/.

[26] Canal de Mario Díez. https://www.youtube.com/channel/UCisGMoxaVxJMcbio2FBHORg.

[27] Chocolatey. https://chocolatey.org/.

[28] GitHub for Desktop. https://desktop.github.com/.

[29] Firebase. https://firebase.google.com/.

[30] Características de Firebase. https://openwebinars.net/blog/que-es-firebase-de-google/.

[31] Redux. https://redux.js.org/.

[32] Integracion de Redux con React Navigation. https://reactnavigation.org/docs/en/redux-integration.html.

[33] Uso de Redux. https://es.stackoverflow.com/questions/41649/redux-cuando-pasar-props-y-cuando-coger-props-del-estado.

[34] MemoriesCloud en la Play Store. https://play.google.com/store/apps/details?id=com.memoriescloud

[35] Aplicaciones de Android en Windows 10. https://winphonemetro.com/2018/02/podriamos-ejecutar-apps-android-en-windows-10-sin-emuladores.

Anexo: React Native

Índice

1. Introducción a React Native

En el mundo del desarrollo de aplicaciones para móvil, los desarrolladores siempre han buscado intentar dirigir mejor los esfuerzos, ya que, ante tanta variedad de dispositivos, programar para cada uno siempre ha sido una tarea demasiado complicada, tanto por la variedad de lenguajes como de sistemas. Si se tiene un equipo lo suficientemente grande, es más factible ya que se pueden dividir recursos, pero ante equipos unipersonales o de pocos miembros, es más rentable buscar otro tipo de soluciones para dirigir mejor los recursos.

Esto es posible, gracias a la llegada de distintos frameworks que ayudan a que el mismo código sea funcional para cualquier dispositivo. El problema se ha limitado a que framework usar, ya que al haber bastantes que solucionan los problemas de forma efectiva, elegir uno u otro es ya simplemente cuestión de enfoque. En la actualidad, varios comparten el podio entre los más usados y mejor valorados. Nosotros también hemos tenido que pasar por esa decisión al encontrar entre todos los que hay, dos framework que se adaptan muy bien a lo que queríamos. Eran Apache Cordova y React Native.

Ambos tenían sus ventajas e inconvenientes.

Ambos de base, son multiplataforma. React Native parte de la base que, aprendiéndolo a usar, se puede escribir código para cualquier plataforma, a pesar de que ciertos componentes habría que reescribirlos dependiendo de la plataforma en la que estuviéramos, mientras que Apache Cordova instaba a crear una aplicación web que luego serviría en cualquier dispositivo sin casi ningún cambio.

Por ello, Apache Cordova es muy flexible, porque permite usar cualquier tecnología web que esté al alcance, y al ser un ecosistema muy variado, tenemos casi cualquier posibilidad con él. Sin embargo, esto crea un retardo en la interfaz hace que sea menos atractivo para nuestra aplicación como explicaremos posteriormente. Estos problemas realmente son solucionables, pero sería gastar recursos en algo que otro framework lo tiene solucionado de base.

React Native por su parte, se escribe con Javascript, más concretamente JSX. Se compila por cada sistema mediante el JavaScriptCore. Con el podemos acceder a todos los controles del dispositivo, y mediante su sistema de etiquetas para los elementos de la aplicación, cada sistema lo traduce a componentes que él mismo entiende, por lo que se personaliza dependiendo del dispositivo. La curva de aprendizaje de este lenguaje es mayor, ya que es muy moderno y aunque tiene muchísima documentación, hay que aprender de nuevo muchos conceptos y formas de crear cosas.

A la hora del desarrollo, ambos frameworks ofrecen muchas opciones y facilidades. Se diferencian sobre todo a la hora de depurar el código. Apache Cordova, al estar basado en tecnología web, nos permite depurar navegando en un navegador, que luego será adaptado a móvil. Esto tiene la desventaja de que, con algunos cambios, podríamos perder el estado de la aplicación, ya que estamos sustituyendo elementos por completo con cada actualización Con React Native, tenemos una emulación de un dispositivo en el que cuando queramos probar un cambio, no hace falta reiniciar por completo, ya que ofrece una actualización de módulos que hace que no perdamos el estado y podamos seguir depurando con los mismos datos. A su vez, ante cualquier problema o fallo, también en la propia simulación tendremos datos más precisos de lo que falla y así poder arreglarlo mejor buscando en el código justo lo que nos indica el simulador que está mal.

La elección no era del todo sencilla, ya que, a pesar de todo, Cordova tenía la ventaja de usar un entorno web que iba a ser más fácil de desarrollar, y menos laborioso de adaptar a todo sistema, ya que con unos pocos ajustes ya se tiene para cualquier dispositivo, mientras que React Native iba a darnos unas aplicaciones más fluidas y con mayor usabilidad, con el coste de tener que redefinir ciertas partes de código para adaptarlo a distintas plataformas a pesar de ser multiplataforma.

A pesar de haber muchas más diferencias, una de las claves para elegir uno u otro fue esta última. React Native, como su propio nombre indica, crea aplicaciones en las que el usuario tiene la sensación de que cada pulsación o tarea que haga, fluye como si fuera una aplicación creada en el código nativo del sistema en el que la esté desarrollando. Apache Cordova, al ser realmente una aplicación web traducida a aplicación, se comporta prácticamente como si fuera una web, ya que realmente se ejecuta en el navegador nativo del sistema. Esto conlleva los inconvenientes de los típicos retrasos entre presionar un link, y la redirección a la nueva página, que como hemos dicho, son factibles de arreglar mediante programación avanzada. Pero nuestro enfoque y objetivo hacían que esos recursos que habría que gastar para solucionarlo, podemos dedicarlos a mejorar más aún otros detalles de la aplicación.

Por todo ello, terminamos eligiendo React Native porque ofrece muchísimas opciones. Tiene de las mejores comunidades, ya que al estar basado en React, avala muchísimo su fuerza en las nuevas tecnologías y hay mucho material y componentes ya creados en la web que nos ayudarán en el desarrollo. También el hecho de que tuviera más fluidez nos gustaba, porque nuestro objetivo era tener una aplicación sencilla, pero que funcionara muy bien, así que el hecho de tener que trabajar e investigar algo más por el mejor funcionamiento de la aplicación era algo que no importaba si la calidad final iba a ser mayor. Otro detalle era el poder usar las entradas de interfaz al completo del dispositivo, por si en alguna iteración del proyecto incluyamos alguna funcionalidad que fuera necesitar tales datos de entrada. Cordova no permite esto ya que, al ser realmente una página web, tiene más limitado el acceso a datos del dispositivo. Había componentes que lo solucionaban, pero era el mismo problema que con los retardos, era investigar algo que React Native ya lo dejaba solucionado.

Después de probar y manejar algo de React Native, llegamos a la conclusión de que habíamos tomado la decisión acertada. Tanto el lenguaje usado, como las herramientas que nos proporcionaba y la facilidad de depurar la aplicación en un simulador de Android en el propio computador nos encantó nada más usarlo. Tiene una simplicidad que ayuda mucho al aprendizaje, y aunque sean conceptos nuevos, van asentándose fácilmente con la mera práctica. Los primeros desarrollos de prueba eran muy sencillos pero bonitos y funcionales, y era lo que queríamos para nuestra aplicación para que el usuario tenga la sensación de estar usando una aplicación simple, pero muy robusta y funcional para lo que único que necesitábamos.

En los siguientes capítulos, detallaremos el desarrollo de aplicación y explicaremos los componentes que hemos usado para crear la aplicación, que son únicamente una pequeña muestra de unas de las muchísimas funcionalidades que nos ofrece este potente framework.

https://www.toptal.com/mobile/comparing-react-native-to-cordova

https://stackshare.io/stackups/apache-cordova-vs-react-native-vs-xamarin

2. Instalación React Native

Para poder empezar a desarrollar código, necesitamos para instalar algunas dependencias el framework Chocolatey:

https://chocolatey.org/

Para instalarlo, hay que ejecutar este código en la consola de administrador:

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
ExecutionPolicy Bypass -Command "iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))" &&
SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

Las dependencias que instalaremos con el serán:

- Node 4
- Python2
- JDK 8

En una consola de administrador se deben ejecutar las siguientes órdenes por separado:

```
choco install nodejs.install
choco install python2
choco install jdk8
```

Nos queda por instalar React Native Command Line Interface (CLI), que se instala con una nueva consola de administrador, ejecutando el siguiente código:

```
npm install -g react-native-cli
```

También opcionalmente podemos instalar Yarn, que nos permitirá organizar e instalar mejor los paquetes y componentes que descarguemos. Se instala con la instrucción de consola:

```
choco install yarn
```

Por último, necesitaremos el entorno de desarrollo para Android, el Android Studio:

https://developer.android.com/studio/index.html

Se debe instalar y, una vez instalado, ejecutarlo y elegir un Custom Setup y comprobar que se selecciona lo siguiente siempre:

- Android SDK
- Android SDK Platform
- Performance (Intel ® HAXM)
- Android Virtual Device

Una vez instalado, debemos en la configuración del Launcher, entrar en SDK Manager (o, en el caso de no verla, al abrir el programa doble shift y buscar “SDK Manager”) e instalar Android 6.0 (Marshmallow), eligiendo con “Show Package Details” las siguientes opciones:

- Google APIs
- Android SDK Platform 23
- Intel x86 Atom_64 System Image
- Google APIs Intel x86 Atom_64 System Image

También debemos ir a la pestaña de SDK Tools y ver los detalles de Android SDK Build-Tools para seleccionar la versión 23.0.1. También seleccionaremos el Android Emulator.

Para terminar con la configuración de Android Studio, debemos agregar la variable de entorno siguiente:

Nombre: ANDROID_HOME
Valor: c:\Users***YOUR_USERNAME***\AppData\Local\Android\Sdk

Donde en valor debemos poner bien el nombre de usuario. Y ya tenemos configurado todo para empezar a crear un proyecto.

3. Control de Versiones con GitHub

El control de versiones lo haremos gracias a la aplicación “GitHub for Desktop”:

https://desktop.github.com/

Combina una potente interfaz gráfica en la que podremos aplicar nuestros cambios y tener un control de las distintas versiones de nuestra aplicación. Con ella, podemos trabajar en paralelo en el mismo proyecto en cualquier dispositivo que instalemos esta herramienta, y también poder visualizar los cambios en la página web de GitHub.

Únicamente, hay que instalarla, registrarse o introducir las credenciales personales de GitHub y clonar el repositorio. La dirección del repositorio de nuestro proyecto es:

ingeniero92/memoriesCloud

Una vez que tenemos el repositorio descargado en nuestro equipo, ya podemos abrir el proyecto con el Android Studio para trabajar con él y desarrollar la aplicación entre los integrantes del equipo.

Es totalmente obligatorio hacer una actualización de los cambios antes de trabajar para evitar los conflictos de archivos en la mayoría de lo posible.

Para descargar/actualizar los cambios, hay que presionar en el botón “Fetch origin”. Esto descargará los nuevos cambios, y en caso de haber cambios en nuestro equipo, nos aparecerá en la lista de “Changes” los archivos disponibles para comitear. Pondremos el título del commit en inglés, y en la descripción un breve texto explicando el cambio. Hay que presionar el botón de “Commit to master” para terminar con la subida de archivos.

Si hubiera algún conflicto, GitHub for Desktop muestra un mensaje de error, indicando que debes resolver los conflictos después de hacer el commit de nuestros cambios. Para ello, el sistema automáticamente escribirá en el archivo que tiene conflictos nuevas líneas con:

```
<<<<<<< HEAD
          Cambios en Local
=======
```

```
            Cambios en Remoto
>>>>>>> a01c7406d03d8e2750052277cd6717d8b20edbef
```

Nuestros cambios estarán en HEAD, y los que ya estaban en el servidor abajo. Debemos borrar las líneas sobrantes en un editor externo y quedarnos con el cambio que más nos interese. Una vez eso, hacemos commit de estos arreglos y ya estará solucionado el error.

Esto suele pasar cuando se ha hecho commit de unos archivos en un equipo, y a la vez en los mismos en otro. Se puede evitar intentando no tocar los mismos archivos si se trabaja en el mismo momento. Para ello la comunicación en el equipo es lo más importante.

En resumen, la forma de solucionarlo es una vez que, de conflicto, resolver en nuestro editor las líneas anteriores hasta quedarnos las que interesen, hasta que GitHub for Windows no incluya un editor de conflictos más potente.

El resto de opciones de la aplicación es muy intuitivo. Tenemos un historial de cambios en el que podremos ver las diferentes versiones de la aplicación y revertir cambios en caso de ser necesario. También se puede dividir el proyecto en distintas ramas de desarrollo. Hemos creado una rama una vez hemos terminado el desarrollo base, para hacer una rama para la versión BETA y otra para la versión 1.0.

4. Crear y Ejecutar Proyecto React Native

Para crear un proyecto nuevo de cero, hay que ejecutar en consola la siguiente orden:

```
react-native init NOMBRE_PROYECTO
```

Se creará en la ruta en la que estemos situados en la consola.

Si ya está creado/bajado de GitHub, únicamente hay que importar el proyecto desde archivos ya existentes seleccionando la carpeta raíz del proyecto.

Hay que aceptar dentro del Android Studio la solicitud de la ventana que nos salga de “Framework Android detected”, para poder acceder a todas las opciones de Android Studio.

Ejecutar aplicación con emulador

Posteriormente, tenemos que configurar el emulador con el AVD Manager. Está en la barra de herramientas de arriba (Tools->Android->AVD Manager). Ahí dentro, podemos crear una nueva máquina virtual. Hay que instalar HAXM si no está instalado, y seleccionar como imagen de sistema la versión Marshmallow 23 de Google API’s de x86_64. Si no está instalada, la descargamos. Luego solo queda configurar el emulador para que acepte entrada de teclado, para poder recargar la página, aunque suele estar activado.

Para ejecutar un emulador manualmente, ejecutamos esto en la consola:

```
C:\Users\USUARIO\AppData\Local\Android\sdk\tools\emulator.exe -avd Nexus_5X_API_23
-netspeed full -netdelay none
```

Cambiamos los datos en negrita subrayados por los datos de nuestro usuario local del equipo y el nombre que le hayamos dado a la máquina virtual.

También se puede ejecutar el emulador desde la aplicación de Android Studio. En Tools->Android->AVD Manager y pulsando en “Ejecutar”.

Para vincular la aplicación con el emulador, ejecutamos este código en la consola de la aplicación (parte inferior) situándonos en la carpeta del proyecto:

```
react-native run-android
```

Si el proyecto no se ha creado previamente, sino que ha sido descargado de GIT por ejemplo, es posible que de error y es necesario antes, ejecutar la siguiente orden estando en la carpeta del proyecto:

```
npm install
```

Ya estaría listo todo. Para abrir el menú desarrollador, usamos Control + M.

Ejecutar aplicación en dispositivo físico Android

Para depurar la aplicación en un dispositivo físico Android, debemos conectarlo vía USB, y en el teléfono, activar la depuración por USB en las opciones de desarrollador. Si al conectarlo nos pregunta por algún método de conexión, elegiremos MTP, que es transferencia de archivos.

Una vez conectado, nos vamos a una consola, y en la raíz del proyecto ejecutamos:

1. `mkdir android/app/src/main/assets` (Seguramente no haga falta, ya está creado)
2. `react-native bundle --platform android --dev false --entry-file index.js --bundle-output android/app/src/main/assets/index.android.bundle --assets-dest android/app/src/main/res`
3. `adb reverse tcp:8081 tcp:8081`
4. `react-native run-android`

Para abrir el menú desarrollador en el dispositivo, tenemos que agitar el teléfono o ejecutar la siguiente orden en una consola:

```
adb shell input keyevent 82
```

Y ya instalará la aplicación y la ejecutará. Para poder refrescar los cambios que hagamos en el ordenador, hay que abrir el menú desarrollador en el teléfono para hacerlo manualmente, o enviar la siguiente orden:

```
adb shell input text “RR”
```

Errores de compilado

Si a la hora de ejecutar el proyecto, da errores, este conjunto de instrucciones de consola suele ayudar a solucionar los problemas:

1. En el directorio de la aplicación, borrar la carpeta build de Android: “/android/build”.
2. En el directorio de la aplicación:

```
npm i
react-native link
```

3. En el directorio “android” de la aplicación:

```
gradlew clean
```

5. Sintaxis React Native

Componentes

Echemos un vistazo al hola mundo que podemos escribir en el archivo index.android.js:

```
import React, { Component } from 'react';
import { AppRegistry, Text } from 'react-native';

export default class HelloWorldApp extends Component {

  _onPressButton() { Alert.alert('You tapped!') }

  render() {
    return (
      <Text>Hello world!</Text>
       <Button
            onPress={this._onPressButton}
            title="Press Me"
          />
    );
  }
}

AppRegistry.registerComponent('AwesomeProject', () => HelloWorldApp);
```

El JavaScript de React Native es el futuro. Se basa en ES6. No es como el que conocemos. En esta página se muestran ciertas mejoras y funcionalidades de este JavaScript.

https://babeljs.io/learn-es2015/

El diseño de aplicaciones en React Native consiste en crear componentes. Estos componentes tienen como único requerimiento una función de render que devuelve el contenido a dibujar.

La clase de HelloWorldApp se debe registrar en el AppRegistry para decirle a React Native que es la raíz para toda la aplicación.

Hay dos tipos de datos que controlan un estado, los Props y el State:

Props y View

Props se encarga de enviar los datos de la vista del componente padre al componente hijo. Son variables para la creación de componentes, aquí un ejemplo en la misma función:

```
let pic = {
      uri:'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
};

return (
      <Image source={pic} style={{width: 193, height: 110}}/>
);
```

Sin embargo, también se pueden usar de la siguiente manera, invocando a nuestro componente tantas veces como se quiera con el prop que le vamos pasando cada vez:

```
class Greeting extends Component {
  render() {
    return (
          <Text>Hello {this.props.name}!</Text>
    );
  }
}

export default class LotsOfGreetings extends Component {
  render() {
    return (
              <View style={{alignItems: 'center'}}>
                        <Greeting name='Rexxar' />
                        <Greeting name='Jaina' />
                        <Greeting name='Valeera' />
              </View>
    );
  }
}
```

Podemos introducir tantos props como queramos, cada uno en una línea distinta.

Introducimos también el concepto de View, que sirve para contener otros componentes y darles de esa manera un cierto estilo. Como una especie de divisor.

Children

Cuando creamos un componente de clase, le solemos pasar props para visualizar contenido en él, o directamente, lo escribimos sin nada, de la siguiente manera:

```
<Card />
```

Ahora bien, si quisiéramos incluir, dentro de Card, un texto, haríamos esto:

```
<Card
    <Text>Soy un texto</Text>
 />
```

Sin embargo, esto no funciona. Es porque dentro del render de Card, debemos incluir Children:

```
{this.props.children}
```

Y así, mostraría el texto, haciendo que Card sea reutilizable dependiendo del uso que se le quiera dar.

Escogeremos Children cuando tengamos que crear un contenedor en el que guardar otras etiquetas de forma dinámica pero controlada, y usaremos Props cuando queremos algo determinado que no varíe en su forma, únicamente por los datos.

State

Se encarga de controlar los datos que se encuentran en la vista. Sirve para guardar el estado de la aplicación, se inicializa en el constructor y se va modificando:

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = {showText: true};

    // Toggle the state every second
    setInterval(() => {
      this.setState(previousState => {
        return { showText: !previousState.showText };
      });
    }, 1000);
  }

  render() {
    let display = this.state.showText ? this.props.text : ' ';
    return (
      <Text>{display}</Text>
```

```
    );
  }
}

export default class BlinkApp extends Component {
  render() {
    return (
      <View>
        <Blink text='I love to blink' />
        <Blink text='Yes blinking is so great' />
        <Blink text='Why did they ever take this out of HTML' />
        <Blink text='Look at me look at me look at me' />
      </View>
    );
  }
}

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => BlinkApp);
```

Sin embargo, es mejor utilizar un state container como Redux o MobX para facilitar la tarea del manejo del estado. Vamos a utilizar Redux por tener más información de el en la red, aunque sea algo más complejo para iniciados en React. En esta página tenemos tutoriales sobre Redux:

https://egghead.io/courses/getting-started-with-redux

Estilos

Con React Native, para nuestros estilos usamos únicamente JavaScript ya que los Props aceptan una propiedad de estilos. El lenguaje de este estilo es similar a CSS, cambiando los guiones por mayúsculas. De la siguiente forma, damos estilos a un Text.

```
...

<Text style={[styles.red, styles.bigblue]}>red, then bigblue</Text>

...

const styles = StyleSheet.create({
  bigblue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});
```

Dimensionando componentes

Para dimensionar componentes, se puede hacer de dos maneras:

- Tamaño Fijo:

  Se añade un ancho y un alto al estilo, como se haría en CSS. No hay unidad de medida en React Native, ya que los números representan densidad de pixeles independientes. De esta forma, sin importar el tamaño de la pantalla se verá siempre del mismo tamaño. Por ejemplo:

```
<View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
```

- Tamaño Flexible:

  En este caso, añadimos la propiedad "flex" a los estilos de un componente. Esto hará que el componente rellene todo el espacio disponible. A esta propiedad se le da un valor. Cuanto más valor, más espacio rellenará respecto a sus componentes al mismo nivel. Solo podemos hacer esto si el componente padre tiene tamaño fijo o también flexible, y en caso contrario, los hijos no serán visibles.

```
<View style={{flex: 1}}>
       <View style={{flex: 1, backgroundColor: 'powderblue'}} />
       <View style={{flex: 2, backgroundColor: 'skyblue'}} />
       <View style={{flex: 3, backgroundColor: 'steelblue'}} />
</View>
```

Disposición de componentes con Flexbox

Usando el algoritmo de flexbox, podemos definir la disposición de los hijos de un componente. Usaremos sobretodo tres propiedades:

- Flex Direction: Determina la disposición del eje principal. Puede ser row o column.

```
<View style={{flex: 1, flexDirection: 'row'}}>
      ...
</View>
```

- Justify Content: Determina cómo se colocan los elementos en el eje principal. Dependiendo de la posicion, seran: flex-start, center, flex-end, space-around y space-between.

```
<View style={{
            flex: 1,
            flexDirection: 'column',
            justifyContent: 'space-between',
   }}>
```

- Align Items: Determina cómo se colocan los elementos en el eje secundario. Dependiendo, pueden ser: flex-start, center, flex-end y stretch. Para que la propiedad stretch tenga efecto, los hijos no pueden tener dimensión fija.

```
<View style={{
           flex: 1,
           flexDirection: 'column',
           justifyContent: 'center',
          alignItems: 'center',
  }}>
```

Hay otras propiedades como:

- Align-self: Alinea de forma especial un elemento en concreto. Se puede rellenar con las propiedades: “flex-grow”, “flex-shrink” y “flex-basis”.

```
.item {
     Align-self: auto | flex-start | flex-end | center ...
    }
```

- Order: Da un orden en específico a los elementos de una secuencia:

```
.item {
     Order: <integer>;
    }
```

Entrada de Texto

Usamos un componente llamado TextInput. Tiene una función “onChangeText” que se llama cada vez que cambia el texto y “onSubmitEditing” que se llama cuando se envía el texto.

```
export default class PizzaTranslator extends Component {
  constructor(props) {
    super(props);
    this.state = {text: ''};
  }

  render() {
    return (
            <View style={{padding: 10}}>
                    <TextInput
                            style={{height: 40}}
                            placeholder="Type here to translate!"
                            onChangeText={(text) => this.setState({text})}
                    />
                    <Text style={{padding: 10, fontSize: 42}}>
                          {this.state.text.split('  ').map((word)  =>  word  &&
                          '✍').join(' ')}
                    </Text>
```

```
            </View>
    );
  }
}
```

Manejando Toques de Pantalla

Para iniciarnos en toques de pantalla, vamos a introducir el componente “Botón”, que lanza el evento de “onpress”. También podemos usar “onLongPress” si necesitamos que algún botón sea pulsado más tiempo.

```
<Button
       onPress={() => { Alert.alert('You tapped the button!')}}
       title="Press Me"
/>

<Button
     onPress={this._onPressButton}
     title="Press Me"
/>
```

Si preferimos usar otros elementos para que se puedan tocar, podemos construir botones propios con la componente “Touchable”. Tenemos distintos componentes:

- TouchableHighlight:El mas comun. Se oscurece el fondo cuando se usa el botón.
- TouchableNativeFeedback: En Android muestra efecto de toque.
- TouchableOpacity: El botón exagera el efecto toque.
- TouchableWithoutFeedback: El boton no refleja toque.

ScrollView

Para elementos largos o listas, necesitaremos hacer scroll. Para ello tenemos este contenedor que puede alojar múltiples componentes y vistas.

```
<ScrollView>
       ...
       <Text style={{fontSize:96}}>Scroll me plz</Text>
       ...
</ScrollView>
```

Se puede configurar para navegar a través de páginas con “pagingEnabled”. También podemos hacer una lista con un solo elemento para que con los props “maximumZoomScale” y “minimumZoomScale” se pueda con gestos ampliar o disminuir.
Estos elementos se renderizan automáticamente incluso si no se ven actualmente, por lo que está pensado para listas pequeñas. Si se desea hacer una lista de muchos elementos es mejor usar “FlatList”.

FlatList

Este elemento solo renderiza los elementos en pantalla, pudiendo cambiar en el tiempo. Requiere de dos props, data que contiene la información y renderItem que lo dibuja.

```
<FlatList
         data={[
           {key: 'Devin'},
           {key: 'Jackson'},
           {key: 'James'},
           {key: 'Joel'},
           {key: 'John'},
           {key: 'Jillian'},
           {key: 'Jimmy'},
           {key: 'Julie'},
         ]}
         renderItem={({item}) => <Text style={styles.item}>{item.key}</Text>}
   />
```

En el siguiente enlace, hay una buena explicación de cómo usarlo:

https://medium.com/react-native-development/how-to-use-the-flatlist-component-react-native-basics-92c482816fe6

SectionList

Si se desea dividir una lista en secciones lógicas, podemos usar “SeccionList”. Por ejemplo:

```
import React, { Component } from 'react';
import { AppRegistry, SectionList, StyleSheet, Text, View } from 'react-native';

export default class SectionListBasics extends Component {
  render() {
    return (
      <View style={styles.container}>
        <SectionList
          sections={[
            {title: 'D', data: ['Devin']},
              {title: 'J', data: ['Jackson', 'James', 'Jillian', 'Jimmy', 'Joel',
              'John', 'Julie']},
          ]}

          renderItem={({item}) => <Text style={styles.item}>{item}</Text>}

       renderSectionHeader={({section})                 =>                 <Text
       style={styles.sectionHeader}>{section.title}</Text>}

        />
      </View>
    );
  }
}
```

```
const styles = StyleSheet.create({
  container: {
   flex: 1,
   paddingTop: 22
  },
  sectionHeader: {
    paddingTop: 2,
    paddingLeft: 10,
    paddingRight: 10,
    paddingBottom: 2,
    fontSize: 14,
    fontWeight: 'bold',
    backgroundColor: 'rgba(247,247,247,1.0)',
  },
  item: {
    padding: 10,
    fontSize: 18,
    height: 44,
  },
})

// skip this line if using Create React Native App
AppRegistry.registerComponent('AwesomeProject', () => SectionListBasics);
```

Llamadas a funciones

La forma habitual en la que otro lenguaje llamaría a una función seria la siguiente:

```
function X (int){
    ...
}
```

Sin embargo, aquí podemos hacerlo de la siguiente manera:

```
export const X = int => {
          ...
}
```

Este comportamiento nos ayudará a poder usar funciones en otros sitios pasándolas por props por ejemplo

Condicionales

En ES6, tenemos una forma muy útil de hacer condicionales de existencia de forma rápida, veamos un ejemplo antes de explicarlo:

```
{ this.state.name
       ? <Text style = {styles.text}>Perfil de {this.state.name}</Text>
       : <Text style = {styles.text}>Perfil de usuario</Text>
}
```

Funciona de la siguiente manera: Se abren corchetes y se escribe la variable que queremos analizar, en este caso “this.state.name”. Luego, escribimos dos líneas, una con una “?” y otra con unos “:”, con código detrás de ellas. Si la variable no es null y está definida, ejecuta lo que aparece tras “?”, y si no, lo que hay tras “:”.

Networking

Las aplicaciones en su mayoría necesitan obtener o enviar datos al exterior. Se usa la API de Fetch. Aquí hay información adicional sobre ello:

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

- Peticiones

```
fetch('https://mywebsite.com/endpoint/', {
	method: 'POST',
	headers: {
		'Accept': 'application/json',
		'Content-Type': 'application/json',
	},
	body: JSON.stringify({
		firstParam: 'yourValue',
		secondParam: 'yourOtherValue',
	})
})
```

https://developer.mozilla.org/en-US/docs/Web/API/Request

- Respuesta

  Es un proceso asíncrono. Se puede hacer de dos maneras:

```
Function getMoviesFromApiAsync() {
	return fetch('https://facebook.github.io/react-native/movies.json')
		.then((response) => response.json())
		.then((responseJson) => {
			return responseJson.movies;
		})
	.catch((error) => {
		console.error(error);
	});
}
```

Ó

```
async function getMoviesFromApi() {
	try {
		let response = await fetch
		('https://facebook.github.io/react-native/movies.json');
		let responseJson = await response.json();
```

```
                        return responseJson.movies;
                    } catch(error) {
                        console.error(error);
                    }
                }
```

Ejemplo Final

```
import React, { Component } from 'react';
import { ActivityIndicator, ListView, Text, View } from 'react-native';

export default class Movies extends Component {

        constructor(props) {
            super(props);
            this.state = {
                  isLoading: true
            }
        }

        componentDidMount() {
            return fetch('https://facebook.github.io/react-native/movies.json')
              .then((response) => response.json())
                     .then((responseJson) => {
                       let ds = new ListView.DataSource({rowHasChanged: (r1, r2) => r1 !==
               r2});
                        this.setState({
                                 isLoading: false,
                                 dataSource: ds.cloneWithRows(responseJson.movies),
                        }, function() {
                              // do something with new state
                        });
              })
              .catch((error) => {
                    console.error(error);
              });
        }

        render() {
               if (this.state.isLoading) {
                        return (
                                <View style={{flex: 1, paddingTop: 20}}>
                                        <ActivityIndicator />
                                </View>
                        );
               }

               return (
                        <View style={{flex: 1, paddingTop: 20}}>
                                <ListView
                                        dataSource={this.state.dataSource}
                                        renderRow={(rowData)    =>    <Text>{rowData.title},
                                        {rowData.releaseYear}</Text>}
                                />
                        </View>
```

```
            );
        }

}
```

Más recursos

Para buscar librerías y componentes:

http://www.awesome-react-native.com/

https://js.coach/react-native

Proyectos ya terminados:

https://github.com/ReactNativeNews/React-Native-Apps

Tutoriales para aplicaciones profesionales:

http://makeitopen.com/tutorials/building-the-f8-app/planning/

6. Componentes de React Native

En este punto, analizaremos los componentes básicos de React Native que necesitaremos para crear la aplicación. Se importan en la cabecera, de la siguiente manera:

```
import{
    Text,
    View,
    StyleSheet,
    FlatList,
    Image
} from 'react-native'
```

ActivityIndicator

Para tener constancia de que un método se está ejecutando el primer plano, como un registro o un logueo de usuario, importamos “ActiviyIndicator”. Únicamente necesitamos tener una variable “loading” en el estado que modificaremos según necesitamos, y este fragmento de código que colocaremos al final de todos nuestros elementos del render()

```
{this.state.loading &&
   <View style={styles.loading}>
       <ActivityIndicator      style={styles.activityIndicator}      size="large"
       color="white" />
   </View>
}
```

Clipboard

La librería “Clipboard” nos da un componente clave para nuestra aplicación. Se encargará de gestionar la información del portapapeles, tanto para obtenerla, como para copiarla. Los métodos para ello son los siguientes:

```
async copyMemoryFromClipboard(){
       var value = await Clipboard.getString()
}

copyToClipboard(text) {
       Clipboard.setString(text)
}
```

BackHandler

Para poder usar el botón Back de Android, importaremos “BackHandler” y usamos los siguientes métodos:

```
componentWillMount() {
       BackHandler.addEventListener('hardwareBackPress', this.backPressed)
}

componentWillUnmount() {
       BackHandler.removeEventListener('hardwareBackPress', this.backPressed)
}

backPressed = () => {

       const { nav } = this.props
       const { routes, index } = nav
       const currentRoute = routes[index];

       if(currentRoute.routeName != "Home"){
           this.props.navigation.pop()
       } else {
           if(this.state.isOpen){
               this.setState({
                   isOpen: false
               })
           } else {
               BackHandler.exitApp()
           }
       }

       return true

   }
```

Esto lo ponemos en el componente raíz de la aplicación para despachar desde ahí todos los backs que se haga, que dependiendo de la ruta actual variará el comportamiento.

Linking

Si queremos redirigir desde una aplicación a otra, usamos “Linking” y en AndroidManifest.xml, incluimos las siguientes líneas:

```
android:name=".MainActivity"
android:launchMode="singleTask"

<intent-filter>
	<action android:name="android.intent.action.VIEW" />
		<category android:name="android.intent.category.DEFAULT"/>
		<category android:name="android.intent.category.BROWSABLE"/>
		<data android:scheme="com.memoriescloud"/>
</intent-filter>
```

Luego, solo habría que llamar a Linking de la siguiente manera:

```
Linking.openURL('com.memoriescloud://').catch(err => console.error(err));
```

AppState

Para comprobar si la aplicación pasa de background a foreground, usamos “AppState” de la siguiente manera:

```
componentDidMount() {
	AppState.addEventListener('change', this._handleAppStateChange);
}

componentWillUnmount() {
	AppState.removeEventListener('change', this._handleAppStateChange);
}

_handleAppStateChange = (nextAppState) => {
	if (this.state.appState.match(/inactive|background/) && nextAppState === 
	'active') {
		this.props.fetchData(this.state.uid)
	}
	this.setState({appState: nextAppState});
}
```

NetInfo

Necesitamos comprobar el estado de red del teléfono. Para ello, “NetInfo” nos provee de métodos para comprobarlo:

```
NetInfo.isConnected.fetch().then(isConnected => {...}
```

7. Componentes de Contribuidores

Instalación de Componentes Necesarios

Cuando necesitamos instalar un componente que no viene por defecto, usamos la siguiente orden de consola:

```
npm install --save “nombre del componente”
```

Y a continuación, enlazamos con React Native el componente recién instalado:

```
react-native link
```

A continuación, veremos componentes que precisan de esta instalación.

Iconos de Fontawesome

Para instalar los iconos de esta librería, hay que ejecutar los siguientes comandos en orden:

```
npm install --save react-native-vector-icons
```

Y luego, usar el siguiente import:

```
import Icon from 'react-native-vector-icons/FontAwesome'
```

Si da un error al compilar la aplicación, hacer:

```
rm node_modules/react-native/local-cli/core/__fixtures__/files/package.json
```

La guía está en:

https://github.com/oblador/react-native-vector-icons

Linear Gradient

Para instalar la librería para Linear Gradient, hacemos:

```
npm install --save react-native-linear-gradient
```

Swiper

Swiper es para hacer un slider de imágenes, se instala de la siguiente manera:

```
npm install --save react-native-swiper
```

Menu

Para crear menús a la derecha, instalamos lo siguiente:

```
npm install --save react-native-side-menu
```

Y en nuestra raíz, lo incluimos para usarlo:

```
import SlideMenu from 'react-native-side-menu'
```

Si queremos que se vuelva transparente el contenido cuando el menú está abierto, debemos ir a node-modules/react-native-side-menu/build/styles.js, hacer lo siguiente:

```
overlay: {
       ...absoluteStretch,
       backgroundColor: 'black',
       opacity: 0.7
}
```

React Navigation

Para navegar a través de la aplicación, usamos React Navigation, para instalarlo usamos:

```
npm install --save react-navigation
```

Puede dar algunos bugs, como la posibilidad de que la pantalla principal, cuando cargue la aplicación se inicialice dos veces. Para ello, en lugar de llamara a StackNavigator a la hora de crear la constante Navigator para nuestra aplicación, podemos llamar a BugFreeStackNavigator, que creamos en "/lib/BugFreeStackNavigator" de la siguiente manera:

```
import { NavigationActions, StackNavigator } from 'react-navigation';

const navigateOnce = (getStateForAction) => (action, state) => {
 const {type, routeName} = action;
 return (
   state &&
   type === NavigationActions.NAVIGATE &&
   routeName === state.routes[state.routes.length - 1].routeName
 ) ? null : getStateForAction(action, state);
};

function BugFreeStackNavigator() {
  var navigator = StackNavigator.apply(null, arguments);
  navigator.router.getStateForAction =
navigateOnce(navigator.router.getStateForAction);
  return navigator;
}

module.exports = BugFreeStackNavigator;
```

Solución encontrada en:

https://github.com/react-navigation/react-navigation/issues/2599

Para cambiar las transiciones, podemos consultar el siguiente artículo:

https://medium.com/async-la/custom-transitions-in-react-navigation-2f759408a053

React Tab View

Para crear sub tabs dentro de una vista, usamos

```
npm install --save react-native-tab-view
```

En el caso de que de error al cargar la tab, es que dentro del paquete de "node-modules/react-native-tab-view/src", habría que modificar los import para que PropType, sea importado de la siguiente manera, ya que este componente se cambió de lugar en recientes versiones de React Native:

```
import PropTypes from 'prop-types'
```

React Video

Para visualizar vídeos a pantalla completa, necesitamos instalar y luego enlazar:

```
npm install --save react-native-video
npm install --save react-native-video-controls
react-native link react-native-video
```

Obligar orientacíon

Para obligar a que la aplicación tenga una orientación u otra, usamos:

```
npm install --save react-native-orientation@git+https://github.com/yamill/react-native-orientation.git
react-native link react-native-orientation
```

En el archivo "android/app/src/main/java/com/#{AppName}/MainApplication.java" fijarse que "new OrientationPackage()" no tenga parametros.

Thunk

Para tener llamadas asíncronas con Redux, usamos Thunk:

```
npm install --save redux-thunk
```

Dropdown Alert

Cuando queremos mostrar algún aviso, en el que el usuario únicamente deba conocer información y no realizar ninguna acción, usamos este componente. Ejemplos de este uso pueden ser alerta de errores al intentar registrar un usuario, o eliminar un recuerdo.

```
npm install --save react-native-dropdownalert
```

Para poder usarlo correctamente, debemos tener en cuenta las siguientes pautas:

1. Importarlo

```
import DropdownAlert from 'react-native-dropdownalert'
```

2. Incluirlo dentro de nuestro render(), en el último nivel del <View> que engloba todo.

```
<DropdownAlert ref={ref => this.dropdown = ref} onClose={data => this.onClose(data)} />
```

3. Crear método para cerrar si necesitamos realizar alguna acción especial

```
onClose(data) {
    // data = {type, title, message, action}
    // action means how the alert was closed.
    // returns: automatic, programmatic, tap, pan or cancel
  }
```

React Native Animatable

Si queremos tener animaciones, usamos el siguiente componente:

```
npm install --save react-native-animatable
```

Para hacer la animación, damos referencia a un objeto Animatable

```
handleViewRef = ref => this.view = ref;

<Animatable.View ref={this.handleViewRef}>
    ...
</Animatable.View>
```

Y en algún lugar donde necesitemos, con esto llamamos a la animación:

```
this.view.rubberBand(2000)
```

React Native Modal

Si queremos mostrar al usuario una ventana emergente avisando de cualquier cosa que necesite su atención inmediata, usaremos este componente:

```
npm install --save react-native-modal
```

Para usarlo, únicamente necesitamos una variable en el estado que sea por ejemplo “isModalVisible” y un método para cambiar ese estado, para usarlo como aparece a continuación:

```
toggleModal = () => this.setState({ isModalVisible: !this.state.isModalVisible })
<Modal
    style={styles.modalContainer}
    isVisible={this.state.isModalVisible}
    supportedOrientations={['portrait', 'landscape']}
    onBackButtonPress={() => this.toggleModal()}
    animationIn = {'pulse'}
    animationOut = {'pulse'}
    animationInTiming = {600}
    animationOutTiming = {200}
    hideModalContentWhileAnimating = {true}
    backdropOpacity = {0.40}
>
    <View style={styles.modalBox}>

        <Text style={styles.modalTitle}>Update Needed</Text>

       <Text style={styles.modalText}>Sorry, but you need to update Memories Cloud.
Please, check in your store the new version.</Text>

        <TouchableHighlight
            onPress={() => this.toggleModal()}
            style={styles.exitButton}
            underlayColor = 'red'
        >
            <Text style={styles.textExitButton}>Back</Text>
        </TouchableHighlight>

    </View>
</Modal>
```

8. Redux

La aplicación, tiene una serie de distintas pantallas que están usando componentes que hemos creado de React Native. Estos, se enlazan para navegar entre ellos mediante un componente externo (React Navigation). Sin embargo, el estado de la aplicación y las variables compartidas, necesitan de un componente externo que lo maneje.

Este es Redux. Es a grandes rasgos, un contenedor del estado, es decir, de los datos de la aplicación. Junto a React Navigation, son los componentes clave para toda la aplicación. Cuando crece el código, se vuelve más complicado añadir funciones y tratar con datos. Redux nos ayudara con eso. Se carga en el archivo inicial de la aplicación, donde cargamos todas las dependencias, el navegador y cualquier elemento que debe inicializarse antes de llamar a la pantalla de carga. Está directamente enlazado con el navegador de la aplicación, y es capaz de mantener las variables que, mediante llamadas, ha almacenado en él para un uso posterior desde cualquier componente que incluya a Redux.

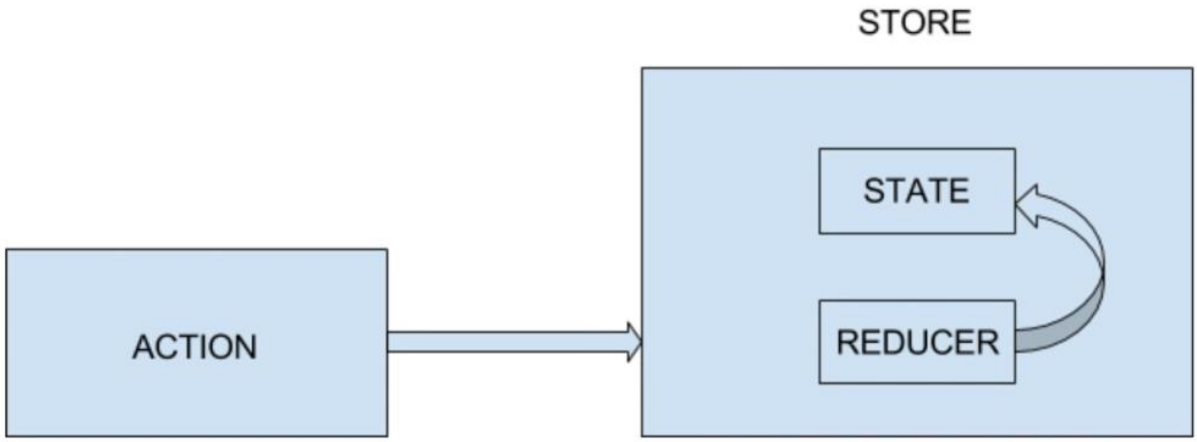
Para instalarlo, hacemos desde consola:

```
npm install --save redux react-redux
```

Para integrarlo con React Navigation, hay que seguir estos pasos:

https://reactnavigation.org/docs/redux-integration.html

El funcionamiento se puede ver en la siguiente figura:



Como se puede ver, Action es el encargado de enviar una solicitud a Reducer. Este es un ejemplo de una Action:

```
export const fetchMemories = (user) => {
   return (dispatch) => {
```

```
        dispatch(getMemories())
        FirebaseHelpers.getMemories(user)
            .then(([response]) => {
                dispatch(getMemoriesSuccess(response))
            })
            .catch((error) => dispatch(getMemoriesFailure(error)))
    }
}
```

Reducer recibe una Action, y dependiendo del tipo, obtiene o modifica el Estado. Reducer y State conforman la Store, que encapsula a estos dos para funcionar de contenedor de estado. Un Reducer puede ser:

```
import {FETCHING_MEMORIES, FETCHING_MEMORIES_SUCCESS, FETCHING_MEMORIES_ERROR} from
'../constants';

const initialState = {
   memories: [],
   isFetching: false,
   error: false,
   success: false
}

export default memoriesReducer = (state = initialState, action) => {
   switch(action.type){
       case FETCHING_MEMORIES:
           return {
               ...state,
               memories: [],
               isFetching: true,
               error: false,
               success: false
           }
       case FETCHING_MEMORIES_SUCCESS:
           return {
               ...state,
               memories: action.data,
               isFetching: false,
               error: false,
               success: true
           }
       case FETCHING_MEMORIES_ERROR:
           return {
               ...state,
               memories: [],
               isFetching: false,
               error: true,
               success: false
           }
       default:
           return state
   }
}
```

Para usar Redux con la aplicación en React Native, hay que englobar el componente principal de la aplicación en un Provider, que es un componente que nos proporcionará la Store que contendrá los datos que depositemos ahí con nuestras Action y Reducer.

```
export default Index = () => {
   return (
       <Provider store={store}>
           <AppIndex />
       </Provider>
   )
}
```

Para combinar Redux con alguno de nuestros componentes, deberemos usar Connect, que hará que ese componente pueda acceder a la Store. Este Connect, tiene dos parámetros, el primero es un método que conecte los Reducers que necesitemos, y el segundo, los Action que usaremos.

```
const mapStateToProps = state => {
   return {memories: state.memories, user: state.user}
}

const mapDispatchToProps = dispatch => {
   return {
       fetchMemories: (user) => dispatch(fetchMemories(user)),
       fetchUser: () => dispatch(fetchUser())
   }
}

export default connect(mapStateToProps,mapDispatchToProps)(List)
```

Una vez conectado, únicamente deberemos acceder a los Prop del componente, para poder obtener los datos de la Store o ejecutar las Action para modificarlo.

Aquí está la página de Redux y unos videotutoriales:

http://redux.js.org/
https://egghead.io/courses/getting-started-with-redux

9. Firebase

Firebase

Para gestionar los usuarios de nuestra aplicación, vamos a usar Firebase, una plataforma que nos permitirá crear el login de la aplicación y poder guardar los datos del usuario.

https://firebase.google.com/

Hay más información en el siguiente link:

https://openwebinars.net/blog/que-es-firebase-de-google/

Con una cuenta de google, se pueden acceder a todas las funciones, de forma gratuita. Si la aplicación crece, siempre se puede migrar a un plan de pago que permita sin problemas todo el flujo de datos.

Se instala en consola con el siguiente comando.

```
npm install --save firebase
```

Para importarlo en nuestros proyectos, hacemos:

```
import * as firebase from 'firebase'
```

Con esto, ya podemos usar la variable firebase y sus numerosos métodos para poder gestionar los usuarios y sus datos.

Debemos inicializar Firebase para que funcione, llamando siempre a esta función en la Home/Loading de nuestro proyecto del siguiente archivo, que podríamos guardar como "lib/firebase.js":

```
import * as firebase from 'firebase'

class Firebase {
   static init(){
       firebase.initializeApp({
           apiKey: "AIzaSyAy_ZDEWc_BYiy9Kbo_OvClOFgJKDFkLxU",
           authDomain: "netflix2-7b1f9.firebaseapp.com",
           databaseURL: "https://netflix2-7b1f9.firebaseio.com",
           projectId: "netflix2-7b1f9",
           storageBucket: "netflix2-7b1f9.appspot.com",
       })
   }
}

module.exports = Firebase
```

Estos datos, hay que copiarlos desde la web de Firebase, ya que podemos crear varios proyectos.

Importándolo y habiendo iniciado, podemos usar ya todas las funciones que trae la librería "firebase", las cuales nos brindan todas las posibilidades que nos da Firebase, entre ellas, aparte de tener el sistema totalmente funcional de registro y sesión, nos permite guardar datos de usuario y poder cargarlos para usar en la aplicación en la base de datos.

También nos permitiría, guardar en su base de datos los recuerdos.

Esto hace, que sea muy conveniente que la primera pantalla de nuestra aplicación, sea un Loading que se encargue de iniciar Firebase, y comprobar si hay sesión activa, para redirigir a una pantalla de logueo, o a la pantalla principal. Después de eso, podríamos obtener todos los datos que necesitamos del usuario activo. Esto se ha debatido en:

https://github.com/react-navigation/react-navigation/issues/620

Una guía completa de la aplicación puede ser encontrada aquí, en el sitio web principal de Firebase:

https://firebase.google.com/docs/guides/

Errores y Warnings de Firebase

La librería de Firebase, aun tiene algunos errores serán siendo depurados en futuras actualizaciones. Son simplemente unos “Warnings” que siempre saldrán, y aunque no tengan relevancia y la aplicación se ejecute normalmente, podemos omitirlos simplemente, con estas líneas de código en nuestro “index.js” principal de la aplicación para facilitar nuestro proceso de depuración:

```
import { YellowBox } from 'react-native';
import _ from 'lodash';
YellowBox.ignoreWarnings(['Setting a timer']);
YellowBox.ignoreWarnings(['FIREBASE WARNING']);
const _console = _.clone(console);
console.warn = message => {
 if (message.indexOf('Setting a timer') <= -1) {
   _console.warn(message);
 }
};
```

10. Extensión Share

Share

En nuestra aplicación, necesitamos obtener cadenas de caracteres de la barra de dirección del navegador, o cualquier fuente de texto compatible con el sistema de portapapeles del dispositivo. Para ello, vamos a usar una extensión llamada: React Native Share Extensión.

Su repositorio en Github es el siguiente:

https://github.com/alinz/react-native-share-extension

La instalación no es trivial, como pasa con otros componentes de React Native. Hay que hacer ajustes extra tanto para Android, como para iOS, siendo la primera vez que haya que hacer trabajo doble para poder seguir teniendo la misma funcionalidad en ambas plataformas.

Hemos realizado los ajustes para Android en la primera iteración, y realizaremos los de iOS en la segunda iteración, una vez que tengamos la aplicación totalmente funcional.

En Android, el funcionamiento de la extensión es el siguiente:

En cualquier lugar que nos permita seleccionar texto libremente, ya sea una barra de direcciones de un navegador, o dentro de una página web, al seleccionarlo se abre una barra que nos permitirá copiar en el portapapeles el contenido. En esa barra, en la mayoría de los dispositivos, hay unos puntos que nos permiten seleccionar más opciones. Una de ellas es "Compartir". Seleccionando compartir, nos permitirá elegir las aplicaciones que permitan este funcionamiento, entre ellas, Memories Cloud.

Esto hace, que se abra una pantalla de nuestra aplicación, que nos permitirá ver el recuerdo que hemos seleccionado, y guardar o cancelar la acción. Esta pantalla, no forma parte de la aplicación principal, por lo que, si tenemos abierta en segundo plano la aplicación, esta pantalla nueva no cambiará el estado de nuestra aplicación. Así mismo, permite varias instancias de ella.

Un ejemplo de aplicación con el mismo funcionamiento, es Dropbox.

En los siguientes puntos, explicaremos como se ha realizado la instalación de la extensión. La guía de instalación está disponible en el repositorio, sin embargo, hay varias cosas desactualizadas que añadimos aquí para que no haya ningún error.

Instalación en Android

Para empezar, instalamos la extensión mediante consola:

```
npm install --save react-native-share-extension
```

Una vez instalada la extensión, tenemos que realizar los siguientes pasos:

Incluimos en el archivo "android/settings.gradle" las siguientes dos siguientes líneas, aunque normalmente, la primera línea ya está, y solamente hay que añadir lo que falta y la segunda:

```
include ':app', ':react-native-share-extension'

project(':react-native-share-extension').projectDir = new
File(rootProject.projectDir, '../node_modules/react-native-share-
extension/android')
```

Después, en el archivo “android/app/build.gradle”, añadimos lo siguiente:

```
dependencies {
    ...
    compile project(':react-native-share-extension')
    compile "com.facebook.react:react-native:+"
}
```

Una vez completado este archivo, creamos una carpeta llamada “share”, de modo que este en la siguiente ruta: “\aplicacion\android\app\src\main\java\com\netflix2\share”

Dentro de ella, creamos dos archivos con el siguiente contenido:

- ShareActivity.java

```
// define your share project, if your main project is com.sample1, then
com.sample1.share makes sense....

package com.aplicacion.share;

// import ReactActivity
import com.facebook.react.ReactActivity;

public class ShareActivity extends ReactActivity {
    @Override
    protected String getMainComponentName() {
      // this is the name AppRegistry will use to launch the Share View
        return "Share";
    }
}
```

- ShareApplication.java

```
// your package you defined in ShareActivity
package com.aplicacion.share;
// import build config
import com.aplicacion.BuildConfig;

import com.alinz.parkerdan.shareextension.SharePackage;

import android.app.Application;

import com.facebook.react.shell.MainReactPackage;
import com.facebook.react.ReactNativeHost;
import com.facebook.react.ReactApplication;
import com.facebook.react.ReactPackage;

import java.util.Arrays;
```

```
import java.util.List;

public class ShareApplication extends Application implements
ReactApplication {
 private final ReactNativeHost mReactNativeHost = new
ReactNativeHost(this) {
   @Override
   protected boolean getUseDeveloperSupport() {
     return BuildConfig.DEBUG;

   }

   @Override
   protected List<ReactPackage> getPackages() {
     return Arrays.<ReactPackage>asList(
         new MainReactPackage(),
         new SharePackage()
     );
   }
 };

 @Override
 public ReactNativeHost getReactNativeHost() {
     return mReactNativeHost;
 }
}
```

En el archivo "\aplicacion\android\app\src\main\java\com\netflix2\MainApplication.java", añadimos el siguiente import:

```
import com.alinz.parkerdan.shareextension.SharePackage;
```

Y modificamos los métodos siguientes, ya que, siendo protegidos, da error, y añadimos la creación de la extensión:

```
@Override
    public boolean getUseDeveloperSupport() {
      return BuildConfig.DEBUG;
    }

    @Override
    public List<ReactPackage> getPackages() {
      return Arrays.<ReactPackage>asList(
          new MainReactPackage(),
          new SharePackage()
      );
    }
  };

  @Override
```

```
  public ReactNativeHost getReactNativeHost() {
      return mReactNativeHost;
  }
```

Buscamos “android/app/src/main/AndroidMainfest.xml” y añadimos el bloque de <activity> después del bloque de “devSettingsActivity”:

```
<activity
android:name="com.facebook.react.devsupport.DevSettingsActivity"/>

<activity
    android:name=".share.ShareActivity"
    android:configChanges="orientation"
    android:label="@string/title_activity_share"
    android:screenOrientation="portrait"
    android:theme="@style/AppTheme" >
   <intent-filter>
     <action android:name="android.intent.action.SEND" />
     <category android:name="android.intent.category.DEFAULT" />
    //  for sharing links include
     <data android:mimeType="text/plain" />
    //  for sharing photos include
    <data android:mimeType="image/*" />
   </intent-filter>
</activity>
```

Hemos usado la variable "@string/title_activity_share", por lo que vamos a definirla en “\aplicacion\android\app\src\main\res\values\strings.xml”. Esta variable, será el string de la cadena de texto de nos aparecerá en “Compartir”.

```
<resources>
    ...
    <string name="title_activity_share">Save in MyShareEx</string>
</resources>
```

Conectar con React Native

Para conectar la extensión con nuestra aplicación, debemos añadir en nuestro index principal, lo siguiente:

```
import Share from './share/share

AppRegistry.registerComponent(Share, () => Share)
```

Donde “Share”, es el nombre que le hemos puesto a la extensión, que debe coincidir con el que pusimos en “ShareActivity.java”.

En el archivo “share.js”, debemos realizar la funcionalidad que queremos hacer con la cadena de texto. Con el siguiente código, que debemos incluir en “share.js”, conseguimos acceder a la cadena y ya poder operar con ella:

```
import ShareExtension from 'react-native-share-extension'

async componentDidMount() {
   try {
     const { type, value } = await ShareExtension.data()
     this.setState({
       type,
       value
     })
     console.log(value)
     ShareExtension.close()
   } catch(e) {
     console.log('err', e)
   }
 }
```

11. Variables y ajustes en Android

Los ajustes para Android, se harán dentro de la carpeta “/android/src”.

Icono Aplicación

En la carpeta “/android/src/res”, tenemos una serie de carpetas que empiezan por mipmap, con el .png por defecto para las aplicaciones en Android, únicamente debemos sustituir este .png por el que queramos. Este icono también será usado para el icono de “Compartir” automáticamente, al no especificar otro. El icono lo hemos creado mediante CSS modificando ligeramente un código encontrado en el siguiente artículo:

> http://www.desarrollolibre.net/blog/tema/89/css/como-hacer-una-simple-nube-en-css#.WoxhOOdG0uU

También hemos usado estas fuentes:

> https://beatrizxe.com/es/blog-diseno-ilustracion/blog-diseno-grafico/tipografias-modernas-para-descargar-gratis.html

Tema y colores de la Aplicación

En la carpeta “/android/src/res”, hay un archivo llamado “styles.xml”. En él, dentro de la etiqueta <resources> podemos ajustar varios parámetros de la configuración del tema de

la aplicación, entre ellos, los colores de la barra de notificaciones, o el color del cursor a la hora de seleccionar texto.

```
<!-- Base application theme. -->
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
      <!-- Customize your theme here. -->
      <item name="colorPrimary">@null</item>
      <item name="colorPrimaryDark">@null</item>
      <!-- sets cursor color -->
      <item name="colorControlActivated">@color/greenlight</item>
</style>
```

También hay más ajustes que pueden ser declarados directamente en el código de React Native, los cuales se pueden encontrar en el siguiente artículo:

http://www.kiodev.com/react-native-on-android-styling-the-cursor/

Crear APK

Para finalmente, lanzar la aplicación al Google Play, hay que crear la APK firmada para poder subirla con seguridad. Para ello, hay que crear una key en el equipo e integrarla en el proyecto para que se pueda compilar con ella. Es importante guardar la key ya que si se desea subir futuras versiones al Google Play. Esta guía explica paso por paso lo que hay que hacer para generar la key y compilar el proyecto.

https://facebook.github.io/react-native/docs/signed-apk-android.html