FORENSICNOTES, UNA *APP* DESARROLLADA CON *IONIC 2* Y *FULL STACK*

Nombre Estudiante: José Antonio Pérez Salvador
Plan de Estudios del Estudiante: Grado en Ingeniería Informática
Área del trabajo final: Desarrollo de aplicaciones para dispositivos móviles (*HTML5* o *Windows Phone*)

Nombre Consultor: Carlos Sánchez Rosa
Nombre Profesor responsable de la asignatura: Carlos Garrigues Olivella

Fecha de Entrega: 14/06/2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*ForensicNotes,* una *app* desarrollada con *Ionic 2* y *Full Stack*
Nombre del autor:	José Antonio Pérez Salvador
Nombre del consultor/a:	Carlos Sánchez Rosa
Nombre del PRA:	Carlos Garrigues Olivella
Fecha de entrega (mm/aaaa):	06/2017
Titulación::	Grado en Ingeniería Informática
Área del Trabajo Final:	Desarrollo de aplicaciones para dispositivos móviles (*HTML5* o *Windows Phone*)
Idioma del trabajo:	Castellano
Palabras clave	"Ionic 2", "dispositivos móviles" y "análisis forense"

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

ForensicNotes es una aplición desarrollada con el *framework* de *Ionic*, que permite crear notas de texto, tomar fotografías y mucho más.

Su principal característica es la capacidad para cambiar el color de fondo de las notas, lo cual ayuda a la hora de organizarse. Otras características incluyen almacenamiento persistente y almacenamiento en la nube.

ForensicNotes funciona como un servicio a la hora de tomar notas, dado que éstas pueden distribuirse de forma síncrona entre dispositivos, que utilizan *Cloudant*.

La aplicación utiliza una interfaz similar tanto en *iOS*, *Android*, como en cualquier navegador *web*, con un fondo e iconos de color azul claro.

En cuanto a las notas, hay que decir que son algo muy personal y varían enormemente de una persona a otra.

Lo principal es que las notas registren todas y cada una de las acciones y a su vez, permitan que cualquiera que repita el mismo proceso obtenga los mismos resultados.

A veces, las personas hacen declaraciones sin tomar notas.

Registrar todo lo que se hace o analiza es una buena praxis que garantiza el éxito de cara al análisis y examen forense, pero no impide que se pueda ser original a la hora de sacar uno sus propias conclusiones.

En casos forenses de pequeña envergadura no existe mucha diferencia entre tomar notas o no, pero en grandes casos las diferencias se acentúan, y puede llegar a convertirse en un problema no saber cómo se ha llegado a una conclusión.

Abstract (in English, 250 words or less):

ForensicNotes is an application developed in Ionic framework. It allows you to create text notes, taking pictures and much more.

Its main feature is the ability to change the background color of notes to help you stay organized. Some other features include internal storage and cloud storage, and more.

It functions as a service for making short text notes, which can be synchronised between devices using Cloudant's service.

The application uses a similar interface on iOS, Android and any web browsers, with a textured paper background for notes and light blue icons.

Notes are a very personal record and vary greatly in form from one person to another.

The main point to remember is that your notes should be a complete record of your actions and should enable anyone who repeats them to have the same results.

Sometimes people use their statements as an addition to their notes and state actions performed that are not recorded in their notes.

A checklist for commonly performed tasks is a good basis for ensuring a consistent approach to every examination but doesn't excuse one from not thinking 'outside the box'.

In small cases it makes no difference but in larger ones it can be a real problem finding how you came to a particular conclusion.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

El proyecto consiste en la elaboración de una herramienta *open source* (aplicación móvil y *back-end*), llamada "ForensicNotes", que permita a los peritos forenses documentar toda la información relacionada con el análisis forense informático de los dispositivos entregados (ya se trate de dispositivos móviles, estaciones de trabajo, soportes de datos convencionales, etc.) y simplifique el proceso.

En general, esto implica desde un primer momento:

- Habilitar un control de acceso en el lado del servidor con *JWT*.
- Organizar los dispositivos que se van a examinar.
- Ubicar la posición inicial de los dispositivos objeto de examen mediante *Google Maps*.
- Tomar notas, notas de voz, fotografías, ubicaciones geográficas, etc… Sobre las evidencias encontradas, subir esta información al servidor y generar información que ayude a elaborar el informe final.
- Dividir el proceso de análisis forense en fases con el objetivo de mantener estructurada la información relevante que se ha extraído del dispositivo.
- Importar información del servidor relacionada con las evidencias encontradas.
- Generar una representación gráfica del orden de creación de las notas (*Timeline*).
- Y compartir la información con otros miembros del laboratorio forense.

Finalmente, indicar que la herramienta no está diseñada para ser instalada en un servidor de *Internet*, dado que las evidencias forenses y los metadatos están considerados como información sensible. Por tanto, es recomendable instalar el *back-end* en una red local.

Sin embargo, si fuera necesario desplegar "ForensicNotes" a través de *Internet*, se recomienda proteger los servicios *online* con *Latch*[1], el cual se encuentra disponible en ***https://latch.elevenpaths.com***.

[1] Esta solución se va a implementar en el proyecto.

1.1. CONTEXTO Y JUSTIFICACIÓN DEL TFG

Los dispositivos móviles han cobrado especial importancia a nivel laboral, donde facilitan no solo la comunicación telefónica, sino también la diversidad de aplicaciones que podemos instalar para facilitar nuestro trabajo (navegador, correos, *apps* especiales para cada sector y un gran etc...).

Extrapolándolo al proceso forense, los dispositivos móviles, por sus características y portabilidad, son herramientas que puedan ayudar a los peritos forenses en la elaboración de sus informes.

Por esa razón, y porque no existe en el mercado un modelo único de aplicación que nos permita recoger información relevante para llevar a cabo el informe forense, debido en parte a que los procedimientos y normas para el análisis en dispositivos móviles se encuentran en desarrollo (es decir, las metodologías), se ha propuesto una aplicación que se adapte a las condiciones del ámbito tecnológico actual y a las etapas del proceso forense.

1.2. OBJETIVOS DEL TFG

Por una parte, el objetivo general del proyecto consiste en construir una aplicación móvil[2] en *Ionic 2* para *smartphones* llamada "ForensicsNotes", que satisfaga los objetivos planteados dentro del proyecto. Esto implica que habrá que codificar la aplicación móvil (tanto para *Android*, como para *iOS*), crear una infraestructura de *back-end* escalable y conectada y aprovechar los *frameworks*, librerías y *APIs* más adecuados para la problemática a resolver.

Por otra parte, los objetivos concretos a alcanzar en este trabajo, son los siguientes:

- Poner en práctica los conocimientos adquiridos a lo largo de toda la titulación.
- Conocer el proceso de desarrollo de una *app* desde su concepción hasta su distribución.
- Ser capaz de concretar una idea de *app* en un proyecto que permita su desarrollo.
- Ser capaz de integrar *Latch* en el *back-end*.
- Saber definir una planificación realista para un proyecto, teniendo en cuenta su alcance y los recursos disponibles.

[2] Hay que tener en cuenta que el desarrollo de *apps* incluye *smartphones*, *tablets* y dispositivos similares, dado que éstos también son considerados dispositivos móviles.

- Ser capaz de documentar y justificar las decisiones tomadas en el desarrollo y los resultados conseguidos.
- Adquirir experiencia en afrontar los retos que supone sacar adelante un proyecto completo.
- Investigar usuarios y recoger requisitos, tanto cuantitativos como cualitativos, que ayuden a conocer los usuarios y definir perfiles.
- Examinar y analizar las condiciones en que se utilizará el sistema para definir su contexto de uso.
- Elaborar un análisis de tareas.
- Elaborar escenarios de uso.
- Definición de los flujos de interacción en el sistema.
- Diseñar y construir un prototipo de alto nivel del sistema teniendo en cuenta los conceptos de las evaluaciones heurísticas y de las particularidades del diseño para dispositivos móviles.
- Plantear la evaluación del prototipo del sistema mediante un *test* con usuarios.
- Mantener la visión de conjunto en todas las etapas de la elaboración de la práctica, identificando los aspectos a mejorar en cada iteración del proceso de DCU.
- Saber implementar funcionalidades concretas en una aplicación móvil utilizando los lenguajes de programación y las herramientas disponibles según la plataforma o plataformas escogidas.
- Conocer mejor la tecnología utilizada en la implementación de la aplicación.
- Saber testear el correcto funcionamiento de una aplicación móvil y depurar los posibles errores.
- Saber completar el desarrollo de una aplicación móvil y preparar su distribución para un usuario final.
- Ser capaz de documentar y justificar las decisiones tomadas en el desarrollo y los resultados logrados.
- Ser capaz de presentar el trabajo realizado a un público no especializado.

1.3. ENFOQUE Y MÉTODO ELEGIDO

Dada la gran fragmentación de plataformas y tipos de aplicaciones que existen, se ha optado por diseñar una aplicación móvil híbrida o *cross-platform*, un

producto nuevo que combine desarrollo nativo con tecnología *web* utilizando *Ionic 2* y *Cordova*, dirigido tanto a la plataforma *Android* (dado que es el sistema operativo más popular del mercado), como a *iOS*[3] (que también cuenta con una importante cuota de mercado) Es decir, utilizar el mismo código independientemente de la plataforma en que se ejecute la aplicación.

Esta estrategia es la más apropiada para conseguir los objetivos, dado que el enfoque híbrido presenta las siguientes ventajas:

- Sólo una línea de desarrollo para varias plataformas (*Android* e *iOS*, por ejemplo). Es decir, escribir el código una vez y ejecutarlo en la plataforma deseada.
- Aplicaciones que podemos instalar en nuestros dispositivos, que tienen la posibilidad de ser distribuidas en los mercados de aplicaciones.
- Dependiendo del caso, el resultado son aplicaciones nativas que aprovechan en gran medida el potencial del dispositivo y ofrecen un diseño y experiencia de usuario idéntica a las aplicaciones desarrolladas sólo para una plataforma.

Sin embargo, también existen aspectos negativos, por ejemplo:

- Soporte parcial. No tienen soporte absoluto para todas las plataformas y versiones.
- Para poder soportar varios dispositivos, suelen tener que hacer el mínimo común denominador de las capacidades. Es decir, que si una plataforma aporta una nueva funcionalidad, no se podrá implementar sin añadir código específico.
- Las nuevas funcionalidades tardan en ser soportadas. Dado que las novedades en las plataformas aparecen a un ritmo muy elevado y en cortos períodos de tiempo.
- El rendimiento normalmente es menor que en aplicaciones desarrolladas de forma nativa.
- Se requiere pagar la licencia, si se tiene, para el entorno de desarrollo, además de la licencia propia de cada plataforma para poder distribuir la aplicación.

En cuanto a la parte del *back-end* se va a utilizar el cuarteto de desarrollo "full-stack" de *JavaScript* (es decir, *MEAN*: *MongoDB*, *Express*, *Angular 2* y *Node.js*), dado que permite utilizar *JavaScript* en el

[3] Al no disponer de un dispositivo *iOS*, esta parte la tendría que simular con el *IDE* de *Apple* para desarrollo de aplicaciones "XCode", indicando la descripción teórica del proceso de compilación y ejecución.

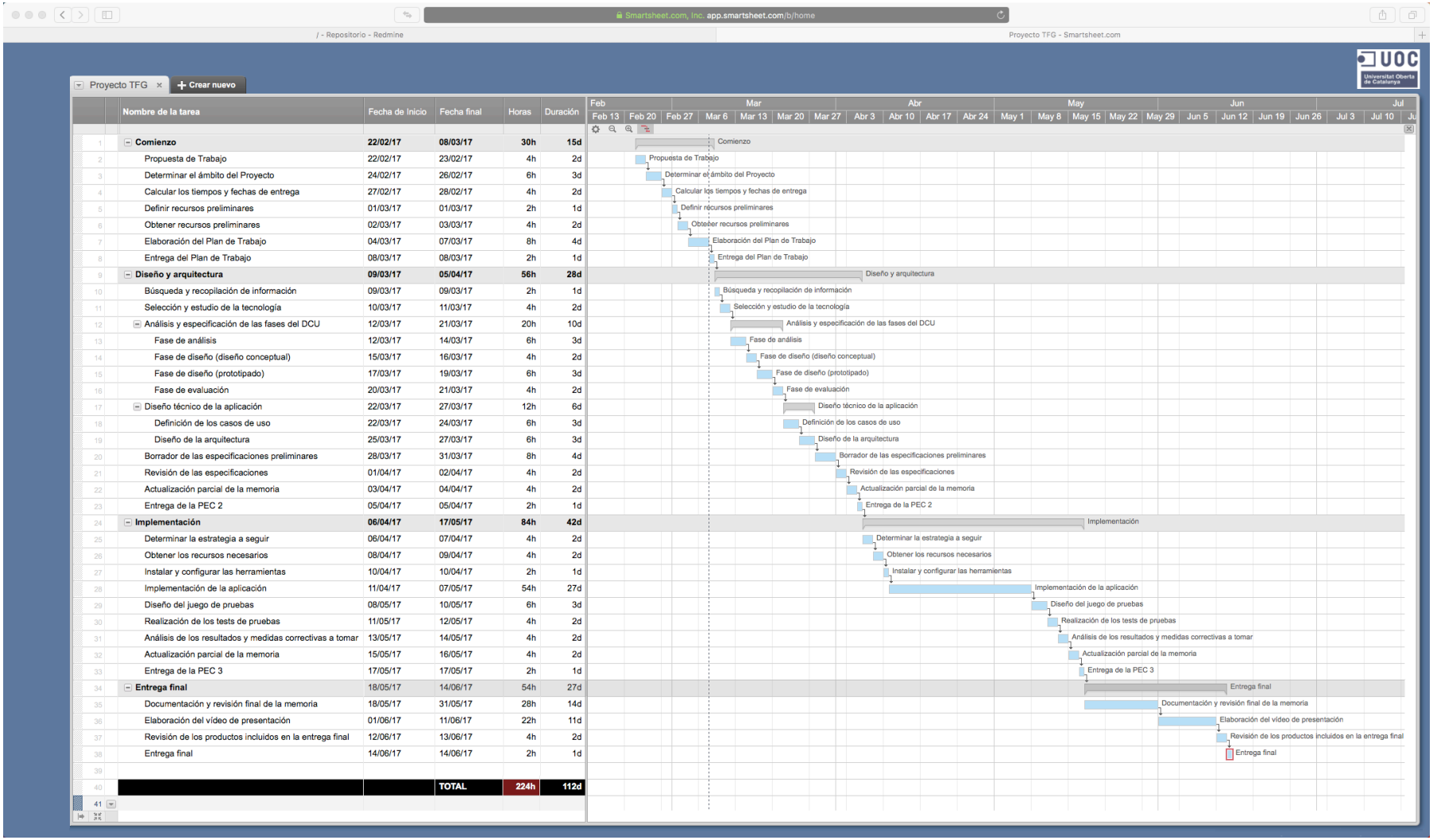
lado del servidor añadiendo nuevas funcionalidades y superando nuevos retos dentro de la programación *Web*.

Quizá se ha obtado por tecnología *Node.js*, dado que está centrada en la velocidad y en la escalabilidad. Es decir, permite crear aplicaciones que vayan a trabajar con muchos usuarios, gestionar datos de una red o dar respuesta a una necesidad en tiempo real.

1.4. PLANIFICACIÓN DEL TRABAJO

Debido a la naturaleza del trabajo a realizar, la planificación viene determinada por la fecha de inicio, que comienza el 22 de febrero de 2017, la entrega final del plan de trabajo, la entrega parcial de dos PEC (Pruebas de Evaluación Continua) de que consta la asignatura, y de una entrega final de carácter esencial e improrrogable, que tiene como fecha de cierre el 14 de junio de 2017.

Por tanto, la estimación se impone, dado que las fechas de entrega se imponen también. En este sentido, se ha diseñado la solución buscando adecuar el tiempo de realización de cada una de las entregas, para alcanzar los objetivos fijados y ajustar el perímetro del proyecto evitando el riesgo.



	Nombre de la tarea	Fecha de Inicio	Fecha final	Horas	Duración
1	**Comienzo**	**22/02/17**	**08/03/17**	**30h**	**15d**
2	Propuesta de Trabajo	22/02/17	23/02/17	4h	2d
3	Determinar el ámbito del Proyecto	24/02/17	26/02/17	6h	3d
4	Calcular los tiempos y fechas de entrega	27/02/17	28/02/17	4h	2d
5	Definir recursos preliminares	01/03/17	01/03/17	2h	1d
6	Obtener recursos preliminares	02/03/17	03/03/17	4h	2d
7	Elaboración del Plan de Trabajo	04/03/17	07/03/17	8h	4d
8	Entrega del Plan de Trabajo	08/03/17	08/03/17	2h	1d
9	**Diseño y arquitectura**	**09/03/17**	**05/04/17**	**56h**	**28d**
10	Búsqueda y recopilación de información	09/03/17	09/03/17	2h	1d
11	Selección y estudio de la tecnología	10/03/17	11/03/17	4h	2d
12	Análisis y especificación de las fases del DCU	12/03/17	21/03/17	20h	10d
13	Fase de análisis	12/03/17	14/03/17	6h	3d
14	Fase de diseño (diseño conceptual)	15/03/17	16/03/17	4h	2d
15	Fase de diseño (prototipado)	17/03/17	19/03/17	6h	3d
16	Fase de evaluación	20/03/17	21/03/17	4h	2d
17	Diseño técnico de la aplicación	22/03/17	27/03/17	12h	6d
18	Definición de los casos de uso	22/03/17	24/03/17	6h	3d
19	Diseño de la arquitectura	25/03/17	27/03/17	6h	3d
20	Borrador de las especificaciones preliminares	28/03/17	31/03/17	8h	4d
21	Revisión de las especificaciones	01/04/17	02/04/17	4h	2d
22	Actualización parcial de la memoria	03/04/17	04/04/17	4h	2d
23	Entrega de la PEC 2	05/04/17	05/04/17	2h	1d
24	**Implementación**	**06/04/17**	**17/05/17**	**84h**	**42d**
25	Determinar la estrategia a seguir	06/04/17	07/04/17	4h	2d
26	Obtener los recursos necesarios	08/04/17	09/04/17	4h	2d
27	Instalar y configurar las herramientas	10/04/17	10/04/17	2h	1d
28	Implementación de la aplicación	11/04/17	07/05/17	54h	27d
29	Diseño del juego de pruebas	08/05/17	10/05/17	6h	3d
30	Realización de los tests de pruebas	11/05/17	12/05/17	4h	2d
31	Análisis de los resultados y medidas correctivas a tomar	13/05/17	14/05/17	4h	2d
32	Actualización parcial de la memoria	15/05/17	16/05/17	4h	2d
33	Entrega de la PEC 3	17/05/17	17/05/17	2h	1d
34	**Entrega final**	18/05/17	14/06/17	54h	27d
35	Documentación y revisión final de la memoria	18/05/17	31/05/17	28h	14d
36	Elaboración del vídeo de presentación	01/06/17	11/06/17	22h	11d
37	Revisión de los productos incluidos en la entrega final	12/06/17	13/06/17	4h	2d
38	Entrega final	14/06/17	14/06/17	2h	1d
39					
40			TOTAL	224h	112d
41					

Figura 1: planificación del Proyecto

1.5. BREVE SUMARIO DE PRODUCTOS OBTENIDOS

Para terminar el Trabajo Final de Grado, hay que concretar la generación de un conjunto de piezas intermedias. Estas piezas son los entregables parciales, que son las piezas que forman el "resultado" del proyecto. Entre los productos a entregar se encuentran:

- El plan de trabajo (PEC 1).
- Un entregable correspondiente a la segunda prueba de evaluación continua (PEC 2).
- El entregable de la PEC 3.

Finalmente, entre los productos incluidos en la entrega final figuran los siguientes:

- Una memoria en formato *PDF* describiendo el trabajo realizado y las decisiones tomadas.
- El código fuente de las dos aplicaciones, junto con las instrucciones de compilación. Es decir, la aplicación móvil para *smartphones Android* desarrollada y empaquetada, y el *back-end* utilizando tecnología *MEAN* (*MongoDB*, *Express.js*, *Angular 2* y *Node*).
- Una presentación en vídeo que sintetiza los objetivos del trabajo, el proceso de diseño y desarrollo y los resultados conseguidos.

1.6. BREVE DESCRIPCIÓN DEL RESTO DE CAPÍTULOS DE LA MEMORIA

El TFG divide la memoria en doce capítulos ordenados cronológicamente, la cual presenta la siguiente estructura:

Capítulo 1 - Introducción. En este capítulo se hace una descripción de los siguientes aspectos: el contexto y justificación del trabajo; los objetivos del proyecto; el enfoque y método elegido; las tareas a realizar y una planificación temporal de cada tarea mediante un diagrama de *Gantt*; un breve sumario de los entregables finales; y finalmente, una explicación de los contenidos de cada capítulo y su relación con el proyecto global.

Capítulo 2 – Introducción a la arquitectura de los sitemas. En este segundo capítulo se va a realizar un estudio de las arquitecturas de aplicación y de tecnologías móviles existentes.

Por una parte, se describen las arquitecturas de aplicación y se presentan los principales elementos de estas arquitecturas.

Por otra parte, se realiza una descripción detallada de la arquitectura interna de los principales sistemas operativos existentes en la actualidad, como son: *Android*, *iOS*, *Windows Mobile* y *BlackBerry.*

Capítulo 3 - DCU y diseño técnico. En este capítulo se aplica la metodología del diseño centrado en el usuario en la aplicación definida en el Plan de Trabajo. Este capítulo está formado por cuatro partes, que siguen las fases del DCU: usuarios y contexto de uso (fase de análisis), diseño conceptual (fase de diseño), prototipado (fase de diseño) y evaluación del mismo (fase de evaluación).

A parte de realizar las fases del DCU, en este capítulo también se realiza el diseño técnico de la aplicación. En concreto, se trabajan dos aspectos del diseño técnico: la definición de los casos de uso y el diseño de la arquitectura.

Capítulo 4 - Herramientas. Este capítulo se dedica a investigar acerca de los *frameworks*, librerías y *APIs* más adecuados para la problemática a resolver y de las herramientas necesarias para la realización del proyecto y cualquier otra necesidad que vaya surgiendo.

Capítulo 5 - Implementación. En el quinto capítulo se realizará la implementación del Trabajo Final según el diseño definido en etapas anteriores. El objetivo es construir tanto la aplicación móvil, como el *back-end*, para lograr los objetivos especificados en el proyecto.

En cualquier caso, habrá que disponer de un prototipo o versión beta de la aplicación robusto y mínimamente testeado. De este modo aseguramos que somos capaces de completarlo en la entrega final y completar el Trabajo Final con éxito.

Capítulo 6 - Pruebas. En este capítulo se hace una descripción breve sobre cómo está previsto probar la aplicación (sobre simulador o dispositivo), el *framework* de *test* que se utilizará y qué tipo de pruebas se realizarán (unitarias, integración, carga, etc...).

Además de lo anterior, se va a entregar junto con la aplicación como mínimo una prueba unitaria de las funcionalidades desarrolladas en la etapa de desarrollo.

Capítulo 7 - Estado del proyecto. El séptimo capítulo trata de analizar brevemente el estado del proyecto en relación a la planificación propuesta, identificando posibles desviaciones y las medidas correctivas a tomar.

Capítulo 8 - Conclusiones y trabajo futuro. En este capítulo se analizará el trabajo realizado, las dificultades encontradas, los resultados conseguidos, las posibles ampliaciones del trabajo, etc.

Capítulo 9 - Recomendaciones. El noveno capítulo se centra esencialmente en explicar cómo migrar un proyecto de *Ionic 2* a *Ionic 3*.

Capítulo 10 - Referencias. Bibliografía, *webgrafía* y otras fuentes de información.

Capítulo 11 - Glosario. Este penúltimo capítulo se centra en el glosario, el cual está concebido a modo de complemento, para ayudar a comprender los tecnicismos utilizados en este trabajo.

Capítulo 12 - Anexos. Este capítulo podría incluir toda la documentación adicional de la memoria si fuese necesario.

2. INTRODUCCIÓN A LA ARQUITECTURA DE LOS SISTEMAS

Antes de empezar a trabajar y empezar a escribir la aplicación para el dispositivo móvil y el *back-end*, es necesario tener una visión general de las arquitecturas existentes, para seleccionar aquellas que mejor se adapten a nuestra solución.

2.1. ARQUITECTURAS DE APLICACIÓN

En general, todas las aplicaciones se estructuran según las siguientes arquitecturas:

- Cliente-Servidor: uno de los terminales (servidor) ofrece servicios que son consumidos por diferentes terminales (clientes).

- *P2P* (*peer-to-peer*): todos los terminales de la aplicación ofrecen y consumen servicios.

Los requisitos de los sistemas modernos como escalabilidad, tolerancia a fallos, redundancia, etc... Han resultado en arquitecturas de aplicación mucho más complejas.

En la práctica, un sistema puede ser dividido en varios componentes que interaccionan entre ellos utilizando las dos arquitecturas antes mencionadas.

A continuación se presentan los principales elementos de estas arquitecturas.

CLIENTES – *HARDWARE*

Los clientes consumen los servicios ofrecidos en la red a través de aplicaciones específicas o el navegador.

Desde el punto de vista del *hardware* se pueden diferenciar los siguientes clientes:

- **Estaciones de trabajo**: acceden desde redes cableadas. Equipos completamente funcionales con alta posibilidad de configuración y personalización. Típicamente usados en empresas, negocios y entornos corporativos. En casas cada vez más se ven reemplazados por portátiles.

- **Portátiles**: igual que la estación de trabajo, pero puede conectarse a la aplicación a través de redes no confiables.

- **Móviles**: similares en capacidades a los portátiles, pero con la posibilidad de personalización y capacidades más limitadas.

- **Dispositivos *IoT*** (*Internet* de las Cosas): cuentan con capacidades de comunicación muy limitadas. Generalmente acceden a los servicios a través de dispositivos de más capacidad a los que son conectados (teléfono móvil o estación de trabajo).

- **Servicio de *back-end***: aplicación de servidor que necesita consumir datos vía *web services* u otros protocolos.

CLIENTES – *SOFTWARE*

Los clientes móviles consumen los servicios ofrecidos por la red utilizando:

- Navegadores *web*: se accede a una aplicación *web* utilizando el protocolo *HTTP*. La aplicación *web* se crea con *HTML5*/*JavaScript*/*CSS*.

- Aplicaciones específicas: se desarrolla una aplicación, necesaria una plataforma, para acceder a la *web*. La comunicación con el servidor se puede realizar utilizando una *API* (*Application Programming Interface*) del tipo:

    - *HTTP* (generalmente a través de servicios *REST* o *SOAP*).

    - Protocolos específicos de la aplicación.

- Para evitar problemas de seguridad, se recomienda que todos los clientes accedan al servicio a través de la misma *API*. De esta forma se desacopla la lógica de negocio de las capas de presentación existente para cada plataforma.

    - Permite aislar los diferentes problemas de seguridad que puedan darse en la plataforma (cada tipo de cliente, incluidos el *web* y el *back-end*, por otro) independientemente del tipo de cliente que acceda al servicio.

SERVIDORES

Un servidor es una entidad que espera, procesa y responde a las peticiones de los clientes.

Dependiendo del tipo de aplicación, existen distintos tipos de servidores, entre otros:

- Servidor de base de datos: para poder trabajar con gran cantidad de información.

- Servidor de aplicaciones: para ofrecer servicios a través de distintas aplicaciones (generalmente *web*).

- Servidor de correo: para el envío y recepción de correo.

Un servicio se puede desplegar principalmente de tres maneras:

- Utilizando un *hardware* e infraestructura propia. En estos casos, el servidor puede ser:

    - Dedicado: si la aplicación no comparte el *hardware* con ningún otro servicio ofrecido.

    - Compartido: si coexisten varias aplicaciones en un mismo servidor. Generalmente, a través de un servidor de aplicaciones instalado en la propia máquina.

- Utilizando servicios en la nube. Estos servicios se basan principalmente en tecnologías de virtualización para emular la existencia de varios servidores en una única máquina física de grandes prestaciones.

- Utilizando una arquitectura híbrida que mezcla infraestructura propia y servicios en la nube.

SERVICIOS EN LA NUBE

Los servicios en la nube reducen el coste en infraestructura, *hardware* y mantenimiento mediante la utilización de *hardware* y servicios de terceros.

La utilización de servicios en la nube debe estudiarse con detenimiento, ya que tienen otras implicaciones como la pérdida del control de los datos, su almacenamiento en servidores externos o la exposisición a riesgos como el cese de actividad de la compañía.

Existen diferentes tipos de servicios:

- Infraestructura como servicio (*IaaS*): ofrece infraestructura, generalmente a través de virtualización, para construir los servicios. Ejemplo: *Amazon EC2*.

- Plataforma como servicio (*PaaS*): añade a la infraestructura un entorno para la ejecución de aplicaciones y bases de datos ya configurados. Ejemplo: *Google App Engine*.

- *Software* como servicio (*SaaS*): ofrece directamente un servicio en particular. Algunos ejemplos son: *Microsoft Office 365*, *Dropbox*, etc...

Desde el punto de vista de la seguridad en dispositivos móviles, son de especial interés aquellos que permiten el desarrollo de *back-end* de forma

sencilla para aplicaciones móviles. Estos tipos de *PaaS* se llaman *Mobile back-end as a service* (*BaaS* o *MbaaS*).

Este tipo de servicios eliminan las tareas de administración y actualización y simplifican el desarrollo ofreciendo librerías con componentes previamente auditados.

MOBILE BACK-END AS A SERVICE

Un *Baas* o *MbaaS* ofrece al desarrollador de aplicaciones un entorno preconfigurado para poder desplegar una aplicación móvil de forma rápida.

Las tareas de administración se simplifican, por lo que el desarrollador de aplicaciones no necesita conocimientos sobre *back-end*.

Principales proveedores:

- ***App Engine*** (*Google*): una vez desarrollado el *back-end*, el *framework* genera librerías para incorporar aplicaciones *iOS* y *Android*. *Back-end* en *Java*, *Python*, *PHP*, *Node*.js o *Go*.

- ***Parse*** (*Facebook*): la mayoría de aplicaciones se realizan desde los clientes a través de las librerías de *Parse*. En el *back-end* se definen políticas de seguridad y tareas periódicas que se programan en *JavaScript*.

- ***Mobile Hub*** (*Amazon*): integración de varios servicios de *Amazon Web Services* para el despliegue de aplicaciones móviles.

Por último, indicar que los desarrolladores inexpertos pueden provocar problemas de seguridad en el lado del servidor.

2.2. ARQUITECTURAS DE SO MÓVILES

Actualmente, *iOS* y *Android* son las plataformas más importantes a nivel de cuota de mercado.

Otras plataformas con menos cuota en usuarios domésticos, pero muy utilizadas y valoradas hoy en día en el mundo empresarial son: *Windows Phone* y *BlackBerry*.

- *Windows Phone*, por su fácil integración con otras soluciones en el mundo empresarial, que están muy extendidas, y por estar fuertemente integrado con los servicios de *Microsoft*: *Office, Skype* y *Xbox*.

  En su última versión (la 10), ha perdido el apelativo “Phone”, para pasar a llamarse *Windows 10 Mobile*.

- *BlackBerry*, dado que fue la primera compañía en desarrollar teléfonos inteligentes con conexión permanente y tarifas competitivas para empresas.

  BlackBerry es la marca de la compañía canadiense, *Research in Motion* (*RIM*), que fabrica dispositivos con su propio sistema operativo.

  Sus principales clientes siguen siendo organizaciones en detrimento de usuarios domésticos.

  Actualmente tiene tres líneas de dispositivos con sistemas operativos diferentes: *BlackBerry OS*, *BlackBerry Playbook* y *BlackBerry 10*.

2.3. ARQUITECTURA INTERNA DEL SO *ANDROID*

INTRODUCCIÓN

En esta introducción se pretende ofrecer una versión general del sistema operativo *Android*.

En concreto, se aborda:

- La arquitectura del sistema operativo.
- Las librerías disponibles.
- El sistema de ficheros y particiones existentes.
- La estructura de un binario.
- Los componentes de una aplicación.
- La comunicación entre aplicaciones.
- El proceso de compilación y creación de una aplicación.

Para obtener más información sobre el sistema operativo *Android* se puede visitar:

http://source.android.com

ESPECIFICACIONES TÉCNICAS

Arquitecturas válidas *ARM* 32 y 64 *bits*, *x86* 32 y 64 *bits*, *MIPS* y *NEON*.

Kernel	*Kernel* de *Linux 3.X* dependiendo de la versión de *Android* y el dispositivo.
Sistema de ficheros	Soporta varios, pero principalmente utiliza *EXT4* o *JFFS2* para el sistema de ficheros internos. Acepta tarjetas de memoria formateadas en *FAT32*. Soporta cifrado de disco desde la versión *4.3*.
Ejecutables	*Bytecode*. Ejecutable por la máquina *Dalvik* o por el más reciente *Android Runtime* (*ART*).
Plataforma del sistema	Basado en *Linux*.
Licencia	Licencia *Apache 2.0*. Los fabricantes modifican el código del sistema y lo adaptan a los dispositivos con libertad dentro de unos parámetros mínimos comunes.

ARQUITECTURA DEL SO

El sistema operativo se organiza en las siguientes capas:

- **Aplicaciones**: aquellas instaladas por el usuario o pre-instaladas en el sistema.
- ***Aplication Framework***: ofrecen servicios a las aplicaciones. Desarrollados en *Java*.
- **Librerías**: módulos que ofrecen servicios al *application framework*. Desarrollados en *C* y compilados a código nativo en cada plataforma.
- ***Android Runtime***: cada aplicación ejecuta su propia instancia de una máquina virtual de *Java*, específica para *Android*. Las aplicaciones nativas se ejecutan directamente.
- ***Kernel***: ejecuta código dependiente del dispositivo (*ARM*, *x86*, *MIPS*, etc.). Ofrece servicios de seguridad a las capas superiores.

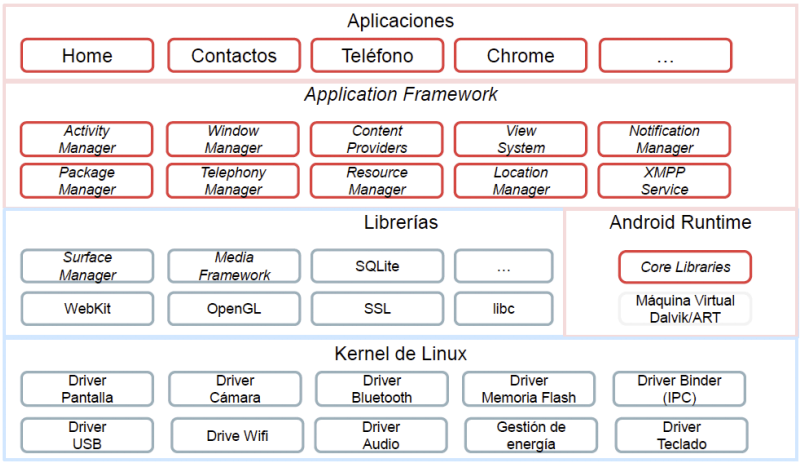


Figura 2: arquitectura del *SO Android*

LIBRERÍAS DISPONIBLES

Las aplicaciones acceden a las librerías a través del *Application Framework*.

- ***Package Manager***: controla la instalación de paquetes.
- ***Activity Manager***: gestiona las actividades sobre la localización del dispositivo.
- ***Location Manager***: ofrece información sobre la localización del dispositivo.
- ***Notification Manager***: permite gestionar las notificaciones recibidas por una aplicación.
- ***Content Providers***: ofrece acceso a datos almacenados por aplicaciones en bases de datos.
- ***View System***: controla las diferentes vistas que se muestran al usuario.
- ***Bluetooth API***: controla las conexiones *Bluetooth* del dispositivo.

Para más información sobre librerías, se puede visitar:

http://developer.android.com/guide/index.html

SISTEMAS DE FICHEROS

Android, por defecto, utiliza múltiples particiones, generalmente formateadas en *Journal Flash File System 2* (*JFFS2*[4]).

Principales directorios:

- ***/system***: directorio donde se guarda la *ROM* del SO (solo lectura).
- ***/proc***: información de los procesos en ejecución.
- ***/mnt***: punto de montaje para otros tipos de almacenamiento.
- ***/sdcard***: redirige a */mnt/sdcard*, punto de montaje de la tarjeta *SD*.
- ***/cache***: guarda la *cache* de datos de las aplicaciones y el sistema.
- ***/data***: directorio en el que se almacenan las apliaciones.

Hay que tener en cuenta, que cada aplicación es guardada en un directorio al que solo ella tiene permisos de acceso.

COMPONENTES DE UNA APLICACIÓN - *ACTIVITY*

Las actividades presentan las siguientes características:

- Constituyen la capa de presentación de la aplicación.
- Ofrecen la interfaz visible de la aplicación.
- Ocupa toda la pantalla (para mostrar elementos de información que no ocupan toda la pantalla se utilizan los *Fragments*).
- Cada actividad contiene una jerarquía de vistas (*Views*) con las que el usuario puede interactuar.
- Cada aplicación puede tener una Actividad principal que será la que lanzará cuando se toque el icono de la aplicación en la pantalla de *apps*.
- Cada actividad de la aplicación tiene un ciclo de vida como el mostrado a continuación.

[4] Cualquier fabricante puede utilizar otro sistema de ficheros o modificar la estructura de ficheros o particiones.

CICLO DE VIDA DE UNA ACTIVIDAD

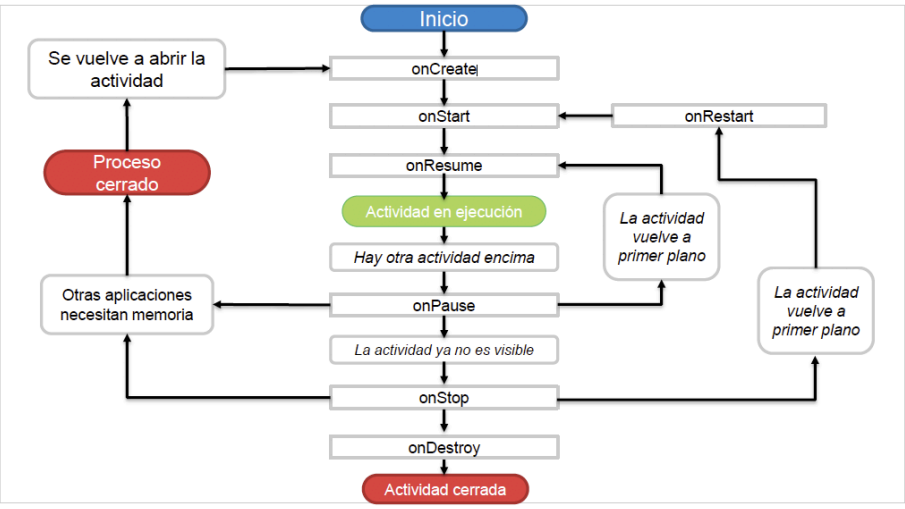


Figura 3: ciclo de vida de una actividad

COMPONENTES DE UNA APLICACIÓN - SERVICE

Los servicios son los encargados de realizar operaciones en el *background*. De hecho, cuando una aplicación deja de estar en primer plano puede seguir ejecutándose a través de servicios.

Generalmente, son utilizados para realizar cómputos, guardar o recibir datos de *internet* o disco.

Una de las formas de ejecutar tareas de servicios es a través de la utilización de *IntentServices*:

- Por cada *IntentService* el sistema crea un único *thread* separado del de la interfaz de usuario.
- Cada tarea enviada al *IntentService* (por medio de un *Intent*) es ejecutada en el thread (no hay ejecución concurrente de varios *Intents*).

COMPONENTES DE UNA APLICACIÓN – *CONTENT PROVIDER*

Los *Content Providers* están diseñados para compartir datos estructurados entre aplicaciones.

Ofrecen un interfaz para que otras aplicaciones puedan acceder a los datos almacenados por la aplicación.

Las aplicaciones que quieren compartir datos con el resto deben contra con sus propios *providers*. Por ese motivo, una buena práctica de seguridad

consiste en requerir permisos para restringir las aplicaciones que pueden acceder a los permisos declarados por la aplicación.

El sistema operativo ofrece por defecto una serie de *Providers* para el acceso a datos del sistema, como los contactos, listado de llamadas y *SMS*. Para acceder a ellos, la aplicación tiene que declarar los permisos correspondientes.

COMPONENTES DE UNA APLICACIÓN – *BROADCAST RECEIVER*

Los *Broadcast Receivers* son tareas que se ejecutan cuando llegan mensajes (*Intents*) generados por otros componentes de la aplicación o por otras aplicaciones.

Las aplicaciones pueden crear *Intents* para enviar mensajes a actividades, servicios o *Broadcast Receivers*.

Cada *Broadcast Receiver* es configurado para escuchar un tipo específico de *Intents*.

Los componentes de una aplicación pueden ser:

- Públicos: otras aplicaciones pueden interactuar con ellos.
- Privados: solo componentes de la misma aplicación (mismo *user ID*) pueden interactuar con ellos.

Todos los componentes dentro de una aplicación se ejecutan dentro del mismo proceso, a no ser que el desarrollador especifique lo contrario.

El desarrollador puede definir restricciones para el paso de mensajes a través de permisos:

- Los que envíen mensajes pueden requerir a las aplicaciones que lo reciban de cierto permiso.
- Los que reciban mensajes pueden aceptar mensajes solo de aplicaciones con ciertos permisos.

COMUNICACIÓN ENTRE APLICACIONES

Las aplicaciones en *Android* se pueden comunicar por cualquier mecanismo de comunicación entre procesos heredado de *UNIX*:

- Sistema de ficheros, *sockets*, etc.

- Los permisos de la aplicación siempre se aplican antes de la comunicación.

Además, *Android* ofrece dos mecanismos adicionales de comunicación entre aplicaciones:

- ***Binder***: sistema *RPC* (*Remote Procedure Call*) implementado sobre *Linux* con un *driver* específico. Las aplicaciones lo utilizan a través de objetos *Intent* que son enviados al sistema para lanzar Servicios, Actividades y *Broadcast Receivers*.
- ***Content Providers***: operaciones de lectura/escritura sobre bases de datos de otras aplicaciones.

COMUNICACIÓN ENTRE APLICACIONES - *INTENTS*

Un *Intent* es un objeto para el envío de mensajes que contiene:

- Información acerca de la operación que se quiere realizar, a través del parámetro *action* o *component*.
- Datos sobre los que se quiere realizar la operación, a través de extras o una *URI* de origen de los datos.
- Información adicional que puede ser de utilidad para el receptor.

Los *Intents* son enviados al sistema operativo que se encarga de entregarlos al receptor correspondiente, y no pueden ser enviados directamente.

Existen *Intents* de dos tipos:

- Explícitos: especifican el receptor a través del nombre de componente.
- Implícitos: indican una acción genérica a realizar (ejemplo: mandar mensaje).

EL FICHERO *ANDROIDMANIFEST*

El *manifest* de una aplicación contiene toda la información necesaria para la instalación y ejecución de la app por parte de *Android*.

- Versión mínima de *Android* con la que la aplicación es compatible.
- Nombre del paquete de la aplicación y versión.
- Componentes incluidos.

    - Define las actividades, servicios y broadcast receivers que utiliza la apliación. Los últimos también pueden declararse dinámicamente durante la ejecución de la aplicación.
- Permisos que la aplicación solicita al usuario para ejecutarse.
- Antes de *Android 6.0*.
- Capacidades del dispositivo necesarias para ejecutar la aplicación.
- Ejemplos: cámara, acelerómetro, etc.
- Librerías del sistema que necesita cargar la aplicación para funcionar.

ESTRUCTURA DE UN *APK*

Las aplicaciones *Android* se empaquetan en ficheros *APK* (*zip*), los cuales contienen los siguientes elementos:

- ***META-INF***: directorio que contiene un listado de ficheros (*MANIFEST.MF*). El certificado de la *app* (*CERT.RSA*), una lista de recursos de la *app* y *hashes SHA*-1 de los ficheros indicados en el listado.
- ***Lib***: directorio con el código nativo utilizado por la *app* para plataformas específicas (*ARM*, *x86*, *MIPS*).
- ***Assets***: directorio con diferentes elementos utilizados por la *app*.
- ***Res***: directorio con recursos utilizados por la *app* (iconos, imágenes, constantes, etc.).
- ***Resources.arsc***: ficheros *XML* pre-compilados con definiciones del interfaz de usuario.
- ***Classes.dex***: código compilado de la aplicación para el *runtime* de *Android*.
- ***AndroidManifest.xml***: fichero de información de la aplicación.

CREACIÓN DE FICHEROS *APK*

Los diferentes ficheros fuente de *java* son compilados en un fichero *JAR* (*Java archive*).

Este formato combina múltiples ficheros *.class* compilados en *bytecode* en un fichero, utilizando la compresión *zip*.

El *bytecode* es transformado al formato “Ejecutable Dalvik” para su ejecución en *Android* (el *runtime* introducido en las últimas versiones de *Android*, es ejecutado por *Dalvik* y *ART*).

El fichero *dex* resultante es comprimido en un fichero *zip* junto con los recursos, *assets* y *manifest* de la aplicación.

Si la aplicación se quiere publicar en *Google Play* se firma con un certificado (también puede ser autofirmado, pero todas las versiones de la *app* deben ser firmadas con el mismo).

2.4. ARQUITECTURA INTERNA DEL SO *IOS*

INTRODUCCIÓN

En esta introducción se pretende ofrecer una visión general del sistema operativo *iOS* de *Apple*.

En concreto, se aboradará:

- La arquitectura del SO.
- Las librerías disponibles.
- El sistema de ficheros y particiones existentes.
- La estructura de un binario.
- Los componentes de una aplicación.
- La comunicación entre aplicaciones.
- El proceso de compilación y creación de una aplicación.

Para obtener más información sobre el sistema operativo se puede visitar:

https://developer.apple.com/ios

ESPECIFICACIONES TÉCNICAS

Arquitecturas válidas	*ARM* 32 y 64 *bits*.
Kernel	*XNU.* Basado en el *Kernel Darwin* de *OS X*.
Sistema de ficheros	HFSX. Basado en HFS+ de iOS. Sin almacenamiento extraíble y con cifrado de disco por defecto.
Ejecutables	Formato *Match-O*, el mismo que el utilizado para aplicaciones de *OS X*.
Plataforma del sistema	Basado en *BSD*, igual que *OS X*.
Licencia	Propietario. No se permite su instalación en equipos que no

sean manufacturados por *Apple*. Su licencia prohíbe la ingeniería inversa del sistema.

ARQUITECTURA DEL SISTEMA OPERATIVO

El sistema operativo se organiza en las siguientes capas:

- **Aplicaciones**: incluye las aplicaciones de terceros y las pre-instaladas por *Apple*. Las últimas tienen acceso a un conjunto de *APIs* adicional.
- ***Cocoa-Touch***: ofrecen la infraestructura básica a todas las aplicaciones (interfaz, *multi-touch*, multitarea, notificaciones, etc.).
- **Media**: contiene las librerías de bajo nivel que se encargan de mostrar gráficos, audio y vídeo.
- ***Core Services***: ofrece diferentes servicios del sistema a las aplicaciones (redes, localización, *iCloud*, etc.).
- ***Core OS***: librerías de bajo nivel que son utilizadas por el resto de capas superiores para la ejecución de tareas. Todos los *frameworks* llaman a estas librerías para interactuar con el núcleo del sistema operativo.

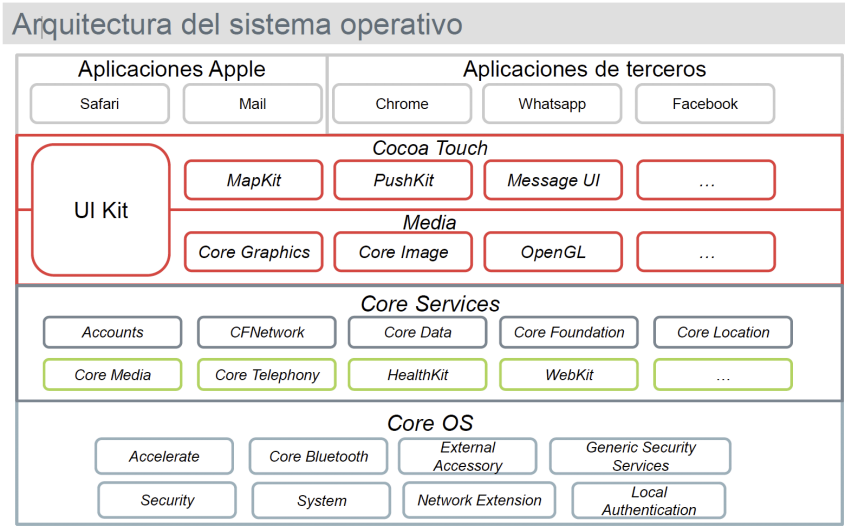


Figura 4: arquitectura del *SO iOS*

LIBRERÍAS DISPONIBLES

iOS ofrece multitud de librerías por defecto para la creación de aplicaciones:

- *UIKit*: se encarga de gran parte de la gestión de las aplicaciones. Algunos ejemplos son: ciclo de vida, creación del interfaz de usuario, multi-*touch*, cámara, etc.
- *Core Graphics*, *Animation*, *Image*, *OpenGL* y *Metal*: ofrecen motores para el dibujado de gráficos en *2D* y *3D*.
- *CFNetwork*: conjunto de librerías para la creación de conexiones de red (*HTTP*, *SSL*, *DNS*, *FTP*, etc.).
- *Core Data*: ofrece persistencia a la aplicación a través de la creación de un modelo de datos.
- *Core Motion*: permite tratar la información recibida por los sensores del dispositivo.
- *Core Location*: permite la utilización del *GPS* del dispositivo por parte de las aplicaciones.

SISTEMA DE FICHEROS

El SO *iOS* utiliza el sistema de Ficheros *HFS+* desarrollado por *Apple.* Cada aplicación se almacena en su propio directorio y no puede acceder a la información de otras aplicaciones.

La estructura de directorios está heredada de *UNIX* con ciertas modificaciones:

- ***/Applications:*** directorio en el que se encuentran instaladas las aplicaciones oficiales de *Apple*.
- ***/private/var:*** contiene la partición con todos los datos del usuario.
- ***/private/var/mobile/Applications:*** directorio en el que se instalan las aplicaciones de terceros.
- ***/boot:*** directorio en el que se guardan los ficheros de actualización.
- ***/private/etc:*** contiene ficheros de configuración del sistema operativo.
- ***/Library:*** contiene las librerías del sistema utilizadas por las aplicaciones.
- ***/Developer:*** utilizado para librerías de desarrollo instaladas por *Xcode*.

SISTEMA DE FICHEROS DE UNA APLICACIÓN

Cada aplicación se instala en un directorio con un *id* único generado en tiempo de instalación.

El directorio *Documents* contiene todos los ficheros generados por la aplicación.

El binario de la aplicación se localiza en una carpeta terminada en *app* en */Containers/Bundle/Application*.

Los otros directorios almacenan preferencias y datos temporales.

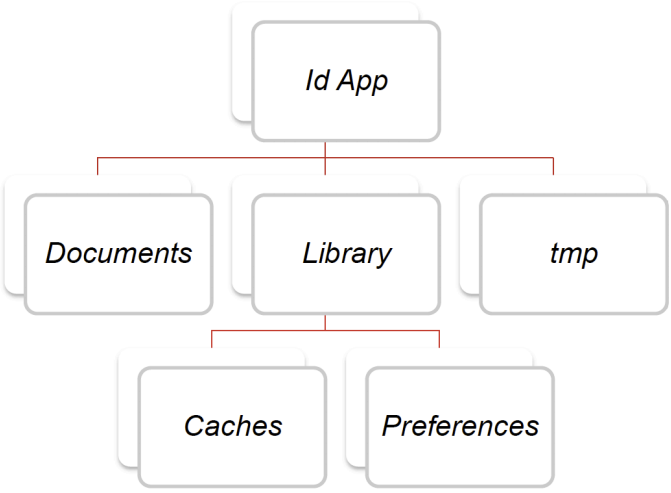


Figura 5: sistema de ficheros de una aplicación *iOS*

COMPONENTES DE UNA APLICACIÓN

Las aplicaciones en *iOS* están compuestas por una serie de elementos comunes:

- ***AppDelegate***: es el encargado de mediar entre el sistema operativo y la aplicación. La mayoría de eventos importantes que pueden interferir en el uso de una *app* (ej.: llamada telefónica) son manejados por él.
- ***ViewControllers***: son los componentes principales de la aplicación. Cada aplicación debe tener al menos uno, aunque suelen tener varios. Un *ViewController* maneja la porción del interfaz mostrado al usuario y gestiona la interacción entre la interfaz de usuario y los datos.
- ***Storyboards***: definen las interfaces de usuario y transiciones entre las diferentes pantallas de la aplicación, aunque también se pueden definir de forma programática.

CICLO DE VIDA DE UN CONTROLADOR

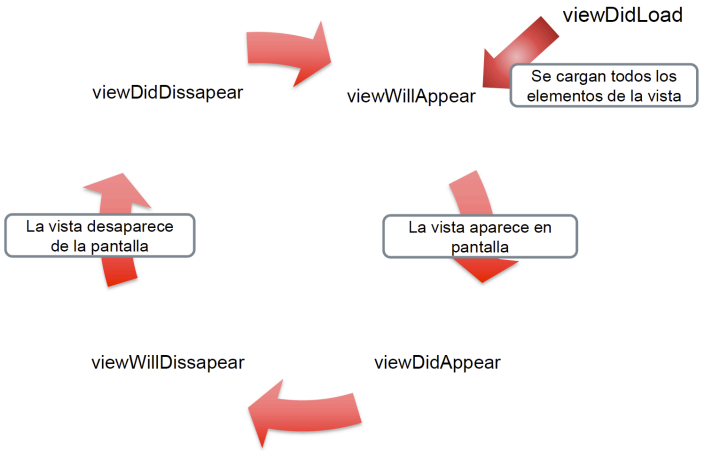


Figura 6: ciclo de vida de un controlador 1

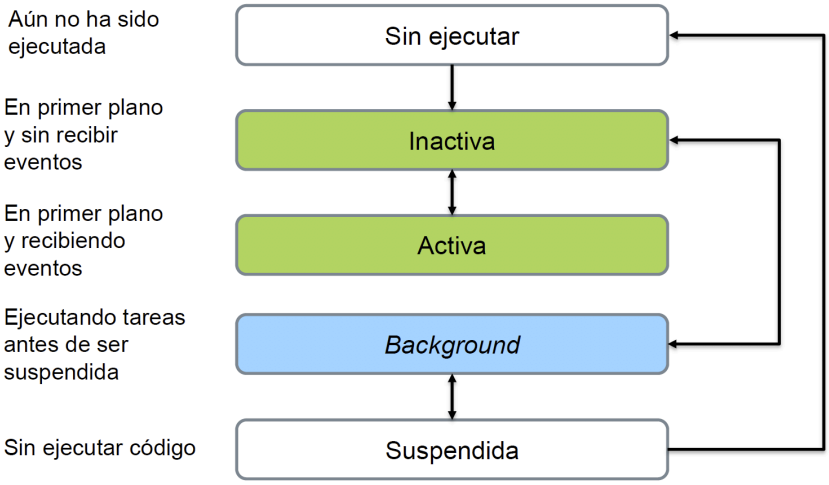


Figura 7: ciclo de vida de un controlador 2

COMUNICACIÓN ENTRE APLICACIONES

La comunicación entre aplicaciones en *iOS* siempre se realiza con la mediación del sistema operativo. El usuario debe iniciar o autorizar la comunicación entre dos aplicaciones.

Existen cuatro opciones principales:

- ***AirDrop***: solo permite la comunicación con aplicaciones en otros dispositivos, no con aplicaciones dentro del mismo dispositivo.
- **Esquemas *URL***: las aplicaciones pueden registrar tipos de *URL* que son capaces de abrir (*SMS*, *Phone*, *HTTP*, *Spotify*, etc.).
- ***App Extensions***: permite extender ciertas funcionalidades del sistema operativo con una aplicación. Sus categorías están restringidas

(almacenamiento en la nube, notificaciones, compartir, etc.) y la interacción se realiza a través de *APIs* específicas.

- ***Document Interaction API***: la aplicación define los documentos que es capaz de abrir. Si la aplicación es seleccionada para abrir el fichero, recibe una *URL* para leerlo.

ESTRUCTURA DE UN BINARIO

Los binarios descargados de la *App Store* de *Apple* son paquetes comprimidos en *Zip* con la extensión *IPA*.

El fichero *IPA* contiene, entre otros, los siguientes directorios y ficheros:

- ***/Payload/Application.app***: directorio que incluye:
  - Aplicación compilada: cifrada con *DRM*, por lo que no se puede acceder directamente al código compilado. Es necesario extraerlo de un dispositivo con *jailbreak*.
  - Recursos utilizados por la *app*: imágenes, vídeo, audio, etc.
  - *Info.plist*: fichero con información de configuración para la instalación y ejecución de la aplicación.
  - *_CodeSignature*: firma de la aplicación realizada con el certificado de desarrollador emitido por *Apple*.
- ***/ItunesArtwork***: contiene las imágenes utilizadas para los iconos de la aplicación.
- ***/ItunesMetadata.plist***: contiene una ficha con información relevante de la app para su instalación y ejecución: *Bundle ID*, *id* del desarrollador, etc.

CREACIÓN DE FICHEROS BINARIOS

Para crear un fichero binario distribuible a través de la *App Store* es necesario disponer de un certificado emitido por *Apple*. Desde *iOS 9*, la instalación de aplicaciones en dispositivos personales a través de *Xcode* no necesita de certificado.

Xcode se encarga de compilar la aplicación a la arquitectura *ARM*, firmar el código y subir el binario a *Apple* (si el certificado y *provisioning profile* del desarrollador y la aplicación están en regla).

Apple revisa la aplicación y protege el código compilado mediante *DRM*, si aprueba la aplicación.

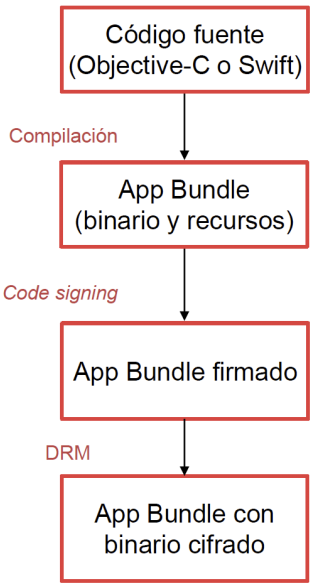


Figura 8: creación de ficheros binarios

2.5. *WINDOWS* Y *BLACKBERRY*

ESPECIFICACIONES TÉCNICAS – *WINDOWS 10*

Arquitecturas válidas	*ARM 32* y *64 bits*, *X86* y *X64* y otras arquitecturas *IoT*.
Kernel	*Windows One Core* para todas las plataformas.
Sistema de ficheros	*NFTS* por defecto. *FAT* en tarjetas *SD*.
Ejecutables	*Universal Windows Apps* (*C#, C*, *C++, JavaScript*).
Plataforma del sistema	*Windows*. Específica para cada dispositivo en el que se instala (*Xbox*, móvil, escritorio, etc.).
Licencia	Propietario.

ARQUITECTURA – *WINDOWS 10*

Windows 10 trata de unificar la arquitectura de todas las plataformas de *Microsoft* en una sola plataforma integrada con:

- Un tipo único de *Kernel*.
- Un único sistema de archivos.
- Un modelo de aplicación único para todas las plataformas.

Windows 10 integra las siguientes plataformas:

- *Windows* para escritorio.
- *Windows Phone*.
- *Xbox One*.
- *Windows* para *IoT*.

De esta manera, una aplicación desarrollada para *Windows 10* podrá ser ejecutada en cualquier plataforma de *Microsoft* (*Universal App Platform*).

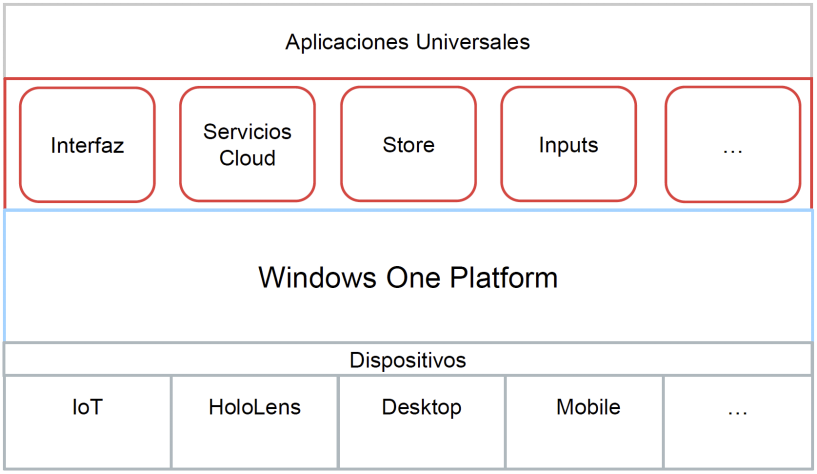


Figura 9: plataforma de *Microsoft* (*Universal App Platform*)

APLICACIONES – *WINDOWS 10*

Las aplicaciones de *Windows* se desarrollan en base a:

- Un código común a todas las plataformas que incluye los interfaces de usuario que se van a mostrar. Debido al diseño de los interfaces de *Windows*, el contenido de los mismos se puede adaptar de forma sencilla a los diferentes dispositivos que ejecutan las aplicaciones.
- Un código específico en forma de extensión por cada plataforma en la que se quiere ejecutar la aplicación.

El código común de una aplicación suele contener la lógica de negocio suficiente para ejecutar la aplicación en dispositivos móviles y escritorio.

Las extensiones permiten a la aplicación acceder a funcionalidad específica de la plataforma en la que se está ejecutando.

Las aplicaciones deben comprobar de forma activa que se están ejecutando en la plataforma antes de acceder a la funcionalidad de la extensión.

ESPECIFICACIONES TÉCNICAS – *BLACKBERRY*

Arquitecturas válidas	*ARM*.
Kernel	*RTOS Microkernel*.
Sistema de ficheros	*QMX* para la memoria interna. *FAT* en tarjetas *SD*.
Ejecutables	*C++, HTML5, Java* para aplicaciones portadas de *Android*.
Plataforma del sistema	*QMX OS* (tipo *UNIX*).
Licencia	Propietario.

ARQUITECTURA - *BLACKBERRY*

BlackBerry es un sistema operativo de micro-*kernel* (*QNX Neutrino RTOS*).

La mayoría de las funciones del sistema se implementan fuera del *kernel*.

El SO y los drivers del sistema utilizan el micro-*kernel* para acceder al *hardware* del dispositivo a bajo nivel.

Por encima del sistema operativo se sitúan los servicios y plataformas que ofrece el sistema operativo a cada plataforma de ejecución: *HTML5*, *Android* y *Cascades* (Nativas).

Cada aplicación es ejecutada en su plataforma específica de ejecución.

El sistema de ficheros de *BlackBerry* se divide en cuatro particiones:

- **Sistema operativo**: partición de solo lectura con el sistema operativo.
- ***Base File System***: contiene ficheros del sistema y también es de solo lectura.
- ***Work File System***: contiene las aplicaciones y datos del entorno de trabajo. Se cifra por defecto.
- ***Personal File System***: contiene las aplicaciones personales del usuario. No se cifra por defecto.

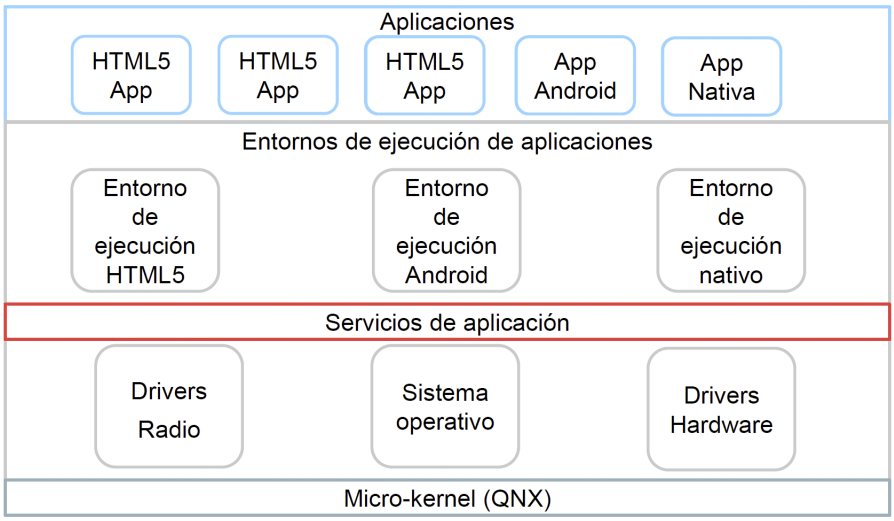


Figura 10: arquitectura de *BlackBerry*

3. DCU Y DISEÑO TÉCNICO

Al contrario de lo que muchas veces se suele argumentar, preguntar directamente a los usuarios ofrece información de gran valor a la hora de afrontar el diseño. El problema surge cuando se pregunta al usuario acerca de cuestiones para las que no tiene una respuesta concreta o cuando sus necesidades se tratan como hechos que no requieren de interpretación.

Por ese motivo, se ha evitado condicionar la repuesta del usuario por lo que cree debería responder o quiere ser oído por quien pregunta. Por tanto, como moderador me he mostrado en todo momento neutral y no he dirigido o condicionado las respuestas del entrevistado, para pretender descubrir información que pudiera orientarme en el diseño.

La razón es simple, cuando se le pregunta a un usuario acerca del porqué de un comportamiento o una decisión que ha tomado, normalmente tiende a racionalizarlo, y a buscar una causa, aunque la desconozca, a sus acciones pasadas.

En este caso, también hay que destacar que no son los usuarios quienes han tenido la responsabilidad ni la capacidad de decidir cómo debía ser el diseño. Sin embargo, sí que han aportado información de gran valor sobre sus necesidades, conocimientos, deseos, motivaciones, valores, satisfacción o experiencia.

Por tanto, el proceso de DCU (Diseño Centrado en el Usuario) propuesto en este trabajo se ha dividido en cuatro fases o etapas:

- Usuarios y contexto de uso.
- Diseño conceptual
- Prototipado.
- Evaluación.

3.1. USUARIOS Y CONTEXTO DE USO

MÉTODOS DE INDAGACIÓN

Uno de los métodos de investigación cualitativo elegidos para la fase de análisis ha sido la entrevista. En realidad, ha sido una variante de ésta, conocida como “focus group”.

La entrevista ha permitido trabajar con un grupo de usuarios compuesto por cuatro personas de distinto sexo y edad de forma conjunta, para buscar información de primera mano acerca de las experiencias, opiniones, actitudes o percepciones del entrevistado.

Si bien la entrevista normalmente se suele afrontar sobre la base de un guión estructurado, en este caso se ha flexibilizado, para permitir una exploración más profunda de los diferentes aspectos o cuestiones tratadas. La interacción entre los participantes ha ofrecido información adicional sobre problemas, experiencias o deseos compartidos.

Aunque las entrevistas también pueden realizarse de forma remota (teléfono, videoconferencia, etc …), en este caso se ha llevado a cabo en persona, para poder reconocer claramente lo que cada entrevistado contaba mediante gestos o expresiones.

El otro método de indagación seguido ha sido la encuesta. Este método ha permitido obtener información subjetiva de los usuarios, pero esta vez cuantitativa.

En este caso, se ha llevado a cabo de forma *online*. Se ha invitado a los usuarios de la entrevista anterior, a responder a una serie de preguntas estructuradas.

Las preguntas han versado sobre cuestiones demográficas (cómo son), tecnológicas (cómo acceden a *Internet*), de necesidades y hábitos (cómo y para qué utilizan *Internet*), competitivas (qué sitios *web* suelen visitar), de satisfacción (acerca del producto), de preferencias (qué les gusta de la aplicación y qué no), y de deseos (qué echan en falta).

En cuanto al diseño del cuestionario se han seguido los consejos de *Chris Gray*, donde las preguntas deben ser:

- Ordenadas de forma lógica.
- Fáciles de comprender.
- Adecuadas para la audiencia a la que se dirige.
- Evitar la doble negación.
- Evitar preguntas que contengan más de un concepto en una misma pregunta.
- Utilizar adecuadamente las tablas de calificación.
- Si la pregunta tiene una serie de respuestas predefinidas, evitar que éstas se solapen.
- Incluir preguntas de respuesta abierta (orientadas a recibir respuestas amplias, destinadas a conocer circunstancias generales, estados de ánimo, sensaciones y opiniones).

PLANTEAMIENTO

En esta prueba se ha pretendido evaluar la usabilidad del diseño partiendo de la premisa de que la mejor forma de comprobar la usabilidad de un diseño es precisamente poniéndolo a prueba con usuarios, observando cómo utilizan el producto y analizando con qué problemas se encuentran, qué errores cometen o qué tipo de tareas no son capaces de resolver satisfactoriamente.

El primer paso ha consistido en definir los objetivos del *test*, lo que se quería evaluar. En este caso, los objetivos han sido de tipo 'formativo', para detectar problemas de usabilidad.

En cuanto a la captación de participantes para la prueba era importante localizar personas con un perfil que se correspondiese al de los usuarios potenciales del producto, y que no estuviesen 'contaminados' por el proyecto, es decir, que no lo conociesen previamente.

Así que se seleccionó un grupo de cinco personas para la prueba, que si bien no formaban una muestra representativa para determinar si un producto era fácil de usar o no, sí que fueron suficientes para ayudarme a detectar problemas significativos de uso.

Antes de comenzar la prueba se definieron los escenarios y tareas. Los escenarios son narraciones de situaciones o contextos imaginarios, que permiten al usuario comprender o imaginarse la motivación o razón de las tareas que va a tener que realizar durante la prueba. Esto ha facilitado que el usuario se pusiera en el contexto.

En cuanto a las tareas, se han hecho de forma que resulten:

- Razonables o naturales para el usuario.
- Específicas. Dado que si la tarea es demasiado abstracta puede dar lugar a malentendidos o a que no podamos comparar la forma que tienen de resolver la misma tarea dos participantes diferentes.
- Factibles. Este factor busca evaluar el producto, no al usuario.
- De duración razonable. Pensando en que para completar la tarea el participante debe dedicar demasiado tiempo, puede desmotivarse y, por tanto, invalidar los resultados.

Por último, indicar que las tareas se han centrado en los procesos interactivos críticos del producto o más importantes desde el punto de vista de los objetivos de negocio.

PROCEDIMIENTO

Antes de comenzar la prueba los participantes respondieron a un breve cuestionario con datos sociodemográficos, intereses, hábitos y conocimientos previos.

Estas pruebas se han llevado a cabo en un “laboratorio”, es decir, en un espacio acondicionado específicamente para la prueba, donde he dirigido el desarrollo de las mismas como facilitador o moderador. En este sentido, he indicado a los participantes qué tareas debían hacer en cada momento, y he dinamizado la prueba para evitar que el usuario se atascase en alguna tarea concreta. Es importante indicar que en ningún caso he ayudado al usuario a realizar la tarea, porque lógicamente invalidaría el resultado.

Justo antes de comenzar las tareas he llevado a cabo una etapa de “test de 5 segundos”. He mostrado el producto únicamente durante cinco segundos y he solicitado a continuación que cada usuario expresase todo lo que recordaba de la interfaz del producto, cuál ha sido su impresión o para qué usos le ha parecido que está destinado el producto.

Durante la ejecución de las tareas, además de observar el comportamiento interactivo del usuario y qué estrategias utilizar para resolverlas, le he solicitado al usuario que expresase verbalmente qué estaba pensando, qué no entendía, y por qué llevaba a cabo una acción o dudaba (protocolo ‘think-aloud’). Esta información verbal fue interpretada, ya que las personas tendemos a reelaborar y reinterpretar nuestros recuerdos y el por qué actuamos como lo hacemos.

Una vez finalizada la prueba los usuarios completaron un cuestionario de satisfacción o usabilidad percibida.

Durante la prueba se ha grabado y registrado toda la información posible, ya que es de gran utilidad para el análisis de los resultados. Para ello, he contado con *software* específico como *Silverback* (*Clearleft*), que permite registrar qué hace el usuario en cada momento, e incluso grabar al usuario a través de cámaras.

También se ha tenido en cuenta que los participantes y su comportamiento son muy fácilmente influenciables en este tipo de pruebas. El comportamiento natural de los participantes se puede ver afectado por el simple hecho de saber que están siendo observados y estudiados. En este sentido, se ha aclarado a los participantes que no es a ellos a quienes se estaba evaluando, sino al producto.

Al mismo tiempo, se ha evitado influir o condicionar en la redacción de las tareas, y qué deben hacer o cómo deben hacerlas para llevarlas a cabo.

RESULTADOS Y CONCLUSIÓN

La conclusión que se puede extraer de las pruebas de usabilidad es que casi todos los participantes tenían dispositivo móvil, no invertían mucho tiempo en buscar aplicaciones, puesto que la mayoría de la información la sacaban de su círculo de amistades o curioseando por *internet*. Sin embargo, se mostraron muy exigentes a la hora de realizar las tareas y de valorar la aplicación (evaluar ciertos aspectos como la eficiencia de las funcionalidades, facilidad de uso,

costo, velocidad, etc…), dado que era un público al que le gustaba el uso de aplicaciones prácticas.

En cuanto a los resultados de los *test*, en general los entrevistados consideraron importantes aspectos como la velocidad, que la *interface* gráfica fuese clara e intuitiva (es decir, que con pocas opciones se pudiese llegar al resultado deseado), y al mismo tiempo consideraron deseable que tanto el texto como los botones fuesen legibles.

Por tanto, podemos considerar que la aplicación alcanza la puntuación necesaria para considerar que los usuarios están satisfechos. Llama la atención el buen rendimiento mostrado por los éstos durante la ejecución de las tareas - todos finalizaron las tareas - y su satisfacción fue alta. Hay que considerar también que para algunos de los usuarios fue la primera vez que hacían uso de un modelo de *smartphone* diferente al que estaban acostumbrados. Aún así, consiguieron terminar los *test* con una sensación de satisfacción con el sistema usado.

FICHA DEL PERFIL DE USUARIO

Se han identificado diversos perfiles, independientemente del género del perito forense o de si trabajaba en un laboratorio o por su cuenta. Por tanto, nos hemos centrado en cuatro grupos, que atendían a la edad de los usuarios:

- Grupo 1: persona entre 18 y 25 años.
- Grupo 2: persona entre 26 y 35 años.
- Grupo 3: persona entre 36 y 45 años.
- Grupo 4: persona entre 46 años y la edad límite de jubilación.

Para cada uno de los perfiles de usuario identificados se ha incluido una ficha.

Cada una de estas fichas nos ayudó a situarnos a la hora de hacer referencia a ciertas características como el perfil (características demográficas, intereses, motivaciones), contextos de uso (es decir, cuándo y en qué entorno harán uso de la aplicación), tareas que necesitaban los usuarios para alcanzar sus objetivos en la aplicación, y aquellas otras características o elementos cuya presencia en la interfaz de la aplicación pudiera resultar útil.

Figura 11: Diseño de la ficha para representar arquetipos

3.2. DISEÑO CONCEPTUAL

ESCENARIOS DE USO

El área de la ciencia forense es la que más ha evolucionado dentro de la seguridad, ya que los incidentes de seguridad se han incrementado en los últimos años. De hecho, cada vez es más habitual encontrar litigios donde la informática forma parte fundamental del proceso forense.

Hoy en día, los ataques informáticos son también diferentes (obligan a actualizar las técnicas de análisis en todo momento) y pueden darse en multitud de escenarios: propiedad intelectual, revelación de secretos, fraudes, espionaje, incumplimiento de contratos, etc... En los que se precisa un informe pericial emitido por expertos informáticos (peritos forenses).

La labor del perito es precisamente esa, analizar y describir los diferentes sucesos probados y las evidencias que los corroboran en un informe, que podrá ser de dos clases: ejecutivo y técnico.

Dicho informe, para que no sea impugnado en un juicio, deberá incluir al menos: un sumario o resumen del caso, las herramientas utilizadas, la adquisición de evidencias, el procesado de las evidencias, el análisis de las evidencias y las conclusiones.

Aunque, dependiendo del objetivo y ámbito del mismo, pueda que sea necesario ajustar la estructura del mismo (informe forense).

A continuación, se presentan algunos de los arquetipos de usuarios que han intervenido en el proceso de diseño y desarrollo:

FICHA 1

DATOS DEL PARTICIPANTE

Nombre: Carlos Javier

Apellidos: Martinez Porras

Edad: 56

Ocupación: Director de un laboratorio pericial forense ubicado en la villa de Madrid.

Descripción "persona": Carlos Javier es un profesional, padre de dos hijos, que cuenta con 16 años de experiencia profesional, y que es experto en criminalística, ciencias forenses, criminología y asesoramiento para la defensa.

Es una persona muy meticulosa a la que le gusta cumplir rigurosamente con los protocolos y procedimientos de calidad más altos.

Actualmente, dispone de un dispositivo móvil con SO *iOS*, el cual utiliza habitualmente.

Escenario: Carlos Javier acaba de salir del trabajo un viernes a las 14:00 horas y se dispone a coger el tren que le lleve a casa. Dispone de mucho tiempo, así que decide realizar el seguimiento de uno de los casos forenses analizados en el laboratorio, para comprobar el trabajo realizado por los miembros del equipo.

FICHA 2

DATOS DEL PARTICIPANTE

Nombre: Lorenzo

Apellidos: Martín Muñoz

Edad: 45

Ocupación: Ingeniero informático en una empresa de seguridad informática en Alicante.

Descripción "persona": Lorenzo es soltero y cuenta con gran experiencia en el mundo de la seguridad informática.

Además de lo anterior, Lorenzo es un reputado ponente en Congresos de seguridad Informática, tanto nacionales como internacionales, con vocación académica y co-fundador de un blog de seguridad informática.

Actualmente, Lorenzo utiliza un portatil para trabajar, pero cuando no está trabajando y necesita consultar el correo electrónico o consultar información de *Internet*, lo hace a través de su terminal *Android*.

Escenario: Ha habido un robo de información privilegiada en la empresa donde trabaja Lorenzo, y su jefe le asigna la investigación del caso a él.

Lorenzo desea tenerlo todo anotado, pero al mismo tiempo quiere compartir las notas del caso con otras personas, aunque no tenga conexión a *Internet* en ese momento.

Al mismo tiempo, quiere gestionar sus notas cómodamente desde su ordenador.

Por tanto, Lorenzo sabe que existe una aplicación con esas características disponible en *Google Play*. Así que, se descarga la *app*, para elaborar el informe forense.

FICHA 3

DATOS DEL PARTICIPANTE

Nombre: Pablo

Apellidos: Fernández Bueno

Edad: 28

Ocupación: es Ingeniero informático titulado por la Universidad de Sevilla, Colegiado nº 93 del Colegio Oficial de Ingenieros en informática de Andalucía y miembro del Cuerpo de Peritos de dicho colegio.

Descripción "persona": Pablo está casado y tiene un hijo de 3 años.

Actualmente, se dedica a la realización de informes periciales informáticos y telemáticos, basándose en sus conocimientos como ingeniero y en su experiencia y formación como perito.

Ha intervenido en numerosos procesos judiciales (solicitados por el juzgado o de parte) aportando pruebas telemáticas fiables y válidas, asegurando su cadena de custodia y realizando una defensa de los mismos ante las autoridades judiciales.

Su terminal móvil *Android* le sirve no sólo para comunicarse, sino también como herramienta de trabajo.

Escenario: Pablo necesita organizar rápidamente la información recabada, para elaborar el informe. Así que de cara a la elaboración del informe forense, se descarga la *app ForensicNotes*, para tomar notas sobre todos los pasos y procesos realizados.

FICHA 4

DATOS DEL PARTICIPANTE

Nombre: José Carlos.

Apellidos: Lorenzana Vilaseca.

Edad: 23.

Ocupación: estudiante de un Master en Seguridad de las TIC

Descripción "persona": José Carlos es una joven universitario que está a punto de finalizar sus estudios.

Este semestre está estudiando la asignatura de Análisis Forense en la universidad, y ha decidido basar el TFM (Trabajo Final de Máster) en esta especialidad.

Actualmente, es propietario de un *smartphone* con SO *iOS*.

Escenario: el profesor de la asignatura de Análisis Forense propone en el aula la realización de un caso práctico en grupos de dos personas, en el que no es necesario extraer las evidencias, es decir, obtener los ficheros, pero sí mostrar sus contenidos.

Así que, José Carlos y su compañera tienen que contestar coordinada y adecuadamente a todas las preguntas planteadas, mencionando las herramientas empleadas y adjuntando una captura de pantalla con los resultados obtenidos para cada pregunta.

La compañera de José Carlos le propone utilizar la *app ForensicNotes*, disponible en las plataformas *Android* e *iOS*, para elaborar un informe pericial que recoja el análisis forense realizado y organizar de forma coherente todas las evidencias localizadas en dicho análisis. Así que, él acepta la propuesta de su colaboradora.

FLUJO DE INTERACCIÓN

Para representar los diagramas de flujo de interacción se ha utilizado el propuesto por *Jesse James Garrett* y posteriormente extendido por *Cecil*.

En estos diagramas, las posibles interacciones del usuario y la respuesta del producto se describe mediante elementos (nodos) y conectores. El significado de cada elemento, así como el de la relación definida por los conectores, se codifica gráficamente, y queda explicado a través de leyendas descriptivas.

A continuación, se muestra el flujo de interacción original, realizado con ayuda de la herramienta *Sketch 3* para *Mac Os*.

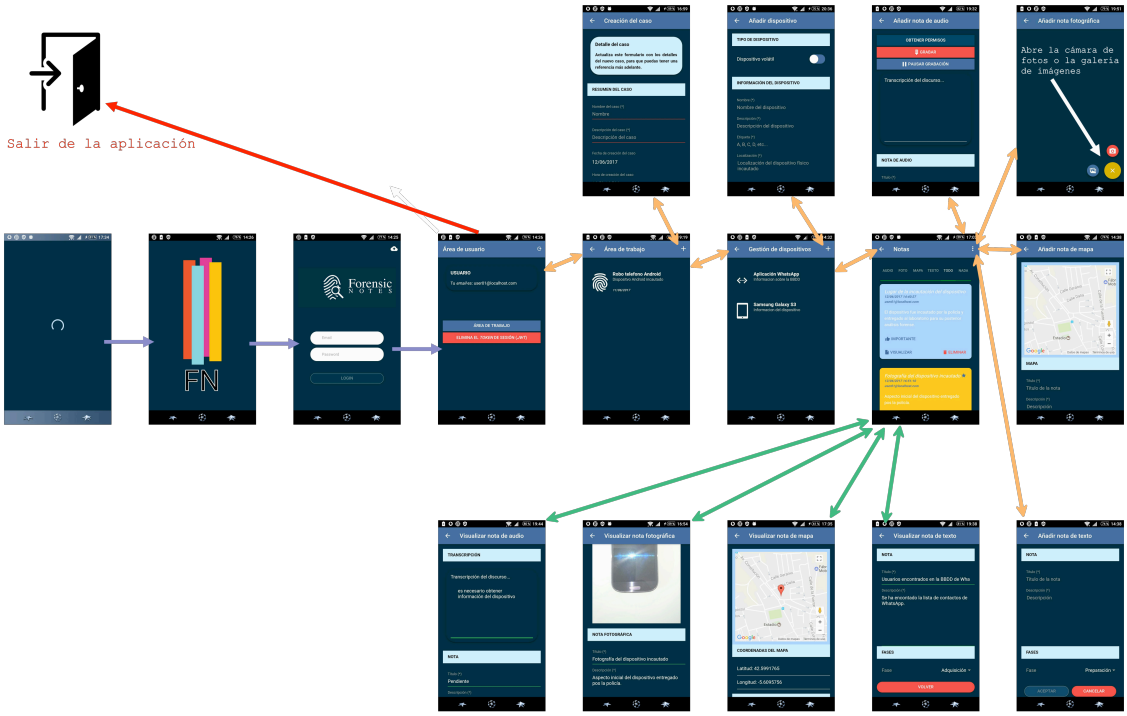


Figura 12: árbol de navegación

La primera pantalla de la aplicación es una *Splash screen*, que hace de punto intermedio entre la pantalla de *loading* y la de *login*. Hay que tener en cuenta que el *hardware* de nuestros dispositivos no es perfecto, y atendiendo al tamaño de la aplicación, puede tardar unos segundos en abrirse.

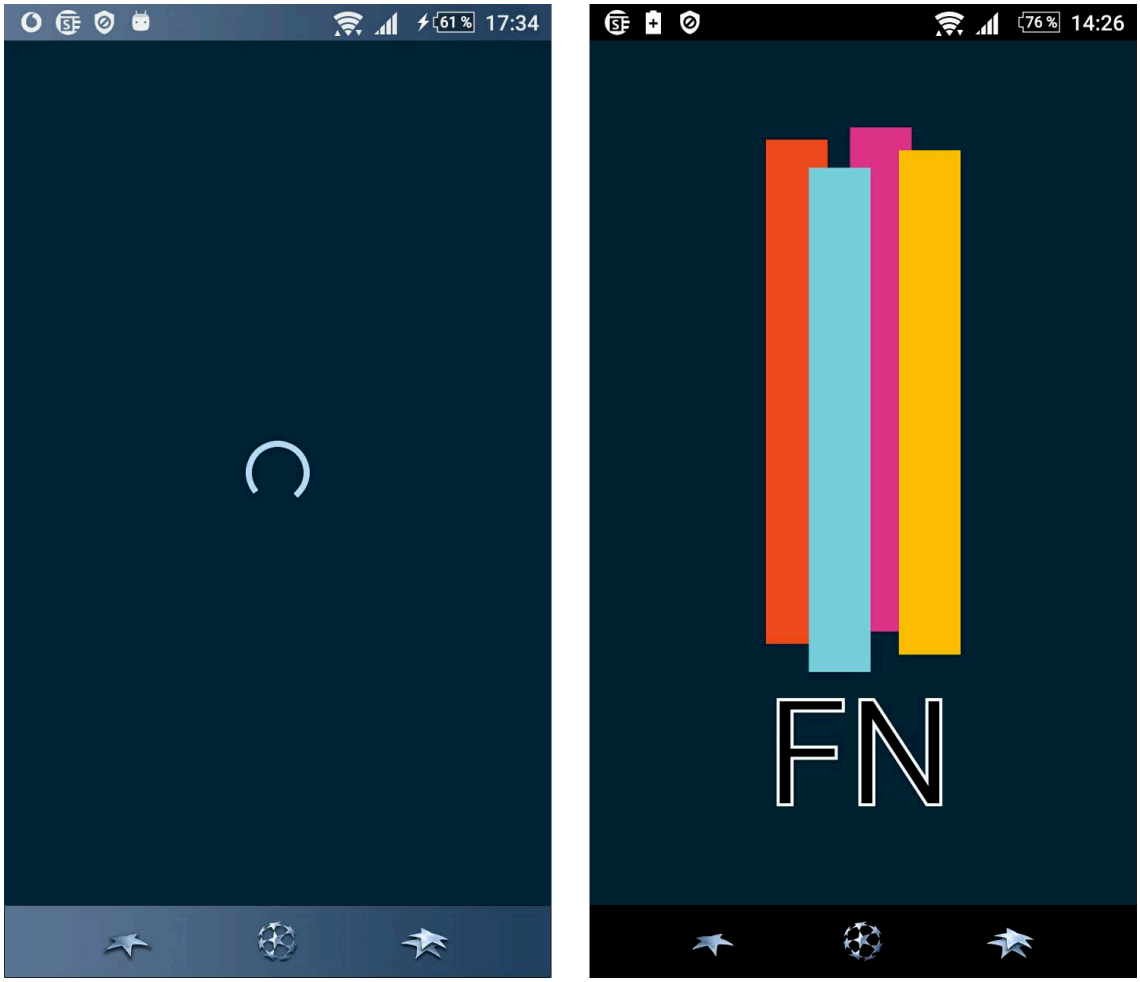


Figura 13: *loading* y *splash screen*

LoginPage es la segunda pantalla de la aplicación. Su función consiste en controlar el acceso individual al sistema en red mediante la utilización de credenciales provistas por el usuario (es decir, un *token JWT*).

Una vez introducida la información del servidor (dirección *IP* del servidor de autenticación, el usuario de la *BBDD* y su correspondiente contraseña), el usuario puede hacer el *login* al sistema, para obtener acceso o puede pulsar sobre el botón “volver atrás” del dispositivo, cuando no precise acceder al mismo.

Hay que tener en cuenta que la validación de los datos de entrada debe llevarse a cabo siempre a través de un sistema de autenticación en red que sea considerado fiable, generalmente en el *back-end*. El dispositivo móvil no se puede considerar como elemento confiable, por lo que la lógica de autenticación no debe estar acoplada a la lógica del recurso al que se accede.

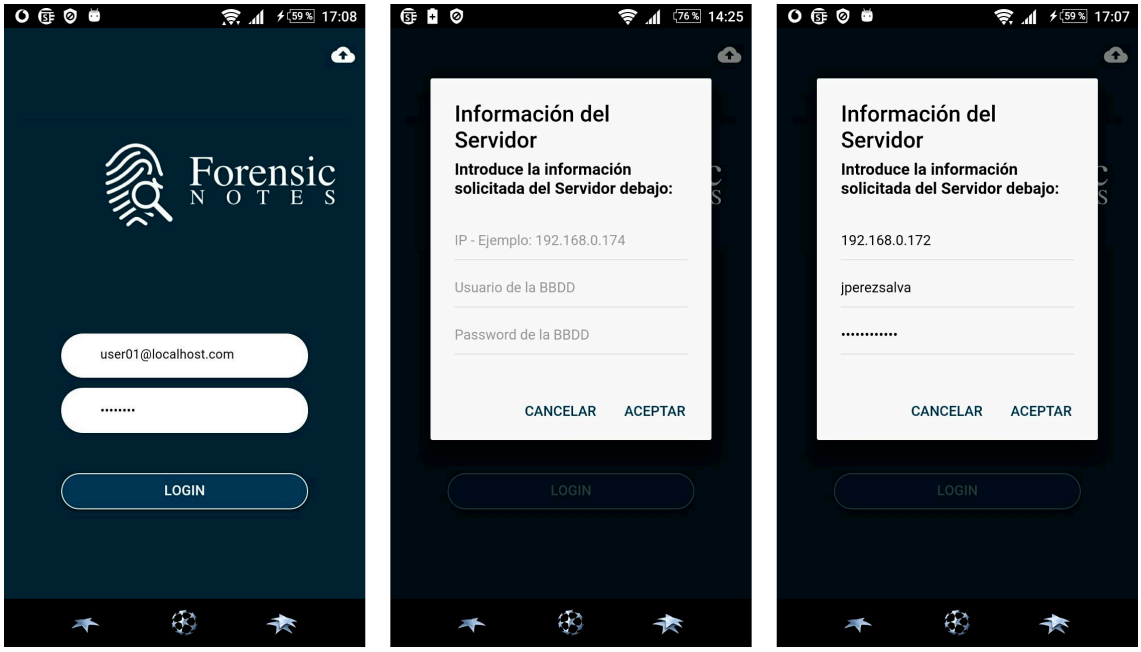


Figura 14: *LoginPage*

Atendiendo a los datos proporcionados y a la respuesta del servidor, podemos obtener acceso a la pantalla principal de la aplicación (*HomePage*) o un error, que puede ser de tres tipos:

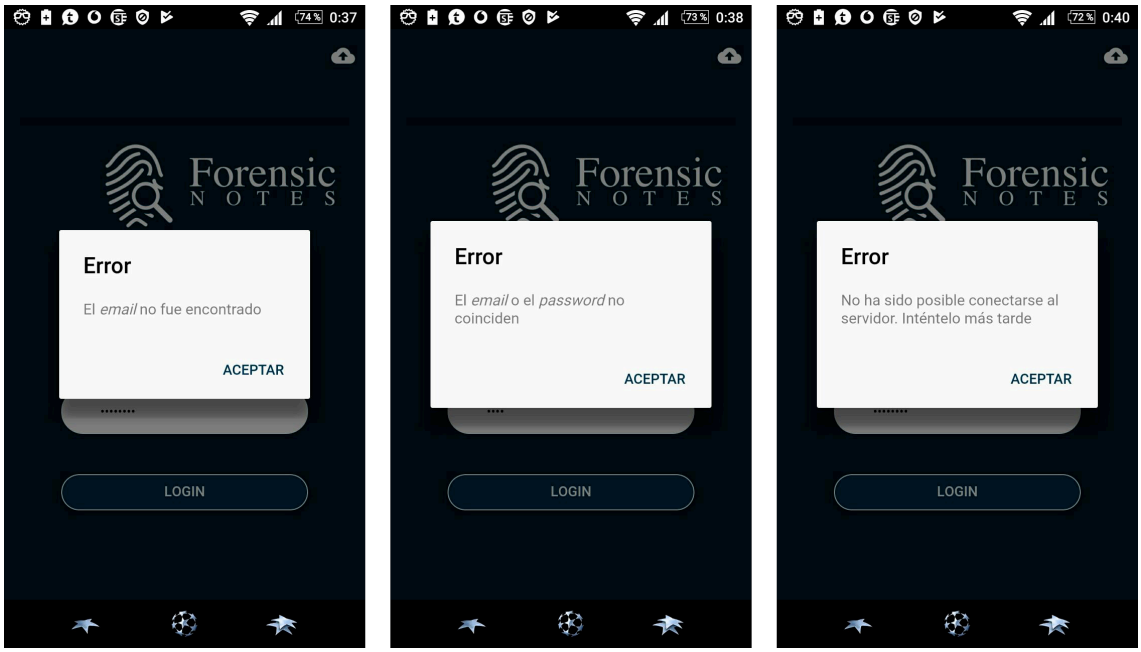


Figura 15: errores obtenidos en la pantalla de *login*

La siguiente vista es la pantalla *HomePage*, que es la pantalla principal de la aplicación. Desde aquí tenemos acceso al área de trabajo.

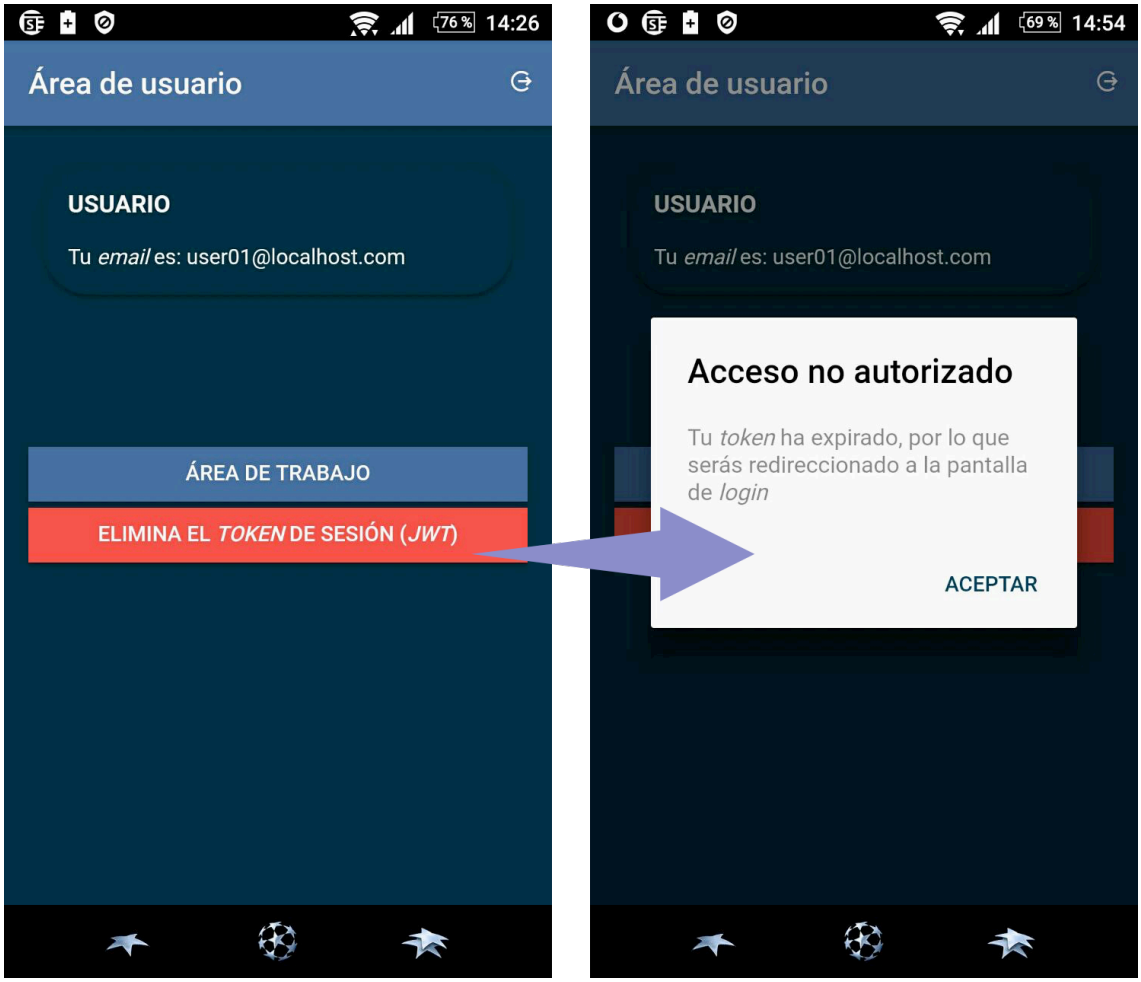


Figura 16: pantalla principal de la aplicación (*HomePage*)

El botón de la esquina superior derecha de la pantalla *HomePage* permite salir de la aplicación.

La razón de poner los botones en esta posición es porque normalmente los usuarios sólo hacen caso a las opciones que se encuentran en la parte superior de la pantalla. Incluso en pantallas pequeñas como las de los móviles, donde parece que toda la información está cerca, los usuarios se fijan únicamente en las opciones que aparecen al principio de la página y prestan menos atención a las demás.

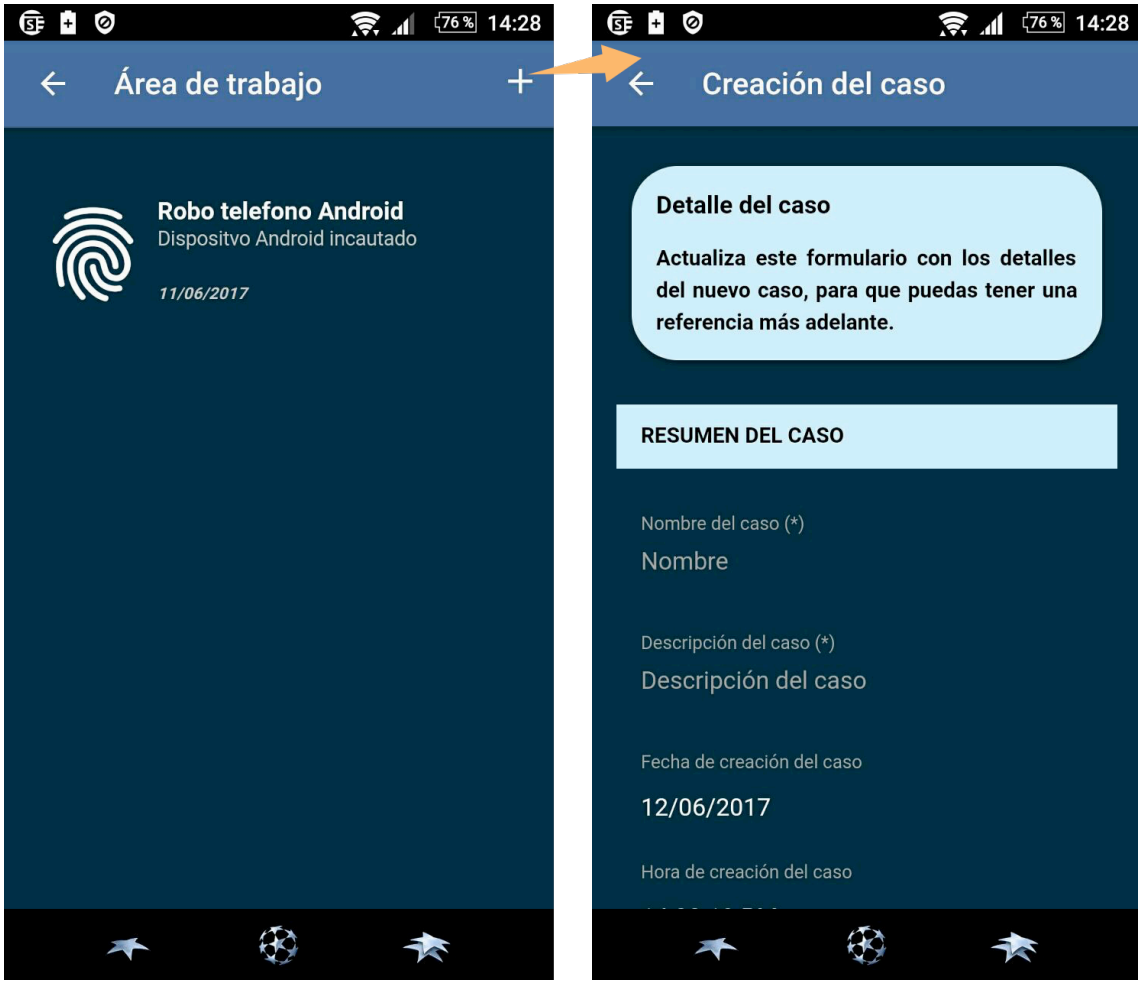


Figura 17: *DetailsPage* y *CreateCasePage*

Antes de añadir los dispositivos físicos o volátiles (*DevicePage*), que van a ser analizados, debemos crear un caso (*CreateCasePage*) y añadir la información correspondiente. También es posible visualizar, actualizar y eliminar los casos.

Una vez añadido el nuevo caso, se mostrará el mensaje “Caso añadido correctamente”. Si por el contrario, la acción es una actualización o borrado, se mostrarán los mensajes “Caso actualizado correctamente” y “Caso eliminado correctamente” respectivamente (siempre que no existan dispositivos asociados al caso, en cuyo caso se mostrará el mensaje “Existen dispositivos pendientes de eliminar”).

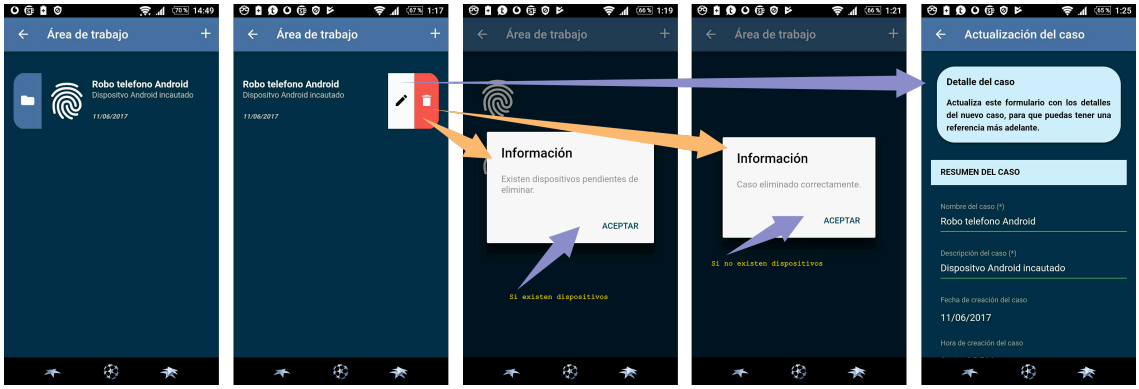


Figura 18: actualización y eliminación de un caso

Al igual que los casos, la pantalla *DevicePage* permite visualizar, añadir, modificar y eliminar los dispositivos físicos y volátiles (dispositivos incautados) que van a ser analizados posteriormente y que contienen las posibles evidencias.

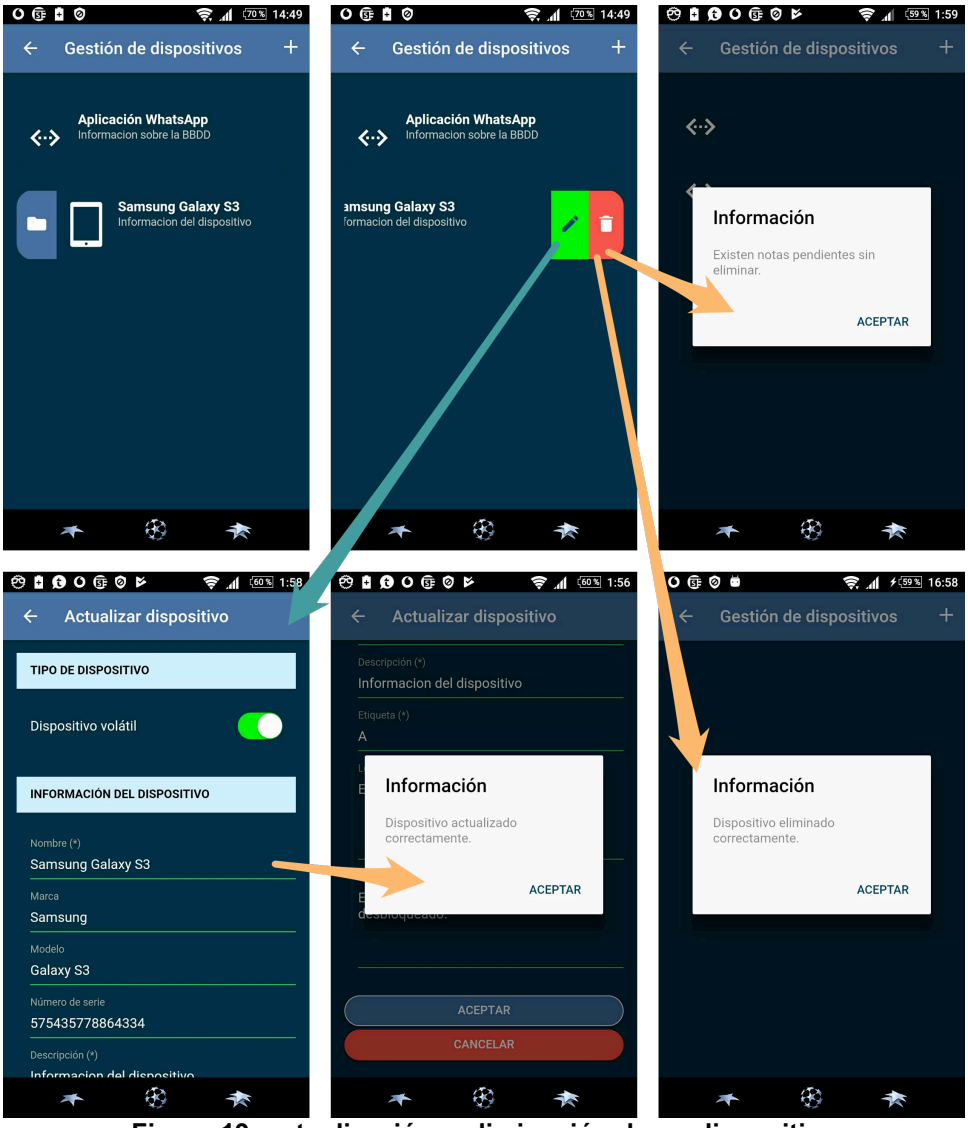


Figura 19: actualización y eliminación de un dispositivo

En cuanto a las evidencias, éstas se podrán agrupar en dos grupos dependiendo de su tiempo de vida:

- Volátil: Son aquellas evidencias que son creadas y destruidas durante la ejecución del sistema (memoria, paquetes de red, ficheros temporales, etc.). Pueden contener contraseñas de cifrado, procesos en ejecución que han sido borrados de disco u otros datos de interés.
- No volátil: Son aquellas evidencias que se pueden obtener del dispositivo una vez ha sido apagado (principalmente dispositivos de almacenamiento).

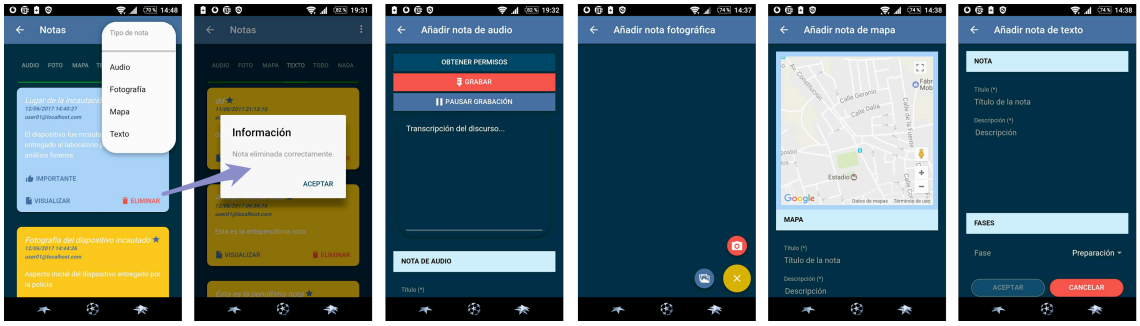


Figura 20: creación y eliminación de notas de un dispositivo

Por último, desde la pantalla de dispositivos accedemos a la de notas, donde el usuario podrá tomar notas (que pueden pertenecer a cualquiera de las etapas del proceso forense), visualizarlas (ordenadas por fecha de más antigua a más reciente) y eliminar aquellas que no sean significativas.

Dado que el proceso de análisis forense se ha dividido en seis etapas, para la correcta consecución del proceso de análisis se recomienda la toma de notas durante cada una de las fases del análisis.

Las notas, cuanto más detalladas mejor, pueden incluir:

- Capturas de pantalla con la cámara.
- Localización de evidencias encontradas.
- Notas manuscritas que reflejen información, como por ejemplo: el estado de las evidencias entregadas; el tamaño y descripción del archivo de evidencias recibido; las limitaciones del análisis realizado; lo que se pide corroborar o comprobar como analista forense; quién ha solicitado el informe forense; las fechas más importantes en relación al informe; datos personales del analista; herramientas utilizadas, etc...

En cuanto a las etapas del análisis forense, comentar que:

- La fase de preparación se ejecuta de forma previa al proceso de análisis, y consiste en identificar los elementos físicos que se van a analizar y las evidencias que se buscarán en cada uno de los elementos analizables.
- El proceso de adquisición se debe realizar teniendo en cuenta la volatilidad de las evidencias, por lo que es necesario recolectar primero las evidencias más volátiles.

  Esta estapa consiste en obtener o capturar las evidencias enumeradas en la fase de preparación.
- La gestión de evidencias está relacionada con la cadena de custodia y la validez de todo el proceso de análisis forense.
- El examen consiste en identificar las evidencias a partir de la información obtenida en la fase de adquisición.
- El análisis consiste en obtener conclusiones a partir de las evidencias obtenidas.
- Finalmente, la presentación consiste en describir los diferentes sucesos probados y las evidencias que los corroboran.

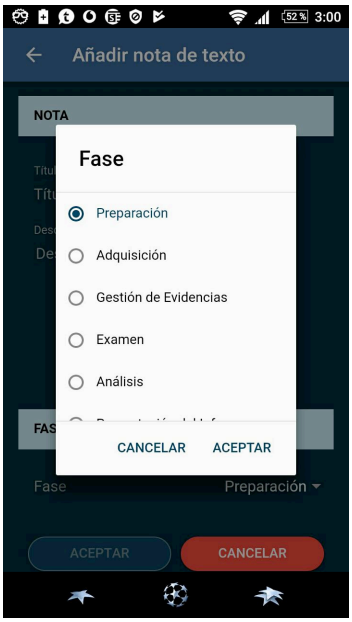


Figura 21: fases del análisis forense

Además, el usuario podrá marcar como importantes aquellas notas que sean relevantes para el proceso de análisis forense.

Indicar también, que tanto la acción de eliminar notas como la de crear notas conlleva su correspondiente mensaje: “Nota añadida correctamente” o “Nota eliminada correctamente”.

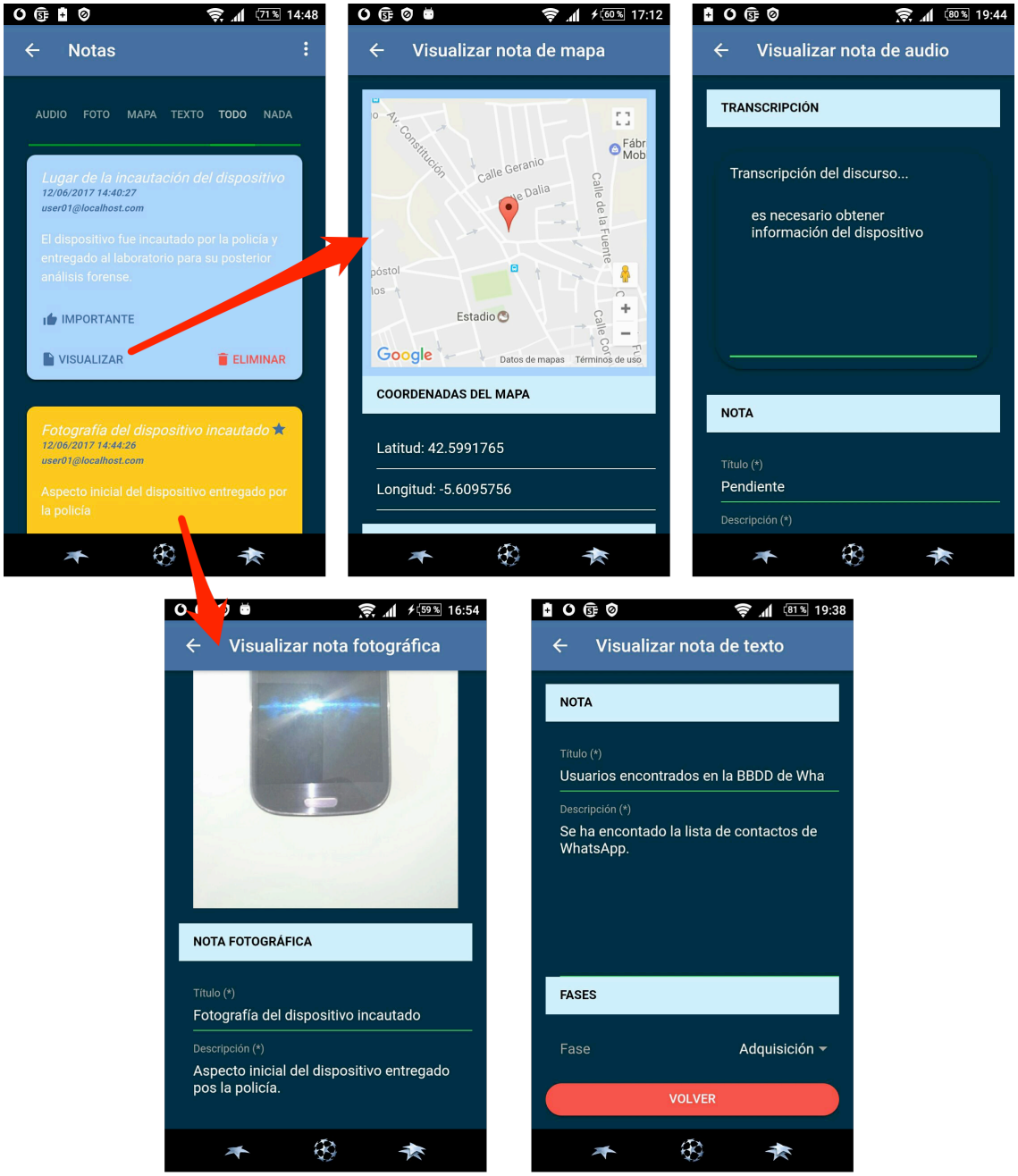


Figura 22: visualización de las notas de un dispositivo

Del mismo modo, que se visualizan las notas en el dispositivo móvil también es posible hacerlo en el navegador *web* de una *tablet* o un equipo de escritorio. Sin embargo, aunque no sea posible tomar fotografías, grabar notas de audio u obtener la posición geográfica del mismo modo que lo hace un dispositivo móvil, si será posible visualizar, editar y eliminar parte de la información obtenida con el mismo.

A continuación, se muestran algunas de las pantallas de la ejecución de la aplicación en un navegador *web* de un equipo de escritorio.

Figura 23: pantalla de *LoginPage* en un navegador *web*

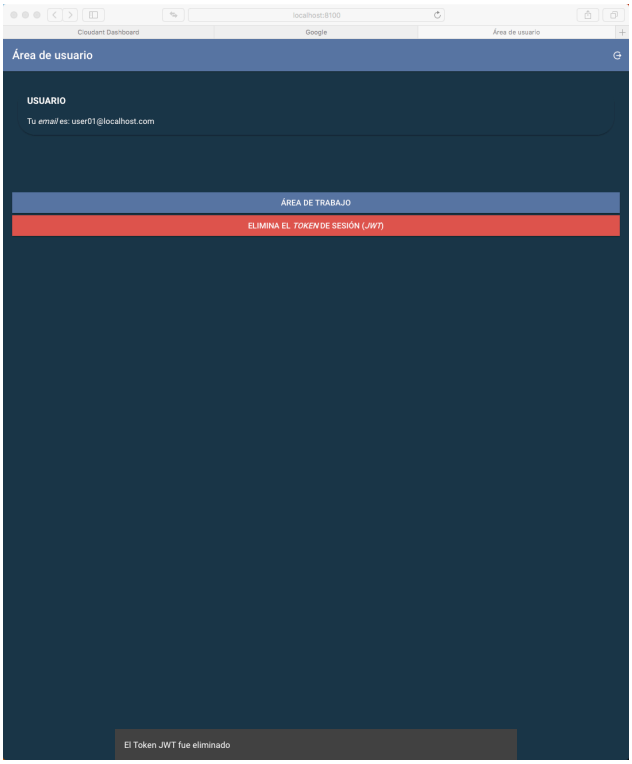


Figura 24: pantalla de *HomePage* en un navegador *web*

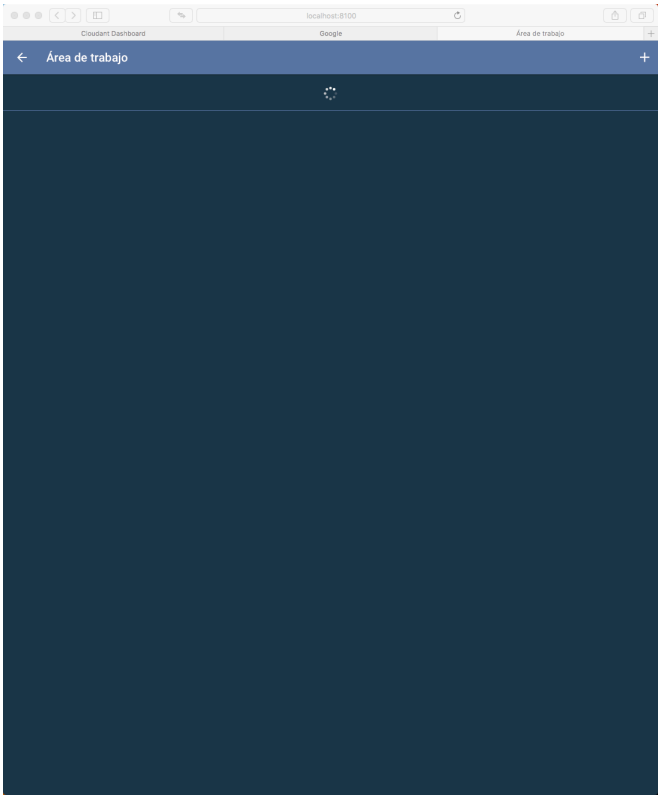


Figura 25: ejecución de un *refresher* en la pantalla *DetailsPage*

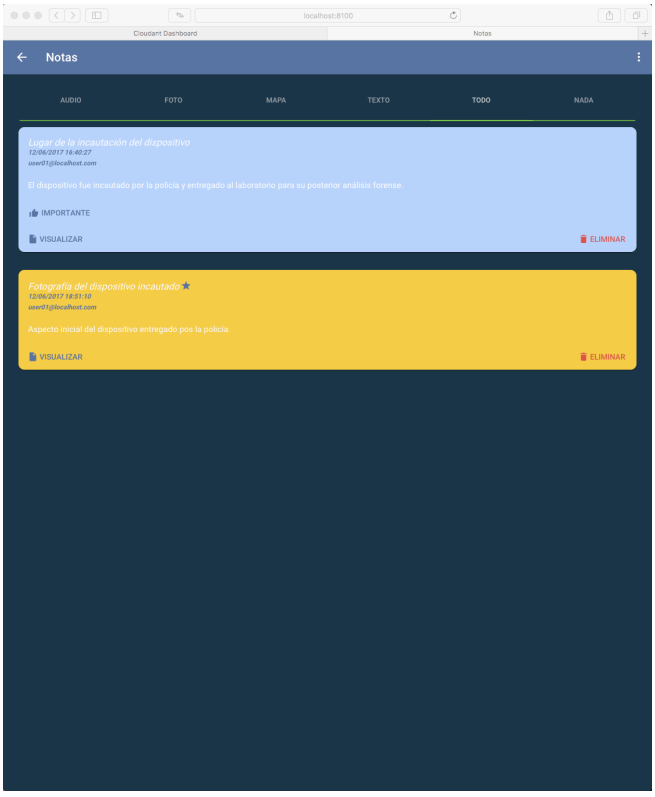


Figura 26: visualización de notas en un navegador *web*

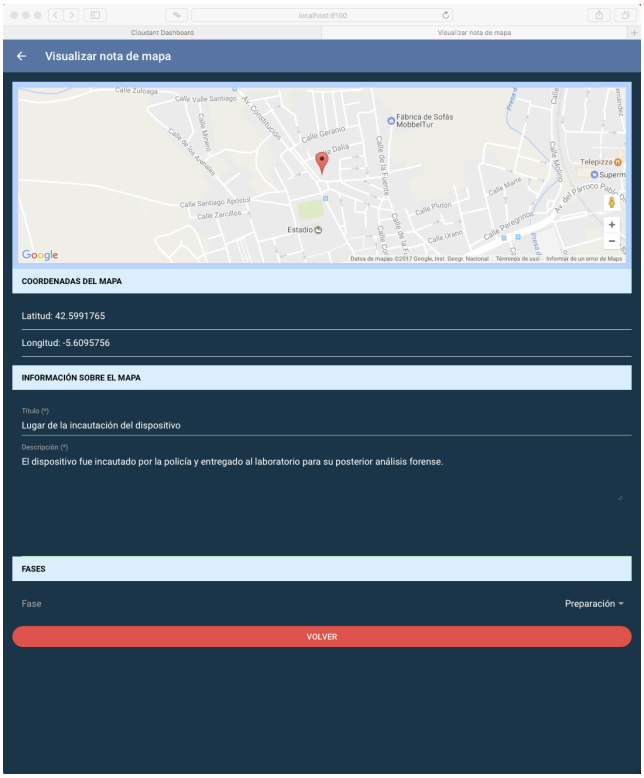


Figura 27: visualización de una nota de mapa en un navegador *web 1*

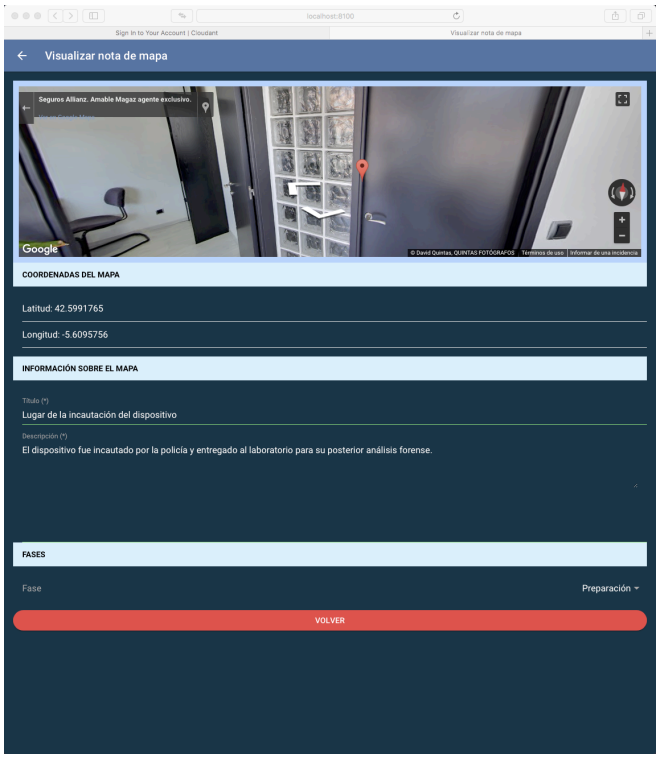


Figura 28: visualización de una nota de mapa en un navegador *web 2*

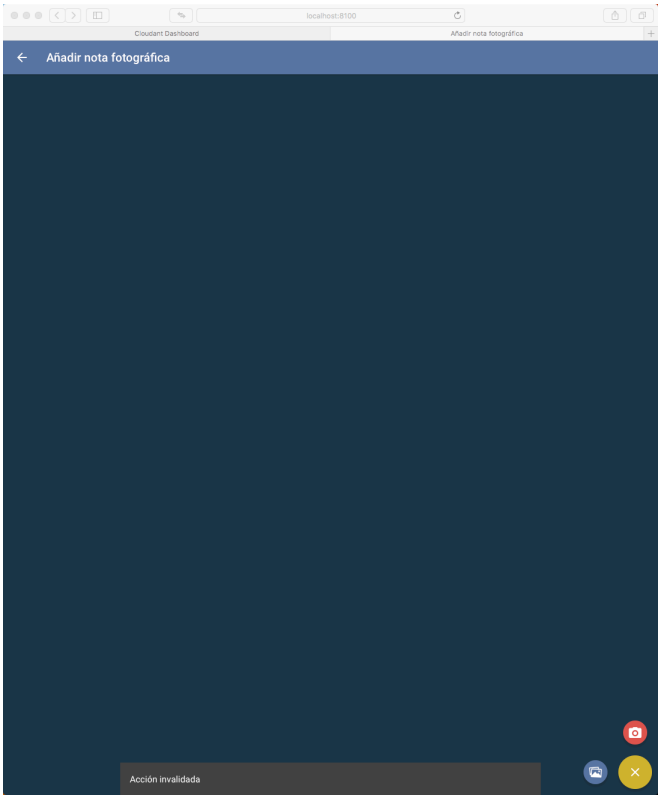


Figura 29: visualización de una nota fotográfica en un navegador *web*

Como se puede observar, cualquier acción que implique acceder a la cámara de fotos será invalidada, dado que *Cordova* solamente se encuentra disponible en dispositivos móviles.

3.3. PROTOTIPADO

Se ha seguido un diseño modular a la hora de diseñar el producto. Es decir, se ha pensado en el diseño en términos de pequeños componentes reutilizables, cada uno con sus propias características, estados, formas de relacionarse, adaptarse e interactuar con el resto de componentes y los diferentes contextos. De este modo, el diseño ha sido más eficiente, flexible y fácil de mantener.

Posibles ejemplos de componentes utilizados en la aplicación serían: botones, gráficas, etc. En todos estos componentes se ha definido o considerado cómo se adaptarán a diferentes contextos: características del dispositivo de acceso (resolución de pantalla) y tipo de usuario (identificado a través de *login*).

En cuanto al diseño de los bocetos, aunque existen numerosas aplicaciones *software* para la elaboración de *wireframes*, al menos en las primeras etapas de diseño, se ha utilizado sin lugar a dudas la más eficaz: el lápiz y el papel.

Por ese motivo, en un primer momento se han creado bocetos o esquemas con *wireframes* de “baja fidelidad” para representar la interfaz, dado que son más fáciles de elaborar y modificar. Es decir, documentos en los que se ha representado el esqueleto de la interfaz, la distribución de los diferentes bloques de contenido y, el rotulado de los diferentes elementos o componentes de la pantalla o página del producto. En este sentido, los *wireframes* también han resultado de gran utilidad a la hora de especificar la relación entre la arquitectura del producto y su aspecto gráfico.

A continuación, podemos ver en la página siguiente algunos ejemplos realizados a mano alzada con ayuda de la herramienta de *Adobe Photoshop Sketch* para *iPad*:

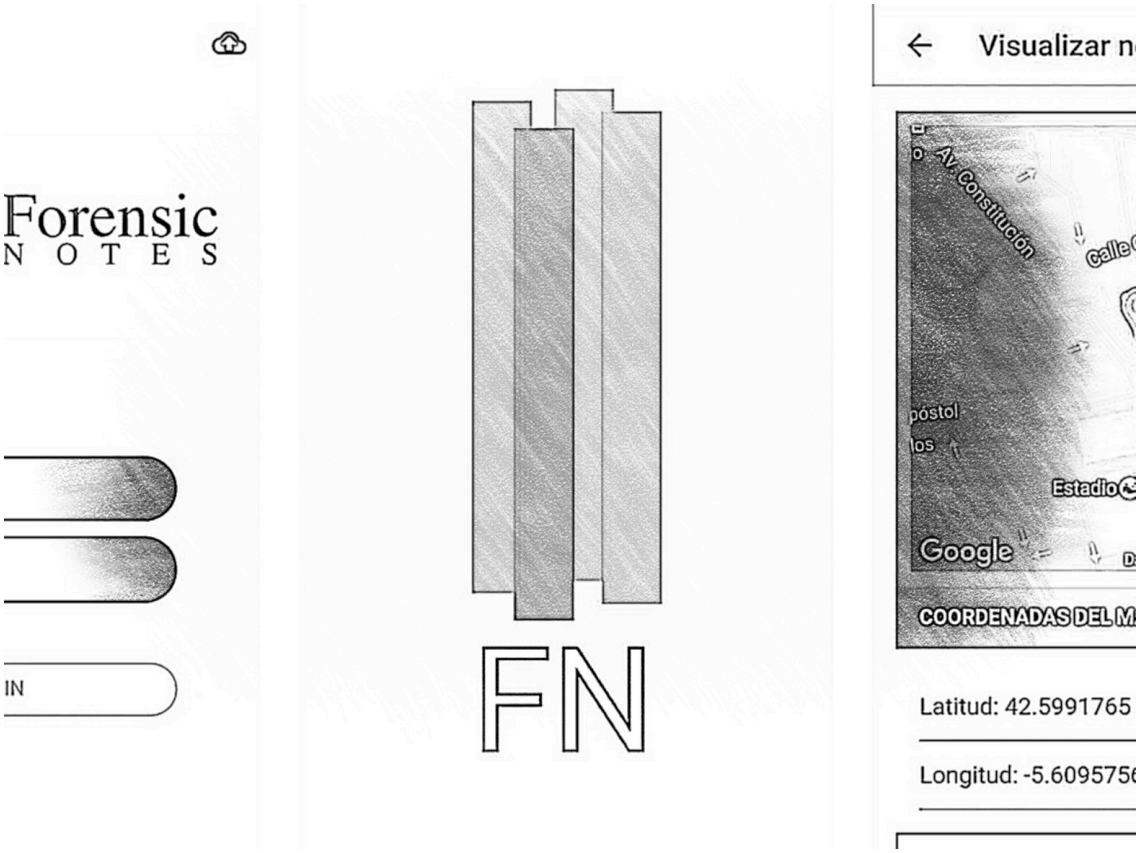


Figura 30: bocetos originales

Posteriormente, se ha procedido a diseñar *wireframes* de “alta fidelidad” con la herramienta *Sketch* 3. Estos incl•uyen mayor detalle visual y precisa del diseño final, como imágenes, colores, tipografías, etc… Las ventajas con respecto a los de baja fidelidad son obvias.

En este caso, no se han añadido anotaciones para explicar o detallar cuestiones relativas al funcionamiento y comportamiento interactivo de cada elemento representado, porque como se puede apreciar el diseño es lo suficientemente claro.

Indicar que en el diseño se ha seguido la tendencia natural a ocupar todo el espacio, rellenando cada hueco con colores claros, e incrementando de este modo innecesariamente su complejidad, la carga visual y cognitiva del usuario.

Este espacio en color claro ha facilitado al usuario diferenciar visualmente elementos o grupos de elementos (el espacio en color claro juega un papel activo en el diseño), permitiendo una exploración visual más calmada, pero también, incrementando la simplicidad, elegancia y apariencia estética del diseño.

Por último comentar, que los elementos incluidos en el diseño son lo suficientemente claros y descriptivos, y no ofrecen lugar a dudas, por lo que no se ha incidido en su descripción. Solamente matizar que existen elementos que pueden desplegar más información al pulsar sobre ellos. Este enfoque permite aprovechar al máximo el espacio disponible en la pantalla del dispositivo. Esta idea también nos permite ahorrar tiempo a la hora de navegar por las diferentes pantallas.

3.4. EVALUACIÓN DEL PROTOTIPO

Antes de comenzar la entrevista se ha llevado a cabo un breve cuestionario con datos sociodemográficos, intereses, hábitos y conocimientos previos. Las preguntas servían para hacer una presentación del usuario, y eran las siguientes:

- ¿Cuál es su nombre [Nombre y apellidos]?
- ¿A qué se dedica [Profesión, Actividad]?
- ¿Qué experiencia tiene en *Internet*?
- ¿Navega habitualmente? ¿Cuántas horas navega al día, a la semana? Incluya el número de horas que utiliza el correo electrónico.
- ¿Qué sitios visita habitualmente?
- ¿Cuáles son sus sitios preferidos?
- Cuando desea encontrar algo en Internet, ¿cómo llega a un sitio que pueda tener esa información? ¿Usa un buscador? ¿Cuál? Si no usa un buscador, ¿cómo lo hace?

Finalizado el cuestionario se dio paso a la entrevista, donde los usuarios realizaron cuatro tareas, siempre en el mismo orden:

- Tarea 1. Localizar y acceder a la tienda de *apps* (conocido más habitualmente como *market*).

- Tarea 2. Localizar la *app* “Evernote” e instalarla.
- Tarea 3. Desinstalar la *app* anterior.
- Tarea 4. Localizar la *app* de *Microsoft* “OneNote” en el *market* y comprarla. La tarea ha finalizado cuando se solicitaban los datos bancarios. Esta tarea es similar a la tarea 2, con la única diferencia de que al ser la segunda vez que instalaban la *app*, había cierta práctica para hacerlo, y por otro lado, conllevaba algunos pasos adicionales al tratarse de una *app* de pago.

Se citó a los cinco usuarios que harían el *test* con su propio teléfono. Ellos servirían de referente para saber cómo se comportan los usuarios que están habituados a un sistema operativo móvil determinado. Tras una breve encuesta demográfica, se les pidió que realizaran las 4 tareas. Tal como se esperaba, los 5 realizaron las tareas sin dificultad. Se anotaron los tiempos requeridos por estos usuarios para cada tarea: 1 segundo para localizar el *market* y entrar en él; 14 segundos (en *iOS*) y 17 segundos (en *Android*) para encontrar la *app* de *Evernote* en el *market* y clicar en el botón que inicia la descarga; 10 segundos (*iOS*) y 40 segundos (*Android*) para desinstalar la *app* recién instalada; y también 10 segundos (en *iOS*) y 40 segundos (en *Android*) para encontrar en el *market* la *app* “OneNote” que les solicité y llegar hasta la interfaz que solicita los datos para el pago.

Estos primeros *tests* permitieron asegurar que las siguientes tareas propuestas se podían realizar sin dificultades, y tener una orientación de cuánto tiempo podían llevar estas tareas a un usuario que conoce su dispositivo y está acostumbrado a instalar y desinstalar *apps*. En vista de que realizaron las tareas sin dificultad, se procedió a continuar con el experimento y se pasó la primera parte del *test* a los cinco usuarios que servirán para obtener resultados de este estudio.

Luego, se llevó a cabo un “test de 5 segundos” y una vez finalizado éste, se planteó a los cinco usuarios que realizaran una serie de acciones sobre un escenario real utilizando un ejemplo de prototipo de la aplicación preparado para el evento:

Tarea A. La primera tarea consistía en dos partes bien diferenciadas. Por una parte, había que registrarse en el sistema.

Por otra parte, se solicitó a los participantes que formasen un equipo forense, para posteriormente crear un caso sobre un informe ficticio documentado correspondiente a varios dispositivos móviles, que habían sido incautados por la policía y que una organización criminal estaba utilizando en el momento de una redada.

Por tanto, con el fin de flexibilizar y facilitar la realización de la prueba se dejó libertad para que uno de los usuarios se hiciese cargo de crear un nuevo caso y añadiese el dispositivo que considerase oportuno de entre los indicados en el dosier.

Tarea B. A continuación, se propuso a los participantes un reto. Debían anotar dentro de la propia aplicación forense cinco elementos de información que considerasen de interés de entre los enumerados en el dosier, con un tiempo límite de 15 minutos. Las notas, cuanto más detalladas mejor, debían incluir al menos: una captura (fotográfica o de video) de cualquiera de las imágenes proporcionadas en el documento, alguna nota manuscrita, la localización geográfica de alguna evidencia encontrada y una nota de audio con su propia voz.

Es decir, un breve listado de cinco elementos con información que pudiese contener del dispositivo involucrado en el proceso forense y que permitiese posteriormente razonar sobre los posibles objetivos, los cómplices del sospechoso o información adicional para continuar las investigaciones.

Tarea C. Luego, se solicitó a los participantes que comprobasen si el *hash* de las tres evidencias elegidas al azar era correcto. Para ello, se les pidió que anotasen los *hash* que habían anotado correctamente, los que habían errado y los que habían dejado en blanco.

Tarea D. Esta tarea fue muy sencilla. Por una parte, consistió en identificar las evidencias a partir de la información obtenida. Los participantes solamente debían descartar las notas que no eran relevantes.

Por otra parte, se pidió a los usuarios que mostrasen el gráfico de evolución de la tarea seleccionada.

Tarea E. Posteriormente, se animó a los participantes a escribir una hipótesis sobre las evidencias obtenidas. Es decir, sobre las notas que habían marcado como relevantes.

Esta tarea fue completamente libre, dado que se había eliminado intencionadamente toda información relacionada con esta parte en el dosier.

Tarea F. Por último, una vez finalizada la tarea anterior, se animó a los participantes a generar el informe y a compartirlo con el resto de participantes. Es decir, subir el informe generado al servidor, sincronizar la aplicación y compararlo con los resultados obtenidos por los compañeros.

En cuanto al desarrollo de las actividades, se permitió a los usuarios comentar en voz alta lo que hacían en cada momento. No se les limitó el tiempo para realizar la mayoría de las tareas, a excepción de la prueba consistente en el reto, y se les indicó que en cualquier momento podían abandonarlas si no veían la forma de hacerlas. Mientras ellos hacían las tareas, como observador cronometraba el tiempo que tardaba el usuario en realizar cada una, anotaba si la finalizaba con éxito y las dificultades que el usuario tenía. Esas anotaciones, se usaron para obtener las siguientes métricas:

- Eficacia. Se entiende por esta medida el grado con el que un usuario resuelve una tarea. La métrica habitualmente usada para medir la

eficacia es la ratio de realización de tarea (*task completion rate*). Para medirla, se optó por un sistema de 3 valores: el usuario termina la tarea sin dificultad, la termina con dificultad (lento o requiere ayuda), o no la realiza con éxito.

- Eficiencia. Durante el reto se midió el tiempo que cada usuario invertía en realizar la tarea, sin cronometrar el tiempo que invertía en tomar una nota sobre una evidencia en particular, ya que puede variar en función de la velocidad de cada participante.
- Satisfacción. Durante la realización de cada una de las tareas se anotaron todos los comentarios de los usuarios y, una vez finalizado, se les invitó a realizar un cuestionario, para conocer su satisfacción con la plataforma y obtener la puntuación promedio de la aplicación.

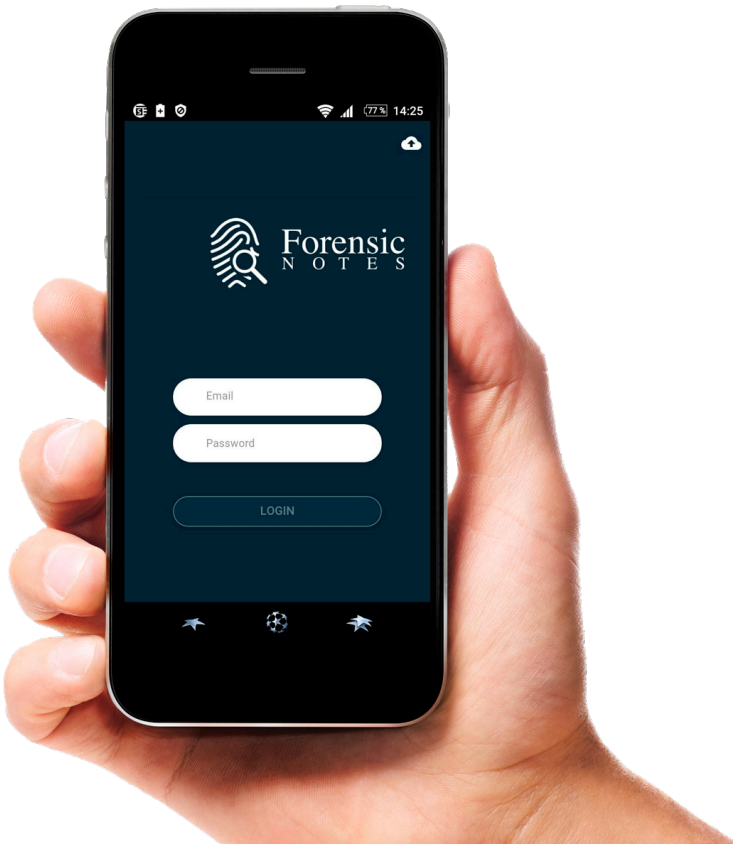


Figura 31: prueba del prototipo en un dispositivo *iPhone*

Así pues, finalizada la entrevista se realizó el cuestionario *online* (*test* heurístico), cuyo contenido era el siguiente:

Estas preguntas estaban pensadas para cuando el usuario estaba mirando la pantalla inicial de la aplicación y antes de comenzar a navegar sobre el contenido.

- **Identidad**

    - ¿Con la información que se ofrece en la pantalla inicial, es posible saber para que fin se va a utilizar la aplicación? ¿Cómo lo sabe?

    - ¿Hay algún elemento gráfico o texto que le haya ayudado a entender más claramente cual era la finalidad de la aplicación?

    - ¿Relaciona los colores predominantes de la aplicación con la finalidad de ésta? ¿Relaciona el nombre de la *app* con su logotipo? ¿Es claro?

    - ¿De los elementos que se muestran en pantalla, hay algo que usted crea que está fuera de lugar?

    - ¿Hacia qué tipo de audiencia cree usted que está dirigida esta aplicación? ¿Por qué?

Estas preguntas estaban pensadas para después de permitir al usuario navegar por el contenido de la aplicación, con el fin de que se formase una opinión acerca de lo que estaba viendo y la forma de navegar por sus contenidos.

- **Diseño**

    - ¿El tamaño de la fuente del texto es lo suficientemente legible? ¿Se utiliza un texto con suficiente contraste para que sea lo más legible posible?

    - ¿Las fechas se muestran en formato internacional?

    - ¿Hay imágenes con marcas de agua, es decir, hay imágenes con texto encima?

    - ¿Los elementos más importantes de la aplicación están visibles en su totalidad sin tener que desplazar verticalmente la pantalla?

    - ¿Le pareció adecuada la forma en que se muestran las imágenes en la pantalla del dispositivo móvil? ¿Son nítidas? ¿Son adecuadas para representar el contenido?

    - ¿Se fijó si el sitio tenía gráficas con animaciones? ¿Hay alguna que le haya llamado la atención? ¿Ninguna?

    - ¿Considera que gráficamente el sitio está equilibrado, muy simple o recargado?

- ¿Recuerda si el sitio tenía banners (avisos) publicitarios? ¿Tuvo intención o llegó a hacer clic sobre alguno? ¿Por qué le hizo clic? ¿Qué le llamó la atención?

- **Contenido**

  - ¿Le parece adecuada la selección de contenidos destacados en la pantalla inicial o usted echó de menos otras áreas de información que le habría gustado ver destacadas?

  - ¿Es posible saber cuando fue la fecha de la última actualización de la aplicación?

  - ¿Los textos usados en los contenidos de las pantallas son suficientemente descriptivos? ¿Se usa un lenguaje dirigido al usuario?

  - En caso de haber información relacionada con la que estaba viendo, ¿se le ofreció de manera simple? ¿O tuvo que volver a otra pantalla para encontrarla?

  - ¿Hay contenido repetido en distintos lugares de la pantalla?

  - ¿Se utilizan correctamente las reglas de estilo de redacción (gramática / ortografía)?

- **Navegación**

  - ¿Puede ver en la pantalla inicial y en las demás pantallas, la forma en que se navega por la aplicación?

  - ¿Existen elementos dentro de las pantallas, que le permiten saber exactamente dónde se encuentra dentro de la aplicación y cómo volver atrás sin necesidad de usar los botones físicos del dispositivo?

  - ¿Cómo vuelve desde cualquier pantalla a la pantalla inicial? ¿Existe alguna otra forma de hacerlo? ¿Le parece claro? ¿Echa de menos alguno?

  - La aplicación tiene varios niveles de navegación y usted ha ingresado y salido de varios de ellos. ¿La información que se le ofrece en pantalla le parece adecuada para entender dónde está ubicado en cualquier momento? ¿Se ha sentido perdido dentro de la aplicación? ¿Si se ha sentido perdido, recuerda en qué área fue? ¿Si no se ha sentido perdido, qué elemento del sitio cree que le ayudó más a orientarse?

- **Búsqueda**

    - ¿Distinguió si en la aplicación se ofrecía un buscador? ¿Dónde está?

    - (Antes de usar el buscador) ¿Cómo haría la operación de buscar? ¿Qué escribiría? ¿Dónde lo escribiría?

    - (Antes de presionar el botón de buscar) ¿Qué espera encontrar?

    - (Al ver la página de resultados) ¿Eso es lo que esperaba encontrar? ¿Le sirve?

    - ¿El cuadro de entrada de texto para la búsqueda, ocupa entre 15 y 30 caracteres visibles?

- **Feedback**

    - ¿Encuentra alguna forma *online* u *offline* de ponerse en contacto con el Administrador de la aplicación, para hacer sugerencias o comentarios?

    - ¿Al mandar datos mediante un formulario, el sistema le avisa si los recibió correctamente o no?

- **Utilidad**

    - ¿Tras un primer vistazo, le queda claro cuál es el objetivo de la aplicación? ¿Qué contenidos y servicios ofrece? ¿Los puede enumerar?

    - ¿Cree que los contenidos y servicios que se ofrecen en la aplicación son de utilidad para su caso personal? ¿Recomendaría la aplicación a algún amigo o familiar?

    - ¿Qué es lo que más te llamó la atención positivamente o negativamente de la utilidad que ofrece la *app*?

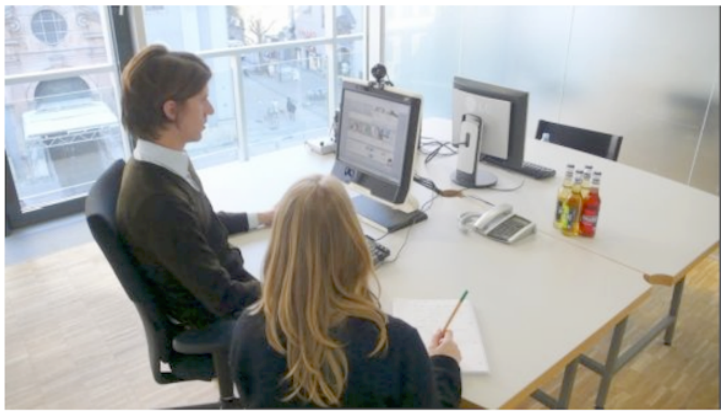


Figura 32: escena de una prueba con usuarios

3.5. DISEÑO TÉCNICO

DIAGRAMA *UML*

Aunque se ha utilizado una *BBDD NoSQL* (*CouchDB*), me he basado en las bases de datos relacionales precisamente para modelar los datos.

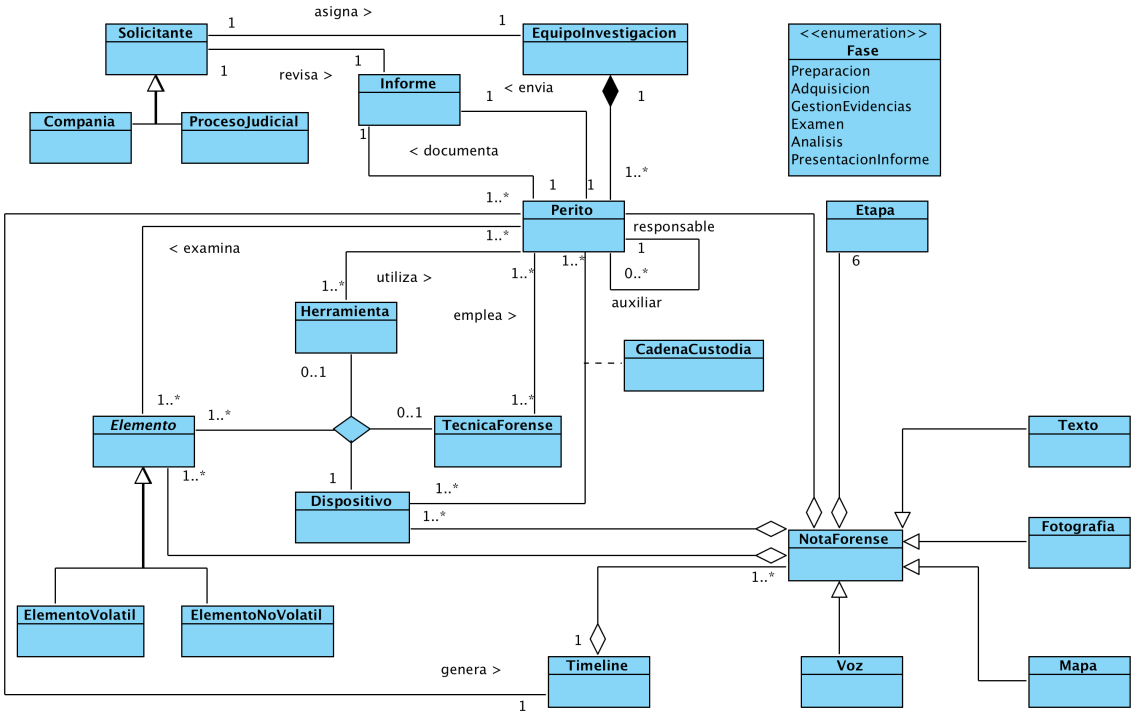


Figura 33: diagrama de clases

Cada caso, dispositivo o nota se salva como una única fila, que contiene toda la información que la describe. Ésto es lo que se llama “contenidos en sí mismos” o “autocontenidos” (*self-contained*”), y es un concepto muy importante a la hora de entender las bases de datos de documentos como *CouchDB*.

La mayoría de los casos, dispositivos y notas contienen más o menos la misma información, que contienen las tablas en un modelo relacional. La forma exacta de esta información puede variar, pero la información representada en general se mantiene igual, y podemos fácilmente reconocer que es un caso, un dispositivo o una nota, dado que se describen como documentos del mundo real.

Una nota de mapa puede tener un campo *coords* (coordenadas), pero no tener un campo *src* (ruta de la imagen de la fotografía). El mero hecho de omitir un campo u otro implica que no tiene.

Podemos ver que los documentos tienen a tener una semántica parecida (el mismo tipo de información), pero pueden variar sustancialmente en sintaxis, o en la estructura de esa información. Los seres humanos manejamos con facilidad este tpo de variación.

Por tanto, mientras que tradicionalmente en una base de datos relacional tenemos que modelar los datos de antemano, el diseño *shema-free* de *CouchDB* nos permite agregar los datos después de creado el documento, tal y como lo haríamos con documentos de verdad.

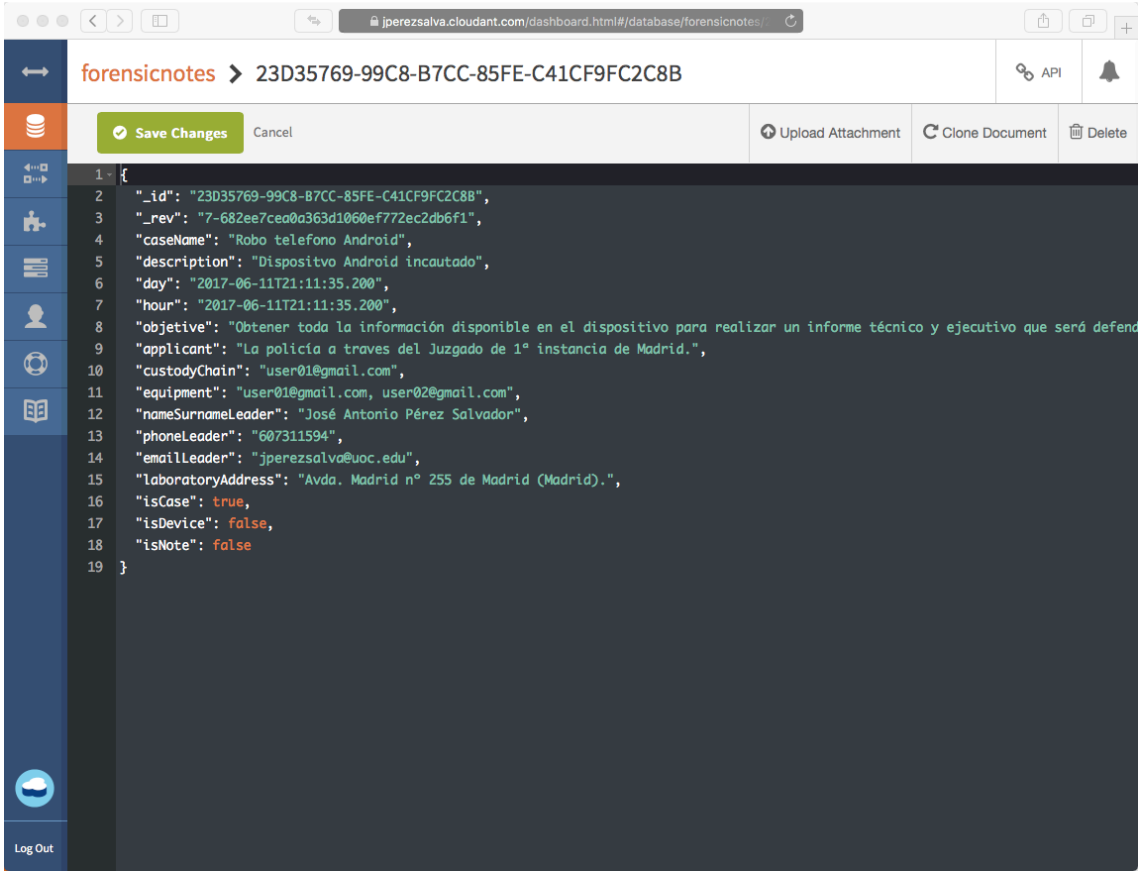


Figura 34: esquema del documento de casos

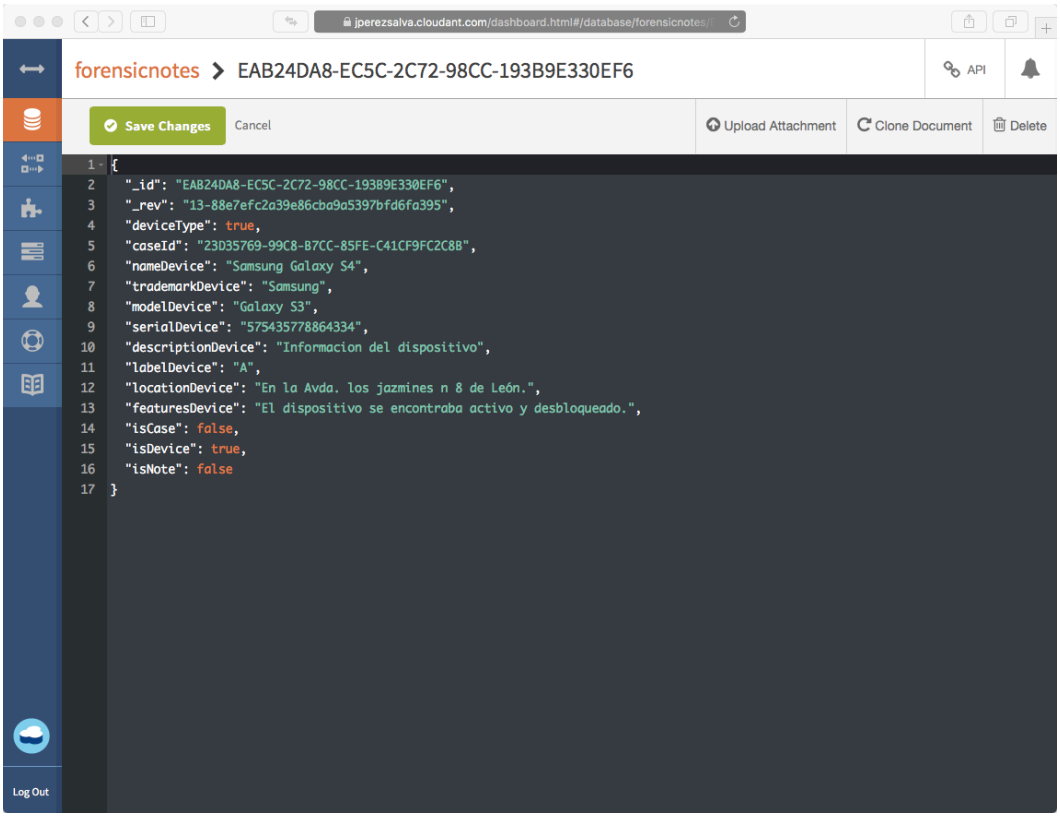


Figura 35: esquema del documento de dispositivos

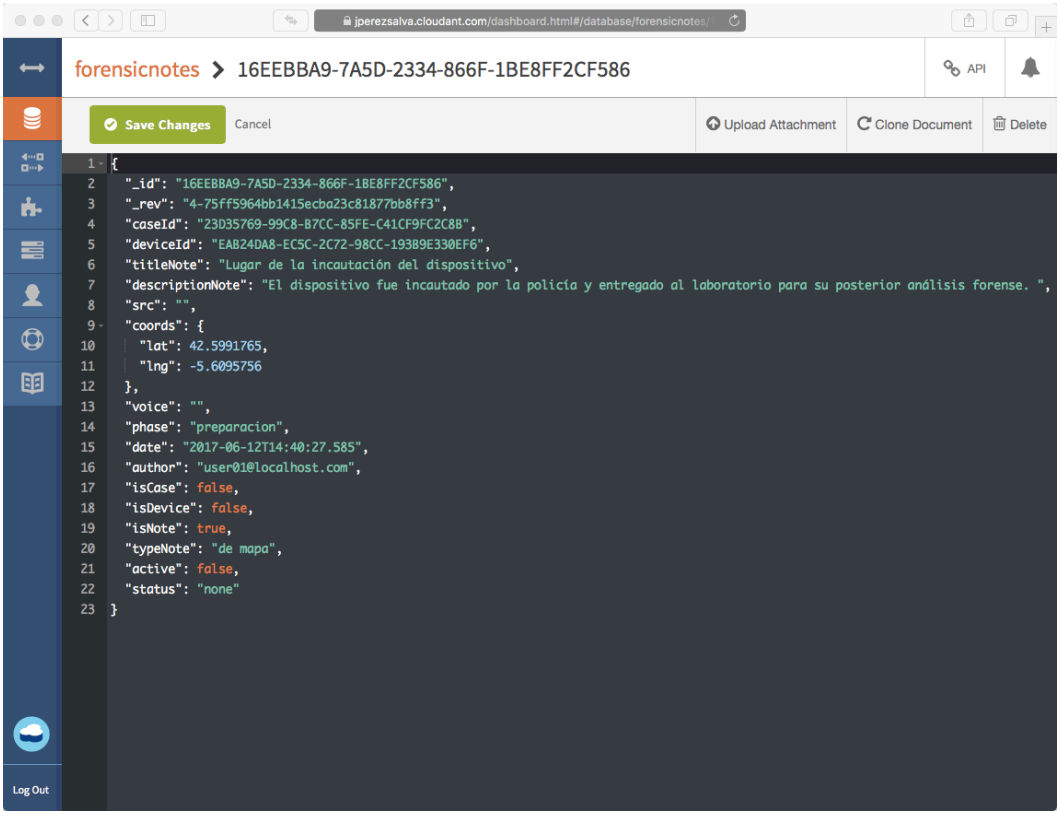


Figura 36: esquema del documento de notas (notas de mapa)

CouchdB almacena los datos en formato *JSON* (*JavaScript Object Notation*), el cual proporciona una alternativa más fácil de usar que *XML*, para serializar los datos de forma que puedan ser enviados a través de una red.

Además, *CouchDB* es muy flexible y nos proporciona suficientes componentes para crear sistemas que se adapten a las necesidades del problema que tengamos que solucionar. Esto no quiere decir que *CouchDB* pueda amoldarse a cualquier problema, pero en el área de almacenamiento de datos, nos puede llevar lejos.

Otro punto importante, es la replicación. Su función fundamental es sincronizar dos o más bases de datos de *CouchDB*. Esto puede sonar simple, pero la simplicidad es la pieza clave para permitir que la replicación solvente varios problemas: sincronizar de manera fiable bases de datos entre múltiples máquinas, para tener almacenamiento redundante; distribuir datos a un cluster de instancias de *CouchDB*, que comparten un subconjunto del total de consultas que llegan a ese *cluster* (balance de carga); y distribuir datos entre lugares geográficamente distantes.

La replicación en *CouchDB* usa la misma *REST API* que todos los demás clientes. El protocolo *HTTP* es transparente y está estudiado en profundidad. La replicación funciona de forma incremental; esto es, que si cualquier cosa sale mal durante la replicación, como una desconexión de red, continuará donde se cortó la próxima vez que se ejecute (gracias a la librería *PouchDB*). Del mismo, solamente transfiere los datos necesarios, para sincronizar las bases de datos.

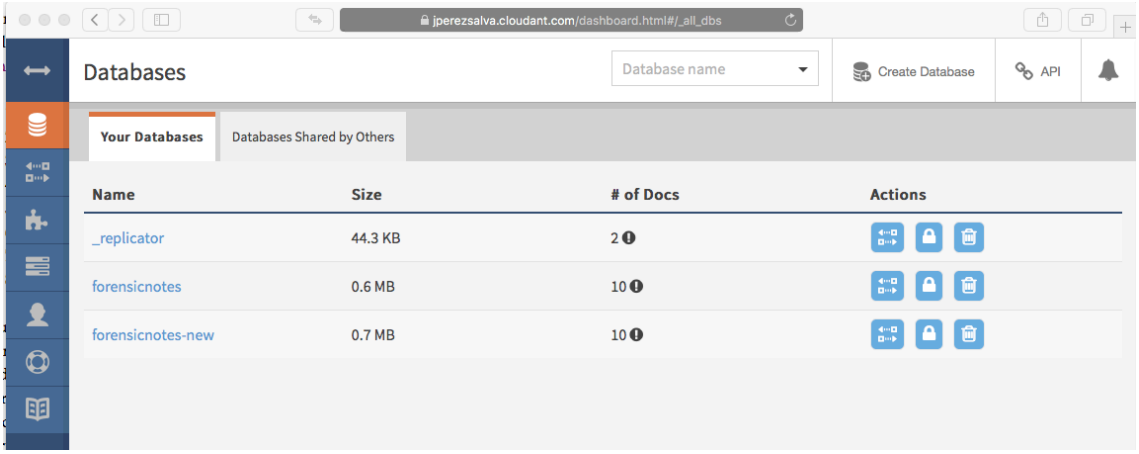


Figura 37: proceso de replicación en *CouchDB 1*

Una de las presunciones fundamentales de *CouchDB* es que las cosas pueden salir mal, la red se puede caer, y está diseñado para recuperarse de errores en vez de asumir que todo irá bien. El diseño incremental del sistema de replicación es lo que mejor representa esta idea.

Algunas herramientas existentes tratan de esconder el hecho de que hay una red y de que cualquiera de las condiciones anteriores o todas ellas pueden no darse en un sistema en particular. Esto normalmente resulta en errores fatales donde al final algo sale mal de verdad. En contraste, *CouchDB* no intenta esconder la red; soporta errores sin caerse y permite intervenir cuando sea necesario.

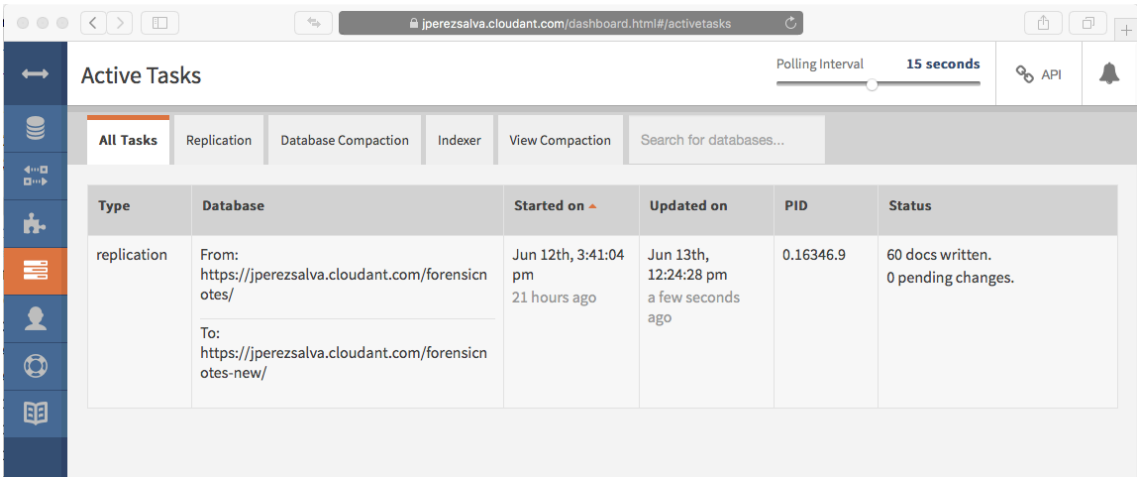


Figura 38: proceso de replicación en CouchDB 2

Para el desarrollo de este proyecto se ha utilizado *Cloudant* (es decir, *CouchDB* en la nube), que es una plataforma de datos *NoSQL* escalable (*DBaaS*), creada para *cloud*.

Por tanto, para poder utilizar el servicio y crear la *BBDD* del proyecto he tenido que visitar la página *https://cloudant.com*.

Una vez registrado en el sistema, he creado una *BBDD* llamada "ForensicNotes", y he anotado la información necesaria para poder acceder posteriormente a la misma desde la aplicación. Es decir, he anotado tanto el nombre de la base de datos ("ForensicNotes"), como el usuario ("jperezsalva") y su *password* ("J9dENDGp5dA6").

CASOS DE USO

A continuación presento los casos de uso. Las extensiones se numeran por la línea de la operación a la cual se aplican, seguida de una letra que permite distinguir las extensiones de una misma línea. Luego, se numeran las operaciones de una extensión de la misma forma que las operaciones del escenario principal.

Caso de uso	*Login* de un usuario.
Actor primario	Usuario.
Sistema	*ForensicNotes.*

Participantes	Usuario.
Nivel	Objetivo acceder al sistema.
Condición previa	El servidor debe estar conectado a una red local o a *Internet*.
Operaciones	
1	Introducir la *IP* del servidor.
2	Introducir el usuario de la *BBDD*.
3	Introducir el *password* de la *BBDD*.
4	Introducir el *email* y el *password*.
5	Acceder al sistema.
Extensiones	
1.A	¿Es correcta la dirección *IP* del servidor?
1.A.1	Si sí, continuar.
1.A.2	Si no, mostrar un mensaje: “la dirección del servidor no es correcta. Inténtelo de nuevo con otra *IP*”.
1.B	¿Se encuentra activo el servidor?
1.B.1	Si sí, continuar.
1.B.2	Si no, mostar un mensaje: “No ha sido posible conectarse al servidor. Inténtelo más tarde”.
2.A	¿Es correcto el usuario de la *BBDD*?
2.A.1	Si sí, continuar.
2.A.2	Si no, se guardan los datos en local (en *PouchDB*) y se actualizarán una vez que las credenciales sean correctas y se establezca conexión con la *BBDD CouchDB* (*Cloudant*) de nuevo.
3.A	¿Es correcto el *password* de la *BBDD*?
3.A.1	Si sí, continuar.
3.A.2	Si no, se guardan los datos en local (en *PouchDB*) y se actualizarán una vez que las credenciales sean correctas y se establezca conexión con la *BBDD CouchDB* (*Cloudant*) de nuevo.
4.A	¿Es correcto el email?
4.A.1	Si sí, continuar.
4.A.2	Si no, mostar un mensaje: “El email no fue encontrado”.
4.B	¿Es correcto el *password*?
4.B.1	Si sí, continuar.
4.B.2	Si no, mostrar un mensaje: “El email o el password no coinciden”.
4.C	¿Se encuentran cumplimentados tanto el *email*, como el *password*?
4.C.1	Si sí, volver a ejecutar las etapas 4.A y 4.B
4.C.2	Si no, mostrar un mensaje: “Debes enviar el email y el password”.
5.A	¿Es posible acceder al sistema?
5.A.1	Si sí, continuar.
5.A.2	Volver a ejecutar las etapas 1.A, 1.B, 4.A, 4.B y 4.C.

Caso de uso	Registro de un usuario.

Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Administrador del servidor.
Nivel	Objetivo dar de alta un usuario nuevo en el sistema.
Condición previa	El usuario no debe existir.
Operaciones	
1	Apagar el servidor.
2	Introducir manualmente el *id*, *email* y *password* del nuevo usuario en el fichero "*Server → user-routes.js"*.
3	Conectar el servidor.
Extensiones	

Caso de uso	Actualización de un usuario.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Administrador del servidor.
Nivel	Objetivo actualizar un usuario existente en el sistema.
Condición previa	El usuario debe existir.
Operaciones	
1	Apagar el servidor.
2	Modificar manualmente el *id*, *email* y *password* del usuario en el fichero "*Server → user-routes.js"*.
3	Conectar el servidor.
Extensiones	

Caso de uso	Eliminar un usuario.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Administrador del servidor.
Nivel	Objetivo dar de baja un usuario existente en el sistema.
Condición previa	El usuario debe existir.
Operaciones	
1	Apagar el servidor.
2	Borrar manualmente el *id*, *email* y *password* del usuario en "*Server → user-routes.js"*.
3	Conectar el servidor.
Extensiones	

Caso de uso	Eliminar el *token* de sesión.
Actor primario	Usuario
Sistema	*ForensicNotes*
Participantes	Usuario.
Nivel	Objetivo usuario
Condición previa	El usuario tiene acceso al sistema.
Operaciones	
1	Eliminar el *token* de sesión.

Extensiones	
1.A	¿Se ha eliminado el *token* de sesión?
1.A.1	Mostrar un mensaje: “Tu token ha expirado, por lo que serás redireccionado a la pantalla de login”.

Caso de uso	Salir de la aplicación.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo salir de la aplicación.
Condición previa	El usuario tiene acceso al sistema.
Operaciones	
1	Salir de la aplicación.
Extensiones	

Caso de uso	Visualizar casos.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar los casos.
Condición previa	El usuario tiene acceso al sistema y los casos deben existir.
Operaciones	
1	Visualizar casos.
Extensiones	
1	¿Existen casos?
1.A.1	Si sí, mostrar por pantalla los casos.
1.A.2	Si no, se vuelve a ejecutar la etapa 1.B
1.B	Una vez refrescada la pantalla, ¿se visualizan los casos?
1.B.1	Si sí, mostrar por pantalla los casos.
1.B.2	Si no, se muestra la pantalla vacía.

Caso de uso	Crear un caso.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo crear un caso nuevo.
Condición previa	El usuario tiene acceso al sistema.
Operaciones	
1	Introducir el nombre del caso.
2	Introducir la descripción del caso.
3	Introduccir la fecha de creación del caso.
4	Introducir la hora de creación del caso.
5	Introducir el objetivo del caso.
6	Introducir la persona que solicita el análisis forense.
7	Introducir la persona responsable de la cadena de

	custodia.
8	Introducir los miembros del equipo.
9	Introducir la persona responsable del caso.
10	Introducir el número de teléfono.
11	Introducir la dirección de correo electrónico.
12	Introducir la dirección del laboratorio forense.
13	Crear el caso.
Extensiones	
1.A	¿Se ha introducido el nombre del caso?
1.A.1	Si sí, se habilita el botón aceptar del formulario.
1.A.2	Si no, no se habilita el botón aceptar del formulario.
2.A	¿Se ha introducido la descripción del caso?
2.A.1	Si sí, se habilita el botón aceptar del formulario.
2.A.2	Si no, no se habilita el botón aceptar del formulario.
6.A	¿Se ha introducido la persona que solicita el análisis forense?
6.A.1	Si sí, se habilita el botón aceptar del formulario.
6.A.2	Si no, no se habilita el botón aceptar del formulario.
9.A	¿Se ha introducido la persona responsable de la cadena de custodia?
9.A.1	Si sí, se vuelve a ejecutar la etapa 7.B.1.
9.A.2	Si no, no se habilita el botón aceptar del formulario.
9.B	¿Son válidos el nombre y apellidos?
9.B.1	Si sí, se habilita el botón aceptar del formulario.
9.B.2	Si no, mostrar un mensaje: “El nombre y apellidos deben ser válidos”.
10.A	¿Se ha introducido el número de teléfono?
10.A.1	Si sí, se vuelve a ejecutar la etapa 10.B.1.
10.A.2	Si no, no se habilita el botón aceptar del formulario.
10.B	¿Es correcto el número de teléfono?
10.B.1	Si sí, se habilita el botón aceptar del formulario.
10.B.2	Si no, mostrar un mensaje: “Debes introducir un número de teléfono válido”.
11.A	¿Se ha introducido la dirección de correo electrónico?
11.A.1	Si sí, se vuelve a ejecutar la etapa 11.B.1.
11.A.2	Si no, no se habilita el botón aceptar del formulario.
11.B	¿Es correcto el formato del *email*?
11.B.1	Si sí, se habilita el botón aceptar del formulario.
11.B.2	Si no, mostrar un mensaje: “Debes introducir una dirección de correo electrónico válida”.
12.A	¿Son correctos todos los campos?
12.A.1	Si sí, mostrar un mensaje: “Caso creado correctamente”.
12.A.2	Si no, no se habilita el botón aceptar.

Caso de uso	Actualizar un caso.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.

Nivel	Objetivo actualizar un caso.
Condición previa	El usuario tiene acceso al sistema y el caso debe existir.
Operaciones	
1	Actualizar el nombre del caso.
2	Actualizar la descripción del caso.
3	Actualizar la fecha de creación del caso.
4	Actualizar la hora de creación del caso.
5	Actualizar el objetivo del caso.
6	Actualizar la persona que solicita el análisis forense.
7	Actualizar la persona responsable de la cadena de custodia.
8	Actualizar los miembros del equipo.
9	Actualizar la persona responsable del caso.
10	Actualizar el número de teléfono.
11	Actualizar la dirección de correo electrónico.
12	Actualizar la dirección del laboratorio forense.
13	Actualizar el caso.
Extensiones	
1.A	¿Se ha actualizado el nombre del caso?
1.A.1	Si sí, continuar.
1.A.2	Si no, se deshabilita el botón aceptar del formulario.
2.A	¿Se ha introducido la descripción del caso?
2.A.1	Si sí, continuar.
2.A.2	Si no, se deshabilita el botón aceptar del formulario.
6.A	¿Se ha introducido la persona que solicita el análisis forense?
6.A.1	Si sí, continuar.
6.A.2	Si no, se deshabilita el botón aceptar del formulario.
9.A	¿Se ha actualizado la persona responsable del caso?
9.A.1	Si sí, se vuelve a ejecutar la etapa 9.B.1.
9.A.2	Si no, no se habilita el botón aceptar del formulario.
9.B	¿Son válidos el nombre y apellidos?
9.B.1	Si sí, se habilita el botón aceptar del formulario.
9.B.2	Si no, mostrar un mensaje: “El nombre y apellidos deben ser válidos”.
10.A	¿Se ha actualizado el número de teléfono?
10.A.1	Si sí, se vuelve a ejecutar la etapa 10.B.1.
10.A.2	Si no, no se habilita el botón aceptar del formulario.
10.B	¿Es correcto el número de teléfono?
10.B.1	Si sí, se habilita el botón aceptar del formulario.
10.B.2	Si no, mostrar un mensaje: “Debes introducir un número de teléfono válido”.
11.A	¿Se ha actualizado la dirección de correo electrónico?
11.A.1	Si sí, se vuelve a ejecutar la etapa 11.B.1.
11.A.2	Si no, no se habilita el botón aceptar del formulario.
11.B	¿Es correcto el formato del *email*?
11.B.1	Si sí, se habilita el botón aceptar del formulario.
11.B.2	Si no, mostrar un mensaje: “Debes introducir una dirección de correo electrónico válida”.

12.A	¿Son correctos todos los campos?
12.A.1	Si sí, mostrar un mensaje: “Caso actualizado correctamente”.
12.A.2	Si no, continuar.

Caso de uso	Borrar un caso.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar un caso
Condición previa	El usuario tiene acceso al sistema.
Operaciones	
1	Borrar un caso.
Extensiones	
1.A	¿Existen dispositivos asociados a un caso?
1.A.1	Si sí, mostrar un mensaje: “Existen dispositivos pendientes de eliminar”.
1.A.2	Si no, mostrar un mensaje: “Caso eliminado correctamente”.

Caso de uso	Visualizar dispositivos.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar dispositivos.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como los dispositivos deben existir.
Operaciones	
1	Visualizar dispositivos.
Extensiones	
1	¿Existen dispositivos?
1.A.1	Si sí, mostrar por pantalla los dispositivos.
1.A.2	Si no, se vuelve a ejecutar la etapa 1.B
1.B	Una vez refrescada la pantalla, ¿se visualizan los dispositivos?
1.B.1	Si sí, mostrar por pantalla los dispositivos.
1.B.2	Si no, se muestra la pantalla vacía.
1.C	¿Es físico o volátil?
1.C.1	Si es físico, mostrar el icono de un dispositivo móvil.
1.C.2	Si es volátil, mostrar un icono con corchetes.

Caso de uso	Crear un dispositivo.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo crear un dispositivo nuevo.

Condición previa	El usuario tiene acceso al sistema y el caso debe existir.
Operaciones	
1	Indicar el tipo de dispositivo (físico o volátil).
2	Introducir el nombre del dispositivo.
3	Introducir la descripción del dispositivo.
4	Introduccir la etiqueta del dispositivo.
5	Introducir la localización del dispositivo.
6	Introducir las características del dispositivo.
7	Crear el dispositivo.
Extensiones	
2.A	¿Se ha introducido el nombre del dispositivo?
2.A.1	Si sí, se habilita el botón aceptar del formulario.
2.A.2	Si no, no se habilita el botón aceptar del formulario.
3.A	¿Se ha introducido la descripción del dispositivo?
3.A.1	Si sí, se habilita el botón aceptar del formulario.
3.A.2	Si no, no se habilita el botón aceptar del formulario.
4.A	¿Se ha introducido la etiqueta del dispositivo?
4.A.1	Si sí, se habilita el botón aceptar del formulario.
4.A.2	Si no, no se habilita el botón aceptar del formulario.
5.A	¿Se ha introducido la localización del dispositivo?
5.A.1	Si sí, se habilita el botón aceptar del formulario.
5.A.2	Si no, no se habilita el botón aceptar del formulario.
6.A	¿Se han introducido las características del dispositivo?
6.A.1	Si sí, se habilita el botón aceptar del formulario
6.A.2	Si no, no se habilita el botón aceptar del formulario.
7.A	¿Son correctos todos los campos?
7.A.1	Si sí, mostar el mensaje: “Dispositivo añadido correctamente”.
7.A.2	Si no, no se habilita el botón aceptar.

Caso de uso	Actualizar un dispositivo.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo actualizar un dispositivo existente..
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo debe existir.
Operaciones	
1	Actualizar el tipo de dispositivo (físico o volátil).
2	Actualizar el nombre del dispositivo.
3	Actualizar la descripción del dispositivo.
4	Actualizar la etiqueta del dispositivo.
5	Actualizar la localización del dispositivo.
6	Actualizar las características del dispositivo.
7	Actualizar el dispositivo.
Extensiones	
2.A	¿Se ha actualizado el nombre del dispositivo?

2.A.1	Si sí, continuar.
2.A.2	Si no, no se habilita el botón aceptar del formulario.
3.A	¿Se ha actualizado la descripción del dispositivo?
3.A.1	Si sí, continuar.
3.A.2	Si no, no se habilita el botón aceptar del formulario.
4.A	¿Se ha actualizado la etiqueta del dispositivo?
4.A.1	Si sí, continuar.
4.A.2	Si no, no se habilita el botón aceptar del formulario.
5.A	¿Se ha actualizado la localización del dispositivo?
5.A.1	Si sí, continuar.
5.A.2	Si no, no se habilita el botón aceptar del formulario.
6.A	¿Se han actualizado las características del dispositivo?
6.A.1	Si sí, continuar.
6.A.2	Si no, no se habilita el botón aceptar del formulario.
7.A	¿Son correctos todos los campos?
7.A.1	Si sí, mostar el mensaje: “Dispositivo actualizado correctamente”.
7.A.2	Si no, no se habilita el botón aceptar.

Caso de uso	Borrar un dispositivo.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar un dispositivo.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo debe existir.
Operaciones	
1	Borrar un dispositivo.
Extensiones	
1.A	¿Existen dispositivos asociados a un caso?
1.A.1	Si sí, mostrar un mensaje: “Existen notas pendientes sin eliminar”.
1.A.2	Si no, mostrar un mensaje: “Dispositivo eliminado correctamente”.

Caso de uso	Visualizar notas.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar notas.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, el dispositivo y las notas deben existir.
Operaciones	
1	Visualizar notas.
Extensiones	
1	¿Existen notas?

1.A.1	Si sí, mostrar por pantalla las notas.
1.A.2	Si no, se vuelve a ejecutar la etapa 1.B
1.B	Una vez refrescada la pantalla, ¿se visualizan las notas?
1.B.1	Si sí, mostrar por pantalla las notas.
1.B.2	Si no, se muestra la pantalla vacía.
1.C	¿Están marcadas las notas como importantes?
1.C.1	Si sí, las notas tienen un color mostaza.
1.C.2	Si no, las notas tienen un color azul cielo.

Caso de uso	Crear notas de audio.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo crear notas de audio.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo deben existir.
Operaciones	
1	Grabar nota de audio.
2	Insertar el título de la nota.
3	Insertar la descripción de la nota.
4	Indicar la fase en la que fue tomada la nota.
5	Añadir la nota.
Extensiones	
1	¿Es un dispositivo móvil?
1.A.1	Si sí, vuelve a ejecutar la etapa 1.B.
1.A.2	Si no, mostrar un mensaje: “Acción invalidada”.
1.B	¿Tiene permisos de grabación?
1.B.1	Si sí, grabar audio.
1.B.2	Si no, mostrar un mensaje del sistema solicitando permisos.
2.A	¿Se ha insertado el título de la nota?
2.A.1	Si sí, continuar.
2.A.2	Si no, el botón aceptar permanece deshabilitado.
3.A	¿Se ha insertado el título de la nota?
3.A.1	Si sí, continuar.
3.A.2	Si no, el botón aceptar permanece deshabilitado.
5.A	¿Son correctos todos los campos?
5.A.1	Si sí, mostrar un mensaje: “Nota añadida correctamente”.
5.A.2	Si no, volver a ejecutar las etapas 2.A y 3.A.

Caso de uso	Visualizar notas de audio.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar notas de audio.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de audio deben existir.

Operaciones	
1	Visualizar la transcripción del discurso.
2	Visualizar el título de la nota de audio.
3	Visualizar la descripción de la nota de audio.
4	Visualizar la fase en la que fue tomada la nota de audio.
Extensiones	

Caso de uso	Borrar una nota de audio.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar una nota de audio.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de audio deben existir.
Operaciones	
1	Borrar una nota de audio.
Extensiones	
1.A	¿Se ha borrado la nota?
1.A.1	Mostrar un mensaje: “Nota eliminada correctamente”.

Caso de uso	Crear notas fotográficas.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo crear notas fotográficas.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo deben existir.
Operaciones	
1	Capturar fotografía o elegir fotografía de la galería de imágenes.
2	Insertar el título de la nota.
3	Insertar la descripción de la nota.
4	Indicar la fase en la que fue tomada la nota.
5	Añadir la nota.
Extensiones	
1	¿Es un dispositivo móvil?
1.A.1	Si sí, vuelve a ejecutar la etapa 1.B.
1.A.2	Si no, mostrar un mensaje: “Acción invalidada”.
1.B	¿Tiene permisos para capturar imágenes o acceder a la galería?
1.B.1	Si sí, capturar o seleccionar imagen.
1.B.2	Si no, mostrar un mensaje del sistema solicitando permisos.
2.A	¿Se ha insertado el título de la nota?
2.A.1	Si sí, continuar.
2.A.2	Si no, el botón aceptar permanece deshabilitado.
3.A	¿Se ha insertado el título de la nota?

3.A.1	Si sí, continuar.
3.A.2	Si no, el botón aceptar permanece deshabilitado.
5.A	¿Son correctos todos los campos?
5.A.1	Si sí, mostrar un mensaje: “Nota añadida correctamente”.
5.A.2	Si no, volver a ejecutar las etapas 2.A y 3.A.

Caso de uso	Visualizar notas fotográficas.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar notas fotográficas.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota fotográfica deben existir.
Operaciones	
1	Visualizar la fotografías.
2	Visualizar el título de la nota fotográfica.
3	Visualizar la descripción de la nota fotográfica.
4	Visualizar la fase en la que fue tomada la nota fotográfica.
Extensiones	

Caso de uso	Borrar una nota fotográfica.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar una nota fotográfica.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota fotográfica deben existir.
Operaciones	
1	Borrar una nota fotográfica.
Extensiones	
1.A	¿Se ha borrado la nota?
1.A.1	Mostrar un mensaje: “Nota eliminada correctamente”.

Caso de uso	Crear notas de mapa.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo crear notas de mapa.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo deben existir.
Operaciones	
1	Añadir marcador en el mapa.
2	Insertar el título de la nota.
3	Insertar la descripción de la nota.
4	Indicar la fase en la que fue tomada la nota.

5	Añadir la nota.
Extensiones	
1	¿Es un dispositivo móvil?
1.A.1	Si sí, se pueden obtener las coordenadas geográficas del dispositivo.
1.A.2	Si no, mostrar un mensaje: “Error: no ha sido posible obtener la localización”. Por tanto, se utilizan las coordenadas establecidas por defecto.
2.A	¿Se ha insertado el título de la nota?
2.A.1	Si sí, continuar.
2.A.2	Si no, el botón aceptar permanece deshabilitado.
3.A	¿Se ha insertado el título de la nota?
3.A.1	Si sí, continuar.
3.A.2	Si no, el botón aceptar permanece deshabilitado.
5.A	¿Son correctos todos los campos?
5.A.1	Si sí, mostrar un mensaje: “Nota añadida correctamente”.
5.A.2	Si no, volver a ejecutar las etapas 2.A y 3.A.

Caso de uso	Visualizar notas de mapa.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar notas de mapa.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de mapa deben existir.
Operaciones	
1	Visualizar el mapa.
2	Visualizar la latitud y la longitud.
2	Visualizar el título de la nota de mapa.
3	Visualizar la descripción de la nota de mapa.
4	Visualizar la fase en la que fue tomada la nota de mapa.
Extensiones	

Caso de uso	Borrar una nota de mapa.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar una nota de mapa.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de mapa deben existir.
Operaciones	
1	Borrar una nota de mapa.
Extensiones	
1.A	¿Se ha borrado la nota?
1.A.1	Mostrar un mensaje: “Nota eliminada correctamente”.

Caso de uso	Crear notas de texto.
Actor primario	Usuario
Sistema	*ForensicNotes*
Participantes	Usuario.
Nivel	Objetivo crear notas de texto.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo deben existir.
Operaciones	
1	Insertar el título de la nota de texto.
2	Insertar la descripción de la nota de texto.
3	Indicar la fase en la que fue tomada la nota de texto.
4	Añadir la nota de texto.
Extensiones	
1.A	¿Se ha insertado el título de la nota de texto?
1.A.1	Si sí, continuar.
1.A.2	Si no, el botón aceptar permanece deshabilitado.
2.A	¿Se ha insertado el título de la nota de texto?
2.A.1	Si sí, continuar.
2.A.2	Si no, el botón aceptar permanece deshabilitado.
4.A	¿Son correctos todos los campos?
4.A.1	Si sí, mostrar un mensaje: “Nota añadida correctamente”.
4.A.2	Si no, volver a ejecutar las etapas 2.A y 3.A.

Caso de uso	Visualizar notas de texto.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo visualizar notas de texto.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de texto deben existir.
Operaciones	
1	Visualizar el título de la nota de texto.
2	Visualizar la descripción de la nota de texto.
3	Visualizar la fase en la que fue tomada la nota de texto.
Extensiones	

Caso de uso	Borrar una nota de texto.
Actor primario	Usuario.
Sistema	*ForensicNotes.*
Participantes	Usuario.
Nivel	Objetivo borrar una nota de texto.
Condición previa	El usuario tiene acceso al sistema y tanto el caso, como el dispositivo y la nota de texto deben existir.
Operaciones	
1	Borrar una nota de texto.
Extensiones	
1.A	¿Se ha borrado la nota de texto?

1.A.1	Mostrar un mensaje: “Nota eliminada correctamente”.

4. HERRAMIENTAS

CLOUDANT

Cloudant es parte de la propuesta *Bluemix* de la plataforma de servicios *cloud* de *IBM*. Está basado en *Apache CouchDB* con algunas mejoras en el campo de autenticación, manejo de información geográfica y búsquedas. Digamos que es la implementación más segura y confiable de *CouchDB* con la gran ventaja de que podemos acceder al nivel gratis del servicio durante 30 días muy fácilmente.

POUCHDB

PouchDB es un proyecto que trata de implementar *CouchDB* en *JavaScript*. Debido a que el soporte de formatos en base de datos en móviles es muy variado, *PouchDB* funciona además como una capa de abstracción de almacenamiento aunque se mayor fortaleza es en la implementación del protocolo de sincronización.

NODE.JS

Es un intérprete *JavaScript* del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sóla máquina física.

IONIC

Es una herramienta para el desarrollo de aplicaciones basadas en *HTML5* con *JavaScript*. Está basado en *Apache Cordova* y permite desarrollar para *Android*, *iOS*, *Windows Phone* y muchas más. Actualmente, *Android* y *iOS* son soportados oficialmente.

Lo más destacable es que *Ionic* está basado en *Angular*, que es un *framework* para el desarrollo de aplicaciones *JavaScript* que facilita el desarrollo de aplicaciones basados en el modelo *MVC*.

ANGULAR

Angular es una solución completa que incluye prácticamente todos los aspectos que se necesitan para crear una aplicación cliente en *JavaScript*. Esto incluye la generación de vistas, el uso de *databinding*, el doble *binding*, las rutas, la organización de componentes, la comunicación con el servidor, etc...

Angular está diseñado para desarrollar aplicaciones, no páginas *web*, por lo que no tiene sentido compararlo con librerías como jQuery, Knockout, etc...

TYPESCRIPT

Para grandes desarrollos, con miles de líneas de código, toda ayuda es poca. *TypeScript* asume todas las mejoras y propuestas del más avanzado *JS* (*JavaScript*) estándar y además aporta tipos.

Esa es la principal razón de su elección como lenguaje en *Angular 2*.

Alrededor de esta piedra angular crece el ecosistema de herramientas. Principalmente, *VSCode* que lo aprovecha ofreciendo intellisense y *refactoring* a la altura de los grandes.

Cabe señalar que *TypeScript* no es ni mucho menos obligatorio. Se puede desarrollar en *ES5* (*EcmaScript 5*) y *ES6* sin problemas. Pero, la idea, los ejemplos, la documentación es mucho más sencilla con *TypeScript*.

VSCODE (VISUAL STUDIO CODE)

Es un editor de código, cuyas características se asemejan a otros, como *Geany* o el más reciente *Brackets* de *Adobe*. Soporta una cantidad considerable de lenguajes, ya sean propios de *Microsoft* o de otros, como *PHP*, *SQL*, *Java*, etc... También soporta *Git* y programación *web* con *HTML*, *CSS* y *JavaScript*, entre otros lenguajes.

VSCode es un software gratuito que cualquier usuario puede descargar en la siguiente dirección *https://code.visualstudio.com*.

SKETCH 3.0

Es una aplicación profesional propietaria para crear gráficos vectoriales e interfaces digitales, desarrollada por *Bohemian Coding*. Sustituye perfectamente a programas como *Adobe Photoshop*, *Illustrator* o *Fireworks*, para la mayoría de las tareas de diseño digital.

GOOGLE CHROME

Es un navegador *web* desarrollado por *Google* y compilado con base en varios componentes e infraestructuras de desarrollo de aplicaciones (*frameworks*) de código abierto, como el motor de renderizado *Blink*.

Se encuentra disponible gratuitamente bajo condiciones específicas del *software* privativo.

5. IMPLEMENTACIÓN

5.1. INSTALACIONES NECESARIAS

Para poder comenzar a trabajar en el proyecto, es necesario realizar una serie de instalaciones mínimas necesarias.

En este sentido, podemos empezar descargando e instalando *Node.js*, que es un entorno de ejecución para *JavaScript* construido con el motor de *JavaScript V8* de *Chrome*, el cual se encuentra disponible en *https://nodejs.org/es/*.

El navegador *web* que se ha utilizado para las pruebas y el desarrollo es *Google Chrome*, dado que permite probar diferentes opciones (sensores para cambiar la ubicación o hacer una simulación, por ejemplo) y pantallas. Pero, también es posible utilizar cualquier otro.

Otra herramienta que es recomendable instalar es *Angular CLI*. Para ello, basta con visitar la página *https://github.com/angular/angular-cli* y ejecutar este comando *npm install –g @angular/cli.*

Si estamos en *Windows* debemos abrir una ventana como administradores. Sin embargo, si nos encontramos en *Linux* o *Mac OS* debemos ejecutarlo con *sudo* de esta forma: *sudo npm install –g @angular/cli.*

En cuanto a la parte del editor de código, he utilizado *VSCode*, el cual se encuentra disponible en *https://code.visualstudio.com/*. He elegido este editor, dado que tiene un montón de *snippets* y características.

Indicar que la parte de *Ionic*, es muy sencilla. Simplemente hay que visitar la página de Ionic en *http://ionicframework.com* y hacer *click* en el botón “Get started”. Luego, bajando un poco tenemos que realizar la instalación ejecutando el comando *npm install –g cordova ionic*, como administrador, en una ventana del terminal.

Una vez finalizada la instalación, ya tendríamos las herramientas necesarias para poder trabajar en *Ionic*. Solamente, faltaría por seguir la guía que nos permita desplegar la aplicación en un dispositivo *Android* o *iOS* utilizando *Cordova*, en *http://ionicframework.com/docs/intro/installation/*.

5.2. ¿QUÉ ES *IONIC*?

Lo primero de todo, hay que aclarar un poquito el panorama.

Cuando nosotros queremos crear aplicaciones para *Windows Phone*, *Android* o *iOS* nosotros necesitamos saber al menos tres lenguajes de programación.

Es probable que nosotros sepamos uno o dos de esos lenguajes. Pero, cuando a nosotros se nos pide crear una aplicación o cuando nosotros tenemos un proyecto que necesitamos que funcione tanto en *iOS*, como en *Android*, eso nos reduce el panorama a saber *Java*, *Swift* u *Objetive C*.

Lo más probable es que nosotros ya conozcamos *HTML*, usando *CSS* y *JavaScript*.

Ahora, qué es *Ionic*. Pues *Ionic*, es un marco de trabajo muy parecido a *Bootstrap*. Es decir, tiene estilos, modals, tiene controles de navegación, tiene alertas. Es decir, tiene un montón de cosas muy interesantes que si nosotros seguimos, nuestra aplicación va a parecer una aplicación nativa, pero ejecutándose en un navegador *web*.

Claro, el producto final es coger todo el código *HTML*, *CSS* y *JavaScript* que escribimos y transformarlo en una aplicación nativa. Para eso, Ionic utiliza *Angular*, como motor de su aplicación.

Es decir, *Angular* es quien va a procesar la información de las peticiones *http*, va a procesar la data, va a conectarse a servicios, etc... Y *Ionic*, es el que se encarga de que la aplicación se mire como se ve en un dispositivo móvil.

Si nosotros seguimos esos requerimientos y lo pasamos por el framework de *Cordova*, terminaremos creando una aplicación totalmente nativa. Es decir, una aplicación que nosotros vamos a poder subir a la Play Store y a la *App Store*.

Es muy importante saber que *Ionic* 1 trabaja con *Angular.JS*. Es decir, la versión 1 de *Angular*.

La primera versión que salió y que es *Ionic* 2, utiliza lo que se conoce ahora como *Angular*. Es decir, la versión 2 o superior.

Con Ionic nosotros tenemos estilos, navegación, *tabs*, alertas, listas, *modals*. Trabajamos con *SASS* y también con *CSS*. Y básicamente, lo que tenemos es *HTML* puro, entre otras cosas. Pero, si nosotros le inyectamos lo que es *Cordova*, que trae una gran cantidad de plugins, nosotros tenemos acceso a lo que es la cámara, el *3D Touch*, los contactos, el giroscopio, la geolocalización, o podemos manejar archivos.

En fin, una gran cantidad de funcionalidades que agregan lo que son los plugins de *Cordova*, que básicamente es una pequeña librería que se utiliza de cierta manera y nos devuelva la información que solicitamos después de realizar un trabajo.

Ahora, referente a lo de *Angular*, es necesario conocer los componentes. Sobretodo, los componentes *HTTP*, los módulos, las directivas, los servicios,

los *pipes* y el *Data binding*, dado que ésto es el corazón de *Ionic*. Esto es lo que hace que funcione como tal una aplicación.

5.3. *IONIC* 2 E *IONIC* 3

Antes de comenzar a escribir líneas de código y empezar a crear la aplicación para dispositivos móviles es muy importante tener claro un concepto interesante: ¿qué es *Ionic* 2 y qué es *Ionic* 3?

Pues en realidad no hay que temer en este cambio. El cambio entre *Ionic* 2 y *Ionic* 3 no es tan drástico con respecto a lo que sucedió con *Ionic* 1 y *Ionic* 2.

La gente que conoce las dos versiones, sabe que se produjo un cambio muy radical, dado que cambió la sintaxis, cambió también la forma de trabajar, cambiaron muchas cosas.

Es algo muy parecido a lo que sucedió con *Angular*.*JS* y *Angular* 2 o *Angular* 4.

Lo primero de todo, indicar que es una simple actualización mayor. Es decir, es una actualización de *Ionic* 2 con algunas mejoras, que merece la pena marcarla como una nueva versión. Pero, todo el conocimiento de *Ionic* 2 es válido para *Ionic* 3.

Bueno, puede que en los cambios exista algo que ya vaya a ser obsoleto de cara al futuro, pero de momento eso no sucede.

Ahora bien, *Ionic* ya utiliza el sistema de versionamiento semántico (X.Y.Z), semejante al de *Angular*, dado que esto ya es un estándar que quiere decir que tendrá tres números: la ***X***, la ***Y*** y la ***Z***. La ***Z*** hay que entenderla como correcciones menores: algo que estaba saliendo mal o algo que se estaba procesando mal. Pero, no implica ningún cambio.

La ***Y*** significa que se han podido añadir características nuevas, pero no son lo suficientemente radicales, como para que sea una actualización mayor.

Sin embargo, la ***X***, como en el caso de *Ionic* 2 y *Ionic* 3, supone un cambio considerable. Entonces, es recomendable que nosotros investiguemos un poco o leamos cuando estos cambios suceden, porque puede darse la circunstancia de que el código que antes utilizábamos ahora deje de funcionar. Así que hay que estar muy atentos.

También tenemos algo muy parecido a lo que sucedió con *Angular* y las versiones que se encuentran actualmente. *Angular* va a ir progresando hacia las versiones 4, 5, 6 y así sucesivamente, pero será el mismo *framework*. Eso mismo, sucederá con Ionic.

Y por último, hay muchas mejoras en la versión 3 de *Ionic*, pero pocos cambios radicales (pero claramente los hay). Esto con el uso de los *plugins* cambia un poco, pero no hay nada que temer, porque en realidad el proyecto realizado

trabaja actualmente con la versión de *Ionic* 3, así que los cambios van a ser transparentes.

5.4. INTRODUCCIÓN A LA DOCUMENTACIÓN DE *IONIC*

Antes de comenzar a crear la aplicación hay que explicar que *Ionic* ha sido gratuito desde sus inicios.

En la página oficial de *Ionic*, que es *http://ionicframework.com/*, exactamente en la parte de la documentación hay mucha información relevante y muy elaborada, que el equipo de *Ionic* ha trabajado bastante para que sea un *framework* de calidad muy fácil de utilizar.

Cuando se trabaja con *Ionic* hay que asegurarse de que la versión de la documentación sea la *2*, porque en la versión *1* los ejemplos utilizan *Angular.JS* y a nosotros eso no nos sirve.

Existen muchas cosas interesantes sobre la documentación. Haciendo una breve introducción, tenemos los componentes, las *API* (que nos permiten visualizar los componentes, pero con un mayor grado de profundidad), el *Native* (que es un listado de *plugins* facilitados por *Cordova* y que nos permite utilizar los recursos nativos del dispositivo, como la cámara, la geolocalización, etc...), el *storage* (para poder almacenar la información en nuestro dispositivo y la *data* [los datos] queden persistentes allí), el *theming* (para cambiar el aspecto de la aplicación), el *ionicons* (que es una gran librería de iconos precargados que podemos utilizar libremente en la aplicación) y el *CLI* (que es el *Command Line Interface*, donde se pueden hacer muchas cosas, como emular el dispositivo *iOS* o *Android*, poder generar la aplicación, poder ejecutar la aplicación, poder montar un servidor para hacer pruebas, etc...)

Si se necesita instalar un *plugin*, lo único que hay que hacer es instalar el *plugin* mediante *CLI* y luego, hacer un *npm install –save* del *plugin*.

La documentación también nos indica los dispositivos que son soportados y nos muestra cómo vamos a usar cada uno de ellos, importando el *plugin* en el componente o servicio donde lo vayamos a utilizar, y luego inyectándolo en el constructor para poder trabajar con él.

Indicar que actualmente, el *CLI* de Ionic se volvió a actualizar *–v2* y *–v1*, ya no son soportadas.

Si intentamos ejecutar *–v2* aparecerá este error:

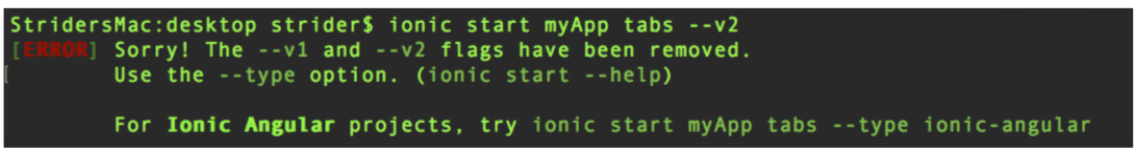


Figura 39: error al intentar *--v2* en *CLI*

Para solucionar este inconveniente, simplemente hay que omitir *–v2*. Es decir, poner por ejemplo: ***ionic start myApp*** en lugar de ***ionic start myApp –v2***.

Para más información sobre los comandos de *Ionic*, ver "referencia a los comandos más utilizados en Ionic" en el espacio de los anexos.

5.5. INSTALACIÓN DEL PROYECTO

Esta parte es solamente para explicar cómo se intala el proyecto, dado que no se va a realizar el despliegue en *Ionic View*, *Google Play* o en la *App Store*.

Haciendo *click* en "Get started" (en *http://ionicframework.com/getting-started/*) de la documentación, podemos ver cómo instalar *Ionic* mediante el comando *npm install –g cordova ionic* y cómo crear la aplicación.

Pero antes, necesitamos crear un directorio donde guardar la aplicación.

Si tuvieramos que crear un proyecto nuevo, tendríamos que abrir una ventana del terminal, y navegar al directorio mediante el comando *cd*, escribiendo en la línea de comandos *ionic start ForensicNotes blank*. Luego, pulsando *Enter,* comenzarían a descargar todos los paquetes necesarios para la instalación y crear los módulos de *Node.js*. Pero, como el proyecto ya está realizado simplemente hay que levantar el servidor e instalar la aplicación en un dispositivo móvil.

Indicar respecto a lo anterior que, *Ionic* es para llamar al *CLI*, *start* es para iniciar el proyecto, *ForensicNotes* es el nombre del proyecto y *blank* es la plantilla que le indica a *Ionic* que se trata de un proyecto vacío. Es decir, que no tiene absolutamente nada.

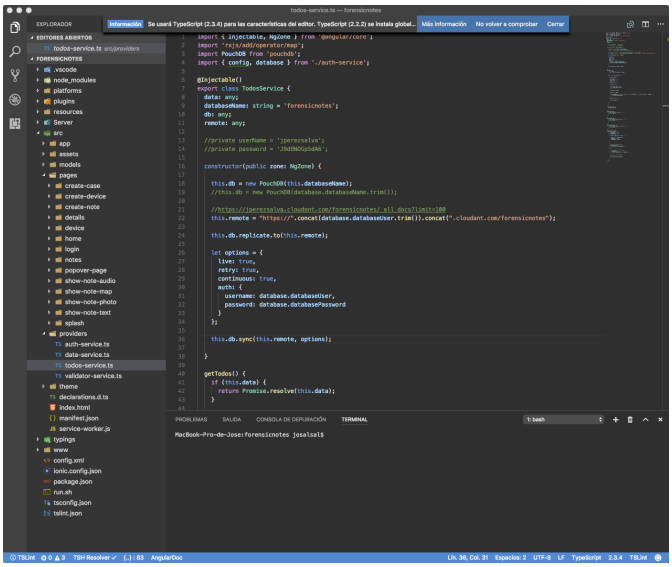


Figura 40: ventana del editor *VSCode*

Figura 37: ventana del editor *VSCode*

Una vez finalizada la instalación, hay que arrastrar el directorio donde se encuentra la aplicación a la ventana del editor *VSCode*, para poder trabajar con el proyecto.

Lo primero que tenemos que hacer es levantar el servidor en la red local. Para poder levantar el servidor, hay que situarse en el directorio *Server* mediante ***cd Server*** y luego escribir en la línea de comandos ***npm start***.

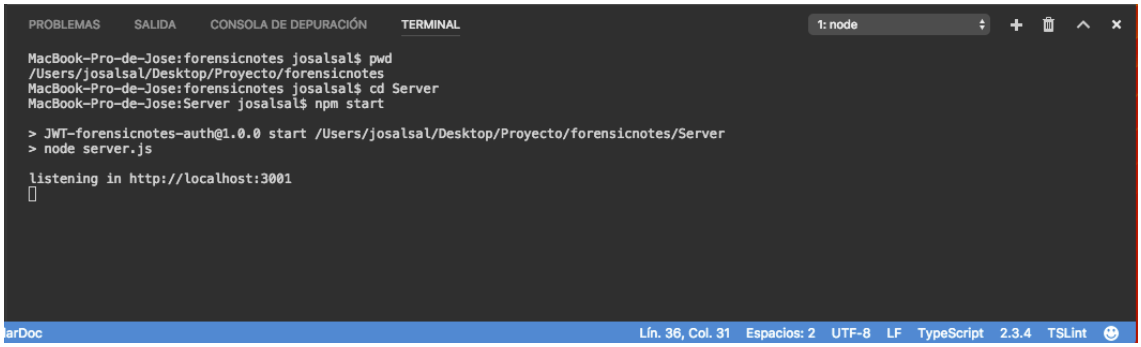


Figura 41: servidor en ejecución mediante *npm start*

Luego, tenemos que instalar la aplicación en el dispositivo móvil. Para ello, abrimos una nueva ventana en el terminal de *VSCode*, nos posicionamos en el directorio raíz "ForensicNotes" y escribimos el comando ***ionic cordova run android***.

También es posible ejecutar la aplicación en un navegador *web* al mismo tiempo que en el dispositivo móvil. Para ello, navegamos dentro de la carpeta "ForensicNotes" y escribimos el comando ***ionic serve***. Luego, abrimos o se nos abre una ventana del navegador en la dirección "http://localhost:8100".

Ionic serve va a levantar el servidor como un *livereload*. Cada vez que lo ejecutemos, va a volver a empaquetar la aplicación y vamos a poder ver en tiempo real los cambios que realicemos en la aplicación.

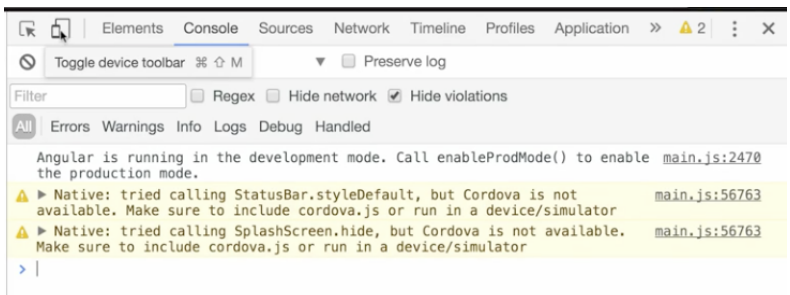


Figura 42: opciones para visualizar la aplicación como dispositivo en *Chrome*

Para poder finalizar la ejecución de la aplicación en el terminal, basta con pulsar la combinación de teclas *ctrl* + *c*.

5.6. ASPECTOS DESTACADOS DE LA IMPLEMENTACIÓN

NAVEGACIÓN

Ionic utiliza su propio sistema de navegación basándose en la navegación nativa de *iOS*.

El objetivo es poder navegar entre páginas, poder navegar entre páginas sólo con comandos *HTML*, tener eventos de regreso a la página anterior o regreso al *root*.

Por lo general, las aplicaciones tienen más de una pantalla, y cuando esto sucede tenemos que trabajar con la navegación.

¿Cómo funciona la navegación? Pues, cuando la aplicación es lanzada en algún lado del código le decimos a *Angular* ésta es la primera página de la aplicación, y será conocida como el *root*. Es decir, ésta es la primera página y no hay nada más.

Si nosotros quisieramos movernos a otra página, podemos ejecutar una función que existe dentro del módulo *navController* para que realice un *push*.

El *push* lo que hace es poner una pantalla nueva sobre la anterior. Esto va creando como una pila de tarjetas. Las páginas que desaparecen, siguen estando ahí, pero no es la página que está más arriba de la pila de targetas que tenemos ahí.

Ahora, el *navController* también tiene un método opuesto, que es el *pop*. El *pop* lo que va a hacer es coger la página que está situada más arriba y la va a eliminar. Luego, puedo volver a realizar otro *pop*, y la va a eliminar en orden consecutivo así como fueron creadas desde la página *root*.

```
160   showNote(note) {
161
162     if (note.typeNote === 'de mapa') {
163       this.navCtrl.push(ShowNoteMapPage, { typeNote: note.typeNote, currentNote: note });
164     } else if (note.typeNote === 'de texto') {
165       this.navCtrl.push(ShowNoteTextPage, { typeNote: note.typeNote, currentNote: note });
166     } else if (note.typeNote === 'de audio') {
167       this.navCtrl.push(ShowNoteAudioPage, { typeNote: note.typeNote, currentNote: note });
168     } else if (note.typeNote === 'fotográfica') {
169       this.navCtrl.push(ShowNotePhotoPage, { typeNote: note.typeNote, currentNote: note });
170     }
171   }
```

Figura 43: navegación en *Ionic*

PIPES

Es una característica propia de *Angular*, por lo que no tiene nada que ver con *Ionic*.

Un *pipe* es simplemente una transformación de la data (los datos), únicamente de forma visual. No cambia el valor, simplemente como se presentan los datos.

Para crear un pipe lo podemos hacer de varias formas. La forma más sencilla es crearlo con el generador de *CLI* escribiendo en el terminal *ionic g pipe nombre_del_pipe*.

Otras fórmulas, pasan por utilizar *Angular* o *JavaScript*.

SERVICIOS

Los servicios permiten organizar la lógica de la aplicación separándolos de las vistas o de los componentes.

En realidad, por separación de conceptos toda la parte lógica debería de estar separada en otro lado, dejando que los componentes se dediquen solamente a gestionar los temas relacionados con la vista.

Para crear un servicio, simplemente hay que escribir en el terminal el comando *ionic g provider nombre_del_servicio*.

Básicamente, se genera una clase inyectable con su constructor, que hay que declarar en el *app.module.ts*.

La principal ventaja de los servicios es que se utilizan de forma global en cualquier componente dentro de la aplicación de *Ionic*.

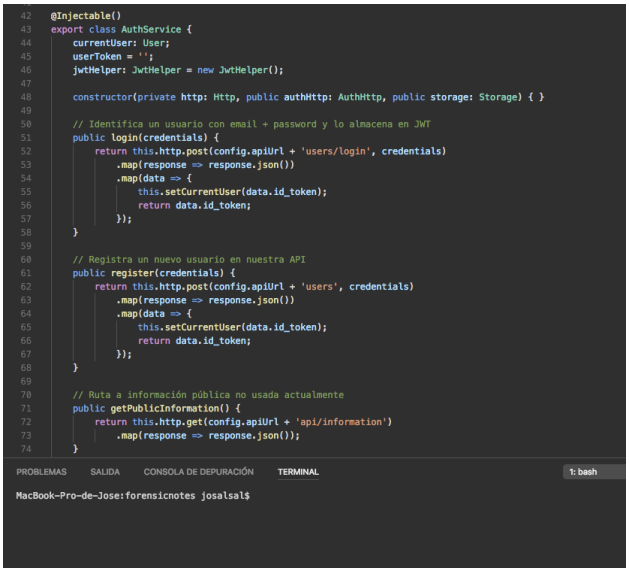


Figura 44: uso de servicios en *Ionic*

6. PRUEBAS

La aplicación se ha probado sobre un dispositivo físico.

Normalmente, para probar una aplicación en *Ionic 2* se utiliza *Jasmine* y *Karma* bajo el concepto *TestBed*[5]. Esto implica tener que instalar y configurar correctamente ambos *frameworks* para crear el juego de pruebas, crear las pruebas y aplicar las pruebas a los módulos.

Jasmine se suele emplear para crear las pruebas unitarias y *Karma* para ejecutar esas pruebas.

Antes de nada, indicar que *Jasmine* es un *framework* de Desarrollo Dirigido por Comportamientos (*Behavior Driven Development*) para realizar pruebas unitarias, que nos ayudan a probar el código de nuestra aplicación. Esto lo hace a través de los siguientes módulos o funciones principales:

- ***Describe()***: donde se define el conjunto de pruebas o especificaciones a realizar.
- ***It()***: que es el contenedor de *unit test*, que define el comportamiento esperado del código que se está probando. Esta función se puede anidar dentro de la función *describe*.
- ***Expect()***: que define el resultado esperado del *test*. A su vez, esta función se puede anidar dentro de la función *it*.

Sin embargo, existen otras formas más sencillas de probar la aplicación, dado que *Angular* nos ofrece *FormBuilder*, que no es otra cosa más que una clase que nos permite controlar y validar los datos introducidos en el formulario de manera eficiente y sencilla.

Como *Ionic* utiliza *Angular*, podemos utilizar la clase *FormBuilder* con los componentes del *SDK* de *Ionic*. Para lograr esto, debemos importar primero *FormBuilder*, y luego, inyectarlo como dependencia en el constructor, como se muestra en la siguiente imagen:

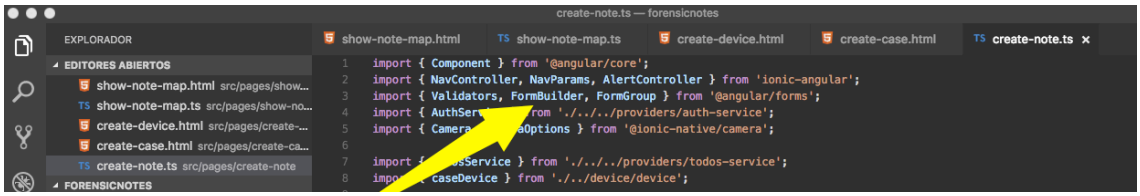
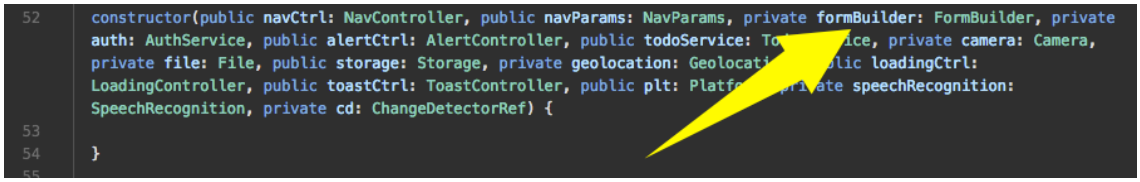


Figura 45: importación de la clase FormBuilder

[5] Es un concepto introducido por *Angular 2*, que nos permite crear un módulo independiente para testar los componentes que se están probando.



```
52  constructor(public navCtrl: NavController, public navParams: NavParams, private formBuilder: FormBuilder, private
    auth: AuthService, public alertCtrl: AlertController, public todoService: To      ice, private camera: Camera,
    private file: File, public storage: Storage, private geolocation: Geolocati      blic loadingCtrl:
    LoadingController, public toastCtrl: ToastController, public plt: Platfo      ate speechRecognition:
    SpeechRecognition, private cd: ChangeDetectorRef) {
53
54  }
```

Figura 46: inyección de *FormBuilder* en el constructor

A continuación, se ha creado el formulario haciendo uso de la variable "this.formBuilder", la cual nos permite controlar cada uno de los campos del formulario y realizar validaciones.

```
97
98  this.note = this.formBuilder.group({
99    _id: [typeof (this.todo) !== 'undefined' ? this.todo._id : ''],
100   _rev: [typeof (this.todo) !== 'undefined' ? this.todo._rev : ''],
101   caseId: caseDevice.caseId,
102   deviceId: caseDevice._id,
103   titleNote: [typeof (this.todo) !== 'undefined' ? this.todo.titleNote : '', Validators.required],
104   descriptionNote: [typeof (this.todo) !== 'undefined' ? this.todo.descriptionNote : '', Validators.required],
105   src: this.selectedFilter === 'fotográfica' ? this.storage.get('imageData').then(data => { return data; }) : '',
106   coords: this.selectedFilter === 'de mapa' ? this.location : '',
107   voice: [this.selectedFilter === 'de audio' ? this.voice : ''],
108   phase: [typeof (this.todo) !== 'undefined' ? this.todo.phase : 'preparacion'],
109   date: localISOTime,
110   author: info.email,
111   isCase: false,
112   isDevice: false,
113   isNote: true,
114   typeNote: this.selectedFilter,
115   active: false,
116   status: 'none'
117 });
118
```

Figura 47: clase FormBuilder

Una vez realizadas manualmente las pruebas unitarias, se han llevado a cabo las pruebas de integración, las cuales se han encargado de validar el funcionamiento del subsistema cuando sus componentes se unen los unos con los otros. Estas pruebas también se han realizado manualmente.

A continuación, se muestra una tabla con algunas de las pruebas de integración que se han realizado:

Conectividad	La *BBDD* local (*PouchDB*) se puede conectar con el servidor *CouchDB*. La aplicación móvil se puede conectar con el servicio de autenticación mediante *JWT* si el servidor de autenticación está activo.
Seguridad	El subsistema se ejecuta en las condiciones de seguridad de entorno de producción. Acceso a la aplicación con *email* y *password* erróneos (denegar el acceso). Acceso a la aplicación sin informar *email* y *password* (denegar acceso).

	Logout, etc...
Rendimiento	La *BBDD NoSQL Cloudant* soporta la carga de más de 1000 consultas por hora. El componente de *login* valida una conexión en menos de 0,1 segundos.
Instalación	La aplicación móvil se puede instalar tanto en la plataforma *Android*, como en *iOS*.
Entorno	El servidor funciona en *Windows*, *Mac OS* y *Linux*.

Y entre las pruebas unitarias realizadas, destacamos las siguientes:

Propósito *LOGIN*

Código	Acciones a verificar	Resultado esperado	Verificación
01	Acceso a a aplicación con *email* y *password* incorrectos.	Denegar el acceso.	**OK**
02	Acceso a la aplicación sin informar *email* y *password*.	No se permite.	**OK**
03	Acceso a la aplicación con un *email* que no cumple el formato.	No se permite.	**OK**
04	Acceso a la aplicación con un *email* no existente.	No se permite.	**OK**
05	Acceso al sistema con la dirección *IP* del servidor incorrecta.	No se permite.	**OK**
06	Si se accede al sistema de autenticación, pero el usuario y password de la BBDD NoSQL Cloudant no son correctos.	Se permite almacenar los datos en local (PouchDB) y actualizar con el servidor de *BBDD Cloudant* una vez que las credenciales sean las correctas.	**OK**
07	Acceso usuario registrado.	Tras el acceso satisfactorio se redirige a la página de inicio del usuario registrado.	**OK**

Propósito *HOMEPAGE*

Código	Acciones a verificar	Resultado esperado	Verificación
01	Acceder a los casos una vez se ha eliminado el *token* de sesión.	No se permite.	**OK**
02	Salir de la aplicación.	Se permite.	**OK**
03	Acceder a la pantalla de casos.	Se permite.	**OK**

Propósito *CASOS*

Código	Acciones a verificar	Resultado esperado	Verificación
01	Crear un caso sin informar los campos obligatorios (*)	No se permite.	**OK**
02	Actualizar un caso sin informar los campos obligatorios (*)	No se permite.	**OK**
03	Crear un caso informando los campos obligatorios (*)	Se permite.	**OK**
04	Actualizar un caso informando los campos obligatorios (*).	Se permite.	**OK**
05	Eliminar un caso con dispositivos pendientes.	No se permite.	**OK**
06	Eliminar un caso sin dispositivos pendientes.	Se permite.	**OK**
07	Acceder a la pantalla de dispositivos.	Se permite.	**OK**
08	Acceder a la pantalla principal (*HomePage*) de la aplicación.	Se permite.	**OK**

Propósito *DISPOSITIVOS*

Código	Acciones a verificar	Resultado esperado	Verificación
01	Crear un dispositivo sin informar los campos obligatorios (*)	No se permite.	**OK**
02	Actualizar un dispositivo sin informar los campos	No se permite.	**OK**

	obligatorios (*)		
03	Crear un dispositivo informando los campos obligatorios (*)	Se permite.	**OK**
04	Actualizar un dispositivo informando los campos obligatorios (*).	Se permite.	**OK**
05	Eliminar un dispositivo con notas pendientes.	No se permite.	**OK**
06	Eliminar un dispositivo sin notas pendientes.	Se permite.	**OK**
07	Acceder a la pantalla de notas.	Se permite.	**OK**
08	Acceder a la pantalla de casos.	Se permite.	**OK**

Propósito *NOTAS*

Código	Acciones a verificar	Resultado esperado	Verificación
01	Crear una nota sin informar los campos obligatorios (*)	No se permite.	**OK**
02	Actualizar notas.	No se permite.	**OK**
03	Crear una nota informando los campos obligatorios (*).	Se permite.	**OK**
04	Eliminar una nota.	Se permite.	**OK**
05	Visualizar una nota.	Se permite.	**OK**
06	Marcar como importante una nota.	Se permite.	**OK**
07	Desmarcar una nota. Es decir, pasar del estado importante a normal.	No se permite.	**OK**
08	Acceder a la pantalla de dispositivos.	Se permite.	**OK**
09	Las notas se encuentran ordenadas por fecha.	Sí.	**OK**

7. ESTADO DEL PROYECTO

Una vez finalizado el proyecto, tengo que indicar que se han conseguido todos los objetivos iniciales.

Por ejemplo, en cuanto a la implementación:

- El *backend* se encuentra totalmente implementado, pero con algunos cambios:
  - El sistema de autenticación se ha realizado con *JWT* en lugar de con *Latch*.
  - He utilizado *Cloudant*, como servicio de base de datos *NoSQL* en lugar de *MongoDB*.

- En cuanto a la *app* móvil, hay que indicar que se encuentran implementadas también todas las funciones:

  - El *login*, la eliminación del *token JWT* y la salida de la aplicación.
  - Añadir, modificar, visualizar y eliminar casos.
  - Añadir, modificar, visualizar y eliminar dispositivos.
  - Añadir, visualizar y eliminar notas.

Por tanto, el proyecto está finalizado completamente.

8. CONCLUSIONES

Como conclusiones solamente indicar, que ha sido muy grato realizar la aplicación en *Ionic*, dado que es un estupendo punto de partida para crear aplicaciones de forma sencilla en la plataforma *Apache Cordova.*

Si a eso le añadimos la combinación de *Angular* y *TypeScript* podemos llegar a crear proyectos de gran complejidad, dado que *TypeScript* mejora la productividad, proporciona detección temprana de errores, ayuda a comprender cómo funciona internamente *JavaScript* y hace que el código sea mejor. Además, los proyectos son de código abierto y la demanda es imparable.

9. RECOMENDACIONES

Para migrar el proyecto de *Ionic 2* a *Ionic 3*, debemos seguir una serie de pasos.

En primer lugar, debemos borrar el directorio *node_modules*, para luego poder instalar las nuevas dependencias del proyecto. Lo podemos hacer con el comando *rm –rf node_modules* (en *Linux* y *Mac OS*) o *rd /s node_modules* (en *Windows*).

El segundo paso, consiste en actualizar el *package.json*. Debemos actualizar las versiones de las dependencias del proyecto y las que *Ionic 3* necesita para trabajar correctamente.

Si existen otras dependencias diferentes de las que maneja *Ionic*, se deben revisar y actualizar. Lo más importante es que sean compatibles con la versión 4 de *Angular*.

El tercer paso, consiste en instalar las nuevas dependencias. Solamente hay que instalar las dependencias que sean nuevas. Para ello, escribimos el comando *npm install* en una ventana del terminal.

Por último, es necesario agregar *BrowserModule* y *HttpModule* en el fichero *app.module.ts*.

10. REFERENCIAS

James Griffiths *(2016). Mastering Ionic 2 – The definitive guide.*

Joyce Justin; Joseph Jude *(2017). Learn Ionic 2. Apress.*

Joshua Morony *(mayo de 2017). Building Mobile Apps with Ionic 2 & 3.*

Simon Reimler*. Ionic 2 from zero to app store.*

Hoc Phan *(2016). Ionic 2 cookbook, Second Edition. Packt Publishing Ltd.*

Adam Freeman *(2017). Pro Angular – Learn to harness the power of modern web browsers from within your application's code, Second Edition. Apress.*

Jeremy Wilken *(2017). Angular in action. Manning Publications.*

Jesse Palmer *(2017). Testing Angular Applications. Manning Publications.*

Yakov Fain; Anton Moiseev *(2017). Angular 2 Development with TypeScript. Manning Publications.*

Vilic Vane *(2016). TypeScript design patterns. Packt Publishing Ltd.*

Jess Chadwick *(2016). Essential TypeScript. LeanPub.*

Mike Cantelon; Alex Young; Marc harter; T.J. Holowaychuk; Nathan Rajlich*. Node.js in Action, Second Edition. Manning Publications.*

Simon Holmes *(November 2015). Getting mean with Mongo, Express, Angular and Node. Manning Publications.*

Chris Sevilleja; Holly Lloyd *(April 2015). Mean Machine – The beginner's guide to the JavaScript Stack. LeanPub.*

Páginas web:

Estas son solamente algunas de las páginas *web* de referencia más importantes citadas en el texto. Para más información sobre *Ionic* 2 y 3 existen en *Internet* gran número de enlaces disponibles directamente a través de *Google* y otros buscadores:

- *http://ionicframework.com/*
- *https://cloudant.com*
- *https://code.visualstudio.com*

11. GLOSARIO

Angular 2: es la segunda versión del *framework* de *JavaScript.* Es de código abierto, mantenido por Google, y se utiliza para crear y mantener aplicaciones *web* de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (*MVC*), en un esfuerzo por hacer que el desarrolo y las pruebas sean más fáciles.

Apache CouchDB: es un gestor de bases de datos de código abierto, cuyo foco está puesto en la facilidad de su uso y en ser "una base de datos que asume la web de manera completa". Se trata de una base de datos *NoSQL*, que emplea *JSON* para almacenar los datos, *JavaScript* como lenguaje de consulta por medio de *MapReduce* y *HTTP* como *API*. Una de sus características más peculiares es la facilidad con la que permite hacer replicaciones.

Ionic: es un *framework* gratuito y open source para desarrollar apliaciones híbridas multiplataforma que utiliza *HTML5*, *CSS* (generado por *SASS*) y *Cordova* como base. Es uno de los *frameworks* del momento por utilizar *Angular* para gestionar las aplicaciones, lo que asegura aplicaciones rápidas y escalables.

Node.js: es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor basado en el lenguaje de programación *ECMAScript*, asíncrono, con *I/O* de datos en una arquitectura orientada a eventos y basado en el motor *V8* de *Google*.

NoSQL: significa que ya no hay esquemas o "tablas", simplemente hay documentos que lo único seguro es que tienen un *id* único. En este proyecto *Cloudant* y *PouchDB* son *NoSQL* y son la columna vertebral de la aplicación móvil.

PouchDB: es una librería *JavaScript* que permite almacenar y consultar datos para aplicaciones *web* que necesitan trabajar sin conexión, para luego sincronizar con una base de datos en línea.

SASS: es un lenguaje de hoja de estilos inicialmente diseñado por *Hampton Catlin* y desarrollado por *Nathan Weizenbaum*. *Sass* es un metalenguaje de Hojas de Estilo en Cascada (*CSS*). Es un lenguaje de *script* que es traducido a *CSS*.

TypeScript: es un lenguaje de programación libre y de código abierto desarrollado y mantenido por *Microsoft*. Es un superconjunto de *JavaScript*, que esencialmente añade tipado estático y objetos basados en clases. *TypeScript* puede ser usado para desarrollar aplicaciones *JavaScript* que se ejecutarán en el lado del cliente o del servidor (*Node*.*js*).

12. ANEXOS

12.1. REFERENCIA A LOS COMANDOS MÁS UTILIZADOS EN *IONIC*

Ionic

CLI v2 -> v3 -- Command Cheat Sheet

Help developers identify the differences in commands used to accomplish primary tasks when migrating to v3.x.x of the Ionic CLI.

- `<param> === required, [param] === optional`
- Your `pwd` must be within an Ionic project to get help or use the following commands: `ionic cordova <command>`
- Commands removed in CLI v3: `setup, share, lib, io, security, push, package, config, service, add, remove, list, hooks, state`

To do this...	In Ionic v2 CLI, I would...	But in Ionic v3 CLI, instead...	Ionic v3 CLI Notes
Create new Ionic app (v1.x)	`ionic start <name> [tmpl] --v1`	`ionic start <name> [tmpl] --type ionic1`	
Create new Ionic app (latest)	`ionic start <name> [tmpl] --v2`	`ionic start <name> [tmpl] [--type <type>]`	`Default type: ionic-angular`
Emulate <platform>	`ionic emulate <platform>`	`ionic cordova emulate <platform>`	
Build <platform>	`ionic build <platform>`	`ionic cordova build <platform>`	
Run app <platform>	`ionic run <platform>`	`ionic cordova run <platform>`	
Get app/CLI/system info	`ionic info`	`ionic info`	`Good for issues :)`
Operate on Cordova platforms	`ionic platform <action> <platforms>`	`ionic cordova platform <action> <platform>`	
Operate on Cordova plugins	`ionic plugins <action> <platforms>`	`ionic cordova plugins <action> <platform>`	
Get command help	*`Unchanged, but just so ya know...`*	`ionic help <command>`	`ionic <command> --help`

Note: We removed `ionic state` because it was a hack we put into the older CLI to manage *cordova platforms* and *plugins*, because *cordova* didn't have a built-in way to accomplish this goal...but now it does:
https://cordova.apache.org/docs/en/latest/platform_plugin_versioning_ref/