Universidad Zaragoza
1542

Trabajo Fin de Grado

Depot Cloud: *Un sistema de gestión de trasteros*
Depot Cloud: A storage room management system

Autor

Rubén Moreno Jimeno

Director

Francisco Javier López Pellicer

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2017

Repositorio la Universidad de Zaragoza – Zaguan http://zaguan.unizar.es

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Rubén Moreno Jimeno,

con nº de DNI 25197158Z en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Depot Cloud: Un sistema de gestión de trasteros,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 08 de mayo de 2017

Fdo: 

AGRADECIMIENTOS

Quiero agradecer el esfuerzo de todos aquellos profesores que se han esforzado en no sólo transmitir conocimientos como profesionales, sino también en transmitir pasión como personas. A todos aquellos que han sabido mantener una amabilidad con sus alumnos y una relación alumno-profesor más cercana, implicándose más con ellos, y dándoles apoyo. A todos ellos que han sabido motivarme para esforzarme al máximo. A los que me han aguantado siendo un pesado en las tutorías improvisadas (incluso a los que no me han matado haciéndoles madrugar a las 8-9 para algunas reuniones). Y en general a todos los que han sabido tratar a los alumnos como personas, y no como números.

Agradecer a Francisco Javier López Pellicer por su dedicación y su ayuda prestada a la hora de realizar este trabajo, del cual me siento especialmente orgulloso gracias a él, habiéndome motivado a refinar cada minucioso detalle del mismo.

Por supuesto, agradecer el apoyo mostrado de todos mis amigos durante estos largos cuatro años. Por haberme apoyado por encima de todo a seguir adelante, pasara lo que pasara y costara lo que costara. Por haber entendido el sacrificio que este largo viaje conllevaba y no haberme olvidado después de tantos planes que he rechazado por los estudios.

En especial, agradecer a Alberto y David, ya que sin ellos nunca habría llegado a explotarme al máximo y sacar todo el rendimiento que podía sacar estos dos últimos cursos, ni haber aprendido todo lo que he aprendido. También por todos esos momentos inolvidables compartidos y por haber sabido como motivarme y apoyarme al máximo durante la carrera.

También agradecer a mis compañeros de viaje de Onirian, mi grupo de música, que durante estos años han sabido adaptarse a mi reducido tiempo sin presionarme. Además de haberme ofrecido todos esos momentos de ensayos, salidas y conciertos con los que disfrutar, desconectar y despejarme en todo momento. Y por supuesto por todo el apoyo que me han dado siempre.

A Andrea por el ánimo, apoyo y paciencia incondicional mostrado en especial durante el transcurso de este trabajo y este curso que ha sido especialmente intenso, ayudándome siempre que fuera posible y animándome en cualquier momento que necesitara.

Por último y más importante, a mis padres y mi hermano, por su apoyo incondicional durante toda mi vida, y que sin ellos probablemente no habría llegado hasta aquí ni a ser lo que soy. Por todo el esfuerzo económico de mis padres para financiarme los estudios, y por toda la ayuda y paciencia prestada por mi hermano en algunas dudas de los mismos.

RESUMEN EJECUTIVO

Cada vez es más frecuente que las familias de dos o más miembros tengan diversos armarios en sus viviendas, uno o más trasteros, y una o más viviendas. Esa situación a medida que crece dificulta la tarea de mantener todo el inventario organizado y bajo control. Es aquí donde nace *Depot Cloud*, un sistema centrado en la gestión de dichos almacenes y orientado a las unidades familiares y usuarios comunes que puedan no estar familiarizados con tecnologías.

El sistema tiene como objetivo crear un cliente para dispositivos móviles para mantener un inventario fiel de los objetos almacenados, poder localizarlos rápidamente, recomendar acciones en base a ellos, y controlar quién los mueve y a dónde. Además, otro objetivo importante es contar con otro cliente para equipos de sobremesa para usuarios administradores. Este cliente ha de ayudar a la gestión del sistema y de los usuarios pudiendo editar al resto de los usuarios y obtener estadísticas del sistema.

El objetivo principal de este trabajo consiste en el desarrollo de dicha aplicación, así como la documentación y gestión de todo el proceso propio de un sistema software profesional. Para conseguirlo, el proyecto ha pasado por cuatro fases organizadas.

- **La definición del producto**: En esta fase se ha definido el producto, empezando por su entorno y necesidades a cubrir, seguido por una concepción de la idea base, y la conceptualización de un modelo de negocio viable. Sobre esta base, y teniendo en cuenta que se trata de un Trabajo de Fin de Grado, se ha establecido el alcance del proyecto, delimitando sus límites, restricciones, dependencias, y lo que se podría realizar en un futuro.
- **El análisis del sistema**: En esta fase se ha realizado un análisis para identificar las características funcionales y no funcionales del sistema. Para conseguirlo, lo primero ha sido establecer una metodología de obtención de requisitos estructurada y formal, así como de su propia redacción. Recopilados los requisitos, se ha realizado un estudio de alternativas junto con la viabilidad de cada una dentro del proyecto. En dichas alternativas entran todas las decisiones de diseño en las que se han contemplado varias posibilidades, y descartado y argumentado las que no servían en el entorno del proyecto. Por último, se ha realizado un análisis de los riesgos asociados a la aproximación elegida identificando cómo solventarlos si surgieran durante el proyecto.
- **El diseño del sistema**: En esta fase se ha diseñado e implementado el sistema. Para ello se han dado soluciones arquitecturales utilizando un patrón Modelo-Vista-Controlador, diseño por capas, y servicios RESTful, soluciones tecnológicas usando el framework MEAN para aprovechar al máximo sus ventajas y los JSON Web Tokens para aportar la seguridad necesaria. Además, para darle calidad al proceso y al proyecto se ha utilizado la metodología de integración continua, una fuerte batería de tests unitarios, de sistema y aceptación, y para que sea un software mantenible se han realizado análisis estáticos del código. Por último, se han utilizado principios generales de diseño e invertido en la usabilidad de la solución, y utilizado herramientas para ayudar a documentar de forma óptima la API del sistema.
- **Gestión del proyecto**: Esta fase se ha llevado paralelamente a lo largo de todo el proyecto. Primero se ha realizado una planificación inicial del proyecto dividiendo el proceso en iteraciones, sin embargo, dado que se trata de un producto novedoso se ha seguido una aproximación ágil, lo cual ha permitido poder evolucionar y refinar el análisis y el diseño del sistema conforme se sabe más acerca de las tecnologías o incluso del mismo dominio del producto. Es por eso que las iteraciones de esta fase han acabado siendo ligeramente distintas, además se ha dejado planteado un posible segundo lanzamiento. Segundo se han recopilado todos los esfuerzos en horas invertidos en el proyecto por tiempo y por actividad. Por último, gracias a la recopilación detallada de las horas invertidas, se ha calculado el presupuesto total del proyecto.

Por último, se han realizado unas conclusiones en las que se ha analizado que el proyecto ha cumplido los objetivos previstos, comprobando que todas las fases anteriores han sido de vital importancia y observando en cuáles de ellas se podría haber mejorado o cuáles han sido las razones de los problemas encontrados.

TABLA DE CONTENIDOS

1. Introducción

Depot Cloud es una idea nace de la necesidad de crear un sistema de automatización y gestión de inventarios para unidades familiares y personas que no están familiarizados con tecnologías, implementando una aplicación sencilla que consuma pocos recursos que permita llegar a un número elevado de personas. Dicha aplicación se ha desarrollado como Trabajo Fin de Grado del Grado de Ingeniería Informática de la Universidad de Zaragoza y este documento contiene su memoria y sus anexos.

El documento recoge la documentación del proyecto propia de sistemas software. Se ha organizado siguiendo una estructura basada en la "Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Informática" del consejo de colegios de Ingenieros en Informática, de forma que hay cuatro apartados principales [1]. En el primer apartado se habla del producto y su viabilidad. Una vez materializada la idea, se describe el análisis del sistema en el segundo apartado. Con el análisis del sistema hecho, se habla en el tercer apartado del diseño de la solución propuesta. En el último apartado, se desarrolla lo relacionado a la gestión del proyecto y su presupuesto, proceso que se ha realizado paralelamente al resto durante el transcurso del proyecto.

A continuación, se realiza una descripción más en detalle de lo que contiene cada uno de los cuatro apartados del documento:

- **Producto**: Incluye las secciones **Depot Cloud**, y **Alcance y restricciones**. En la primera se habla sobre las necesidades que han dado lugar al producto, la idea del mismo, y su estudio de viabilidad. En la segunda, se aclara y delimita el alcance del proyecto con vistas al futuro, y se enumeran las restricciones, siendo algunas de ellas decisivas a la hora de tomar decisiones.
- **Análisis**: Incluye las secciones **Documentación de requisitos, Estudio de alternativas y viabilidad**, y **Análisis de riesgos**. En la primera se habla de las alternativas que han surgido a lo largo del proyecto a la hora de tomar decisiones y los argumentos que han dado lugar a unas u otras. En la segunda se habla del proceso de la documentación y recogida de requisitos que se ha seguido. En la tercera se describe el análisis de los riesgos captados para el proyecto.
- **Diseño**: Incluye la sección de **Descripción de la solución propuesta**, e **Implementación de la solución.** En la primera se describe la solución propuesta para el proyecto. En la segunda aspectos básicos de la implementación de la solución.
- **Gestión**: Incluye la sección **Gestión del proyecto**. En ella se encuentran los esfuerzos en horas totales del proyecto clasificados de distintas formas. y se calcula el presupuesto que tendría el proyecto si se hubiera desarrollado como aplicación real y comercial.

Además, el documento cuenta con una diversidad de anexos que son referenciados durante el resto de la memoria para ampliar cada una de las secciones si se desea, ya que en ellos se detalla todo a un nivel muy bajo y exhaustivo, mientras que en el núcleo de la memoria se describe todo a un nivel más alto.

Por último, dada la naturaleza académica del proyecto, se cuenta con un apartado de **Conclusiones y trabajo futuro**. Además, ya que este proyecto se plantea como un producto para una empresa real, lo óptimo es que se llevara a cabo por un equipo de desarrolladores de más de una persona. Es por eso que para que suene "más natural" cada vez que se haga referencias a "los desarrolladores" o al "equipo de desarrollado" en el documento, se refiere al alumno realizador del Trabajo de Fin de Grado.

2. Definiciones

A los efectos del presente documento, sus definiciones técnicas se entenderán por:

- **IDE**: Entorno de desarrollo integrado, aplicación informática que proporciona servicios integrales para facilitar al programador el desarrollo de software (interpretación de [2]).
- **Front-end:** Capa de abstracción de un sistema software que engloba la capa de presentación del mismo (interpretación de [3]).
- **Back-end:** Capa de abstracción de un sistema software que engloba la capa de acceso a datos y lógica de negocio del mismo (interpretación de [3]).
- **Framework**: Estructura conceptual y tecnológica con artefactos o módulos concretos de software que puede servir de base para la organización y desarrollo de software. Puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, para ayudar a desarrollar y unir diferentes componentes en un proyecto (interpretación de [4]).
- **Middleware**: Software que actúa como intermediario entre dos lógicas, programas, y/o capas separadas normalmente complejas y ya existentes, de tal forma que se encarga de comunicarlos abstrayendo al desarrollador (interpretación de [5]).
- **Template**: Documento HTML que proporciona la estructura y código de una plantilla, vista, o componente de UI (interpretación del autor).
- **Modal**: Es un cuadro de diálogo/ventana emergente que se muestra en un nivel superpuesto al resto de la UI (interpretación del autor).
- **Dashboard**: Componente de la interfaz de usuario que su finalidad es la de mostrar al usuario un tipo de información, en el caso de esta aplicación en formato de lista (interpretación del autor).
- **Quality Assurance**: Conjunto de actividades planificadas y sistemáticas aplicadas en un sistema de gestión de la calidad para que los requisitos de calidad de un producto o servicio sean satisfechos (interpretación de [6]).

A los efectos del presente documento, sus definiciones del domino del problema se entenderán por:

- **Almacén/Depósito**: Casa, trastero, o armario que pertenece a una unidad familiar, puede ser gestionada por ella, y en el que se van a guardar objetos con posterioridad.
- **Miembros de la unidad familiar**: Cualquier persona que forme parte de una unidad familiar o que se quiera que pertenezca a la misma y comparta la información de la cuenta (p.ej.: el padre, la madre, y lo/s hijo/s).
- **Actividad de una unidad familiar**: Creación, modificación, o eliminación de un miembro, almacén u objeto perteneciente a una unidad familiar.

3. Depot Cloud

3.1 Necesidades y entorno

Hoy en día es bastante común que una unidad familiar o incluso que un individuo en particular cuente con más de una vivienda, (ya sea adquirida por sus propios medios, heredada, o compartida), uno o más trasteros, y por supuesto, varios armarios. Es por eso que un problema bastante común al que se tienen que enfrentar dichas personas es a una buena gestión de sus inventarios en dichos almacenes, ya cuando la situación empieza a crecer, se vuelve cada vez más difícil de solucionar sin un sistema que lo controle todo adecuadamente. Este problema se vuelve complicado de solucionar cuando se tiene en cuenta que cada caso en particular es completamente distinto en distribución y número de almacenes.

Existen, sin embargo, algunos sistemas ya especializados en la gestión de inventarios, almacenes y/o bibliotecas. No obstante, estas soluciones suelen estar pensadas para el sector empresarial, para inventarios a gran escala, control de producción, optimización de recursos, o gestión de un tipo de objetos específico, y no para el uso de unidades familiares a una escala muy inferior y con finalidades distintas.

Es aquí donde nace *Depot Cloud*, tratando de resolver dichos problemas intentando crear una solución general y de uso intuitivo para un amplio sector de usuarios, yendo desde los más expertos en tecnologías hasta los menos familiarizados con las mismas.

3.2 Materialización

Depot Cloud debería ser un sistema centrado en la gestión de almacenes (entendiendo como tales las viviendas, los armarios, y los trasteros) y orientado a las unidades familiares y usuarios comunes. La aplicación debería contar con dos clientes distintos orientados a usuarios con funcionalidades y finalidades diferentes (los administradores y las unidades familiares).

Los usuarios de las unidades familiares deberían disponer de funcionalidades tales como la gestión de miembros de la misma (es decir, las personas que pertenecen a ella), de almacenes, así como de los objetos que pertenecen a los mismos (además de posteriormente poder realizar búsquedas sobre ellos). Tanto los almacenes como los objetos deberían contener una amplia información que les caracterizase para poder categorizarlos y poder describirlos ampliamente. Además, todas las actividades realizadas en la unidad familiar quedan deberían quedar registradas para que ellos posteriormente las puedan consultar manteniendo así el control de sus inventarios. La aplicación también debería ofrecer a los usuarios unas recomendaciones que realizara de forma periódica en base al uso de sus inventarios y el contenido que éstos tienen. Por último, dado que se trata de una aplicación que trata de ser usable para usuarios no expertos o familiarizados con tecnologías, debería contar con un breve tutorial dentro de la misma aplicación.

Los usuarios de tipo administrador deberían disponer de la capacidad de listar todos los usuarios del sistema, pudiendo editarles la información de la cuenta, o activarles la misma en caso de que la hubieran borrado anteriormente. También deberían poder visualizar estadísticas a nivel de interés del sistema.

3.3 Modelo de negocio

El análisis no estaría completo sin un modelo de negocio. El modelo de negocio nos da una visión de conjunto del proyecto en la que la solución resultante de este TFG sería una pieza más. De esta forma le damos un marco de referencia al análisis, a los requisitos capturados, a algunas decisiones de diseño, y a aspectos como la valoración del coste de este TFG. La elaboración del modelo de negocio del producto se ha basado en la herramienta desarrollada por *Alex Osterwalder* denominada *Business Model Canvas* [7]. El modelo consta de 9 campos o bloques que permiten tener una visión integrada del modelo de negocio y se ha representado en la **Tabla 1**. El número que aparece entre paréntesis en los títulos de cada bloque representa el orden de trabajo en el que se han completado los módulos. Gracias a ese orden es como se consigue refinar y conceptualizar el modelo de negocio.

Tabla 1. Business Model Canvas.

<table>
<tr>
<td rowspan="2">SOCIOS CLAVE (8)

• Empresas de alquiler de trasteros.</td>
<td>ACTIVIDADES CLAVE (7)

• Identificación de nuevas necesidades.
• Mantenimiento de la aplicación.
• Servicio al cliente.</td>
<td rowspan="2">PROPUESTA DE VALOR (2)

• Localizar con un solo click dónde está lo que alguna vez se guardó en un trastero y ahora no se localiza.
• Mantener ordenados y libres de trastos innecesarios los trasteros familiares.
• Localizar cualquier trasto que alguien haya movido a la casa del pueblo sin informar.</td>
<td>RELACIONES CON LOS CLIENTES (4)

• Sitio web para administradores que ayuden con súper privilegios a los usuarios.
• Redes sociales y correo electrónico.</td>
<td rowspan="2">SEGMENTOS DE CLIENTES (1)

Unidades familiares conformadas por dos o más miembros:
• Con múltiples viviendas, trasteros y armarios.
• Les gusta tener todos sus objetos organizados y controlados.
• Tienen dificultad para acordarse de dónde dejan las cosas.
• Realizan mudanzas con frecuencia.</td>
</tr>
<tr>
<td>RECURSOS CLAVE (6)

Personal:
• Equipo de desarrolladores.</td>
<td>CANALES (3)

• Redes sociales.
• Sitio web.</td>
</tr>
<tr>
<td colspan="2">ESTRUCTURA DE COSTES (9)

Unos costes totales aproximados de 10.000€ repartidos entre:
• Dominio y hosting (gasto anual).
• Conexión a teléfono e internet (gasto anual).
• Mobiliario de oficinas (inversión inicial).
• Equipos informáticos (inversión inicial).
• Programas informáticos (gasto anual).
• Desarrollo de la aplicación móvil y sitio web (inversión inicial).</td>
<td colspan="3">FLUJO DE INGRESOS (5)

• Los clientes tendrán 30 días de uso inicial gratuito y posteriormente tendrán que pagar 10€/mes. Este coste para los usuarios se ha estimado en base a otras aplicaciones que funcionan de forma similar en sus métodos de pago (p.ej. Spotify), y para que, además, con este flujo de ingresos se pueda amortizar con 1000 clientes en el primer mes.
• El cliente puede hacer sus pagos a través de pago en línea (transferencia electrónica, PayPal…).</td>
</tr>
</table>

4. Alcance y restricciones

Existiendo unas restricciones de duración del proyecto al tratarse de un Trabajo de Fin de Grado de 12 créditos ECTS (que se traduce en 300 horas de trabajo por parte del alumno), habiéndose superado dicho límite en más de 400 horas, y además con un plazo de entrega fijado, quedan fuera del alcance del proyecto las siguientes mejoras que han propuesto algunos de los potenciales usuarios de una solución como *Depot Cloud*:

- Detectar objetos potencialmente registrados dos veces o que se encuentran registrados en dos almacenes diferentes utilizando las descripciones y las fotos y avisar al usuario con una recomendación.
- Compartir información si lo deseas de tus almacenes u objetos utilizando sistemas de mensajería de terceros.
- Permitir delegar ciertas operaciones de creación, modificación, borrado y búsqueda a una persona ajena a la unidad familiar (orientado hacia empresas de mudanzas).
- Creación de un enfoque de uso atractivo centrándose en empresas de mudanza (*check lists*) o de gestores de almacenes (personalización de la interfaz para su reventa por terceros).
- Optimizar la aplicación para cumplir aspectos de accesibilidad.

El proyecto cuenta con las siguientes hipótesis y dependencias:

- La base de datos y el servidor web se podrá realizar sobre una base de datos conocida por el desarrollador para cumplir con los plazos establecidos, front-end se podrá implementar con tecnologías híbridas como Ionic 2 y AngularJS.

El proyecto cuenta con las siguientes restricciones derivadas del alcance:

- El proyecto, incluida esta memoria, ha de ser terminado antes del 18 de septiembre de 2017.
- El proyecto debe poder realizarse por un alumno durante 300 horas siendo éste el único desarrollador de la aplicación y un director durante 35 horas.
- El control de la localización es solo virtual, es decir, no se puede realizar ninguna modificación a los objetos o almacenes familiares físicos.
- La solución en el cliente para dispositivos móviles no debe requerir de un gran número de recursos computacionales ya que se quiere que se ejecute en un sector del mercado lo más amplio posible.
- La solución no puede asumir nunca que la información existente en el sistema, de los objetos y almacenes familiares, pertenecientes a una unidad familiar es actualizada al 100% constantemente.

5. Documentación de requisitos

Para la captura y obtención de requisitos, la primera tarea realizada ha sido la identificación de los usuarios implicados que se detallan en la **Tabla 2**. Después, ha tenido lugar una fase de descubrimiento de requisitos, esta fase ha consistido en encontrar fuentes de información relevantes para el sistema y tratar de comprender sus necesidades e inquietudes. Es por eso que se han buscado personas que tuvieran el perfil del usuario objetivo del sistema para posteriormente poder realizarles algunas entrevistas informales acerca de sus necesidades o de qué les gustaría que llevara una aplicación del estilo. Posteriormente, se ha tratado de eliminar cualquier conflicto entre los requisitos, detectar duplicados, y minimizar el número de requisitos, todo ello para empezar a estructurar la documentación de los mismos, y volviendo a empezar la fase de descubrimiento cada vez que se tuviera dudas de alguno de ellos. Por último, se han estructurado por tipo, por prioridad y por los usuarios implicados de la siguiente sección. De tal forma que se ha obtenido primero la clasificación entre requisitos funcionales y no funcionales. Después se han ordenado por prioridad. Y, por último, dentro de los funcionales, se clasifican en tres tipos:

- **Requisitos Funcionales de unidades familiares y usuarios administradores**: Aquí se recogen las funcionalidades comunes para todos los usuarios implicados en el sistema.
- **Requisitos Funcionales de unidades familiares**: Aquí se recogen las funcionalidades orientadas a los usuarios comunes/unidades familiares.
- **Requisitos Funcionales de usuarios administradores**: Aquí se recogen las funcionalidades orientadas a los usuarios administradores.

Así, el sistema cuenta con un total de 3 requisitos funcionales del primer tipo, 8 requisitos funcionales del segundo tipo, 4 requisitos funcionales del tercer tipo, y 20 requisitos no funcionales. El listado completo, detallado y estructurado de todos los requisitos del sistema se encuentra en el anexo **Requisitos.**

En cuanto a la redacción de los mismos, se ha seguido una estructura formal a la hora de escribirlos y así, mantener una consistencia en todos ellos. Esta forma consiste en la clasificación de los requisitos mediante un código identificativo dentro del documento, de forma que se pueda referenciar desde cualquier parte del mismo de una forma cómoda y que no dé pie a ambigüedades. Dicho código consiste en dos partes, la primera son unas letras correspondientes a las iniciales de las palabras del tipo al que pertenecen (RFU para los requisitos funcionales comunes, RFUF para los requisitos funcionales exclusivos de unidades familiares, RFUA para los requisitos funcionales exclusivos de usuarios administradores, y RNF para los no funcionales) seguido de un número que le identifica dentro del propio tipo otorgándole la prioridad. A continuación, se puede ver un ejemplo en la **Tabla 3**.

Tabla 2. Usuarios implicados en el sistema.

Usuario común/Unidad familiar	Usuarios habituales de la aplicación, los cuales han de registrarse primero y después iniciar sesión para poder acceder a sus funcionalidades. Hacen uso del cliente en dispositivos móviles, y a todas las funcionalidades que se ofrecen en esas plataformas.
Usuario administrador	Usuario que se encarga de gestionar y mantener el sistema. No se pueden registrar nuevos usuarios de este tipo, desde ninguno de los clientes, para prevenir temas de seguridad. Hacen uso del cliente en dispositivos de escritorio, y a todas las funcionalidades que se ofrecen en esas plataformas.

Tabla 3. Ejemplo de requisitos del sistema.

RFU-2	El sistema debe permitir que el usuario cambie información de una cuenta de usuario únicamente si se trata de la suya
RFUF-1	El sistema debe permitir que se registren nuevas cuentas para unidades familiares.
RFUA-4	El sistema debe permitir al usuario mostrar estadísticas con datos relevantes del sistema.
RNF-12	Las recomendaciones que se van a mostrar a una cuenta familiar deben mostrarse al usuario de forma cronológica descendente.

6. Estudio de alternativas y viabilidad

Dado que se trata de un proyecto de ingeniería, es muy importante que haya una fase de análisis, y en ella, un estudio de alternativas investigando posibles candidatos y alternativas para diversos fines, los cuales se han de evaluar y en caso necesario descartar por opciones que resulten ser más viables en el entorno y situación en el que se encuentre el proyecto. Como se acaba de decir, para poder tomar correctamente las decisiones, primero se han de resumir los aspectos más relevantes del entorno del proyecto.

Se presenta pues, un proyecto que cuenta con unas restricciones importantes en cuanto a plazos y esfuerzos, los cuales idealmente no deberían pasar de 300 horas siendo su fecha límite el 22 de septiembre. Junto a estas restricciones, se tiene que sólo hay un desarrollador a cargo de dicho proyecto, es por eso que cuando se presenten alternativas de viabilidad semejante, se tenderá a coger aquella que ya sea dominada por él o cuyo plazo de aprendizaje sea asumible. Por otro lado, la aplicación del proyecto ha de ser usada por un mínimo de 1000 usuarios (cifra estimada en la sección **Modelo de negocio**) con la mejor relación experiencia de usuario/coste. Por eso también es importante que el sistema escale adecuadamente dado el amplio número de usuarios al que se quiere llegar. Además, puesto que se trata de un producto completamente nuevo y en evolución, se necesita un sistema flexible que pueda evolucionar a la vez que lo hace el dominio del problema. Por último, se estima que de entre todas las operaciones que se realicen en la base de datos, entre un 50%-80% sean de lectura, ya que los usuarios normalmente tendrán una pequeña fase de población de datos inicial, pero posteriormente disminuirá (dado que ya sólo tendrán que meter objetos nuevos que adquieran o algunos existentes que quieran modificar), mientras que sin embargo, el ritmo de las consultas para saber dónde tienen sus objetos o en qué lugares los tienen, se estima que sea constante de inicio a fin, por lo que de ahí se estima el hecho de que más de la mitad de las operaciones sea lectura, pero que tampoco se trate de un sistema únicamente centrado en ellas.

Una vez analizado el entorno, se van a describir las decisiones tomadas en el proyecto, las cuales se estructuran, como se puede ver en la **Figura 1**, en las decisiones tomadas para dispositivos clientes móviles y sobremesa, servidor web, comunicaciones en tiempo real, y base de datos. Al final de la sección se encuentra la **Tabla 4** con el resumen de alternativas y opción elegida para cada uno de los apartados.

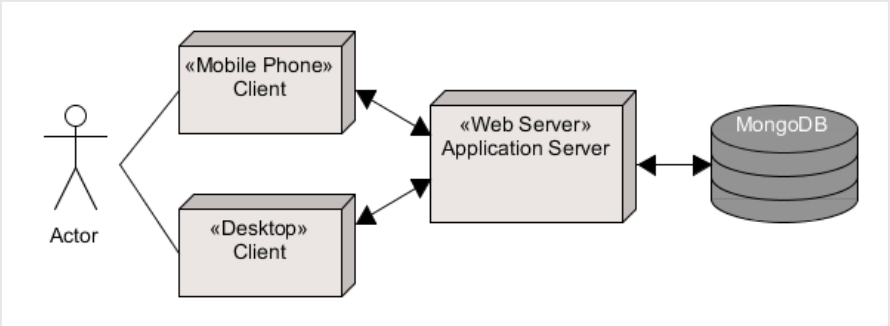

Figura 1. Diagrama de alto nivel de los principales componentes del sistema.

Dispositivo clientes para móviles: Aparecieron dos alternativas viables, la primera de ellas desarrollar la aplicación en lenguaje Android [8] e iOS [9] nativo, y la segunda en una tecnología híbrida como el framework Ionic 2 [10]. Las decisiones que llevaron a finalmente, desarrollarlo con el segundo, fueron las siguientes:

- La tecnología del framework Ionic 2, permitía desarrollar una sola aplicación, que gracias a ser híbrida funcionando sobre el navegador web Cordova, funcionaba independientemente de la plataforma móvil, es decir, con el desarrollo de un mismo cliente móvil, se cubrían las necesidades de Android, iOS y Windows Phone, mientras que, con Android o iOS, solo se cubría uno de los dispositivos móviles.
- Mientras que para desarrollar Android o iOS necesitas desarrolladores especializada únicamente en dichos lenguajes, con el framework Ionic 2 únicamente necesitas desarrolladores que sepan implementar aplicaciones Web independientemente de si es para móvil o para dispositivos de escritorio, lo que hace más fácil ampliar el equipo de desarrolladores con vistas al futuro.

Dispositivo clientes de sobremesa: Las alternativas que surgieron fueron entre las diversas aplicaciones con capacidad de implementar tecnología AJAX y así hacer el sistema más escalable, descartando por completo tecnologías de clientes web basadas en templates como JSP. Aquí se plantearon cuatro tipos de alternativas distintas: JavaScript plano [11], jQuery [12], React [13], y AngularJS[1] [14]:

- El primero de ellos se descartó debido a que implementar una aplicación web cliente hoy en día sin utilizar ningún framework y utilizando únicamente JavaScript, es bastante contraproducente y se ralentiza el ritmo del trabajo, además de que el sistema pueda acabar teniendo serios problemas de seguridad.
- El segundo quedó descartado puesto que únicamente es una librería del lenguaje JavaScript para manipular el DOM [15], haciendo que sea más difícil también implementar algunas funcionalidades que con un framework moderno sería más rápido y fácil.
- Por último, las razones de peso para elegir entre React o AngularJS fueron que, además de que el desarrollador del producto dominaba AngularJS con antelación (pudiendo implementar el sistema sin problemas, mientras que no se tenían conocimientos acerca de React), existen unas cláusulas de la empresa Facebook [16] que pueden revocar la licencia de React y caer en asuntos judiciales si existe algún tipo de competencia con dicha empresa. Puesto que es una aplicación joven y no se sabe a ciencia cierta hacia dónde podría evolucionar en un futuro, es una razón de peso para tener en cuenta.

Servidor Web: Se presentaban dos opciones claras en la capa de la lógica de controladores. Dado que se buscaba un framework para desarrollar el back-end de una aplicación web, para evitar los problemas de trabajar sin ellos en un determinado lenguaje, se presentaban dos opciones viables por conocimiento del desarrollador: Spring Framework [17] y Express Framework [18]. El lenguaje del primero es Java, mientras que del segundo es JavaScript.

La decisión aquí fue casi inmediata, se contaba ya con un sistema casi puro escrito en JavaScript y JSON, orientado a recursos tipo RESTful, y, además, se tenía más conocimiento de NodeJS y Express que de Spring en cuanto a desarrollo back-end de aplicaciones web. Es por eso que se tomó la decisión de implementarlo con Express y NodeJS, para que, de esta forma, agregando Mongoose como driver de base de datos, se pudiera integrar todo el sistema sin problemas dando lugar a lo que se conoce como el framework MEAN [19].

Comunicación en tiempo real: La decisión aquí fue si abordar o no abordar un sistema de comunicación en tiempo real usando el protocolo Websockets u otras alternativas entre cliente y servidor. Principalmente este sistema de comunicación se quería implementar en los requisitos **RFUF-5** (ver actividades de la cuenta familiar) y **[RFUF-6]** (ver recomendaciones de la cuenta familiar). Dado que Websockets es una tecnología en tiempo real que abre un canal *full-dúplex* (bidireccional) entre cliente y servidor, abriendo el canal de comunicación mientras dure la conexión con una latencia muy baja, se empezó investigando alternativas para implementarla. Después de una tarea inicial de investigación, se consiguió reducir las posibilidades a dos librerías que implementan el protocolo Websockets, la primera de ellas llamada "SocketIO" [20], y la segunda "ws" [21]. Sin embargo, al final se acabaron descartando ambas y con ellas la idea de una comunicación por Websockets por las siguientes razones:

- En la librería de "SocketIO" el problema principal era de compatibilidad entre el estándar de Websockets y ella. Debido al protocolo de implementación, se necesita usar obligatoriamente la misma librería tanto en cliente como en servidor, sin dar la posibilidad a que alguien pueda conectarse al sistema con un protocolo estándar de Websockets.
- En la librería de "ws" no se pudo desarrollar una forma adecuada de implementar más de un canal de comunicación distinto en el mismo servidor, imposibilitando de esta forma obtener una API organizada y optimizada por funciones dependiendo de las rutas a los canales de acceso. Además, tampoco se pudo

[1] **:** En este documento cuando se hace referencia a AngularJS es a las versiones del framework 1.X, y cuando se hace a Angular, es a las versiones del framework 2.0 en adelante

encontrar una forma de poder encriptar las comunicaciones o hacer que a una cuenta familiar le llegara únicamente su información, y no la del resto de usuarios.

Esas razones dieron lugar a descartar las ideas. En este punto en un desarrollo con otras circunstancias, la siguiente alternativa viable que se hubiera planteado sería realizar dichas comunicaciones mediante, o tecnologías de servidores *push* para móviles [22], o el mecanismo de *polling* [23] con la tecnología de AJAX. En la situación actual, la técnica de *polling* se habría descartado por el gran consumo de recursos que origina, lo cual sería un problema serio en dispositivos móviles, y se habría escogido la opción de servidores *push*. Sin embargo, dada la investigación anterior de Websockets ya no se disponía de tiempo para poder implementar dicha solución. Puesto que formaba parte de los riesgos asumidos, y que, además, no supone un daño de gran impacto en la aplicación (dado que normalmente las unidades familiares suelen contener de 2-5 miembros, y es poco común que, con esas características, se exprimiera la naturaleza de tiempo real), se descartaron ambas opciones y con ello dicha forma de comunicación.

Base de datos: La primera decisión aquí ha sido entre bases de datos SQL clásicas, NoSQL [24], y las denominadas NewSQL [25]. La decisión resultante tomada ha sido escoger una base de datos no relacional debido a las siguientes razones:

- Ya que se trata de un proyecto nuevo, con una metodología ágil, en el cual aún no se tiene clara la dirección y evolución del mismo, esto otorga una mayor flexibilidad, adaptándose a necesidades nuevas que surjan pudiendo hacer cambios de los esquemas sin tener que parar bases de datos.
- En lo referente al rendimiento y escalabilidad, las bases de datos relacionales necesitan un mayor procesamiento en recursos y rendimiento cuanto más compleja sea la base de datos y sus relaciones (principalmente debido a la atomicidad). Esto es un problema ya que se busca una escalabilidad horizontal, ampliando el número de máquinas en lugar de máquinas más potentes. Esto permite que se pueda empezar el proyecto con un menor presupuesto y posteriormente poder integrarlo con más facilidad en servicios como Amazon Web Services y hacer que el sistema posea una alta escalabilidad.
- Por último, en lo relacionado a la atomicidad, las NoSQL salen perdiendo con la consistencia eventual que las caracteriza (por ejemplo, en MongoDB a nivel de documento y no de colección). Sin embargo, solo existiría una operación que incluiría varios documentos a la vez (**[RFUF-2.4]** borrado de almacenes), que en caso de que fallara, no supondría un daño crítico al sistema y que se podría resolver dejando un proceso en segundo plano que purgara los objetos huérfanos de almacén. Es por eso que se asume dicho riesgo ya que no supondría un problema en el sistema.

Una vez escogida la decisión entre bases de datos relacionales o no relacionales se abre un abanico de diversos tipos de base de datos NoSQL, entre las cuales también se ha tenido que tomar una decisión. La decisión tomada en este punto se ha basado en un informe realizado por *Altoros Systems Inc* [26] que contiene unas analíticas de bases de datos como Riak (almacenamiento clave-valor), Cassandra (almacenamiento de familia de columnas), HBase (almacenamiento de familia de columnas), MongoDB (almacenamiento orientado a documentos), MySQL Cluster (implementación NewSQL), y Sharded MySQL (implementación NewSQL). En él, se han utilizado máquinas virtuales de Amazon para garantizar unos resultados reproducibles y verificables, y la transparencia de la investigación (además de, por supuesto, minimizar errores debidos a la heterogeneidad del software).

En la **Figura 2** se puede observar que cuando dichas bases de datos se someten a una carga de trabajo de 50% actualizaciones, y 50% lecturas, durante la fase de lectura las tres mejores son Cassandra, HBase y MongoDB, mientras que en la **Figura 3** se obtiene que con un ratio de carga de trabajo de 5% actualizaciones y 95% lecturas, durante la fase de lectura, Sharded MySQL es la mejor, seguida de MongoDB. De aquí se ha extraído que, puesto que estamos ante un sistema que probablemente tenga más lecturas que escrituras, pero tampoco se conoce exactamente la ratio, las mejores bases de datos son Cassandra y MongoDB, ya que, por ejemplo, HBase empieza a reducir su rendimiento en cuanto se aumentan las lecturas mucho y bajan las actualizaciones, y Sharded MySQL baja casi exponencialmente su rendimiento en cuanto empiezan a aumentar un poco las actualizaciones con respecto a las lecturas. Por último y debido a restricciones temporales del

proyecto, dado que el desarrollador cuenta con conocimientos suficientes como para desarrollar este proyecto a tiempo en MongoDB, pero no en Cassandra, y tampoco existe una diferencia grande que pueda suponer problemas al sistema dada su estimación de carga de trabajo, se ha decidido desarrollar la capa de datos en MongoDB.

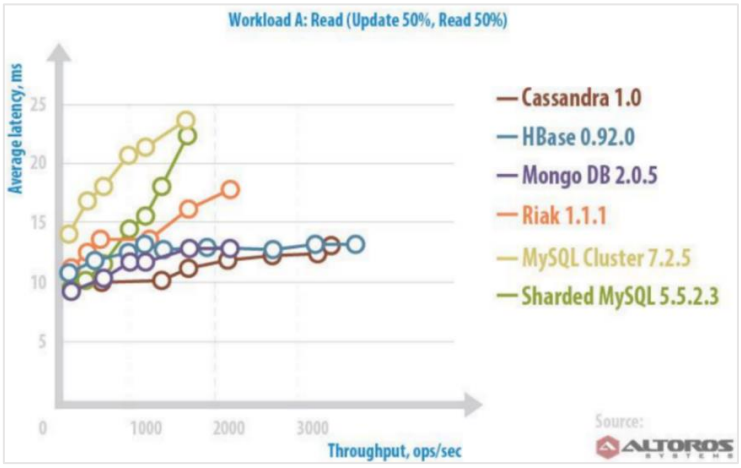

Figura 2. Comparación BBDD NoSQL en fase de lectura: 50% actualizaciones - 50% lecturas

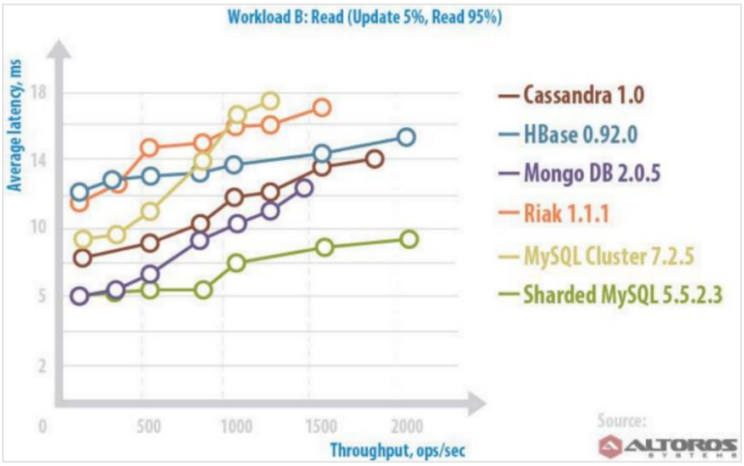

Figura 3. Comparación BBDD NoSQL en fase de lectura: 5% actualizaciones - 95% lecturas

Tabla 4. Posibles alternativas junto con su opción elegida.

	Opción elegida	Alternativas
Cliente móvil	Framework Ionic 2	Android, e iOS
Cliente escritorio	AngularJS	JavaScript, jQuery, y React
Base de datos	MongoDB con Mongoose	Base de datos relacional, Cassandra, HBase, Riak, MySQL cluster, y Sharded MySQL
Servidor web	NodeJS con Express	Spring Framework
Comunicación en tiempo real	Ninguna	Websockets con SocketIO o ws, polling con AJAX, y servidores push de notificaciones

Por último, como se comentaba al principio de la sección, en la tabla anterior se recogen todas las posibles alternativas que se podían haber escogido para el proyecto, las cuales hacen un total de 840 combinaciones (teniendo que escoger únicamente una), junto con las decisiones elegidas finalmente. Remarcar que dichas decisiones no tienen por qué ser mejores tecnologías que otras, sino que, en la situación y entorno del proyecto presentado, encajan de una forma mejor que otras.

7. Análisis de riesgos

Esta sección realiza un análisis de los riesgos potenciales del proceso de desarrollo más importantes. Anticipando lo que puede ocurrir, es posible mitigar el riesgo tomando las medidas oportunas. Los riesgos se aplican directamente al proceso de desarrollo o tienen una consecuencia directa sobre el mismo. El registro y monitorización de estos riesgos es una tarea que se ha de realizar de forma continua incluso después de que se haya redactado este informe, ya que es un proceso que nunca acaba hasta la muerte del proyecto.

Todos los riesgos identificados se encuentran en la **Tabla 5,** entendiéndose la columna I como Impacto (categorizado del 1-5), P como probabilidad, así como R exposición al riesgo (Impacto*Probabilidad). Como se puede observar y como resultado del análisis, se han identificado un total de 3 riesgos de nivel alto, 3 de nivel medio, y 2 riesgos de nivel bajo:

Tabla 5. Análisis de Riesgos.

Nº	Evento	Consecuencia	I	P	R	Medidas
1	La idea de Depot Cloud no está suficientemente definida.	Retraso en general del proyecto, inestabilidad durante el desarrollo, alcance no definido y estimaciones imprecisas.	5	3	15	Apoyarse en entrevistas a los usuarios implicados. Realizar una visión de alto nivel del modelo de negocio para ayudar a definir la solución. Tener más cuidado en la toma de requisitos, estableciendo una forma estructurada y formal de recogida de los mismos.
2	Falta de uso, en la gestión del proceso, de métricas e indicadores.	No poder analizar y mejorar la gestión del proceso en próximas iteraciones y lanzamientos.	4	2	8	Realizar una recopilación y análisis de métricas del proceso durante la duración del mismo.
3	Falta de formación en el uso de algunas herramientas de desarrollo.	Retraso en general en el proyecto debido a una baja eficiencia del desarrollo y/o posibilidad de no finalización del proyecto.	2	2	4	Establecer un periodo de aprendizaje para formar correctamente en las herramientas, lenguajes y componentes necesarios, dándole una mayor dedicación de esfuerzos durante el proyecto.
4	Novedad de la tecnología y los lenguajes de programación a construir en la organización.		4	4	16	
5	Creación de nuevos componentes para los requisitos.		4	4	16	
6	Los requisitos ponen restricciones excesivas de rendimiento del producto.	Posibilidad de que el proyecto no sea eficiente en entornos más restrictivos en rendimiento como la plataforma de dispositivo móvil.	4	2	8	Estudiar las restricciones que caracterizan los escenarios más estrictos y adaptarse a ellos.
7	El proyecto requiere más esfuerzo que el inicialmente previsto.		4	1	4	Establecer una definición de requisitos y un alcance durante el desarrollo del proyecto e ir

Nº	Evento	Consecuencia	I	P	R	Medidas
		Posibilidad de no finalización del proyecto o de no conseguir un producto mínimo viable.				trabajando sobre iteraciones continuas.
8	Desarrollo del proyecto de forma simultánea a otros proyectos (académicos).		5	1	5	Tenerlo en cuenta para las estimaciones.

El equipo de desarrollo es consciente de los riesgos y monitoriza las medidas adoptadas durante el análisis.

8. Descripción de la solución propuesta

Una vez terminada la fase de análisis del proyecto, con todos los estudios de alternativas y viabilidades, es cuando empieza la de diseño, la cual va a ser descrita en esta sección. El diseño de la aplicación ha sido orientado pensando en qué características eran las más importantes de cara a los usuarios objetivos, y el funcionamiento del sistema. Además, se han intentado llevar unos estándares de calidad con respecto al código y el proyecto, de forma que el proceso de desarrollo fuera lo más automático posible, además de reproducible y lo más cercano a lo que sería el desarrollo de un producto real con unos niveles de mantenimiento y calidad adecuados.

Los objetivos mencionados en el párrafo anterior, han sido determinantes a la hora de los estudios de viabilidad de la sección **Estudio de alternativas y viabilidad**, y del desarrollo de las tecnologías y la arquitectura escogidas, así como de la solución final. Estos objetivos principalmente eran la usabilidad, la escalabilidad, y el rendimiento. En la sección **Solución arquitectural** se tiene una visión gráfica y descriptiva de alto nivel de la arquitectura del sistema. En **Solución tecnológica,** se hace una descripción de los aspectos de escalabilidad y rendimiento, que han dado lugar a la solución actual con sus diferentes tecnologías. En **Integración continua, calidad y mantenimiento**, se detallan las acciones realizadas para conseguir un software con la metodología de integración continua, que tenga calidad, y que sea mantenible. En **Usabilidad de la solución** se detallará la usabilidad, su proceso y los principios de diseño en los que se ha basado la solución. Por último, en **Documentación de la API** se detalla cómo se ha documentado la API durante el desarrollo del proyecto y cómo se pretende abrir hacia el público.

8.1 Solución arquitectural

Como resultado de los requisitos se tiene que el proyecto tiene que implementar una aplicación web con dos clientes, uno para dispositivos de escritorio y otro para dispositivos móviles. Ya que se trata de una aplicación que no trata ningún problema en específico ni necesita de una arquitectura especial, sino que es una aplicación web común, se ha aplicado el patrón de diseño arquitectural Modelo-Vista-Controlador, implementando el sistema en capas de abstracción, con todas las ventajas que ello supone, pudiéndose ver una aproximación de muy alto nivel en la **Figura 4**. Además, se ha intentado simplificar el modelo de datos al máximo posible, a la vez que intentando emularlo con el dominio real del problema como refleja la **Figura 5** en un nivel muy alto. Toda la información detallada de la arquitectura del sistema se encuentra en el anexo **Análisis y Diseño del sistema.**

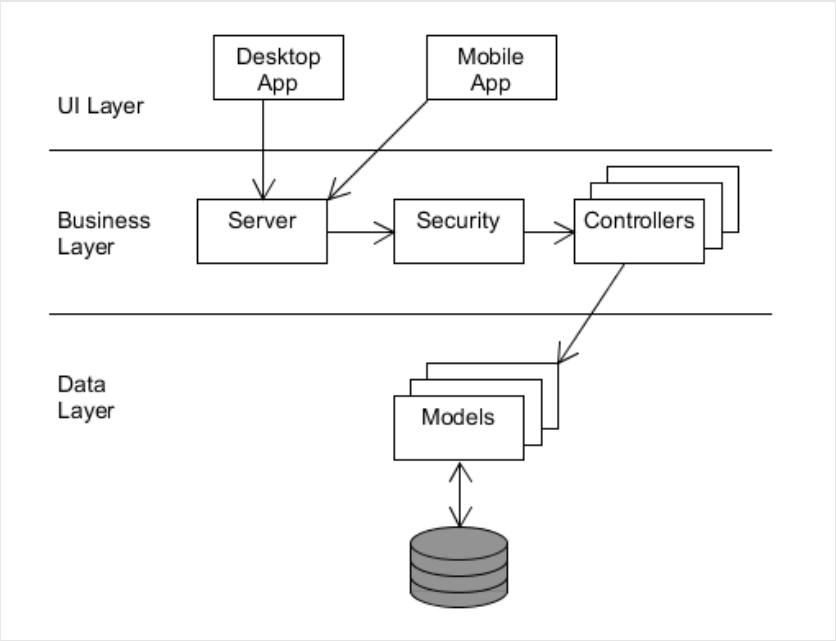

Figura 4. Visión gráfica arquitectural de alto nivel de la solución.

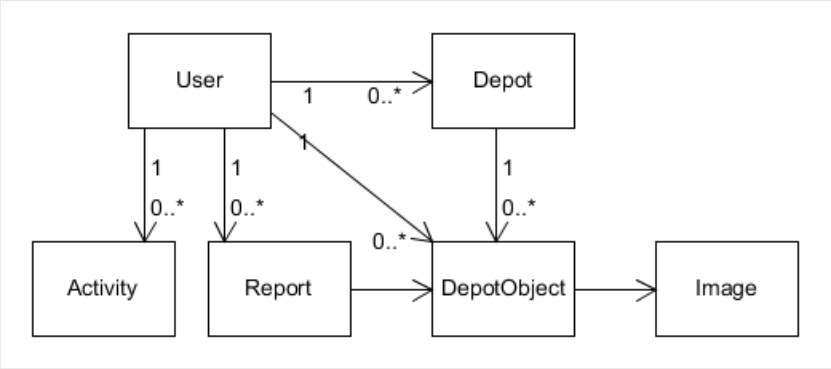

Figura 5. Visión gráfica de alto nivel del modelo de datos de la solución.

8.2 Solución tecnológica

Al haberse implementado el sistema con el framework MEAN, se ha conseguido un sistema como el de la **Figura 6**. Todo ello ha otorgado algunas características esenciales en el sistema, las cuales se pueden ver en la **Tabla 6**.

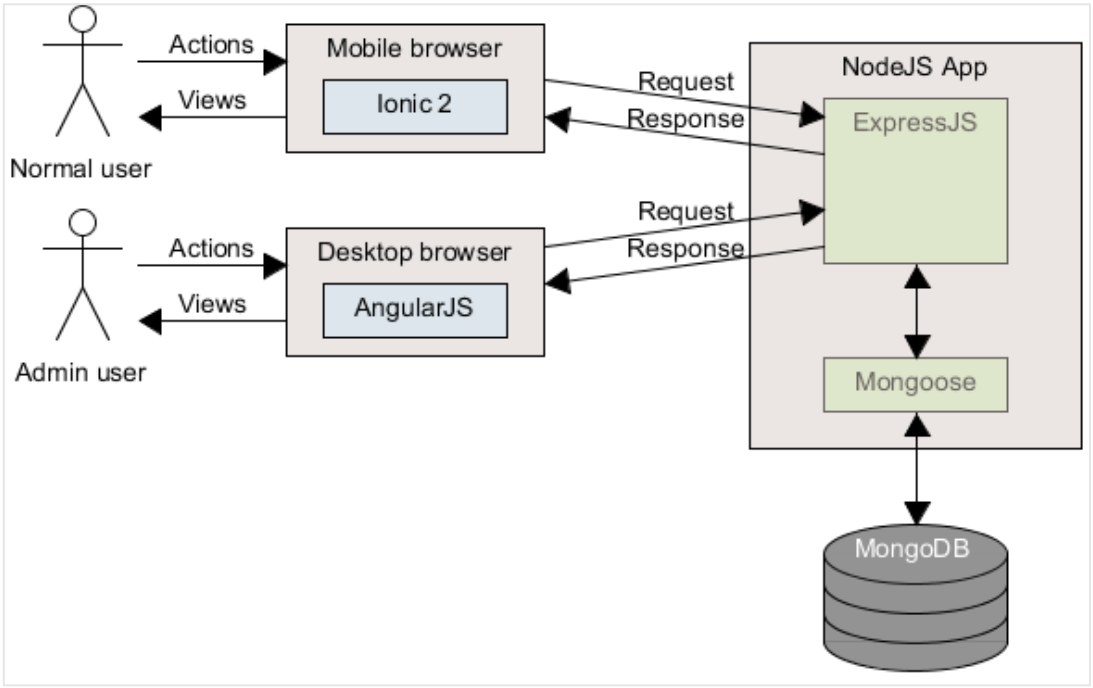

Figura 6. Visión gráfica conceptual de alto nivel de la solución.

Tabla 6. Características esenciales del sistema con el framework MEAN.

Capa de presentación	Desarrollada con las tecnologías AngularJS e Ionic 2, siguiendo un estilo SPA que permite establecer peticiones HTTP con la tecnología AJAX. Esto influye en reducir la carga de los servidores, y ofrecer una mejor experiencia de usuario, pudiendo realizar consultas asíncronas al servidor sin tener que recargar la página completamente
Capa de servicios web y controladores	El diseño que se ha seguido es utilizando el patrón *API-First*, construyendo el sistema a partir del diseño de la API. Puesto que se partía de un sistema desde cero y completamente nuevo, era la mejor forma de empezar a diseñarlo. Esto permite tener divisiones más claras entre la capa de servicios y la capa de front-end, consiguiendo menor acoplamiento en el sistema, repitiendo menos código en los controladores y manteniéndolos unificados. Además, dado que se trata de una

	aplicación claramente orientada a recursos, y haciendo el sistema lo más escalable posible, se ha utilizado el paradigma RESTful para diseñar la API y el comportamiento del sistema. De esta forma, se aprovecha al máximo el estándar HTTP, protocolo que, al no tener estado, y además ser fácilmente cacheable, da la ventaja a la hora de aumentar la escalabilidad. Por otro lado, la tecnología usada de NodeJS permite muy fácilmente replicar el sistema horizontalmente, ya que está pensado para generar peticiones no bloqueantes.
Capa de persistencia y búsqueda	Al haberse utilizado la tecnología de MongoDB, su principal ventaja es el uso de la técnica de *sharding* [27]. Esta técnica permite a la base de datos particionar los documentos en árboles para poder replicarlos fácil e independientemente. Además, dicha técnica MongoDB la implementa de forma transparente gracias a su tecnología, proporcionando mecanismos internos para realizarla sin que sea necesario implementarla a nivel de aplicación.

Por último, en lo que respecta a seguridad, se ha implementado en el sistema la tecnología de los JSON Web Tokens. Esta tecnología permite crear unos tokens en el momento de la autenticación, para su posterior envío en cada una de las peticiones HTTP que se realizan en el sistema. Esta tecnología permite gracias a eso cifrar la comunicación dentro del token, y además simular una sesión activa (ya que se les puede otorgar un tiempo de expiración a los mismos). Además, puesto que únicamente se trata del cifrado de un token, esto permite tener una sesión en el sistema, y añadirle un nivel de seguridad, consumiendo un número mínimo de recursos y sin que influya en la escalabilidad del sistema.

Lo comentado hasta ahora es lo que el sistema puede hacer actualmente, sin embargo, se ha diseñado pensando también en aspectos futuros, para poder ampliar la escalabilidad sin tener problemas de integración. La propuesta tecnológica que se plantea para la siguiente iteración consistiría en lo siguiente:

- En los temas de seguridad, se puede integrar fácilmente el uso de la tecnología de reCAPTCHA de Google [28] para evitar un uso de BOTs y SPAM en el sistema en el requisito **[RFUF-1]** (registrar cuenta de usuario).
- En los temas de escalabilidad, resiliencia, y recuperación, se puede integrar de una forma casi inmediata los servicios de Amazon S3 (*Amazon Simple Storage Service*) [29]. Utilizando dichos servicios se externalizarían estos riesgos (con lo que ello conlleva). Dichos servicios garantizan un 99.99% de disponibilidad en la nube (lo que se traduce en que el servicio está inactivo para los usuarios únicamente durante 48 minutos al año), y un 99,999999999% de durabilidad de los objetos en la nube (lo que se traduce en que, si se guardan 10.000 objetos, de media se pierde 1 de ellos cada 10 millones de años). Además, el servicio *cloud* de Amazon permite el escalado horizontal a demanda del sistema (integrándose con la escalabilidad actual del framework MEAN), de forma que el presupuesto es bastante flexible y se amoldaría a las necesidades de la aplicación en cada momento determinado de demanda. Esto permite tener un sistema altamente escalable y disponible a un precio flexible y ajustado.

8.3 Integración continua, calidad y mantenimiento

En cuanto al proceso de calidad y mantenimiento seguida durante el desarrollo del producto, se han utilizado algunas herramientas, tecnologías y metodologías para poder conseguirlo. Primero, se ha aplicado la metodología de integración continua (como se puede ver en la **Figura 7**), integrando el código todo el tiempo, pasando los tests correspondientes, y permitiendo que cada poco tiempo se puedan hacer entregas con un valor potencial para el cliente teniendo siempre un producto mínimo viable. Cada vez que se realiza un *commit*, se construye todo automáticamente, todo se encuentra bajo un control de versiones, y, además, se cuenta con unas definiciones de hecho para que las entregas sean de mayor calidad. Estos procesos se detallan en los anexos **Definiciones de hecho y estándares del proyecto**, **Estrategia para la construcción automática del software,** y **Estrategia de control de versiones**.

En segundo lugar, para asegurarse un producto con una calidad adecuada, se cuenta con una batería de tests automáticos. Dicha batería consiste en un total de 136 tests unitarios que suponen un 90.55% de cobertura de código, y 10 caminos críticos realizados para tests de aceptación y de sistema. Todos lo tests unitarios se

lanzan automáticamente en cada integración del software a mano de la herramienta de TravisCI [30] siguiendo las pautas de la integración continua. También se cuenta con una herramienta llamada SonarQube [31] que es un analizador estático de código, para ayudar a hacer un software de mayor calidad que consiga tener unos niveles altos de mantenimiento. Estos procesos se detallan en los anexos **Estrategia y herramientas usadas para los tests** y **Mantenibilidad y Quality Assurance**. Además, se ha hecho uso de la tecnología de EditorConfig [32], una herramienta que automatiza el estilo de la codificación para conseguir un código consistente y uniforme.

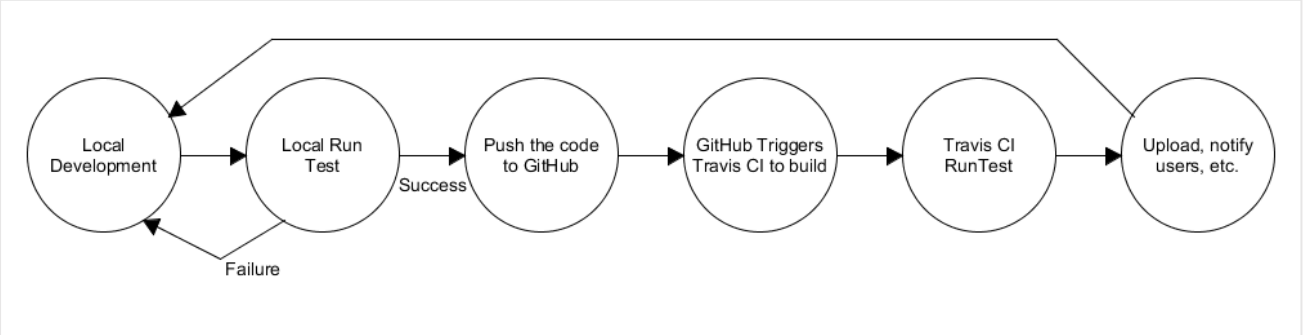

Figura 7. Diagrama de alto nivel del proceso seguido de integración continua.

8.4 Usabilidad de la solución

Se ha realizado un gran esfuerzo para conseguir una aplicación usable en el sistema, ya que principalmente está orientado a un público muy amplio que no siempre tiene porqué tener experiencia con la tecnología. Para conseguirlo, se han aplicado algunos principios de diseño generales en la capa de presentación y la UI los cuales se pueden ver en la **Tabla 7**, dando como resultado las vistas de la aplicación (pudiendo ver algún ejemplo en la **Figura 8** y la **Figura 9**).

Tabla 7. Principios de diseño generales de la UI.

Consistencia	Se ha intentado que toda la aplicación funcione con los mismos estilos, iconos y formas de manipulación de interfaz, para que el usuario una vez sepa hacer cualquier acción dentro de ella, el resto le sean fácilmente recordables y la curva de aprendizaje sea más rápida. Además, se mantiene la consistencia gracias a la tecnología de Ionic 2, en el uso de iconos y distribución de la UI, con los estilos generales dependiendo del dispositivo móvil que lo esté ejecutando (iOS, Android, o Windows Phone), de forma que para ellos algunos componentes les resulten familiares (p.ej.: El botón del navbar).
Retroalimentación	Todas las acciones que realiza el usuario tienen su propia retroalimentación con mensajes llamativos y textos explicativos que permiten al usuario saber lo que está pasando gracias a sus acciones en todo momento, así como el estado de la aplicación.
Número mágico	Gracias al estudio de George A. Miller [33] se sabe que la memoria de trabajo de las personas puede almacenar una cantidad máxima de 7 (+/- 2) elementos o de unidades de información que se pueda recordar, se ha seguido este principio también para no sobrecargar al usuario en ninguna pantalla con más acciones de las que pueda recordar.

Además, uno de los objetivos principales de la usabilidad era que la realización de las tareas fuera muy rápida, sin tener que interactuar muchas veces, ya que los usuarios solo buscan realizar una consulta o introducir un nuevo dato. Es por eso que el objetivo aquí era conseguir que las tareas tuvieran un tiempo de realización de entre 1-3 minutos (obteniendo la mayor de ellas 154.5s), que la escala de satisfacción del usuario fuera muy positiva, y que hubiera más de un 75% de usuarios que alcanzaran sus objetivos y terminaran con éxito sus tareas (obteniendo que un 100% de los usuarios han conseguido terminar con éxito las tareas analizadas).

Como se puede ver en la sección **Usabilidad del sistema**, en la cual se han realizado unos análisis con técnicas de validación de usabilidad (KLM [34] para los tiempos de realización, recorrido cognitivo [35] para las tareas con éxito y heurísticas de Nielsen [36] para el sistema en general), se han cumplido los objetivos propuestos. Por último, se puede encontrar un mapa de navegación en el anexo **Mapa de navegación** y el estado final de la aplicación con cada una de las vistas y los requisitos que pertenecen a ellas en el anexo **Estado de la aplicación final**.

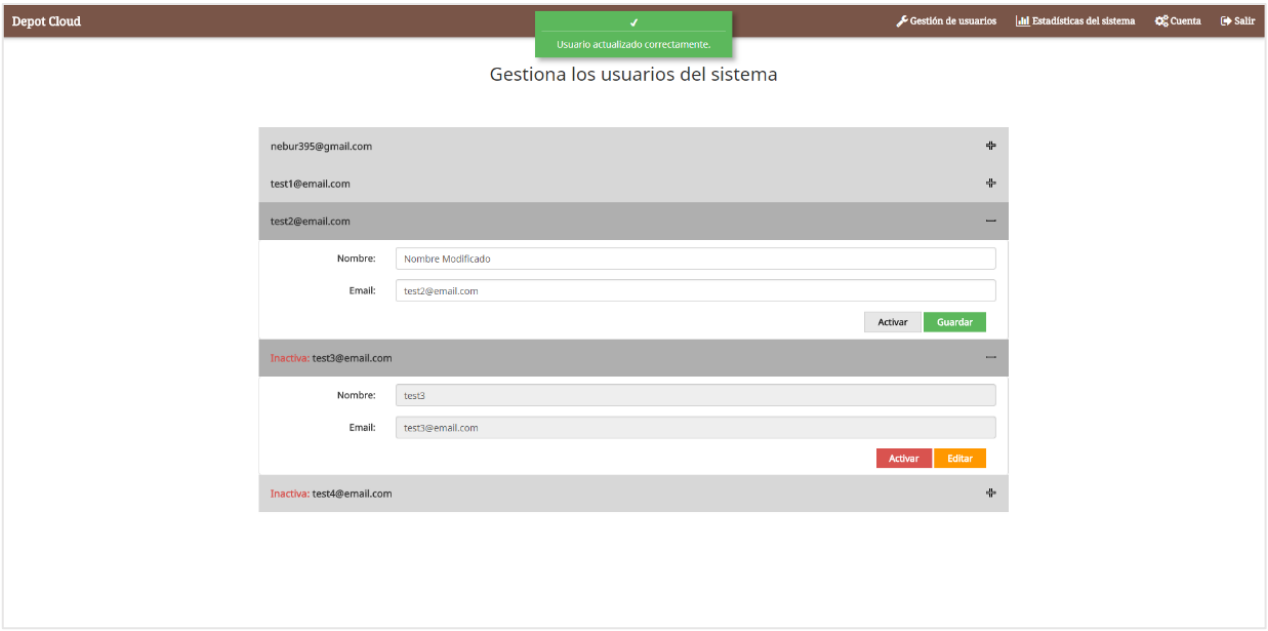

Figura 8. Ejemplo de vista para dispositivos de sobremesa.

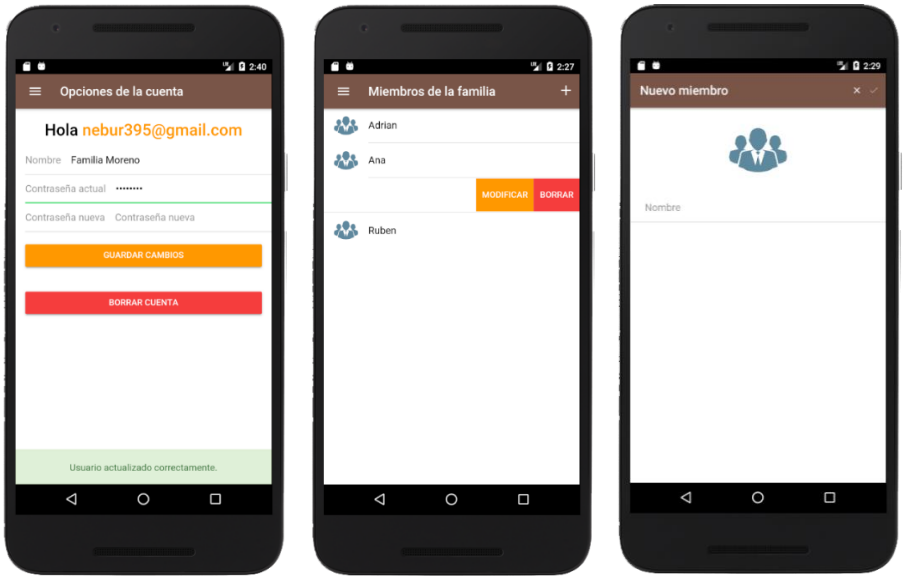

Figura 9. Ejemplo de vista para dispositivos móviles.

8.5 Documentación de la API

Para que la API RESTful pública del sistema sea accesible por cualquier entidad externa, y esté correctamente documentada, se ha utilizado la herramienta de Swagger [37], la cual sincroniza automáticamente la documentación del código para transformarla en formato JSON y posteriormente, con el servidor en ejecución, la presenta por pantalla en las direcciones “localhost:8080/swagger.json” sin formatear y “localhost:8080/api-docs/” ya formateada. Esto hace que la API tenga un formato de presentación intuitivo y fácilmente accesible

por entidades de terceros, y además se mantenga sincronizada la documentación de la misma con el código, dando un resultado como en la **Figura 10** y la **Figura 11** . Toda la API en detalle del sistema, con un total de 36 operaciones, se encuentra en el anexo **API RESTful del sistema**.

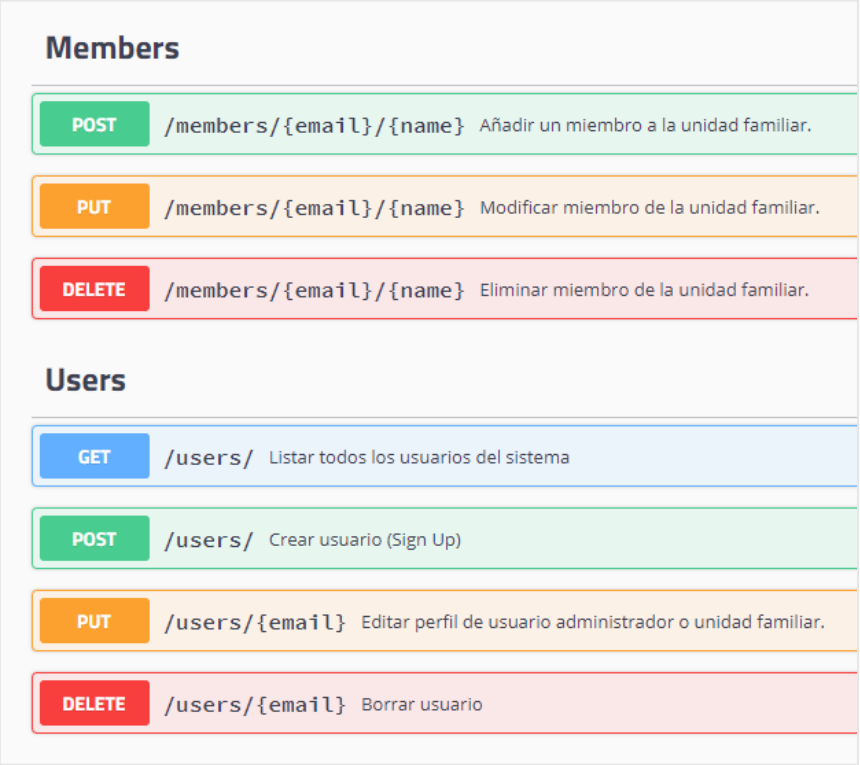

Figura 10. Ejemplo de Swagger: Lista de operaciones.

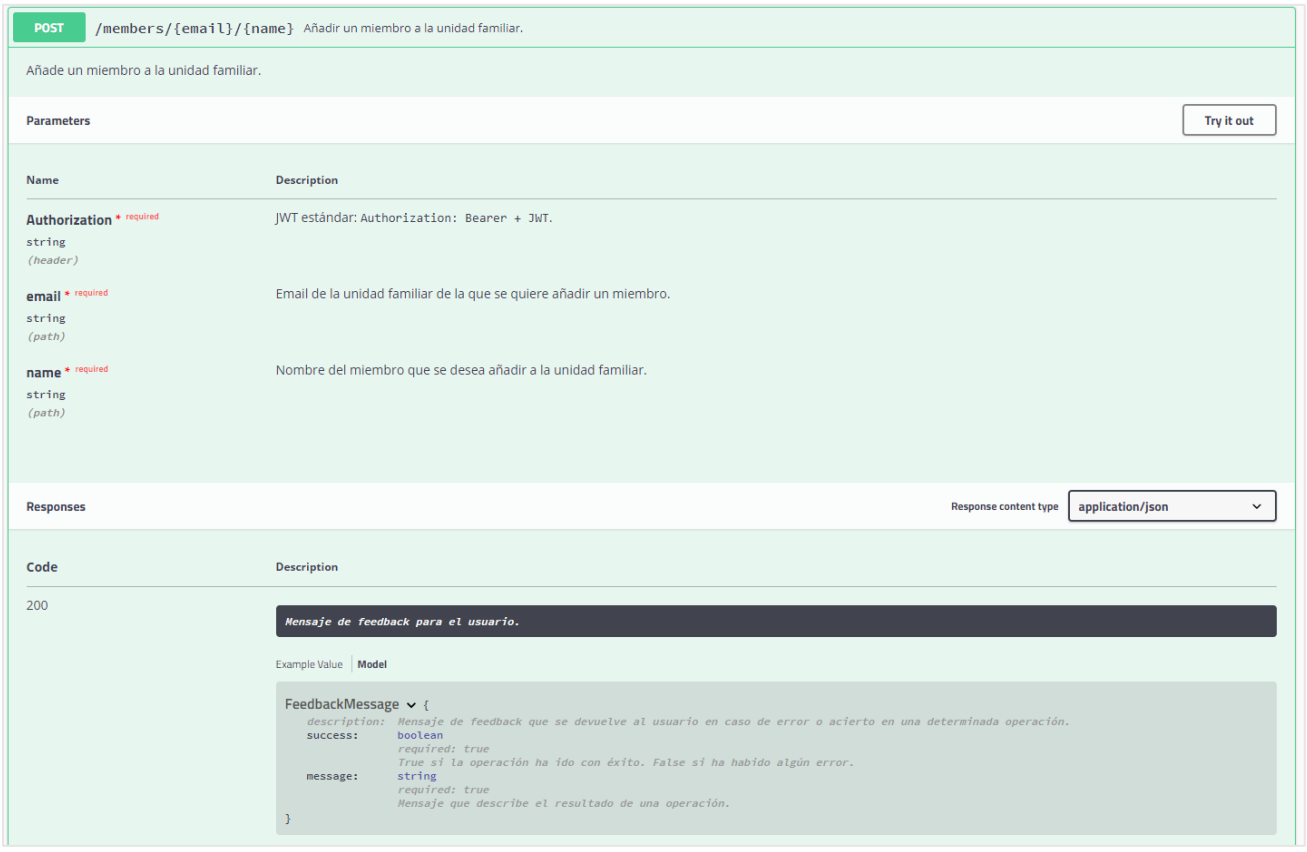

Figura 11. Ejemplo de Swagger: Operación ampliada.

9. Implementación de la solución

En este apartado se describe dos aspectos de la implementación clave: el proceso de implementación y los retos técnicos de la implementación. Si se desea profundizar en los aspectos técnicos de la implementación se debe acudir al anexo **Detalles de implementación del sistema** donde se describe la implementación de los clientes, los servicios y la persistencia. Además, en **Estadísticas implementadas** se recogen los diferentes informes que es capaz de elaborar el sistema.

La implementación de la solución se ha realizado siguiendo un proceso determinado (el cual se puede ver en la **Figura 12**):

- Por cada funcionalidad a implementar se creaba una *issue* en el VCS de GitHub con una pequeña explicación de lo que tenía que incluir. En ella, se asignaba a las personas correspondientes de llevarla a cabo, y se comentaba todo lo relevante a dicha funcionalidad (problemas, soluciones, actualizaciones de estado, etc.). Por último, se añadía al tablero Kanban de GitHub, y se ponía a la cola de prioridades en la columna TODO hasta que el encargado de implementarla se ponía a trabajar en ella y la movía a la columna de DOING (como se puede ver en la **Figura 13**).
- En el desarrollo de los clientes del sistema, se utilizaba la técnica de despliegue en caliente, la cual permite reflejar los cambios que se realizan en el código en los navegadores web cada pocos segundos, de forma que se conseguía una forma rápida y usable de depurar e implementar la interfaz y el cliente web. Además, se colocaban mockups simulando respuestas del servidor en caso de que estas no estuvieran implementadas.
- En el desarrollo del servidor web y la persistencia se desarrollaban las funcionalidades mientras se hacían pruebas manuales para depurar con la herramienta de pruebas para APIs de Postman, y con el cliente de terminal de MongoDB realizando consultas directas a la base de datos.
- Cuando una funcionalidad se había acabado en ambos puntos, se quitaban los mockups correspondientes y se probaba la integración de forma manual.

Además, cada uno de los dos procesos, contaba con unos pasos comunes:

- Cada vez que se finalizaba el desarrollo de una funcionalidad se establecía una batería de tests para ella, que a partir de entonces se tendría que integrar y pasar cada vez que se realizara cualquier cambio siguiendo la práctica de integración continua, para comprobar en el futuro que no hay nada que pueda introducir fallos en dicha funcionalidad.
- Habiendo completado la batería de tests correspondientes a la funcionalidad, se hacía un escaneo de la herramienta de SonarQube para comprobar si los niveles de calidad y mantenibilidad del código pasaban las reglas establecidas, y en caso de que no los pasaran corregir los problemas añadidos.

Una vez finalizado todo el proceso anterior, se daba por concluida la funcionalidad correspondiente cerrando la issue y moviéndola a la columna de DONE en el tablero Kanban.

Por último, añadir que dado que se trata de un sistema que ha seguido el framework MEAN, no ha tenido problemas de integración. Sin embargo, el cliente de dispositivos móviles implementado con el framework Ionic 2, que no forma parte del framework MEAN, ocasionó un problema crítico durante el desarrollo al contar con su propio servidor web. Este problema es conocido como CORS [38]. Dicho problema consiste en que cuando se está desarrollando, se usa el comando '**ionic serve',** lo que hace que un servidor web local arranque, y el navegador se abra apuntando a la dirección local ("localhost:8100"). Esto hace que el campo header "origin" de las peticiones HTTP de AJAX sea "localhost:8100", el cual no coincide con el host al que se envía la petición (en este caso inicializado en un servidor web distinto con "localhost:8080"). Esto ocasiona que el "origin" del servidor que ha lanzado la petición tenga que preguntar al servidor si se le aprueba el acceso al recurso en un HTTP OPTIONS, que, en este caso, era rechazado dando lugar a un error.

Para resolver esto se plantearon inicialmente dos opciones: permitir todos los headers “origin” desde el back-end, o implementar un servidor proxy en el cliente Ionic 2 que no especifique un “origin” en las peticiones. Puesto que en este caso se tiene acceso a modificar el back-end, se ha escogido la primera opción, dado que la segunda, posee problemas de mantenimiento a largo plazo. Ya que este problema solo aparece cuando se está ejecutando o depurando la aplicación cliente de Ionic 2 con los comandos “**ionic serve”** o **“ionic run (-1)”** se ha utilizado una librería que se detalla en el anexo **Back-end** para atacar la solución tomada.

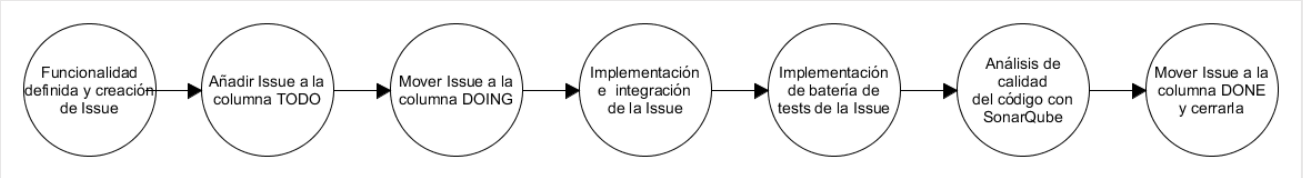

Figura 12. Proceso seguido durante la implementación del sistema.

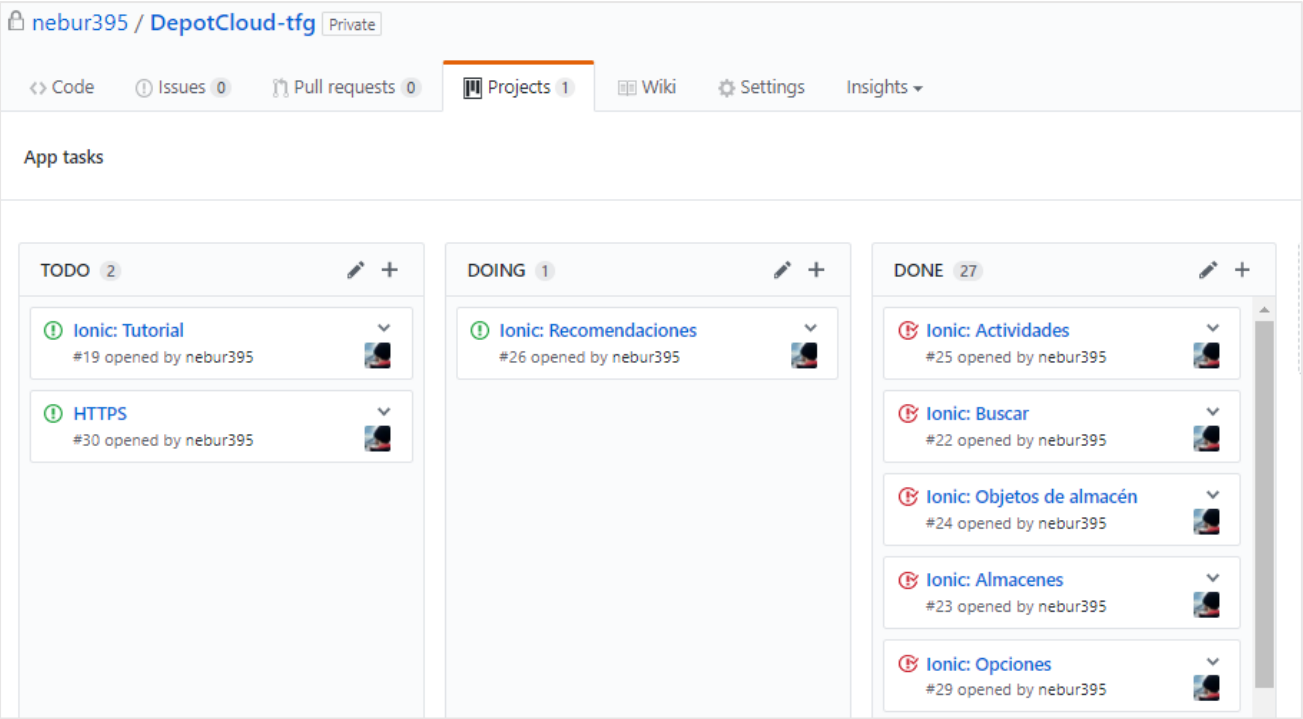

Figura 13. Kanban del proyecto usando GitHub.

10. Gestión del proyecto

10.1 Planificación del proyecto

El proyecto ha seguido una aproximación ágil, dividiendo el desarrollo en varias iteraciones para poder conseguir cuanto antes un producto mínimo viable y tener siempre algo de valor para poder trabajar sobre él, ampliarlo poco a poco y en un caso extremo, incluso tener algo que poder entregar como producto final recortando alcance y/o calidad.

Además, esta aproximación que se ha seguido se adapta a la situación actual, ya que el desconocimiento de las tecnologías o poca experiencia trabajando con ellas, además de tener los plazos fijados, pero el alcance y calidad variables, ha ayudado a que el análisis de requisitos y diseño del sistema vaya evolucionando a la vez que el proyecto, refinándose así los anteriores continuamente hasta el final de la aplicación.

En la **Figura 14**, se puede ver el diagrama de Gantt realizado a priori de la planificación del proyecto y en la **Tabla 8** una descripción de sus lanzamientos e iteraciones.

Tabla 8. Planificación a priori: Lanzamientos e iteraciones del proyecto.

Lanzamiento del proyecto	**2017/02/17 - 2017/03/20**
Reuniones de lanzamiento del proyecto en las que se decide la motivación de la idea, un análisis y diseño inicial del proyecto, incluyendo la maduración del plan de producto y la captura inicial de requisitos, una planificación, y una investigación de las diferentes tecnologías a usar.	
Primer lanzamiento – Primera iteración	**2017/03/21 - 2017/03/31**
Implementación de la infraestructura del proyecto, así como el primer producto mínimo viable, e investigar en profundidad las tecnologías elegidas en la iteración anterior.	
Primer lanzamiento – Segunda iteración	**2017/04/01 - 2017/05/31**
Implementación completa del resto del proyecto (las tres partes al completo, es decir, tanto el back-end, como el cliente móvil, como el cliente de escritorio), refinando las partes implementadas en la iteración anterior, así como requisitos o arquitectura si fuera necesario.	
Primer lanzamiento – Tercera iteración	**2017/06/01 - 2017/06/30**
Finalización de redacción de la memoria y el proyecto, dando los toques finales al mismo.	

En la **Figura 15**, se puede ver el diagrama de Gantt realizado a posteriori de la planificación del proyecto y en la **Tabla 9** una descripción de sus lanzamientos e iteraciones:

Tabla 9. Planificación a posteriori: Lanzamientos e iteraciones del proyecto.

Lanzamiento del proyecto	**2017/02/17 – 2017/04/09**
Reuniones de lanzamiento del proyecto en las que se decide la motivación de la idea, se refina el plan de producto y su estructura inicial, requisitos iniciales, tecnologías a usar, arquitectura software del sistema inicial, y planificación, gestión y organización del proyecto. Además, se inicia un periodo de investigación en profundidad y aprendizaje de las tecnologías elegidas para su posterior implementación.	
Primer lanzamiento – Primera iteración	**2017/04/10- 2017/05/08**
Realización de la primera iteración del primer lanzamiento del producto, incluyendo únicamente la integración de todas las diferentes tecnologías, entre las que se encuentra el servidor del back-end en Express y MongoDB, el front-end de cliente escritorio con AngularJS, el front-end de cliente móvil con Ionic 2 y Angular, la infraestructura de testing con protractor, mocha y chai, la infraestructura de QA con SonarQube, la seguridad del sistema con JSON Web Tokens, y las herramientas de integración continua de Git, GitHub, TravisCI, y Codecov.	
Primer lanzamiento – Segunda iteración	**2017/05/09- 2017/06/23**
Realización de la segunda iteración del primer lanzamiento del producto, incluyendo la finalización del cliente de escritorio para usuarios administradores, y toda la implementación del back-end del proyecto entero, incluyendo funcionalidades tanto de usuario administrador como de usuarios comunes.	

Primer lanzamiento – Tercera iteración	2017/06/24- 2017/07/18
Realización de la tercera iteración del primer lanzamiento del producto, incluyendo la finalización del cliente de dispositivos móviles y finalizando así la implementación del proyecto haciendo funcionales todos los requisitos establecidos para el proyecto.	
Primer lanzamiento – Cuarta iteración	**2017/07/19- 2017/09/18**
Finalización de la documentación del proyecto, terminando de completar el presente documento.	

Como se puede ver, existe una clara diferencia tanto en la fecha de final del proyecto como en el número de iteraciones. Esto es debido a dos razones principales.

La primera es que durante la investigación de las tecnologías realizada en el lanzamiento del proyecto, se pudo ver que el cliente de dispositivos móviles realizado con la tecnología del framework Ionic 2 iba a ser más complicada de lo que se había pensado en un principio debido a su desconocimiento total, además de alargarse en el tiempo con respecto a lo estimado. Es por eso que, debido a dicho inconveniente, se decidió separar completamente la implementación de dicho cliente en una iteración distinta al resto del desarrollo del sistema.

La segunda razón, es que dado que se contaba también con poca experiencia en el framework MEAN, una de las ideas principales fue implantar algunas funcionalidades con el protocolo de Websockets. Sin embargo, al final esto no fue posible como se ha explicado anteriormente en este mismo documento, y la investigación que se hizo sobre diferentes tecnologías, así como sus diferentes pruebas de concepto, resultaron en un retraso considerable del proyecto.

Por último, se plantea un segundo lanzamiento con vistas al futuro incluyendo funcionalidades futuras. Puesto que el alcance del proyecto terminaría aquí, y no se saben las condiciones del proyecto ni de si el equipo de desarrollo cambiaría, se deja como un único lanzamiento sin desglosar en pequeñas iteraciones. Es por eso, que lo primero que se debería hacer al realizar el segundo lanzamiento es realizar una reunión de refinamiento de producto y planificación, para establecer y definir bien unos requisitos iniciales y planificar en cuantas iteraciones habría que desarrollarlos:

Tabla 10. Planificación del segundo lanzamiento.

Segundo lanzamiento	2017/09/18- 2017/12/31
Realización del segundo lanzamiento, incluyendo las funcionalidades del primero y las funcionalidades que se han descrito en la sección "**Alcance**".	

10.2 Esfuerzos distribuidos por tiempo y actividad

El proyecto desarrollado ha tenido una duración de 471 horas, 368 de ellas de desarrollo, y 103 de formación en nuevas tecnologías. Se ha llevado a cabo una recopilación de los esfuerzos realizados en horas en cada una de las actividades distribuidos a lo largo de la duración del proyecto. Dicha recopilación se puede ver en la **Tabla 11** y en la **Tabla 12** que contienen el desgranado de las horas por cada mes y actividad respectivamente.

Tabla 11. Distribución de horas por meses.

	Actividades	Formación	TOTAL
Febrero	11	0	**11**
Marzo	14	16	**30**
Abril	13	34	**47**
Mayo	21	6	**27**
Junio	91	23	**114**
Julio	128	24	**152**
Agosto	90	0	**90**
Septiembre	0	0	**0**
TOTAL	**368**	**103**	**471**

Tabla 12. Distribución de horas por actividades

Análisis y diseño del sistema	22
Desarrollo	136
Pruebas	29
Gestión del proyecto y documentación	163
Gestión de configuraciones	10
Quality Assurance	8

10.3 Presupuesto

En esta sección se realiza un cálculo aproximado del presupuesto que tendría el proyecto realizado. Para ello, se ha tenido en cuenta el material utilizado durante su desarrollo, y las horas de trabajo acumuladas calculadas en la sección anterior. En primer lugar, se ha calculado el coste del desarrollador por las horas trabajadas en el proyecto. Para ello se ha consultado el Instituto Aragonés de Estadística, el cual indica que un trabajador en el sector servicios tiene un salario de media de unos 15€ la hora aproximadamente [39]. En segundo lugar, se han calculado los siguientes costes extra por mes:

- **Mobiliario de oficina**: Los costes aquí han sido de una silla usada durante el trabajo (con un precio de 95€), y de una mesa (con un precio de 150€). Para el cálculo de la amortización se ha sacado el 10%[2] estimado de su vida útil (consiguiendo así el coste anual), y posteriormente se ha dividido por 12 para sacar el coste mensual, de tal forma que la amortización queda con la fórmula siguiente: $\frac{((95+150)*0,1)}{12} = 2,04\ €$
- **Equipo informático**: Los costes aquí han sido de un ordenador (con un precio de 500€), una impresora (con un precio de 300€), y un móvil (con un precio de 200€). La amortización se ha calculado de la misma forma mostrada anteriormente, pero con el 25%: $\frac{(500+300+200)*0.25}{12} = 20,83\ €$
- **Programas informáticos**: Los costes aquí han sido de los programas Microsoft Office (con un precio de 149€), y de IntelliJ IDEA (con un precio de 300€). La amortización se ha calculado de la misma forma mostrada anteriormente, pero con el 33%: $\frac{(149+300)*0.33}{12} = 12,35\ €$
- **Teléfono e internet**: 35,00€ mensuales aproximados, basados en datos históricos del desarrollador.

Por último, se han sumado todos los gastos, obteniendo así el presupuesto total del proyecto, el cual se queda dentro del estimado en la sección **Modelo de negocio**:

Tabla 13. Cálculo total aproximado del presupuesto del proyecto.

Tareas	Horas/Ítem	Coste/hora	Meses	Coste/Mes	Coste (€)
Desarrollo	368	15,00 €			5.520,00 €
Formación	103	15,00 €			1.545,00 €
Amortización: mobiliario de oficina			8	2,04 €	16,33 €
Amortización: equipo informático			8	20,83 €	166,67 €
Amortización: Programas informáticos			8	12,35 €	98,78 €
Teléfono e internet			8	35,00 €	280,00 €
TOTAL					7.626,78 €

[2] Porcentajes sacados de las tablas de amortización fiscal en: http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml

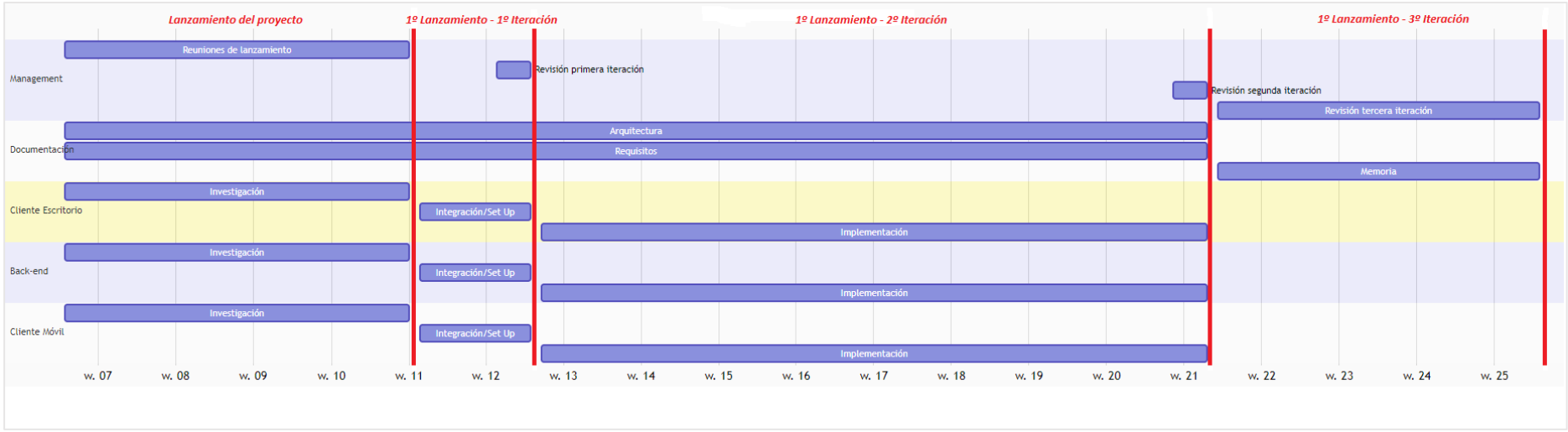

Figura 14. Diagrama de Gantt a priori del proyecto.

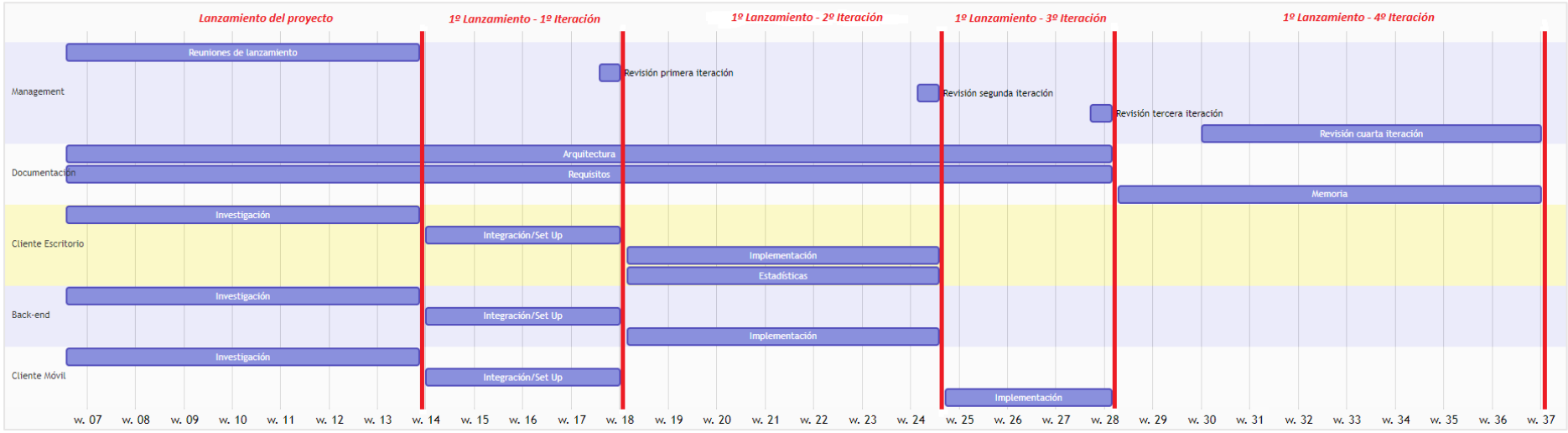

Figura 15. Diagrama de Gantt a posteriori del proyecto.

11. Conclusiones y trabajo futuro

Como se ha comentado a lo largo del documento, los principales objetivos del trabajo eran: crear una aplicación web de gestión de trasteros para unidades familiares y usuarios comunes, y documentar y gestionar todo el proceso propio de un sistema software profesional. Se considera que dichos objetivos se han completado satisfactoriamente a lo largo del desarrollo, viéndose los resultados en el documento presente, tanto de la aplicación implementada como de la documentación desarrollada.

Como conclusiones generales, cabe destacar la importancia que ha tenido un buen proceso de gestión, documentación y planificación de principio a fin del proyecto, teniendo en cuenta cada una de las fases documentadas aquí. Esto ha permitido poder evolucionar y desarrollar una idea novedosa como es *Depot Cloud* de la que no se tenía clara la trayectoria, cumpliendo las restricciones establecidas:

- Gracias a la fase inicial de definición del producto, la idea consiguió tomar forma rápidamente pudiendo de esta forma empezar a trabajar en fases posteriores como el análisis y el diseño, así como tener una idea más clara de qué era lo que se quería y cuál era el alcance del proyecto, así como poder ver si podría resultar viable o no.
- Gracias a la fase de análisis del sistema se pudo establecer un procedimiento riguroso de recogida de requisitos para que la tarea fuera más estructurada y a la vez flexible. Se pudieron descartar las numerosas combinaciones de tecnologías que se presentaban para finalmente poder escoger una que funcionara bien en el entorno del proyecto. Además, otro aspecto muy importante ha sido el análisis de riesgos, el cual ha permitido que se desarrolle el proyecto sin imprevistos, y la capacidad de saber en qué fases había que invertir más o menos esfuerzo. Se pueden observar algunas consecuencias inmediatas de dicho análisis de riesgos en algunas secciones del documento, pudiéndose observar la gran importancia que se les ha dado a los mismos y cómo han servido para que el proyecto se desarrollara con éxito:
  - Se observa como gracias a las secciones **Depot Cloud**, **Alcance y restricciones**, y **Documentación de requisitos**, se ha invertido gran esfuerzo en mitigar el riesgo número 1, para conseguir una idea sólida de la solución que se quería definir.
  - Se observa como durante el proyecto se han recogido algunas métricas e indicadores para mitigar el riesgo número 2, como los que aparecen en la sección **Gestión del proyecto**, consiguiendo con ellos poder mejorar la gestión del proceso en próximas iteraciones.
  - Se observa como a lo largo del proyecto se ha invertido un gran esfuerzo en horas de formación para mitigar los riesgos 3, 4 y 5. Esto se puede ver en la sección **Esfuerzos distribuidos por tiempo y actividad**, en la cual se puede comprobar que casi 1/3 de los esfuerzos del proyecto se han dedicado únicamente a la formación.
  - Se observa como durante la sección **Estudio de alternativas y viabilidad**, y **Descripción de la solución propuesta**, se ha tenido en cuenta el entorno y situación del proyecto para descartar algunas posibilidades (como por ejemplo la técnica de *polling* con AJAX), debido a que reduciría el rendimiento de los dispositivos móviles y mitigando así el riesgo número 6.
  - Se observa a lo largo del documento, y más específicamente durante las secciones **Alcance y restricciones** y **Gestión del proyecto** como se han mitigado los riesgos 7 y 8 adoptando unas metodologías ágiles de desarrollo, así como un buen análisis del alcance del proyecto, y estableciendo una planificación detallada en pequeñas iteraciones teniendo en cuenta el entorno académico del alumno.
- Gracias a la fase de diseño se pudo empezar a implementar correctamente el sistema habiendo asentado unas bases y una trayectoria estables para que el proceso fuera de una calidad alta y se tuviera claro siempre qué hacer, y en qué invertir esfuerzo o no para que el sistema pudiera funcionar de la forma más óptima. Además, gracias a las metodologías seguidas, esta fase se consiguió que fuera lo más eficiente y rápida posible.

- Por último, gracias a la fase de gestión del proyecto se pudo hacer una planificación adecuada y ágil del proyecto, pudiendo así finalmente cumplir los plazos de las restricciones y que todo acabara desarrollándose sin ningún problema crítico. Además, gracias a la recopilación rigurosa de esfuerzos durante la vida del mismo, se pudo posteriormente hacer una aproximación más exacta del presupuesto que tendría el proyecto completo.

Cabe destacar en estas conclusiones que unos de los problemas más comunes a lo largo de este desarrollo (recogidos todos en el análisis inicial de riesgos) han sido los retrasos de plazos y algunas alternativas descartadas por los mismos. Estos inconvenientes (pese a haberse resuelto de una forma óptima sin restarle calidad a la solución final, gracias a que ya se tenía en cuenta que pudiera pasar) han surgido precisamente por ser la primera iteración y lanzamiento del producto, y no haber tenido datos anteriores con los que poder realizar unas estimaciones más precisas. Sin embargo, gracias al presente documento en el que se ha detallado todo el proceso, se cuenta con muchos más datos y conocimientos sobre el proyecto para poder realizar unas estimaciones que sean mucho mejores.

Por último, con vistas al trabajo futuro ya se han ido dejando posibles mejoras durante el documento. En concreto habría que plantear, refinar y planificar el segundo lanzamiento propuesto, y realizar un análisis de investigación y pruebas de concepto de comunicación en tiempo real con la tecnología de servidores *push* de notificaciones para móviles. Para ello, habría que realizar una recopilación de los datos de este documento, en cuanto a riesgos, alternativas, problemas surgidos, y, sobre todo, esfuerzos invertidos, para poder planificar las siguientes iteraciones de una forma precisa.

Bibliografía

[1] Consejo General de Colegios Profesionales de Ingeniería en Informática, «Norma Técnica para la realización de la Documentación de Proyectos en Ingeniería Inofrmática,» CCII, Mayo 2016. [En línea]. Available: http://cpiicm.es/wp-content/uploads/sites/3/2016/11/CCII-N2016-02-Norma-Tecnica-para-la-realizacion-de-la-Documentacion-de-Proyectos-en-Ingenieria-Informatica-V1.0-f.pdf. [Último acceso: 22 Agosto 2017].

[2] M. Rouse, «Integrated development environment (IDE),» Junio 2016. [En línea]. Available: http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment. [Último acceso: 1 Agosto 2017].

[3] M. Wales, «Front-end vs Back-end vs Full-stack,» 8 Diciembre 2014. [En línea]. Available: http://blog.udacity.com/2014/12/front-end-vs-back-end-vs-full-stack-web-developers.html. [Último acceso: 1 Agosto 2017].

[4] M. Rouse, «Framework,» Febrero 2017. [En línea]. Available: http://whatis.techtarget.com/definition/framework. [Último acceso: 1 Agosto 2017].

[5] M. Rouse, «Middleware,» Junio 2015. [En línea]. Available: http://searchmicroservices.techtarget.com/definition/middleware. [Último acceso: 1 Agosto 2017].

[6] M. Rouse, «Quality Assurance (QA),» Febrero 2017. [En línea]. Available: http://searchsoftwarequality.techtarget.com/definition/quality-assurance. [Último acceso: 1 Agosto 2017].

[7] A. Bernardo, «9 pasos para que tu negocio sea un éxito a través del modelo Canvas,» 4 Septiembre 2013. [En línea]. Available: http://blogthinkbig.com/modelo-canvas-9-pasos-exito-negocio/. [Último acceso: 6 Agosto 2017].

[8] Android, [En línea]. Available: https://developer.android.com/index.html. [Último acceso: 22 Agosto 2017].

[9] Apple Inc., «Apple Developer,» [En línea]. Available: https://developer.apple.com/. [Último acceso: 22 Agosto 2017].

[10] Ionic Framework, [En línea]. Available: https://ionicframework.com/. [Último acceso: 22 Agosto 2017].

[11] DesarrolloWeb, «Javascript a fondo,» [En línea]. Available: https://desarrolloweb.com/javascript/. [Último acceso: 22 Agosto 2017].

[12] jQuery, [En línea]. Available: https://jquery.com/. [Último acceso: 22 Agosto 2017].

[13] React, [En línea]. Available: https://facebook.github.io/react/. [Último acceso: 22 Agosto 2017].

[14] AngularJS, [En línea]. Available: https://angularjs.org/. [Último acceso: 22 Agosto 2017].

[15] w3schools, «The HTML DOM (Document Object Model),» [En línea]. Available: https://www.w3schools.com/js/js_htmldom.asp. [Último acceso: 22 Agosto 2017].

[16] Jorgé, «Your license to use React.js can be revoked if you compete with Facebook,» 16 Julio 2016. [En línea]. Available: https://react-etc.net/entry/your-license-to-use-react-js-can-be-revoked-if-you-compete-with-facebook. [Último acceso: 2 Agosto 2017].

[17] Spring Framework, Pivotal Software Inc., [En línea]. Available: https://spring.io/. [Último acceso: 22 Agosto 2017].

[18] Express Framework, StrongLoop Inc., [En línea]. Available: http://expressjs.com/es/. [Último acceso: 22 Agosto 2017].

[19] J. Raj, «An introduction to the MEAN Stack,» 17 Abril 2014. [En línea]. Available: https://www.sitepoint.com/introduction-mean-stack/. [Último acceso: 2 Agosto 2017].

[20] «Socket.IO,» [En línea]. Available: https://socket.io/. [Último acceso: 11 Septiembre 2017].

[21] E. O. Stangvik, «WS,» [En línea]. Available: http://websockets.github.io/ws/. [Último acceso: 9 Septiembre 2017].

[22] M. Gaunt, «Agregado de notificaciones push a una app web,» 13 Julio 2017. [En línea]. Available: https://developers.google.com/web/fundamentals/getting-started/codelabs/push-notifications/?hl=es. [Último acceso: 23 Agosto 2017].

[23] TVD, «Simple Long Polling Example with JavaScript and jQuery,» 16 Octubre 2011. [En línea]. Available: https://techoctave.com/c7/posts/60-simple-long-polling-example-with-javascript-and-jquery. [Último acceso: 9 Agosto 2017].

[24] P. Sadalage, «NoSQL Databases: An Overview,» 2 Octubre 2014. [En línea]. Available: https://www.thoughtworks.com/insights/blog/nosql-databases-overview. [Último acceso: 11 Septiembre 2017].

[25] E. Saravia, «NewSQL - The Future of Databases?,» 4 Octubre 2016. [En línea]. Available: https://es.slideshare.net/ibelmopan/newsql-the-future-of-databases. [Último acceso: 11 Septiembre 2017].

[26] S. Bushik, «A Vendor-independent Comparison of NoSQL Databases: Cassandra, HBase, MongoDB, Riak,» Altoros Systems Inc, [En línea]. Available: https://www.altoros.com/vendor_independent_comparison_of_nosql_databases.html. [Último acceso: 2 Agosto 2017].

[27] MongoDB, «Sharding,» [En línea]. Available: https://docs.mongodb.com/manual/sharding/. [Último acceso: 22 Agosto 2017].

[28] Google's reCaptcha, [En línea]. Available: https://www.google.com/recaptcha/intro/android.html. [Último acceso: 23 Agosto 2017].

[29] Amazon Web Services, «Amazon S3,» Amazon Web Services Inc, [En línea]. Available: https://aws.amazon.com/es/s3/. [Último acceso: 4 Agosto 2017].

[30] TravisCI, [En línea]. Available: https://travis-ci.org/. [Último acceso: 23 Agosto 2017].

[31] SonarQube, [En línea]. Available: https://www.sonarqube.org/. [Último acceso: 23 Agosto 2017].

[32] EditorConfig, [En línea]. Available: http://editorconfig.org. [Último acceso: 23 Agosto 2017].

[33] G. A. Miller, «The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information,» *Psychological Review,* vol. 63, pp. 81-97, 1956.

[34] P. Latorre, S. Baldassarri y E. Cerezo, «Evaluación sin usuarios: KLM,» de *Interacción Persona - Ordenador: Evaluación*, Zaragoza, 2015, pp. 23-27.

[35] P. Latorre, S. Baldassarri y E. Cerezo, «Evaluación sin usuarios: Recorrido cognitivo,» de *Interacción Persona - Ordenador: Evaluación*, Zaragoza, 2015, pp. 17-21.

[36] P. Latorre, S. Baldassarri y E. Cerezo, «Evaluación sin usuarios: Heurísticas,» de *Interacción Persona - Ordenador: Evaluación*, Zaragoza, 2015, pp. 10-14.

[37] «Swagger,» [En línea]. Available: https://swagger.io/. [Último acceso: 11 Septiembre 2017].

[38] J. Bavari, «Handling CORS issues in Ionic,» 24 Febrero 2015. [En línea]. Available: http://blog.ionic.io/handling-cors-issues-in-ionic/. [Último acceso: 3 Agosto 2017].

[39] Instituto Aragonés de Estadística, «Encuesta anual de estructura salarial,» Gobierno de Aragón, [En línea]. Available: http://www.aragon.es/DepartamentosOrganismosPublicos/Institutos/InstitutoAragonesEstadistica/pcaxis/ci.Aplicacion_axis_EAES.detalleDepartamento. [Último acceso: 9 Septiembre 2017].

[40] Google, «Google HTML/CSS Style Guide,» [En línea]. Available: https://google.github.io/styleguide/htmlcssguide.html. [Último acceso: 23 Agosto 2017].

[41] Google, «Google JavaScript Style Guide,» [En línea]. Available: https://google.github.io/styleguide/javascriptguide.xml. [Último acceso: 23 Agosto 2017].

[42] Mocha, [En línea]. Available: https://mochajs.org/. [Último acceso: 23 Agosto 2017].

[43] Protractor, [En línea]. Available: http://www.protractortest.org/#/. [Último acceso: 23 Agosto 2017].

[44] S. Panda, «Understanding Angular's $apply() and $digest(),» 7 Mayo 2014. [En línea]. Available: https://www.sitepoint.com/understanding-angulars-apply-digest/. [Último acceso: 4 Agosto 2017].

[45] T. Felix, «Using Page Objects to Overcome Protractor's Shortcomings,» 19 Junio 2014. [En línea]. Available: https://www.thoughtworks.com/insights/blog/using-page-objects-overcome-protractors-shortcomings. [Último acceso: 4 Agosto 2017].

[46] Codecov, [En línea]. Available: https://codecov.io/. [Último acceso: 23 Agosto 2017].

[47] Ionic Framework, «Ionic CLI Guide,» [En línea]. Available: https://ionicframework.com/docs/cli/. [Último acceso: 4 Agosto 2017].

[48] Git, «Distributed Git - Distributed Workflows,» [En línea]. Available: https://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows. [Último acceso: 4 Agosto 2017].

[49] «Basic Access Authentication,» 30 Julio 2017. [En línea]. Available: https://en.wikipedia.org/wiki/Basic_access_authentication. [Último acceso: 4 Agosto 2017].

Lista de Figuras

Lista de Tablas

Lista de Abreviaturas

A los efectos del presente documento, sus abreviaturas se entenderán por:

- **UI:** Interfaz de Usuario (User Interface).
- **CLI**: Interfaz de línea de comandos (*Command-Line-Interface*).
- **Navbar**: Barra de navegación (*Navigation bar*).
- **DOM**: Modelo de Objetos del Documento (*Document Object Model*).
- **VCS:** Sistema de control de versiones (*Version Control System*).
- **QA:** Aseguramiento de la calidad (*Quality Assurance*).
- **MEAN:** MongoDB, ExpressJS, AngularJS, NodeJS.
- **RFU:** Requisitos Funcionales de unidades familiares y usuarios administradores.
- **RFUF:** Requisitos Funcionales de Unidades Familiares.
- **RFUA:** Requisitos Funcionales de Usuarios Administradores.
- **RNF:** Requisito No Funcional.
- **AJAX:** *Asynchronous JavaScript And XML*.
- **SPA:** *Single Page Application*.
- **KLM:** *Keystroke-Level Model.*
- **CORS:** *Cross Origin Resource Sharing.*

Anexo A: Requisitos del sistema

En este anexo se encuentran redactados todos los requisitos del sistema fruto del proceso de recogida explicado y detallado en la sección **Documentación de requisitos**.

A.1 Requisitos funcionales

Anexo que incluyen los requisitos funcionales del sistema, los cuales se han clasificado en tres tipos, dependiendo del usuario implicado al que pertenecen:

Requisitos Funcionales de unidades familiares y usuarios administradores:

Tabla 14. Requisitos Funcionales de unidades familiares y usuarios administradores.

RFU-1	El sistema debe permitir que el usuario que ya se haya registrado pueda iniciar sesión en el sistema para poder acceder a las funcionalidades que le corresponda según el tipo de usuario implicado que sea.
RFU-2	El sistema debe permitir que el usuario cambie información de una cuenta de usuario únicamente si se trata de la suya.
RFU-3	El sistema debe permitir que el usuario borre una cuenta de usuario únicamente si se trata de la suya.

Requisitos Funcionales de unidades familiares:

Tabla 15. Requisitos Funcionales de unidades familiares.

RFUF-1	El sistema debe permitir que se registren nuevas cuentas para unidades familiares.
RFUF-2	El sistema debe permitir al usuario gestionar los miembros de la unidad familiar:
RFUF-2.1	El sistema debe permitir crear un miembro de la unidad familiar.
RFUF-2.2	El sistema debe permitir listar todos los miembros de la unidad familiar.
RFUF-2.3	El sistema debe permitir modificar un miembro de la unidad familiar.
RFUF-2.4	El sistema debe permitir eliminar un miembro de la unidad familiar.
RFUF-3	El sistema debe permitir al usuario gestionar los almacenes de la unidad familiar:
RFUF-3.1	El sistema debe permitir crear un almacén en la unidad familiar.
RFUF-3.2	El sistema debe permitir listar todos los almacenes de la unidad familiar.
RFUF-3.3	El sistema debe permitir modificar un almacén en la unidad familiar.
RFUF-3.4	El sistema debe permitir eliminar un almacén en la unidad familiar.
RFUF-4	El sistema debe permitir al usuario gestionar los objetos de los almacenes de la unidad familiar:
RFUF-4.1	El sistema debe permitir crear un objeto en un almacén de la unidad familiar.
RFUF-4.2	El sistema debe permitir listar todos los objetos que pertenezcan a un almacén de la unidad familiar.
RFUF-4.3	El sistema debe permitir modificar un objeto en un almacén de la unidad familiar.
RFUF-4.4	El sistema debe permitir eliminar un objeto en un almacén de la unidad familiar.
RFUF-5	El sistema debe permitir al usuario mostrar un dashboard de las actividades de su cuenta familiar.
RFUF-6	El sistema debe permitir al usuario mostrar un dashboard de recomendaciones para su cuenta familiar.
RFUF-7	El sistema debe permitir al usuario poder realizar búsquedas de los objetos que pertenecen a su cuenta familiar independientemente del almacén al que pertenezcan.
RFUF-8	El sistema debe ofrecer al usuario un tutorial que recoja las principales funcionalidades del sistema dentro de la propia aplicación.

Requisitos Funcionales de usuarios administradores:

Tabla 16. Requisitos Funcionales de usuarios administradores.

RFUA-1	El sistema debe permitir al usuario listar todas las cuentas de usuario (salvo las de usuarios administradores) del sistema.
RFUA-2	El sistema debe permitir al usuario modificar los datos de una cuenta de usuario (salvo las de usuarios administradores).
RFUA-3	El sistema debe permitir al usuario reactivar una cuenta de usuario si ésta ha sido borrada anteriormente (salvo si es de usuarios administradores).
RFUA-4	El sistema debe permitir al usuario mostrar estadísticas con datos relevantes del sistema.

A.2 Requisitos no funcionales

Anexo que incluye todos los requisitos no funcionales del sistema:

Tabla 17. Requisitos no Funcionales

RNF-1	El sistema debe tener un cliente para dispositivos móviles que ofrezca las funcionalidades para los usuarios comunes/unidades familiares.
RNF-2	El sistema debe tener un cliente para dispositivos de escritorio que ofrezca las funcionalidades para los usuarios administradores.
RNF-3	El inicio de sesión consiste en introducir el email de una cuenta de usuario junto con su contraseña.
RNF-4	El sistema debe de pedir los siguientes datos al usuario cuando éste desee crear una cuenta: Correo electrónico (el cual servirá de identificador único en el sistema), nombre, y dos veces la contraseña (las cuales se han de comprobar para asegurarse de que coincidan).
RNF-5	La información que un usuario puede cambiar de la cuenta consiste en el nombre y la contraseña.
RNF-6	El sistema debe validar pidiendo una contraseña siempre que se quiera cambiar un dato de una cuenta de usuario, o eliminarla, para asegurarse que la persona que intenta cambiar la información se trata del propietario.
RNF-7	Un miembro de la unidad familiar consiste únicamente en un nombre que no tiene porqué ser único.
RNF-8	Una cuenta de tipo de unidad familiar necesita identificarse como uno de los miembros de la misma antes de acceder a los siguientes requisitos: [RFUF-3.1], [RFUF-3.3], [RFUF-3.4], [RFUF-4.1], [RFUF-4.3], y [RFUF-4.4].
RNF-9	El sistema debe pedir los siguientes datos al usuario cuando éste desee crear un almacén para la unidad familiar: Nombre, localización, tipo, distancia que hay del almacén a crear al domicilio habitual, y descripción.
RFUF-9.1	El tipo del almacén se discretiza en los siguientes: Trastero, Vivienda, Armario.
RFUF-9.2	La distancia del almacén se discretiza en los siguientes: [0-1km], [1km-2km], [2km-10km], [10km-100km], [100km-300km], [300km, +].
RNF-10	El sistema debe pedir los siguientes datos al usuario cuando éste desee crear un objeto de un almacén de la unidad familiar: Nombre, imagen/foto, fecha de garantía, fecha de caducidad, y descripción.
RFUF-10.1	Las fechas de caducidad y de garantía siguen el siguiente formato: (YYYY-MM-DD).
RNF-11	Las recomendaciones que se van a mostrar a una cuenta familiar tienen que generarse automáticamente y de forma periódica, consistiendo en las siguientes: Comprobación de cuando la fecha de garantía de un objeto ha expirado, comprobación de cuando la fecha de caducidad de un objeto ha expirado, y comprobación de cuando un objeto tiene asociadas más de 20 actividades y existe un almacén más cercano al cual moverse que en el que se encuentra actualmente.

RNF-12	Las recomendaciones que se van a mostrar a una cuenta familiar deben mostrarse al usuario de forma cronológica descendiente.
RNF-13	La información a mostrar tras la búsqueda de un objeto de una unidad familiar es todos los objetos que coincidan con el criterio de búsqueda en forma de lista conteniendo la siguiente información: El nombre y la imagen/foto.
RNF-14	El criterio para realizar una búsqueda de un objeto de una unidad familiar es su nombre, el cual no tiene por qué coincidir exactamente, solo contener la cadena de caracteres que el usuario ha introducido, y no es sensible a minúsculas ni mayúsculas.
RNF-15	La UI debe cumplir aspectos de usabilidad sometiéndose al análisis de la técnica KLM, siendo los tiempos analizados resultantes de las actividades principales inferiores a 3 minutos.
RNF-16	La captura de información de la imagen/foto de los objetos de un almacén de la unidad familiar tiene que realizarse con ayuda de un dispositivo móvil.
RNF-17	Las imágenes/fotos que se van a tratar en el sistema tienen que tener un tamaño máximo de 20mb.
RNF-18	El sistema debe tener una forma automática de introducir mínimo una cuenta de usuario administrador en el sistema.
RNF-19	La información que un usuario administrador puede modificar de una cuenta de usuario consiste en el nombre y el correo electrónico de la misma.
RNF-20	Las estadísticas que se muestran al usuario administrador tienen que poderse visualizar en forma de gráficos.

Anexo B: Definiciones de hecho y estándares del proyecto

En este documento se definen los estándares usados en la aplicación *Depot Cloud* para el Trabajo de Fin de Grado de Rubén Moreno. El documento se divide en tres partes definiendo unos estándares determinados para cada una de ellas: Documentación del código, API y Codificación. Además, todas las funcionalidades del sistema tienen que cumplir las siguientes definiciones de hecho para que se consideren completadas correctamente

- El producto ha sido incluido en la rama master del repositorio del *owner* y supera todos los test unitarios y de integración automáticos con TravisCI.
- El código del producto está correctamente documentado siguiendo los estándares definidos en la sección **Documentación del código**.
- El código del producto sigue los estándares de codificación definidos en la sección **Codificación**.
- Los nombres de las variables, métodos y clases tienen que ser mínimamente descriptivos, así como guardar una consistencia semántica entre las distintas capas de implementación si se refieren al mismo concepto.

B.1 Documentación del código

B.1.1 Estándar

Todos los ficheros de código del proyecto de *Depot Cloud* sean del lenguaje que sean deben seguir los siguientes requisitos:

- Toda función o método que no sea un método de acceso a atributos debe tener una breve explicación de la descripción de su funcionamiento o ser tan sencilla que baste con un título autodescriptivo.
- Los métodos de acceso a atributos tipo "*getAtributo*" o "*setAtributo*" pueden prescindir de una explicación si poseen un nombre intuitivo.
- La documentación debe estar redactada en inglés.

B.1.2 Verificación

Para cada uno de los lenguajes se han de seguir los siguientes pasos para verificar que cumple con los estándares del código:

1. Se colocan todos los ficheros en una misma carpeta
2. Se selecciona uno al azar.
3. Se comprueban los requisitos de estándar para todo el fichero.
4. Si cumple el estándar se acaba la verificación, si no, se realiza el paso anterior para todos los ficheros asumiendo que puede haber fallo en más de uno.

B.2 Codificación

B.2.1 Estándar

Depot Cloud utiliza diferentes lenguajes. Cada uno de ellos tienen que seguir las guías de estilo citadas a continuación:

HTML y CSS (basado en [40]**):**

- Se usa siempre el protocolo HTTPS para importar recursos siempre que sea posible.
- El código tiene una sangría de 2 espacios por cada línea y no se utilizan tabuladores.

- El código está escrito en minúscula. Aplicado a nombres de elementos, atributos, valores de atributos, selectores CSS, propiedades y valores de propiedades (menos cadenas de caracteres).
- Se borran todos los espacios al final de las líneas.
- Se utiliza la codificación UTF-8.
- Todos los documentos HTML empiezan con el elemento "<!*DOCTYPE html*>".
- Todos los elementos que no tienen elemento de cerradura se escriben con una barra al final (p.ej.: "<*br* />" en vez de "<*br*>").
- Todos los elementos que tienen un elemento de cerradura se tienen que cerrar con la misma.
- Todos los elementos "<*img*>" tienen un atributo "*alt*".
- Todas las clases o IDs de CSS tienen nombres significativos.
- No se usan unidades después del valor 0 en un atributo CSS.
- Se usa un espacio después de los dos puntos de una propiedad CSS antes de escribir el valor.

JavaScript (basado en [41]**):**

- Cualquier variable que se declare tiene que estar especificada con "*var*", "*let*" o "*const*".
- Todas las expresiones de función o variables tienen que terminar con punto y coma. Las declaraciones de función no tienen que terminar con punto y coma.
- Se pueden usar funciones anidadas.
- El código tiene una sangría de 4 espacios por cada línea y no se utilizan tabuladores.
- Todas las variables o nombres de funciones o nombres de ficheros empiezan por minúscula. Si tienen varias palabras se escriben sin espacios y cada palabra que sigue a la primera empieza por mayúscula (p.ej.: "ficheroDePrueba.js").

B.2.2 Verificación

Para cada uno de los lenguajes se han de seguir los siguientes pasos para verificar que cumple con los estándares del código.

1. Se colocan todos los ficheros en una misma carpeta
2. Se selecciona uno al azar.
3. Se comprueban los requisitos de estándar para todo el fichero.
4. Si cumple el estándar se acaba la verificación, si no, se realiza el paso anterior para todos los ficheros asumiendo que puede haber fallo en más de uno.

Anexo C: Estrategia y herramientas usadas para los tests

Para la realización de la verificación y validación de la aplicación se han realizado tests unitarios, de sistema y de aceptación, todos ellos automáticos.

Detallando en primer lugar los tests unitarios implementados, se ha empleado la herramienta Mocha [42], un framework de testing para Node.js. Junto con Mocha, a fin de contar con aserciones para realizar los mencionados tests unitarios, se ha empleado también Chai, una librería de aserciones, también para Node.js, enfocado al desarrollo dirigido por pruebas o por comportamiento. Con la integración de ambas dos herramientas se han conseguido implementar varias suites de tests para las diferentes funcionalidades de la aplicación.

En segundo lugar, se han implementado los test de sistema y aceptación utilizando la herramienta Protractor [43], un framework de testing end-to-end para aplicaciones Angular y AngularJS. Se ha seleccionado esta herramienta para este tipo de tests ya que actualmente la parte del cliente está desarrollada con AngularJS para escritorio, y Ionic 2 (que funciona sobre Angular) para dispositivos móviles, y esto hace que los test de Protractor se sincronicen con los ciclos de *digest* [44], lo que elimina los típicos problemas de tener que introducir esperas activas en este tipo de tests para que le dé tiempo al navegador web a realizar las acciones que se le han encargado.

Ya que este tipo de tests son más costosos en tiempo y complejos para probar todas las posibles combinaciones de acciones realizadas por un usuario, se ha decidido centrar los esfuerzos de estos tests en los caminos más críticos de la aplicación. Como se puede ver en la **Figura 16**, se ha conseguido un total de 10 caminos críticos a probar.

Además, se ha utilizado uno de los patrones de diseño de test *end-to-end* más comunes: Page Objects [45]. Esto ayuda a escribir tests más claros y limpios encapsulando la información acerca de los elementos de la página de la aplicación. De esa forma, los Page Objects pueden ser reusados a través de múltiples tests, y si el template de la aplicación cambia, sólo se tiene que actualizar el Page Object.

```
10 specs, 0 failures
Finished in 23.632 seconds

[01:43:47] I/launcher - 0 instance(s) of WebDriver still running
[01:43:47] I/launcher - chrome #01 passed
```

Figura 16. Cobertura con tests de Protractor

En relación a la cobertura de los tests unitarios que se han mencionado, se ha decidido integrar en el proyecto una nueva herramienta que permite, mediante un análisis del código, mostrar en cada estado del proyecto la cobertura del código en ese momento. Dicha herramienta es Codecov [46], la cual se ha integrado con GitHub, dándole acceso al repositorio para realizar el análisis tras cada *commit* y *pull request*. De esta forma, Codecov muestra si la cobertura del código ha aumentado, descendido o si sigue igual tras añadir el contenido que se incluya en dicho *commit* o *pull request*. No obstante, dado que sólo comprueba los tests del servidor y no tests del sistema completo, Codecov no tiene en cuenta para el total de cobertura del proyecto los tests de Protractor, porque le es imposible saber cuántas líneas de código cubre al realizar cada petición.

Finalmente, se muestran las estadísticas de cobertura al momento de finalizar el proyecto, presentando así un 90.55% de cobertura de código tan sólo con los tests unitarios realizados con Mocha en un total de 136 tests. Se puede ver una evolución de la cobertura de tests durante toda la duración del proyecto en la **Figura 17** y el total de porcentaje de cobertura de test distribuidos por ficheros en la **Figura 18**.

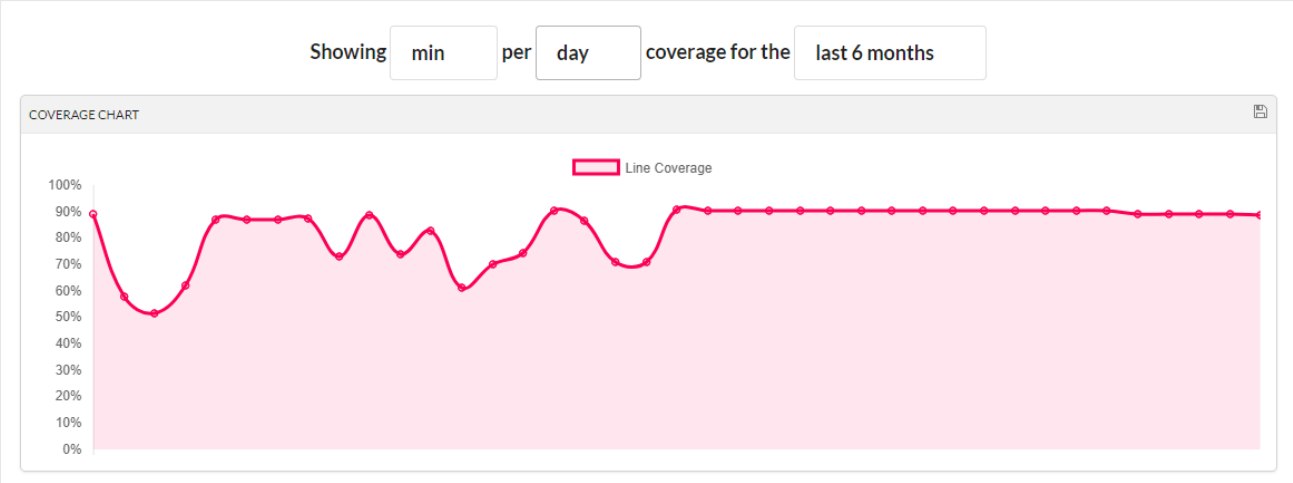

Figura 17. Evolución de la cobertura de tests durante el proyecto.

Files					Coverage
models	21	21	0	0	100.00%
routes	887	800	0	87	90.19%
config.js	1	1	0	0	100.00%
security/jwt-handler.js	7	6	0	1	85.71%
server.js	37	35	0	2	94.59%
Project Totals (21 files)	953	863	0	90	90.55%

Figura 18. Cobertura del sistema al finalizar el proyecto.

Anexo D: Estrategia para la construcción automática del software

Ya que la aplicación se ha desarrollado empleando el framework MEAN, una de las estrategias de construcción automática empleada ha sido el gestor de paquetes npm. Gracias a que se pueden configurar numerosos comandos en el fichero package.json del proyecto, npm facilita la ejecución de la aplicación con un solo comando: 'npm start' (que simplemente ejecutará el comando 'node' en el fichero principal que lanza el servidor, en este caso el fichero server.js).

Además, dado que su función principal es la gestión de paquetes y dependencias del proyecto, npm cuenta con el comando 'npm install' que descarga automáticamente todas las dependencias necesarias para el proyecto definidas en el package.json.

Se han configurado también en dicho fichero tres scripts que facilitan la construcción y despliegue de la aplicación: 'postinstall' e 'install', que se ejecutan después de que todos los paquetes de dependencias se hayan instalado tras ejecutar el comando 'install', y 'test', que se ejecuta con el comando 'npm test'. 'Postinstall' ejecuta el comando 'npm install' para el package.json del directorio public y public-ionic, para así instalar las dependencias de las vistas de la aplicación. 'Install', por su parte, ejecuta el comando 'update' sobre el webdriver-manager que ejecuta los tests de Protractor, con el fin de mantenerlo actualizado. Por último, 'test' ejecuta los tests de mocha que se encuentren disponibles.

Por otro lado, en la parte del cliente de Ionic 2, dado que este framework cuenta con su propio CLI [47], existen algunos comandos que se necesitan saber. 'ionic serve' se utiliza para empezar un servidor local de desarrollo para el desarrollo/testing de la aplicación. 'ionic cordova build [<plataform>] construye (prepara + compila) el proyecto de Ionic para una determinada plataforma, pudiendo ser iOS, Android o Windows. 'ionic cordova run [<plataform>]' ejecuta el proyecto de Ionic en un dispositivo conectado. 'ionic cordova emulate [<plataform>]' emula el proyecto de Ionic en un emulador o simulador. Estos son los comandos más importantes que se han utilizado durante el proyecto para la construcción automática del cliente para dispositivos móviles, ya que como se ha explicado con anterioridad, toda la dependencia de paquetes se ha instalado previamente.

Anexo E: Estrategia de control de versiones

Dada la importancia de agilizar el proceso de desarrollo del proyecto, pero sin dejar de lado un buen control de versiones, se ha decidido usar la herramienta GitHub creando un repositorio llamado "DepotCloud-tfg" (https://github.com/nebur395/DepotCloud-tfg) y siguiendo un *workflow* tipo "*Integration-manager workflow*" [48] modificado. De esta forma se posee un repositorio central al cual no se puede realizar un *push* directo (salvo en excepciones y situaciones extremas) hasta haber pasado localmente todos los test correspondientes. Los cambios del proyecto se van realizando en local. Cuando se quiere integrar algo lo suficientemente interesante y estable al repositorio central, se realiza el *push* correspondiente. Una vez hecho, se comprueba que se han pasado con éxito los tests automáticos de TravisCI y todas las definiciones de hecho establecidas.

Así se consigue mantener una frecuencia de trabajo rápida, una rama en el repositorio central lo más estable posible, y un control mínimo de que todo el trabajo integrado en la misma ha pasado una revisión y posee un grado de calidad aceptable.

Para la organización de los documentos y de la gestión del proyecto se ha usado la plataforma de Google Drive, en la cual se ha mantenido un seguimiento de los documentos que se han ido actualizando. Se ha creado una organización en el directorio para que se pueda realizar una gestión adecuada del mismo siguiendo la siguiente estructura:

Tabla 18. Estructura del directorio creado en Google Drive.

1_Gestión	Temas relacionados con la gestión del proyecto.
1.01_Hoja de esfuerzos	Hojas de esfuerzos y control de horas del Trabajo Fin de Grado.
1.02_Planificación	Planificación del proyecto.
2_Análisis y diseño	Temas relacionados con todo el análisis y diseño del proyecto.
2.01_Diagramas	Diagramas del análisis y diseño del sistema.
2.01_Mapas de navegación	Mapas de navegación de la UI del sistema.
3_Documentación	Temas relacionados con toda la documentación del proyecto, incluidas las entregas finales.
4_Presentaciones	Presentaciones que se van a realizar (en este caso la defensa del Trabajo Fin de Grado).
5_Multimedia	Contenidos varios del Trabajo Fin de Grado, mayormente multimedia.
Z_Cosas	Otros.

Anexo F: Mantenibilidad y Quality Assurance

Se ha decidido integrar la herramienta de Quality Assurance y análisis estático del código SonarQube con el proyecto, para mejorar la misma y la mantenibilidad del código. Para la configuración de la herramienta se ha hecho uso principalmente de dos módulos llamados Sonar Way: uno para los perfiles de calidad de Web, y otro para los perfiles de calidad de JavaScript. Dado que ya se cuenta con una herramienta de análisis de cobertura de código, se ha desactivado esa sección de la herramienta.

Para el análisis del código se han activado las reglas que recomienda SonarQube en su página para cada uno de los módulos utilizados, estas se pueden ver en el siguiente enlace:

- **JavaScript**: *https://sonarqube.com/coding_rules#qprofile=js-sonar-way-56838|activation=true*
- **Web**: *https://sonarqube.com/coding_rules#qprofile=web-sonar-way-50375|activation=true*

Gracias a los estándares definidos internamente junto con las definiciones de hecho, el equipo de desarrollo ha intentado hacer un esfuerzo extra para conseguir la máxima calidad en mantenibilidad del código en el proyecto. Los análisis obtenidos se encuentran en la tabla X mientras que una vista general del análisis se puede visualizar en la **Figura 19**.

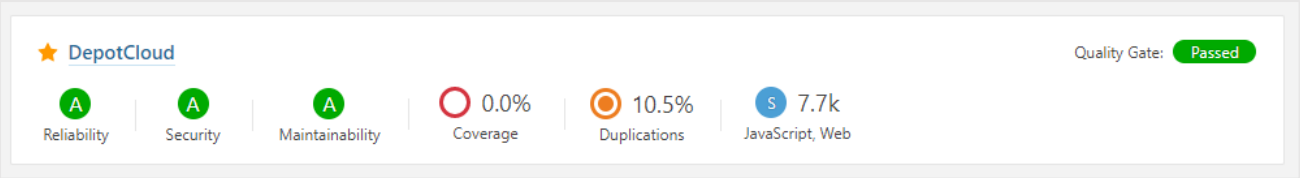

Figura 19. Vista general del análisis de SonarQube.

Medida	Definición	Resultado
Reliability	Número de errores de programación en el código.	**Figura 20**
Security	Número de vulnerabilidades en el código.	**Figura 21**
Maintainability	Número de incidencias de código deficiente. Se entiende por código deficiente cualquier síntoma en el código fuente de un programa que posiblemente indique un problema más profundo. Usualmente no son errores de programación, es decir, no son técnicamente incorrectos y en realidad no impiden que el programa funcione correctamente, sin embargo, indican deficiencias en el diseño que puede ralentizar el desarrollo o aumentan el riesgo de errores o fallos en el futuro.	**Figura 22**
Duplications	Número de bloques duplicados de líneas.	**Figura 23**
Size	Tamaño de líneas de código.	**Figura 24**
Complexity	Es la complejidad calculada en base al número de rutas a través del código. Siempre que el flujo de control de una función se divide, el contador de complejidad se incrementa en uno. Cada función tiene una complejidad mínima de 1. Este cálculo varía ligeramente según el idioma.	**Figura 25**
Issues	Número de incidencias.	**Figura 26**

Figura 20. Análisis de SonarQube: Reliability.

Figura 21. Análisis de SonarQube:Security.

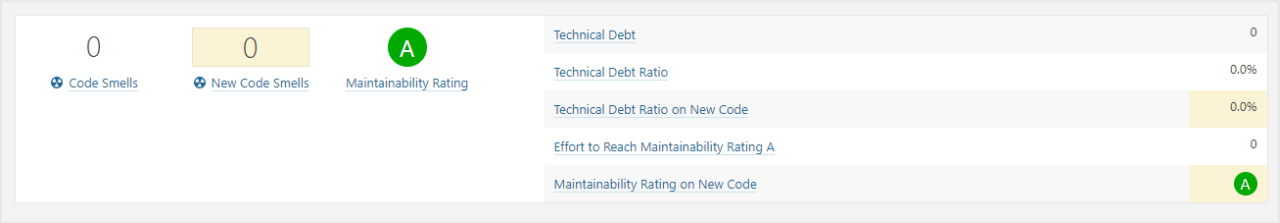

Figura 22. Análisis de SonarQube: Maintainability.

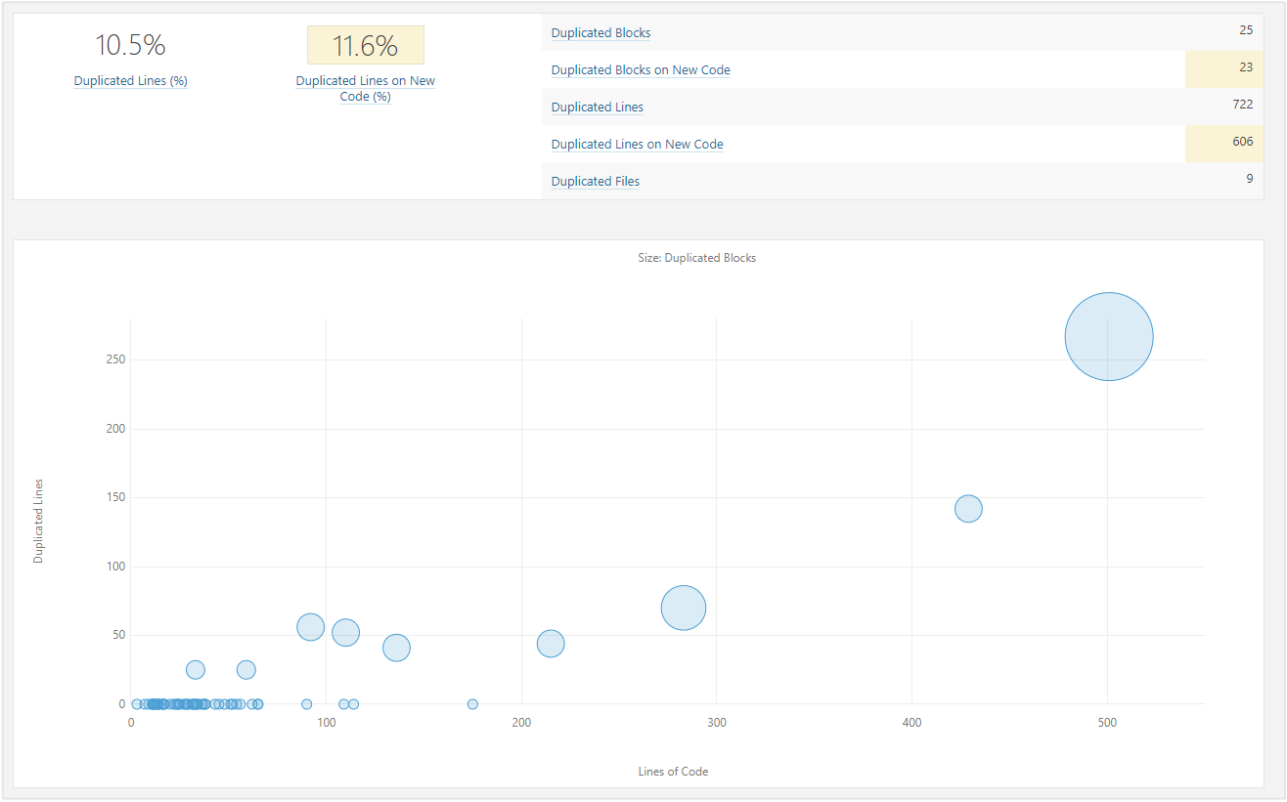

Figura 23. Análisis de SonarQube: Duplications.

Figura 24. Análisis de SonarQube: Size.

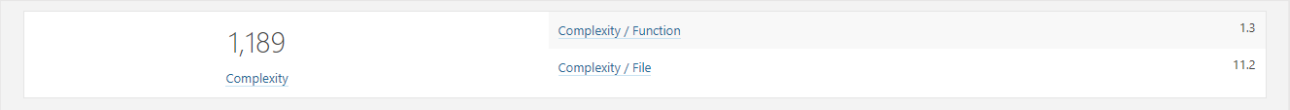

Figura 25. Análisis de SonarQube: Complexity.

Figura 26. Análisis de SonarQube: Issues.

Como se puede observar, el proyecto tiene el nivel A en todas las métricas establecidas en el proyecto con las correspondientes reglas. Además, cuenta con un porcentaje bastante bajo de código duplicado, con tan solo un 11.6% en sus 7701 líneas de código totales. Se cuenta actualmente con un total de 0 issues que añadan problemas al proyecto. No obstante, hay 27 de ellas que se han marcado como "*won't fix*" que se han introducido en el sistema intencionadamente con otros propósitos resultando así en falsos positivos. Gracias a todo ello, el proyecto no cuenta con una deuda técnica y consigue de esta forma unos niveles altos de calidad en producción.

Anexo G: Estado de la aplicación final

Se pueden ver los requisitos funcionales que corresponden a cada una de las pantallas de la aplicación y clasificadas en los tipos de clientes en esta sección:

Cliente de dispositivo móvil:

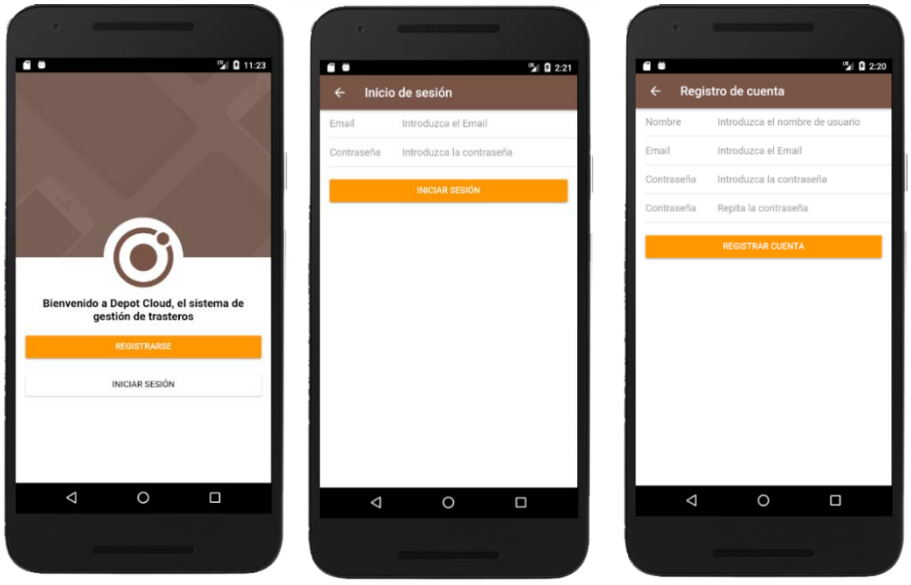

Figura 27. De izquierda a derecha: Pantalla de bienvenida, pantalla de iniciar sesión (RFU-1), pantalla de registrar usuario (RFUF-1).

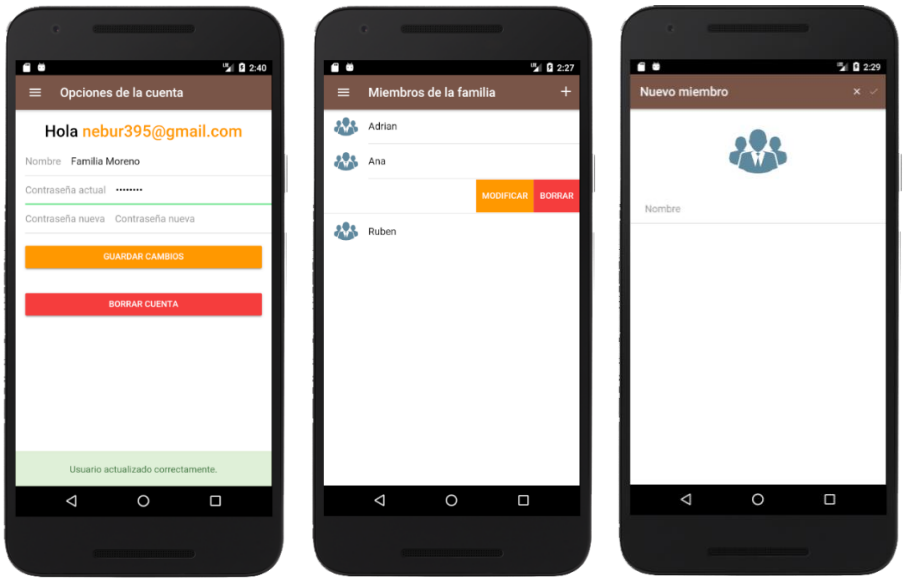

Figura 28. De izquierda a derecha: Pantalla de opciones de usuario (RFU-2, RFU-3), pantalla de gestionar miembros junto con el modal para crear y modificar (RFUF-2).

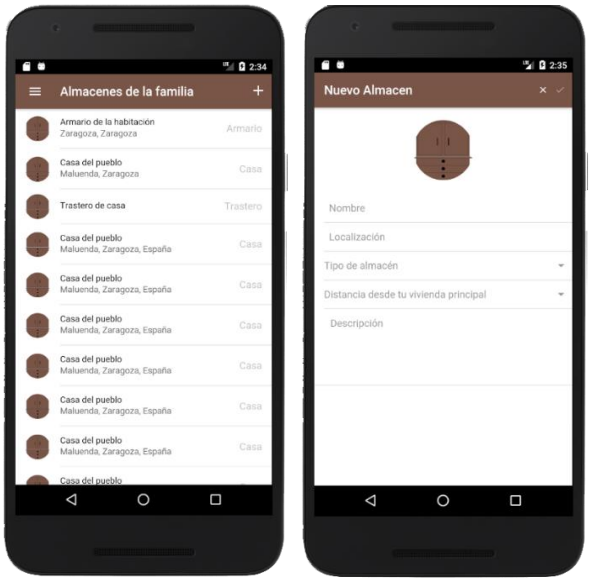

Figura 29. Pantalla de gestionar almacenes junto con el modal para crear y modificar (RFUF-3).

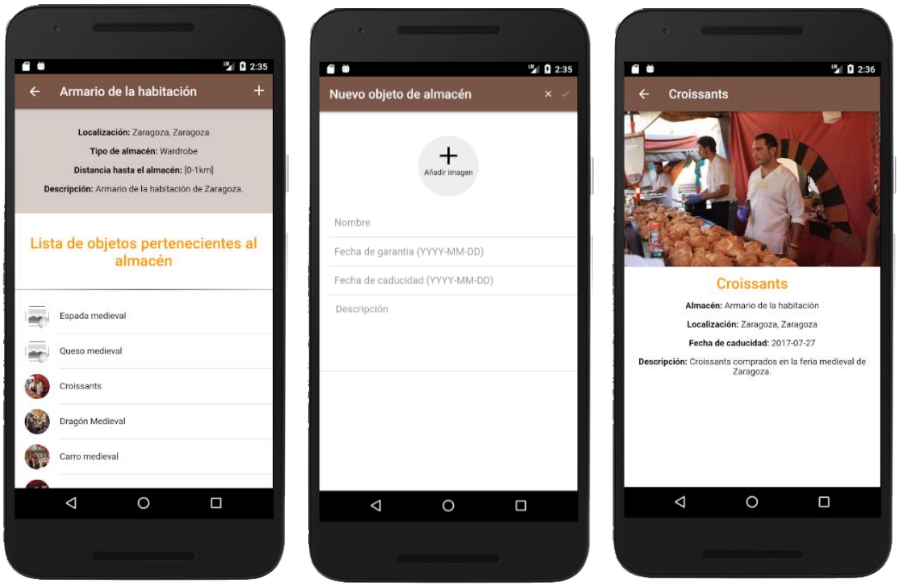

Figura 30. Pantalla de gestionar objetos de almacén junto con el modal para crear y modificar y la de detalles de objeto (RFUF-4).

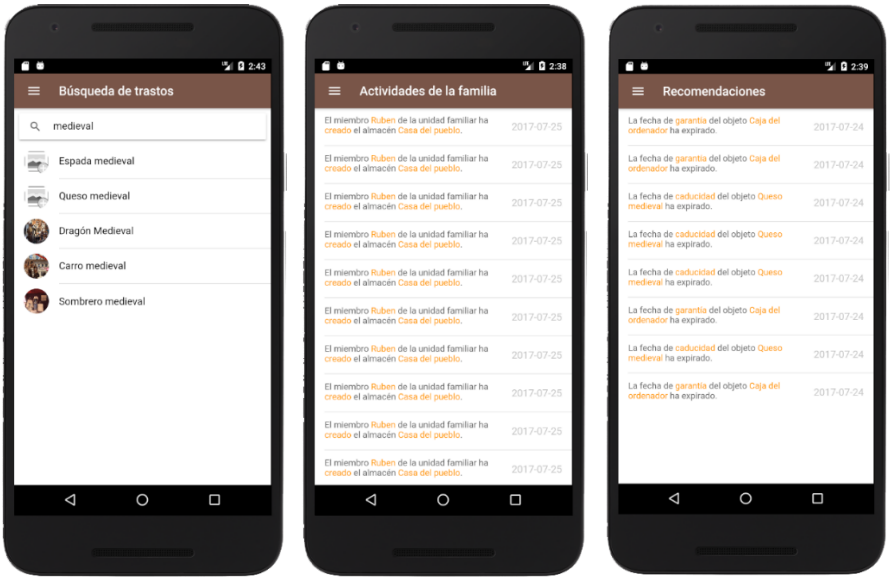

Figura 31. De izquierda a derecha: Pantalla de búsqueda (RFUF-7), pantalla de actividades (RFUF-5), pantalla de recomendaciones (RFUF-6).

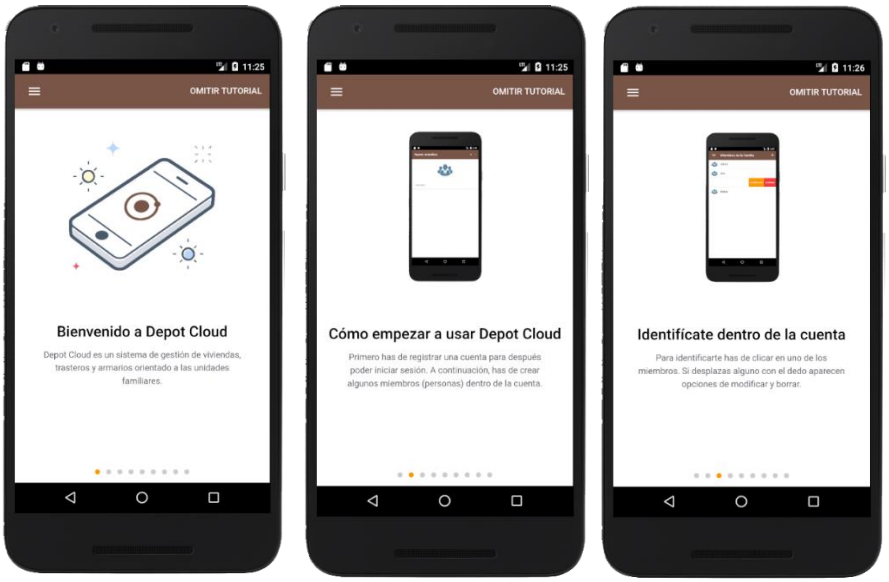

Figura 32. Pantallas de tutorial 1 (RFUF-8).

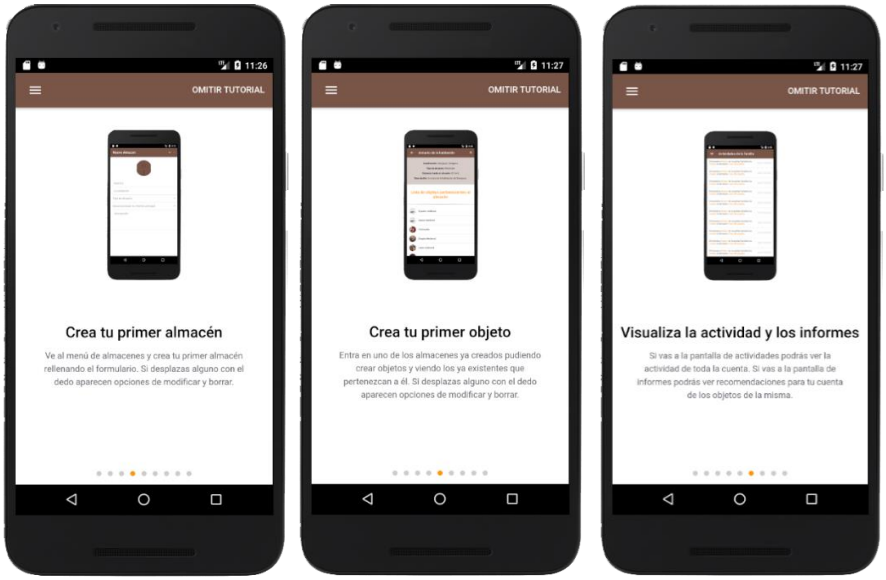

Figura 33. Pantallas de tutorial 2 (RFUF-8).

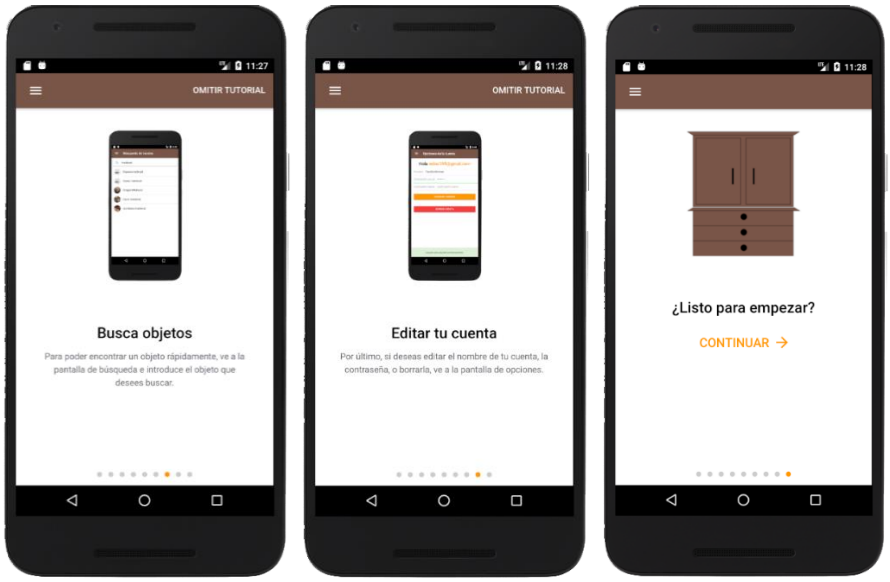

Figura 34. Pantallas de tutorial 3 (RFUF-8).

Cliente de escritorio:

Figura 35. Pantalla de iniciar sesión (RFU-1).

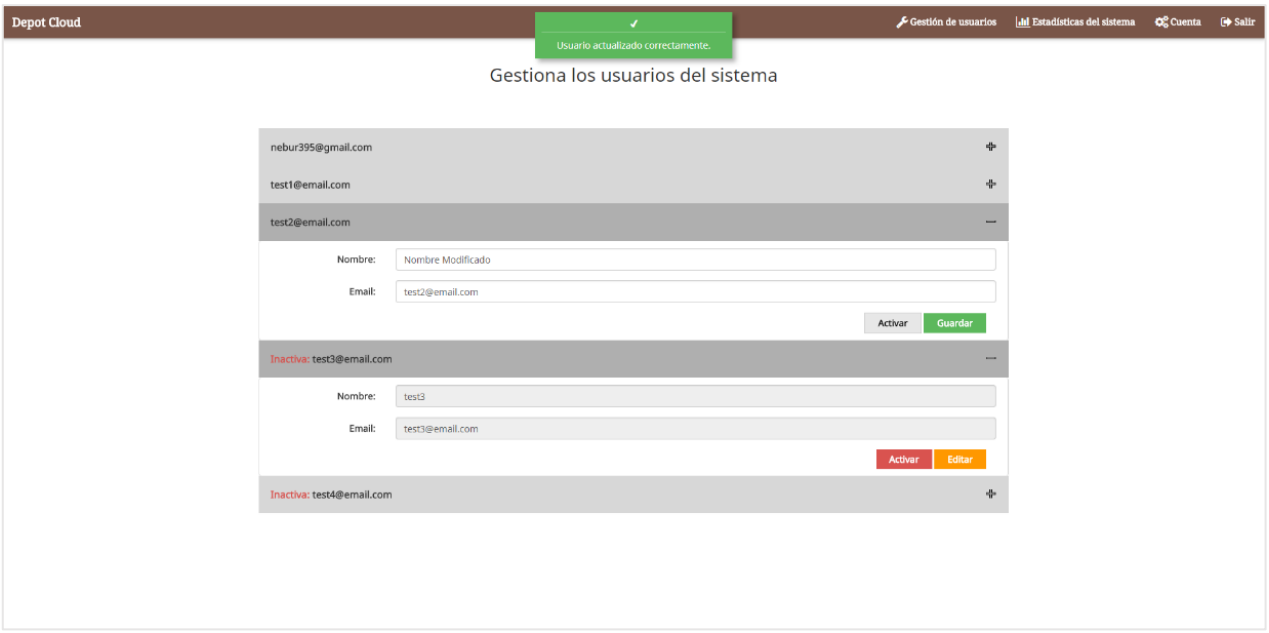

Figura 36. Pantallas de gestión de usuarios (RFUA-1, RFUA-2, RFUA-3).

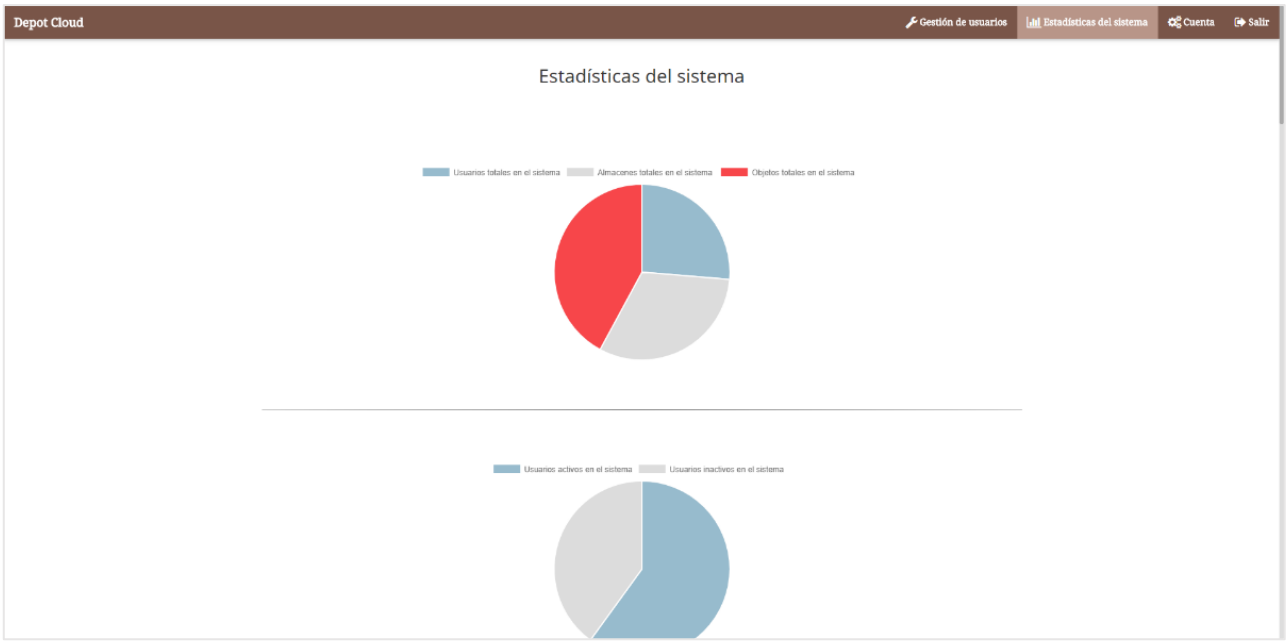

Figura 37. Pantalla de estadísticas (RFU-4).

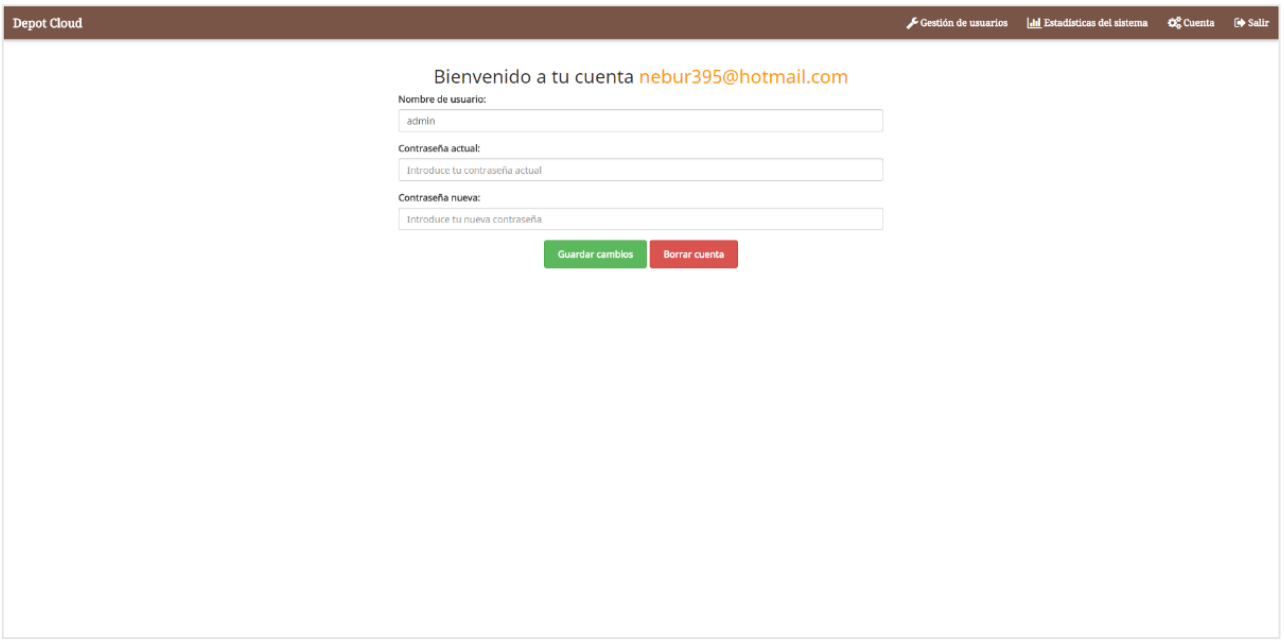

Figura 38. Pantallas de opciones de usuario (RFU-2, RFU-3).

Anexo H: Mapa de navegación

La aplicación cuenta con dos mapas de navegación. Dado que cuenta con un número elevado de pantallas, se ha simplificado el mapa de navegación para que se puedan visualizar de una forma más legible en el formato actual del documento. Los nombres de las pantallas que aparecen se refieren a las mismas que las del apartado anterior.

Cliente de dispositivo móvil:

En este mapa de navegación existen algunas conexiones que no aparecen, esto se debe a que esas pantallas son accesibles a través del navbar lateral, una vez se haya iniciado sesión, desde cualquier otro estado. Estas pantallas se hay representado en el mapa con el borde rojo:

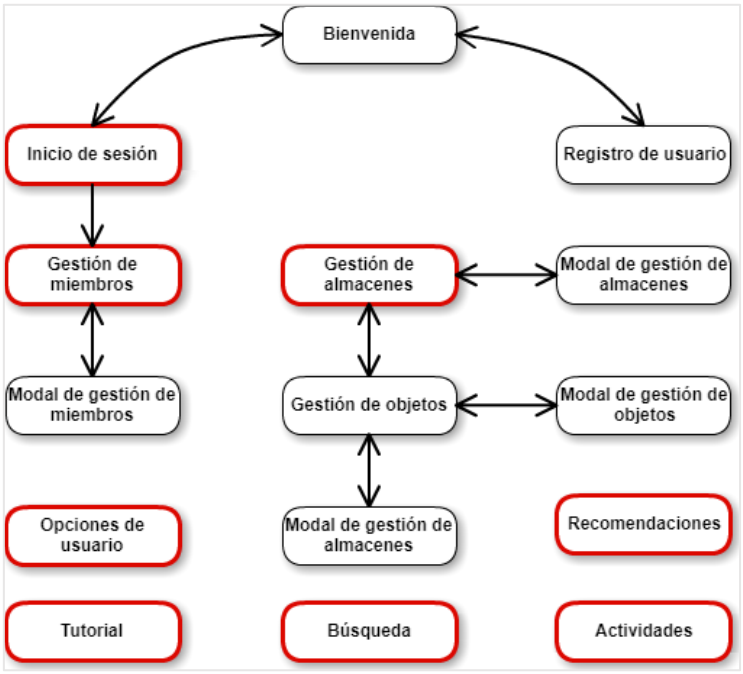

Figura 39. Mapa de navegación del cliente de dispositivos móviles.

Cliente de escritorio:

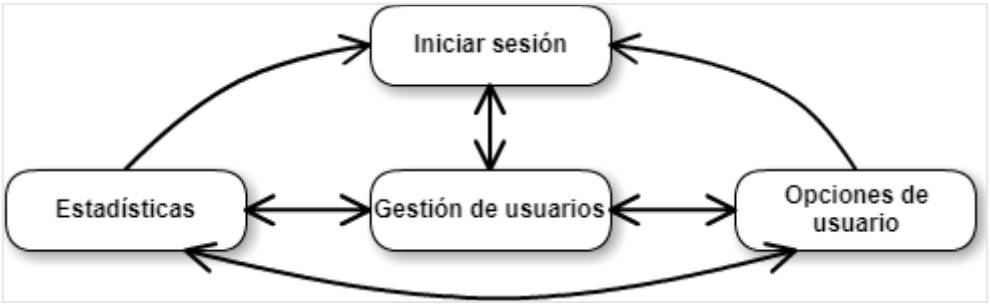

Figura 40. Mapa de navegación del cliente de dispositivos móviles.

Anexo I: Usabilidad del sistema

Para analizar la usabilidad del sistema, se han escogido cuatro de los escenarios más comunes representativos y largos del sistema, tres de ellos en clientes móviles y dos de ellos en clientes de escritorio.

A continuación, se tiene una sección por cada uno de los escenarios a analizar. En ellas, se han utilizado dos técnicas de análisis de la usabilidad sin usuarios. Se han escogido dichas técnicas en vez de otras que involucran usuarios para agilizar el proceso debido a las restricciones de tiempo del proyecto. Estas técnicas han sido: recorrido cognitivo, y método KLM.

El recorrido cognitivo [35] es un método de inspección que evalúa la facilidad de aprendizaje. Está basado en los recorridos estructurales de la ingeniería de software. Se evalúa una propuesta de prototipo de interfaz en el contexto de una o más tareas. En cada acción el evaluador responde las siguientes preguntas:

1. ¿Coincide el efecto de la acción con el objetivo del usuario en ese punto?
2. ¿Percibirán los usuarios que está disponible la acción correcta?
3. Una vez encontrada la acción en la interfaz, ¿asociarán estos usuarios la acción correcta al efecto que se alcanzará?
4. Una vez realizada la acción, ¿entenderán los usuarios la realimentación del sistema?

El método KLM (*Keystroke-Level Model*) [34] es un sistema para la evaluación de prestaciones. Asigna un tiempo determinado a cada acción que se realiza en el sistema a manos de los usuarios. Existe un sistema ampliado y modificado del método para sistemas móviles. Las tablas con los correspondientes tiempos que se van a usar en este documento se pueden ver en la **Figura 41**.

Operator	Description	Time
K	Press a key or mouse button	0.2s
P	Point with mouse	1.1s
H	Home on keyboard, mouse or other device.	0.4s
M	Mentally prepare	1.35s
R	System response time	Needs to be measured
B	Press or reléase mouse button	0.1s
BB	Click mouse button	0.2s

Operator		Time	Quartile 1	Quartile 3
***A*, Action**	picture/marker	**1.23**	0.61	1.44
	NFC	**0.00**	-	-
	in general	*variable, input to model*		
D, Drawing		*not applicable*		
***F*, Finger Movement**		**0.23**	0.20	0.29
***G*, Gestures**		**0.80**	0.73	0.87
H, Homing		**0.95**	0.81	1.00
***I*, Initial Act**	externally	**5.32**	3.98	7.51
	internally	**3.89**	2.23	4.89
	optimal setting	**1.18**	1.10	1.26
	no assumptions	**4.61**	-	-
K, Keystroke	keypad average	**0.39**	0.37	0.48
	keypad quick	**0.33**	0.32	0.37
	hot Key	**0.16**	0.15	0.20
M, Mental Act		**1.35**	-	-
P, Pointing		**1.00**	0.84	1.20
R, System Response Time	NFC	**2.58**	2.46	2.80
	visual marker	**2.22**	2.09	2.82
	general	*variable, input to model*		
S_{Macro}, Macro Attention Shift		**0.36**	0.28	0.44
S_{Micro}, Micro Attention Shift	keypad ↔ display	**0.14**	0.14	0.19
	hotkey ↔ display	**0.12**	0.02	0.14
	keypad ↔ hotkey	**0.04**	0.02	0.12
	in general	**0.14**	0.10	0.16
***X*, Distraction**	slight	**6 %**	3 %	13 %
	strong	**21 %**	11 %	25 %

Figura 41. Tabla KLM original (a la izquierda) y tabla KLM para móviles (a la derecha).

Por último, para analizar globalmente la interfaz, se han escogido las heurísticas de Nielsen [36], que son un método de evaluación que se basan en principios reconocidos de usabilidad. El método consiste en 10 reglas que son evaluadas por expertos para validar la interfaz. Cada evaluador ha puntuado del 0-7 (siendo 0 NS/NC, 1 totalmente en desacuerdo, y 7 totalmente de acuerdo) cada una de las heurísticas. Los resultados obtenidos se pueden ver en la **Tabla 19** y las 10 reglas que se han utilizado para evaluar han sido las siguientes:

1. **Visibilidad del estado del sistema**: El sitio mantiene informados a los usuarios de lo que está ocurriendo a través de retroalimentación apropiada.
2. **Relación entre el sistema y el mundo real**: El sitio habla el lenguaje de los usuarios mediante conceptos que son familiares al usuario.
3. **Control y libertad del usuario:** El sitio permite "salidas de emergencia" claramente marcadas, como "deshacer" y "rehacer".
4. **Consistencia y estándares:** el sitio sigue las convenciones establecidas. Los usuarios nunca tienen duda de que diferentes elementos significan en realidad la misma cosa.
5. **Prevención de errores:** El sitio tiene un diseño cuidadoso que previene la ocurrencia de problemas.
6. **Reconocimiento antes que recuerdo:** El sitio hace visibles los objetos, acciones y opciones. Las instrucciones para su uso están a la vista o son fácilmente recuperables cuando sea necesario.
7. **Flexibilidad y eficiencia de uso:** El sitio incluye la presencia de aceleradores que no son vistos por los usuarios novatos, pero ofrecen una interacción más rápida a los usuarios expertos.
8. **Estética y diseño minimalista:** El sitio no contiene información que es irrelevante o poco usada.
9. **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores:** el sitio incluye mensajes de error en un lenguaje claro y simple.
10. **Ayuda y documentación:** El sitio ofrece ayuda y documentación. Dicha información es fácil de buscar.

Tabla 19. Heurísticas de Nielsen aplicadas al proyecto.

Heurísticas	NS/NC	Totalmente en desacuerdo						Totalmente de acuerdo
	0	1	2	3	4	5	6	7
1								✓
2								✓
3						✓		
4								✓
5							✓	
6							✓	
7						✓		
8								✓
9								✓
10								✓

I.1 Escenario I: Crear almacén

En este escenario se va a analizar la funcionalidad de crear almacén. Se presupone que el usuario ya ha iniciado sesión, seleccionado el miembro de la unidad familiar con el que desea realizar la acción, y ya se encuentra en la pantalla de gestión de almacenes:

Recorrido cognitivo:

Tabla 20. Recorrido cognitivo de crear almacén.

Nº	Acción del usuario	Respuesta del sistema	P1	P2	P3	P4
1	Selecciona el botón de añadir en la pantalla de gestión de almacenes.	Abre un modal con un formulario a completar para crear el almacén.	✓	✓	✓	✓

Nº	Acción del usuario	Respuesta del sistema	P1	P2	P3	P4
2	Completa el formulario y le da al botón de guardar.	Cierra el modal y le avisa con un mensaje lo que ha sucedido. Si es éxito, se añade a la lista actual, si es error, se indica el porqué.	✓	✓	✓	✓

Método KLM:

Tabla 21. KLM de crear almacén.

Acción	Operador	Tiempo
Leer Pantalla de Gestión de almacenes	M	1.35
Botón de añadir almacén: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Leer el formulario	M	1.35
Escribir nombre de almacén: • Apuntar • Click con el dedo • Pensar nombre • Escribir (media: 10 letras)	 F K M 10*K	 0.23 0.33 1.35 3.3
Escribir localización de almacén: • Apuntar • Click con el dedo • Pensar nombre • Escribir (media: 10 letras)	 F K M 10*K	 0.23 0.33 1.35 3.3
Seleccionar tipo de almacén: • Leer 3 tipos • Apuntar opción • Click con el dedo • Apuntar “OK” • Click con el dedo	 3*M F K F K	 4.05 0.23 0.33 0.23 0.33
Seleccionar distancia de almacén: • Leer 6 tipos • Apuntar opción • Click con el dedo • Apuntar “OK” Click con el dedo	 6*M F K F K	 8.1 0.23 0.33 0.23 0.33
Escribir descripción de almacén: • Apuntar • Click con el dedo • Pensar nombre • Escribir (media: 20 letras)	 F K M 20*K	 0.23 0.33 1.35 6.6
Botón de aceptar: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Distracción	X	6%
TIEMPO TOTAL		39.37s

I.2 Escenario II: Crear objeto

En este escenario se va a analizar la funcionalidad de crear objeto. Se presupone que el usuario ya ha iniciado sesión, seleccionado el miembro de la unidad familiar con el que desea realizar la acción, creado el almacén en el que se va a introducir el objeto, y ya se encuentra en la pantalla de gestión de objetos:

Recorrido cognitivo:

Tabla 22. Recorrido cognitivo de crear objeto.

Nº	Acción del usuario	Respuesta del sistema	P1	P2	P3	P4
1	Selecciona el botón de añadir en la pantalla de gestión de objetos.	Abre un modal con un formulario a completar para crear el objeto.	✓	✓	✓	✓
2	Hace click en el botón de añadir imagen	Da a elegir si se desea sacar foto (si el dispositivo cuenta con cámara) o seleccionar archivo de la galería.	✓	✓	✓	✓
4	Selecciona saca una foto con la cámara o selecciona una de la galería.	Sustituye el botón de añadir imagen por la que se acaba de sacar o seleccionar.	✓	✓	✓	✓
5	Completa el resto del formulario y le da al botón de guardar.	Cierra el modal y le avisa con un mensaje lo que ha sucedido. Si es éxito, se añade a la lista actual, si es error, se indica el porqué.	✓	✓	✓	✓

Método KLM:

Tabla 23. KLM de crear objeto.

Acción	Operador	Tiempo
Leer Pantalla de Gestión de objetos	M	1.35
Botón de añadir objeto: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Leer el formulario	M	1.35
Añadir imagen: • Apuntar • Click con el dedo • Pensar opciones de añadir imagen • Sacar foto/Seleccionar de galería	 F K M A	 0.23 0.33 1.35 1.23
Escribir nombre de objeto: • Apuntar • Click con el dedo • Pensar nombre • Escribir (media: 10 letras)	 F K M 10*K	 0.23 0.33 1.35 3.3
Escribir fecha de caducidad: • Apuntar • Click con el dedo • Pensar nombre • Escribir (10 letras con el formato correcto)	 F K M 10*K	 0.23 0.33 1.35 3.3
Escribir fecha de garantía: • Apuntar • Click con el dedo • Pensar nombre	 F K M	 0.23 0.33 1.35

Acción	Operador	Tiempo
• Escribir (10 letras con el formato correcto)	10*K	3.3
Escribir descripción de almacén: • Apuntar • Click con el dedo • Pensar nombre • Escribir (media: 20 letras)	 F K M 20*K	 0.23 0.33 1.35 6.6
Botón de aceptar: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Distracción	X	6%
TIEMPO TOTAL		32.97s

I.3 Escenario III: Ver detalles de objeto

En este escenario se va a analizar la funcionalidad de ver detalles de un objeto. Se presupone que el usuario ya ha iniciado sesión, ha creado el objeto que desea visualizar, y se encuentra en la primera pantalla que aparece al iniciar sesión:

Recorrido cognitivo:

Tabla 24. Recorrido cognitivo de ver detalles de objeto.

Nº	Acción del usuario	Respuesta del sistema	P1	P2	P3	P4
1	Hace click en el botón de menú.	Abre el menú lateral con las opciones a elegir.	✓	✓	✓	✓
2	Selecciona la opción “Almacenes”.	Cambia la pantalla a la de gestión de almacenes.	✓	✓	✓	✓
4	Selecciona el almacén donde se encuentra el objeto que quiere visualizar.	Cambia la pantalla a la de gestión de objetos, mostrando la información del almacén seleccionado y una lista de objetos pertenecientes al mismo.	✓	✓	✓	✓
5	Selecciona el objeto que se desea visualizar.	Cambia la pantalla a la de detalles de objeto, mostrando toda la información que se tiene del mismo.	✓	✓	✓	✓

Método KLM:

Tabla 25. KLM de ver detalles de objeto.

Acción	Operador	Tiempo
Abrir el menú lateral: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Leer el menú lateral	M	1.35
Seleccionar el botón “Almacenes”: • Apuntar • Click con el dedo	 F K	 0.23 0.33
Leer Pantalla de Gestión de almacenes	M	1.35
Seleccionar el almacén en el que se encuentra el objeto: • Ver almacenes (media: 3 almacenes) • Apuntar opción • Click con el dedo	 3*M F K	 4.05 0.23 0.33
Seleccionar el objeto que se desea visualizar:		

Acción	Operador	Tiempo
• Ver objetos (media: 10 objetos) • Apuntar opción • Click con el dedo	10*M F K	13.5 0.23 0.33
Visualizar Pantalla de Detalles de objeto	M	1.35
Distracción	X	6%
TIEMPO TOTAL		25.27s

I.4 Escenario IV: Editar información de usuario

En este escenario se va a analizar la funcionalidad de administrador de editar información de usuario. Se presupone que el usuario ya ha iniciado sesión, y ya se encuentra en la pantalla de gestión de usuarios:

Recorrido cognitivo:

Tabla 26. Recorrido cognitivo de editar información de usuario.

Nº	Acción del usuario	Respuesta del sistema	P1	P2	P3	P4
1	Busca y selecciona el usuario que desea modificar.	Expande un acordeón del usuario clicado con su nombre y correo correspondiente, así como opciones para activar y editar cuenta.	✓	✓	✓	✓
2	Hace click en el botón de “Editar”.	Desbloquea los campos de texto de nombre y correo electrónico del usuario para que éste pueda editarlos. Sustituye el botón de “Editar” por “Guardar”, cambiándole el color de naranja a verde para darle más retroalimentación.	✓	✓	✓	✓
4	Edita la información que desea guardar y hace click en el botón de “Guardar”.	Avisa al usuario con un mensaje de lo que ha sucedido. Si es éxito, vuelve a bloquear los campos de texto de nombre y correo electrónico del usuario para que no se puedan editar, y sustituye el botón de “Guardar” por “Editar”, cambiándole el color de verde a naranja para darle más retroalimentación. Si es error, se indica por qué en un mensaje.	✓	✓	✓	✓

Método KLM:

Tabla 27. KLM de editar información de usuario.

Acción	Operador	Tiempo
Leer Pantalla de Gestión de usuarios	M	1.35
Busca el usuario que se desea editar: • Ver usuarios (media: 100 usuarios encontrados) • Apuntar opción • Click de ratón	 100*M P K	 135 1.1 0.2
Leer el acordeón desplegado	M	1.35
Pulsar el botón “Editar”: • Apuntar • Click de ratón	 P K	 1.1 0.2
Editar nombre de usuario: • Apuntar • Click de ratón • Pensar nombre • Cambiar a teclado • Escribir (media: 10 letras)	 P K M H 10*K	 1.1 0.2 1.35 0.4 2

Acción	Operador	Tiempo
Editar correo electrónico de usuario:		
• Cambiar a ratón	H	0.4
• Apuntar	P	1.1
• Click de ratón	K	0.2
• Pensar nombre	M	1.35
• Cambiar a teclado	H	0.4
• Escribir (media: 20 letras)	20*K	4
Pulsar el botón “Guardar”:		
• Cambiar a ratón	H	0.4
• Apuntar	P	1.1
• Click de ratón	K	0.2
TIEMPO TOTAL		154.5s

Anexo J: Análisis y Diseño del sistema

J.1 Modelo de datos

Presentación primaria

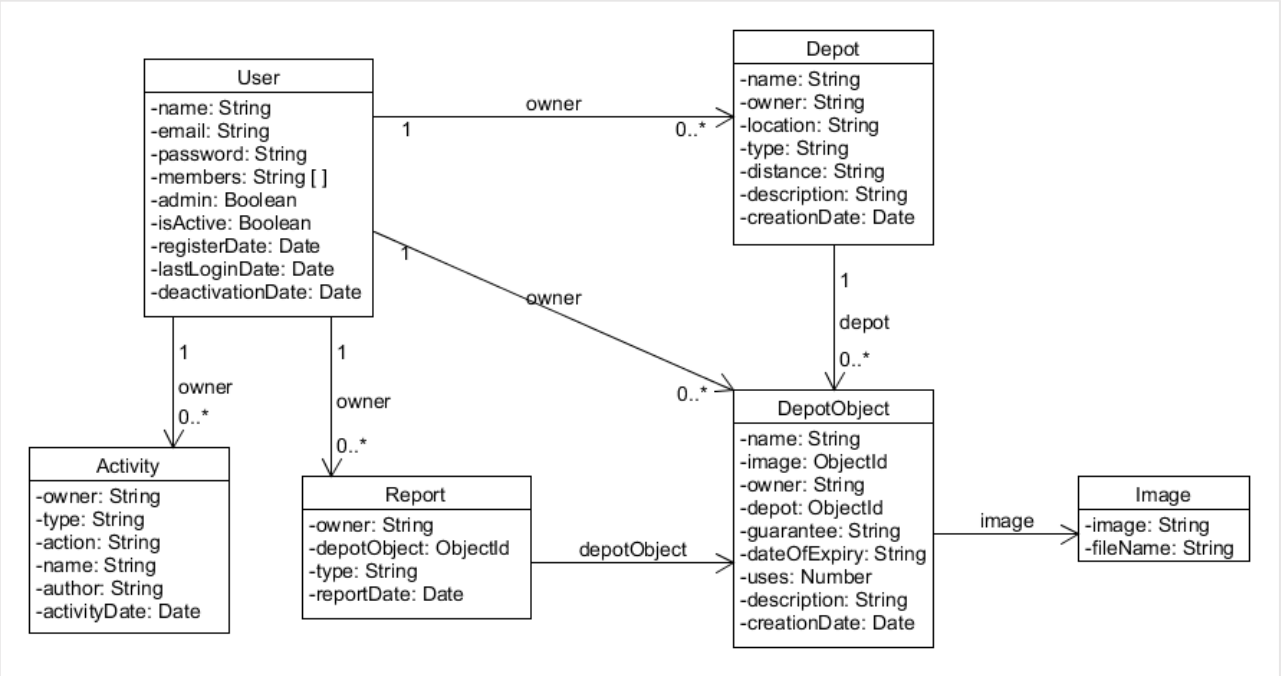

Figura 42. Diagrama de modelo de datos del sistema.

Catálogo de la vista

- **User**: Un usuario registrado en el sistema. Incluye los datos de registro del usuario, información para saber si es administrador, si la cuenta está activa, los miembros de la unidad familiar si los tiene, y fechas de registro, último acceso y borrado.
- **Depot:** Un almacén creado en el sistema. Incluye datos descriptivos de un almacén como su nombre, localización y descripción, el usuario creador, su caracterización por tipo y distancia que se encuentra de la vivienda principal, y la fecha de creación del mismo.
- **DepotObject:** Un objeto creado en el sistema. Incluye datos descriptivos de un objeto como su nombre, imagen, fecha de garantía y de caducidad, descripción, fecha de creación, usuario creador, almacén al que pertenece y un contador de cuantas veces se ha modificado para posteriormente la creación de informes.
- **Activity:** Una actividad creada en el sistema. Incluye el usuario al que pertenece, el tipo y nombre de entidad sobre la que ha sido realizada, la acción realizada, el miembro y autor de la misma, y su fecha de creación.
- **Report:** Un informe creado en el sistema. Incluye el usuario al que pertenece, el objeto relacionado con él, su tipo, y su fecha de creación.
- **Image**: Una imagen adjuntada a un POI y almacenada en el sistema. Incluye la imagen en base 64 y el nombre con el que se ha guardado la imagen.

Exposición de razones

Se han tomado numerosas decisiones de diseño a lo largo del desarrollo del proyecto en lo referente al modelo de datos del mismo. En primer lugar, centrando la atención en la entidad User, se ha decidido que los miembros de la unidad familiar formaran parte de la misma como un array de strings. Esto se ha hecho porque

realmente no aportan nada de valor salvo la identificación dentro de la propia unidad familiar para distinguir quién ha realizado qué actividad, y no requería de ningún tipo de lógica adicional. Algunas de las alternativas que se barajaron para atacar esta situación, fue crear los miembros como entidades independientes, con sus propias credenciales, que fueran descendientes de una cuenta "padre" que fuera una entidad con significado de ser la unidad familiar. Otra de ellas fue como la alternativa anterior, pero en vez de con una entidad padre, que los objetos pudieran ser compartidos entre distintas cuentas. Sin embargo, dichas soluciones se descartaron porque añadían una complejidad considerable de relaciones al sistema, así como una complejidad innecesaria de cara al usuario de traspaso de permisos de la cuenta padre o las entidades compartidas, control de quién está invitado y quién no, etc., y eso es algo que no se quería ya que está pensada para un público inexperto en tecnologías.

Por otro lado, en cuanto a las entidades de objetos, una de las decisiones más importantes fue añadir el atributo de "uses". Este atributo permite generar el informe de la comprobación de cuando un objeto tiene asociadas más de 20 actividades, y existe un almacén más cercano al cual moverse que en el que se encuentra actualmente. Este atributo es un contador que incrementa cada vez que se realiza cualquier modificación sobre la entidad, de tal forma que esto permite optimizar las búsquedas que realiza en segundo plano el generador de informes, ya que en vez de tener que realizar búsquedas de las entidades objeto y actividad a la vez para ver cuantas actividades estaban relacionadas con qué objetos, y ver si superaba el límite o no, de esta forma únicamente hay que comprobar si dicho atributo sobrepasa el número establecido de modificaciones.

J.2 Vista de Componentes y Conectores

En el diagrama presentado a continuación pueden diferenciarse las tres capas de la arquitectura Modelo-Vista-Controlador. Después, se van a especificar los componentes pertenecientes a cada una de las capas.

Presentación primaria

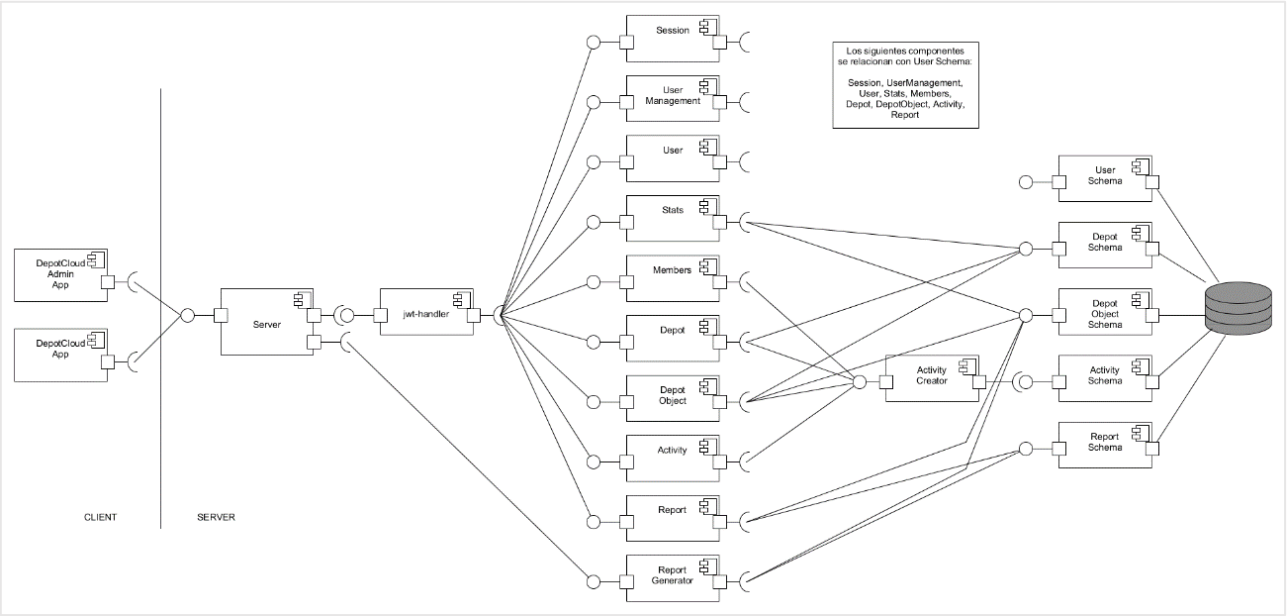

Figura 43. Diagrama de CyC.

Catálogo de la vista

- **Vista**:
  - **DepotCloud App**: Es el cliente para dispositivos móviles de la aplicación. Contiene todo el cliente desarrollado sobre Ionic 2 y se comunica vía HTTP haciendo las conexiones con los end-points vía API REST, salvo con el componente Stats y UserManagement con los cuales no se comunica.

- **DepotCloud Admin App**: Es el cliente para dispositivos desktop de la aplicación. Contiene todo el cliente desarrollado sobre AngularJS para los usuarios administradores y se comunica vía HTTP o HTTPS haciendo las conexiones con los end-points vía API REST. Este componente solo va a hacer uso de las interfaces RESTful de los componentes User, UserManagement, Session y Stats.

- **Controlador**:
  - **Server**: Representación del server.js que inicializa todo el sistema, ejecuta en segundo plano el componente ReportGenerator, y realiza todo el enrutado a los módulos que gestionan las peticiones, así como del módulo del control de acceso de usuarios. Requiere el componente jwt-handler dado que todas las peticiones pasan primero por el filtro para comprobar que la petición la ha realizado un usuario válido y que ha iniciado sesión.
  - **jwt-handler**: Es el encargado de gestionar el control de acceso al sistema. Este control de accesos se va a realizar con la tecnología JSON Web Tokens. Solo se puede hacer uso de los componentes habiendo iniciado sesión. Los usuarios normales pueden acceder a todos los componentes salvo UserManagement y Stats.
  - **Session**: Es el encargado de gestionar el inicio de sesión y generar un JSON Web Token válido y con la información del usuario.
  - **User Management**: Ofrece la interfaz para acceder a las funcionalidades relacionadas con la gestión de usuarios del administrador (editar y reactivar usuario).
  - **User**: Ofrece la interfaz para acceder a las funcionalidades relacionadas con la gestión de usuarios.
  - **Stats**: Ofrece la interfaz para acceder a las estadísticas que necesita el usuario administrador para visualizar en el cliente de escritorio.
  - **Members**: Ofrece la interfaz para acceder a las funcionalidades relacionadas con la gestión de los miembros de la unidad familiar.
  - **Depot**: Ofrece la interfaz para acceder a las funcionalidades relacionadas con la gestión de almacenes.
  - **Depot Object**: Ofrece la interfaz para acceder a las funcionalidades relacionadas con la gestión de los objetos que se guardan en los almacenes.
  - **Activity**: Es el componente que ofrece el listado de actividades a todos los miembros de la unidad familiar acerca de cualquier acción realizada sobre los almacenes y los objetos almacenados.
  - **Report**: Es el componente que se encarga de ofrecer el listado de todos los informes generados de los objetos en el sistema.
  - **Report Generator**: Componente en *background* (segundo plano) establecido con un *timeout* e inicializado desde el componente Server, al arrancar el sistema, que se encarga de generar los informes correspondientes.
  - **Activity Creator**: Es el encargado de generar una actividad relacionada con una acción y guardarla en el sistema.
- **Modelo:** Todos los componentes del modelo están conectados directamente con la base de datos MongoDB mediante el driver Mongoose.
  - **User Schema:** Componente que representa el esquema del modelo de datos de un usuario.
  - **Depot Schema:** Componente que representa el esquema del modelo de datos de un almacén.
  - **DepotObject Schema:** Componente que representa el esquema del modelo de datos de un objeto.
  - **Activity Schema:** Componente que representa el esquema del modelo de datos de una actividad.
  - **Report Schema:** Componente que representa el esquema del modelo de datos de un informe.

Exposición de razones

El framework MEAN lleva de forma casi natural a realizar una implementación siguiendo el patrón de diseño Modelo-Vista-Controlador, ofreciendo servicios RESTful orientados a recursos, por lo que se ha decidido usar dichos patrones para el desarrollo de la aplicación

Es por eso que nos encontramos los controladores divididos en componentes que se identifican como recursos web que pueden ser accedidos e identificados vía URI, que en este caso, son accedidas desde la vista.

J.3 Vista de módulos

El diagrama de módulos presentado en esta sección representa las cuatro capas del sistema desarrollado: vista, controlador, modelo e infraestructura. Se va a explicar a continuación el contenido de cada capa, sin entrar demasiado en detalle.

Presentación primaria

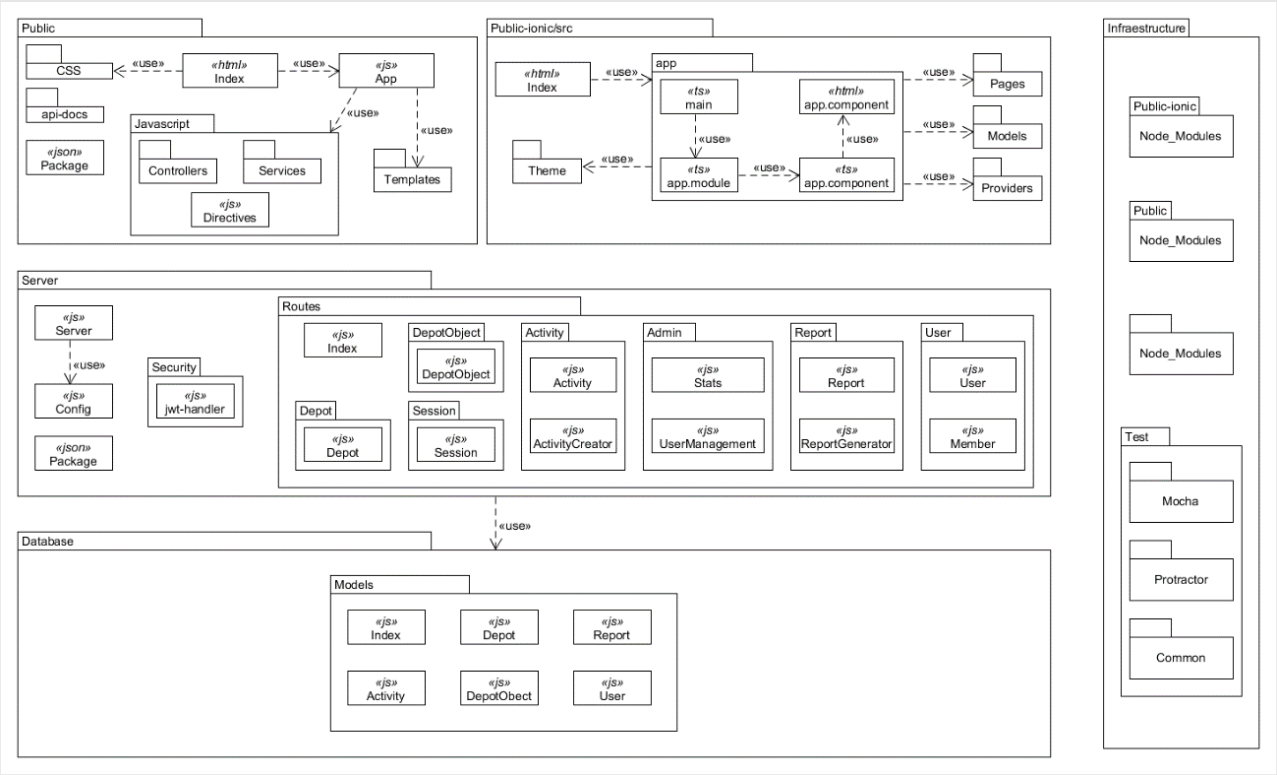

Figura 44. Diagrama de paquetes.

Catálogo de la vista

- **Módulo "Public":** Representa los módulos y ficheros que conforman la parte de la vista de la aplicación del cliente de escritorio.
  - **Index**: El fichero index.html de la aplicación web. Usa los módulos de CSS, así como el fichero app.js de la aplicación.
  - **CSS**: Módulo que contiene todos los ficheros CSS necesarios para la vista de la aplicación.
  - **App**: El fichero app.js de la vista de la aplicación. Usa las plantillas definidas en el módulo Templates y los ficheros JavaScript definidos en el módulo JavaScript.
  - **Templates**: Módulo que contiene las plantillas HTML necesarias para la vista de la aplicación.

- **JavaScript**: Módulo que contiene los ficheros JavaScript que manejan la vista de la aplicación. Lo conforman los ficheros services.js y directives.js, además del módulo Controllers que contiene todos los controladores de la vista.
- **Api-docs**: Módulo que contiene los ficheros necesarios para la documentación de la API en Swagger.
- **Package**: Fichero package.json del módulo Public que contiene las dependencias de módulos de la vista de la aplicación.

- **Módulo "Public-ionic":** Representa los módulos y ficheros que conforman la parte de la vista de la aplicación del cliente móvil.
  - **Index**: El fichero index.html de la aplicación web. Usa el módulo app.js de la aplicación para inicializarla.
  - **Theme**: Módulo que contiene los colores principales de la aplicación para poder ser usados en cualquier componente de la misma.
  - **App**: Este módulo sirve para inicializar y ensamblar el sistema. Crea el componente principal del sistema en el que se desarrollarán los subsiguientes, así como inyecta las dependencias de los servicios del módulo providers y de librerías externas si las hay.
  - **Pages**: Módulo que contiene un directorio por cada componente/página de la vista. Cada directorio contiene su fichero de lógica TypeScript, su fichero template de HTML y su fichero de estilo SCSS.
  - **Models**: Módulo que contiene los modelos de objetos utilizados para la vista.
  - **Providers**: Módulo que contiene los servicios con diferentes funciones para ser utilizadas en los componentes de la vista que se requiera.
- **Módulo "Server":** Representa los módulos y ficheros que conforman la parte del controlador de la aplicación.
  - **Server**: Fichero server.js de la aplicación, encargado de crear el servidor, la conexión con la base de datos y fijar todas las rutas y módulos globales necesarios.
  - **Config**: Fichero config.js de la aplicación. Contiene la clave secreta empleada para la firma de los JSON Web Tokens emitidos.
  - **Package**: Fichero package.json de la aplicación. Contiene dependencias, dependencias de desarrollo y la definición de algunos comandos para lanzar con npm.
  - **Security**: Módulo que contiene los ficheros encargados de la gestión de los JSON Web Tokens empleados en la aplicación.
  - **Routes**: Módulo que contiene todos los ficheros con los endpoints y la lógica de la aplicación.
    - **Index**: Fichero que contiene la definición de las rutas para cada uno de los ficheros del módulo.
    - **User**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre los usuarios y miembros de unidades familiares del sistema.
    - **Report**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre los informes en el sistema.
    - **Admin**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles para el administrador del sistema y las estadísticas que se muestran para éste.
    - **Report**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre los informes en el sistema.

        - **Activity**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre las actividades generadas por los miembros familiares en el sistema.
        - **Depot**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre los almacenes en el sistema.
        - **DepotObject**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre los objetos en el sistema.
        - **Session**: Módulo que cuenta con los ficheros que contienen la lógica de la aplicación en lo que se refiere a las funciones disponibles sobre el control del inicio de sesión en el sistema.
- **Módulo "Database":** Representa los módulos y ficheros que conforman la parte del modelo de la aplicación.
    - **Models**: Módulo que contiene los ficheros de definición de modelos de datos de la aplicación.
        - **Index**: Fichero que exporta globalmente los modelos de datos de la aplicación.
        - **User**: Fichero que contiene el modelo de datos de un usuario en el sistema.
        - **Depot:** Fichero que contiene el modelo de datos de un almacén en el sistema.
        - **DepotObject:** Fichero que contiene el modelo de datos de un objeto en el sistema.
        - **Activity:** Fichero que contiene el modelo de datos de una actividad en el sistema.
        - **Report:** Fichero que contiene el modelo de datos de un informe en el sistema.
- **Módulo "Infraestructure":** Representa los módulos y ficheros que conforman la parte de infraestructura de la aplicación.
    - **Node_modules**: Módulo que contiene los módulos de npm empleados en la parte del controlador de la aplicación.
    - **Public/Node_modules**: Módulo que contiene los módulos de npm empleados en la parte de la vista de la aplicación.
    - **Test**: Módulo que contiene todos los ficheros y submódulos de testing del sistema, tanto unitarios, como de sistema y aceptación.

Exposición de razones

Como ya se ha expuesto anteriormente, la aplicación sigue un patrón Modelo-Vista-Controlador, por lo que en el diagrama de módulos se han representado las distintas capas agrupando los módulos que pertenecen a cada una. No obstante, como se puede apreciar en dicho diagrama, la organización que se ha seguido en la aplicación es casi representativa de dicho patrón, agrupando los ficheros de cada capa en sus módulos correspondientes.

Tal y como dicta la arquitectura de capas que se ha representado, se ha respetado que tan sólo las capas superiores pueden utilizar las inferiores y nunca viceversa. La capa de infraestructura, aunque representada a un lateral, se sitúa en la parte más baja y todas las capas pueden acceder a ella.

J.4 Vista de distribución

El despliegue de la aplicación se hace actualmente en local y de manera automática como se explica en la sección **Instrucciones y despliegue del sistema**.

Presentación primaria

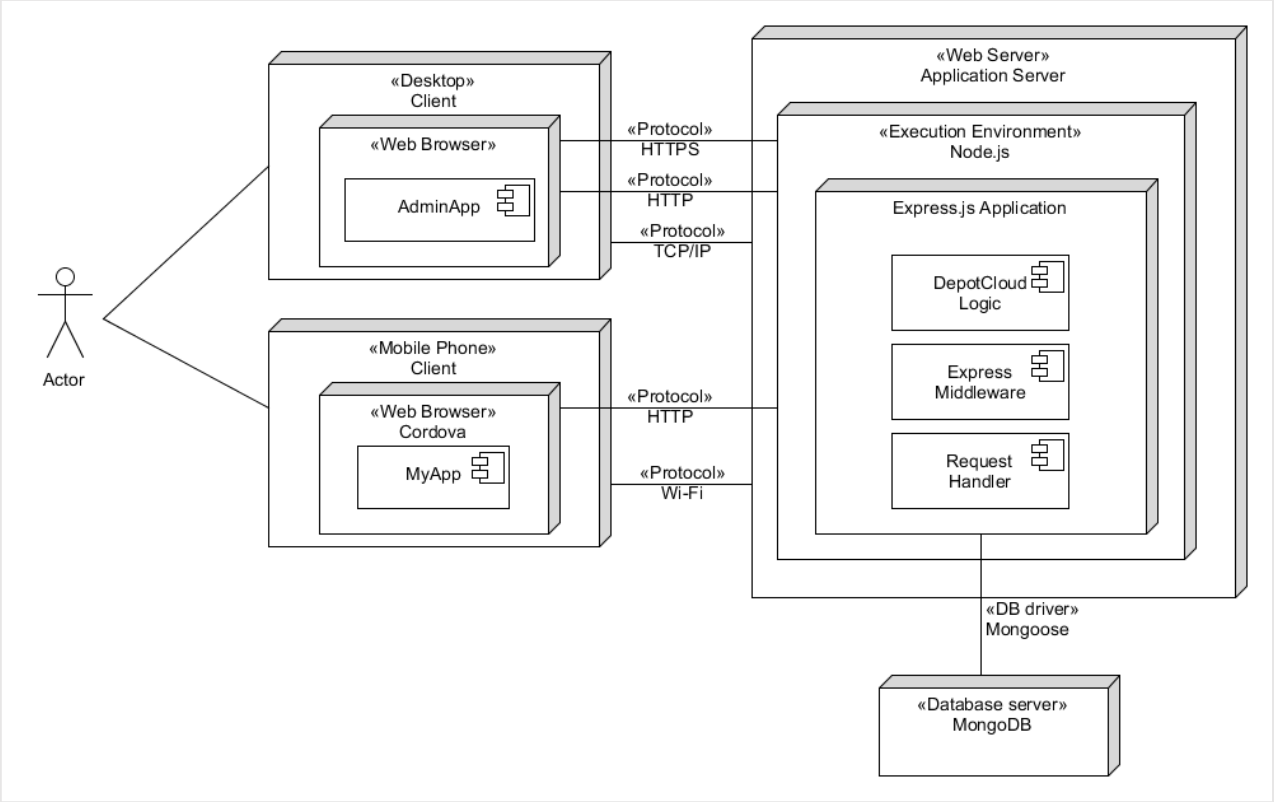

Figura 45. Diagrama de despliegue.

Catálogo de la vista

Siguiendo un orden de flujo de interacción se encuentran los siguientes componentes:

- El cliente Mobile Phone, componiéndose de un dispositivo móvil que puede ser iOS, Android, y Windows Phone. Incluye el navegador web Cordova para conseguir el funcionamiento híbrido. Este nodo se encarga de contener el cliente de la aplicación para usuarios no administradores, comunicándose con los protocolos HTTP sobre Wi-Fi con el servidor en el puerto 8100.
- El cliente Desktop, que se compone de un dispositivo desktop, en el cual se incluye un navegador web. Este nodo se encarga de contener el cliente para usuarios de tipo administrador comunicándose con los protocolos HTTP o HTTPS sobre TCP/IP con el servidor en el puerto 8080 y 8443 respectivamente.
- El servidor web, componiéndose del entorno de ejecución Node.js en el cual se despliega Express, una aplicación framework que actúa como middleware, conteniendo a su vez la lógica de la aplicación.
- El servidor de base de datos documental MongoDB, el cual se conecta con la aplicación del servidor mediante un DB Driver, en este caso Mongoose.

Exposición de razones

Dado que la aplicación sigue el despliegue habitual para cualquier aplicación desarrollada con el framework MEAN, no se han tomado decisiones relevantes en cuanto a la distribución del sistema.

En la parte del cliente, puesto que la aplicación va a contar con dos tipos de clientes diferentes, uno para dispositivos móviles orientado a usuarios comunes, y otro para dispositivos desktop orientado a usuarios administradores, se presentan dos nodos independientes actuando como clientes del servidor web con una arquitectura web cliente-servidor. Además, se ha tomado la decisión de dejar abiertos los dos servidores HTTP y HTTPS del cliente de escritorio para que se pueda usar el que se desee, al menos hasta que el producto salga a producción y se replantee dicha decisión.

Anexo K: Detalles de implementación del sistema

K.1 Front-end – Cliente de escritorio

En lo referente a los detalles de la implementación del front-end de escritorio realizado con AngularJS, se van a aclarar algunos detalles y nombrar las librerías de las que se ha hecho uso.

Primero hay que aclarar que se ha seguido uno de los estándares que marca AngularJS para la organización del proyecto del cliente. De esta forma, se han agrupado todas las templates en un directorio, todos los controladores, directivas y servicios en otro, y en la raíz el index.html junto con el package.json y el app.js.

Se ha decidido usar el gestor de vistas de ui-router en vez del ngRouter puesto que además de ofrecer un mayor potencial y mejora de la flexibilidad gracias a su gestión de los estados, el equipo de desarrollo ya tenía experiencia en ese gestor.

Las librerías que se han utilizado han sido las siguientes:

- **Angular-base64**: Librería que permite codificar y decodificar datos en base 64, utilizado en el requisito [RFU-1] para implementar el estándar basic authorization [49].
- **Angular-char.js y chart.js**: Se han utilizado estas dos librerías para poder hacer uso de las gráficas que se muestran en la pantalla de estadísticas.
- **Angular-jwt**: Librería utilizada para codificar y decodificar los JWT utilizados en el sistema.
- **Angular-ui-notification**: Librería para mostrar notificaciones de diversa naturaleza.

Por último añadir que para el correcto funcionamiento del control de accesos en la aplicación, se han utilizado interceptores de peticiones HTTP para incluir automáticamente el JWT si existe en cada una de las peticiones antes de ser enviada, o para comprobar si el servidor ha devuelto un código HTTP 401 (debido a problemas con el JWT) o HTTP 403 (debido a un problema de permisos) y realizar las acciones determinadas (p.ej.: comprobar que no hay errores en el JWT o si no existe y ha caducado, salir de la sesión…) antes de que la página siga con su flujo de ejecución correspondiente.

K.2 Front-end – Cliente de dispositivo móvil

En lo referente a los detalles de la implementación del front-end de dispositivos móviles realizado con Ionic 2 y Angular, se van a aclarar algunos detalles y nombrar las librerías de las que se ha hecho uso.

Primero hay que aclarar que se ha seguido uno de los estándares que marca Angular para la organización del proyecto del cliente, introduciendo todo el contenido no compilado de TypeScript en la carpeta **/src**. De esta forma, se ha agrupado las páginas por directorios (dentro de la carpeta **/pages**), incluyendo cada una de ellas su propia template, controlador y fichero de estilo.

En la carpeta **/app** se incluye el fichero de inicialización de la aplicación **main.ts**, el módulo **app.module.**ts para acoplar e inyectar todas las dependencias globales del sistema e indicar como inicializarlo, y el fichero **app.component.ts** que actúa como componente general de la aplicación, en la que van a ejecutar sus ciclos de vida el resto de componentes de la aplicación (además, este componente incluye su propia template actuando como navbar lateral).

Por último, en la carpeta **/assets** se encuentra todo el contenido multimedia utilizado en la aplicación. En la carpeta **/models** todos los modelos de entidades fijas. En la carpeta **/providers** todos los servicios utilizados. Y en la carpeta **/theme** los colores principales de la aplicación.

El resto de directorios y ficheros son los usados para compilar la aplicación y propios de los frameworks de Ionic 2 y Angular.

La única librería usada de forma extra a las que vienen por defecto en el package.json de aplicaciones de Ionic 2 ha sido la siguiente:

- **Angular2-jwt**: Librería utilizada para codificar y decodificar los JWT utilizados en el sistema.

K.3 Back-end

A continuación, se van a exponer algunos de las librerías (no pertenecientes al core del framework Express) y decisiones de implementación de las mismas tomadas a lo largo del desarrollo del proyecto en lo referente al back-end de la aplicación:

- **async**: Módulo que facilita un conjunto de funciones para tratar con el asincronismo de JavaScript y, en particular, de Node.js. Dado que no se recomienda el uso de forEach para iterar arrays, se han empleado las funciones each del módulo async para iterar listas de POIs. La función each de JavaScript habría sido igualmente válida de no ser porque se ejecuta de forma asíncrona y, en estos casos, era necesario que la iteración finalizase antes de enviar una respuesta al cliente. La función each de async permite saber cuándo ha finalizado la iteración para realizar las acciones deseadas posteriormente.
- **base-64**: Módulo que permite codificar y decodificar ristras de bytes o caracteres a o en base 64. Se emplea para decodificar la información del email y la contraseña del usuario enviados por el cliente en codificación base 64 al realizar el inicio de sesión del usuario.
- **cors**: Módulo que actúa como middleware en las peticiones HTTP configurando la activación del CORS (*Cross-origin resource sharing)* [38], se ha utilizado en este caso para permitir que durante el desarrollo se puedan comunicar el servidor web que contiene la aplicación de Ionic 2 con el servidor web que contiene el back-end de Express sin ningún tipo de problema.
- **crypto**: Módulo que sirve para cifrar cadenas de caracteres. En este caso se ha usado para cifrar todas las contraseñas que se guardan en la base de datos.
- **express-jwt**: Módulo que actúa como middleware interceptando las peticiones que llegan al back-end para comprobar que las rutas que estén protegidas solo puedan ser accedidos por peticiones que contengan JSON Web Tokens válidos.
- **gridfs-stream**: Módulo que facilita el uso de canales de lectura y escritura siguiendo la especificación GridFS de MongoDB. Se emplea para el almacenamiento y lectura de las imágenes adjuntas a los objetos de almacén del sistema. Se ha decidido usar GridFS para evitar el límite de tamaño de un documento estándar de MongoDB de 16MB y así poder subir imágenes grandes y de buena calidad.
- **https**: Módulo que sirve para lanzar un servidor HTTPS en Node.js.
- **jsonwebtoken**: Módulo que ofrece varias funciones sobre JSON Web Tokens. Se emplea para firmar los tokens que se envían en el inicio de sesión de un usuario.
- **morgan:** Módulo que actúa como middleware para hacer un seguimiento e imprimir por pantalla las peticiones HTTP del servidor y así ayudar a tareas de depuración.
- **protractor:** Módulo que permite utilizar la herramienta de Protractor.
- **stream**: Módulo que facilita la creación y uso de *streams*. Se emplea para crear un *stream* con la imagen en base 64 que envía el cliente de cara a guardarla con gridfs-stream.
- **swagger-jsdoc:** Módulo que permite utilizar la herramienta de Swagger.
- **utf8**: La decodificación con base-64 da lugar a una ristra de bytes que más adelante hay que interpretar. Para ello, se usa el módulo utf8 para pasarlos a la codificación deseada.

K.4 Persistencia y capa de datos

Aunque back-end y persistencia van muy unidos en el framework MEAN, sí se pueden destacar algunos detalles de esta parte de la implementación.

El driver empleado para la comunicación con la base de datos MongoDB ha sido Mongoose, que, como se ha mencionado, ha facilitado la creación de esquemas y modelos para las colecciones en la base de datos. La conexión se crea en el fichero server.js, el primero en ejecutarse al lanzar la aplicación. Dicha conexión se crea con el puerto por defecto de MongoDB (27017) y con la base de datos "depotCloudDb". Además, también en el fichero **server.js**, se han asignado las rutas de los modelos en la aplicación con 'app.models'. De esta forma, resulta mucho más sencillo acceder a ellos desde cualquier parte del back-end de la aplicación.

Además, ha sido necesario modificar la variable "Promise" de Mongoose, fijándola a "global.Promise" dado que, de no hacerlo, se indicaba que las promesas que se estaban usando con ese modelo en algunas partes del código estaban *deprecated*.

Anexo L: Estadísticas implementadas

Las estadísticas implementadas en el sistema para que los usuarios administradores las puedan visualizar se centran en conocer información de los usuarios del sistema, y el sistema en general, por lo tanto, se tienen en cuenta los datos de todos los usuarios para estas estadísticas.

Se cuenta con un total de 13 estadísticas implementada, las cuales son:

- **Total de usuarios en el sistema:** Total de usuarios incluyendo activos e inactivos en el sistema, a excepción de usuarios tipo administrador (ver **Figura 46**).
- **Total de almacenes en el sistema:** Total de almacenes existentes en el sistema (ver **Figura 46**).
- **Total de objetos de almacén en el sistema:** Total de objetos de almacén existentes en el sistema (ver **Figura 46**).
- **Estado de los usuarios del sistema:** Distribución del número de usuarios entre cuentas activas e inactivas, a excepción de usuarios tipo administrador (ver **Figura 47**).
- **Número medio de almacenes por usuario:** Número medio de almacenes totales creados en el sistema en función del número de usuarios totales registrados en el mismo (ver **Figura 48**).
- **Número medio de objetos de almacén por usuario:** Número medio de objetos de almacén totales creados en el sistema en función del número de usuarios totales registrados en el mismo (ver **Figura 48**).
- **Accesos a lo largo del tiempo:** Número de cuentas de usuario que han accedido a la aplicación iniciando sesión a lo largo del último año, mes a mes (ver **Figura 49**).
- **Nuevos registros a lo largo del tiempo:** Número de registros de cuentas de usuario a lo largo del último año, mes a mes (ver **Figura 50**).
- **Cuentas borradas a lo largo del tiempo:** Número de cuentas de usuario borradas a lo largo del último año, mes a mes (ver **Figura 51**).
- **Nuevos almacenes a lo largo del tiempo:** Número de creaciones de almacenes a lo largo del último año, mes a mes (ver **Figura 52**).
- **Nuevos objetos de almacén a lo largo del tiempo:** Número de creaciones de objetos de almacén a lo largo del último año, mes a mes (ver **Figura 53**).
- **Almacenes según tipo:** Distribución del número de almacenes creados dependiendo del tipo (viviendas, trasteros, armarios) (ver **Figura 54**).
- **Almacenes según distancia:** Distribución del número de almacenes creados dependiendo de la distancia ([0-1km], [1km-2km], [2km-10km], [10km-100km], [100km-300km], [300km, +]) (ver **Figura 55**).

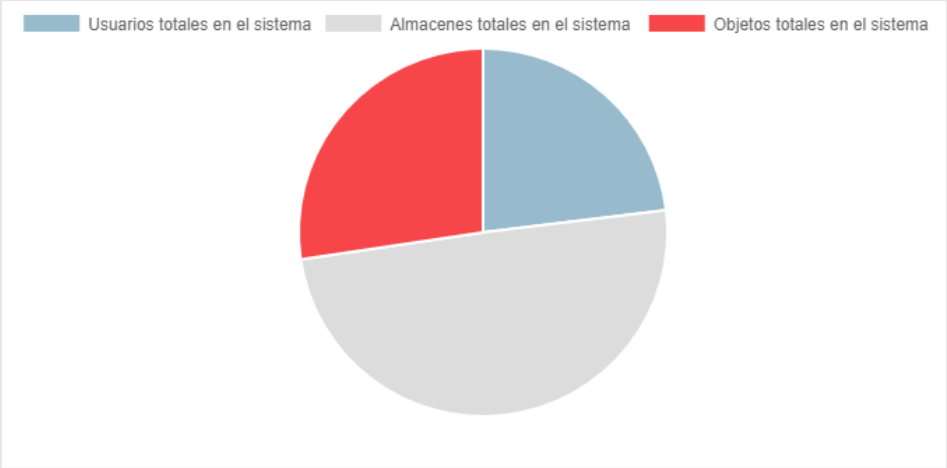

Figura 46. Estadística de usuarios totales, almacenes totales y objetos totales.

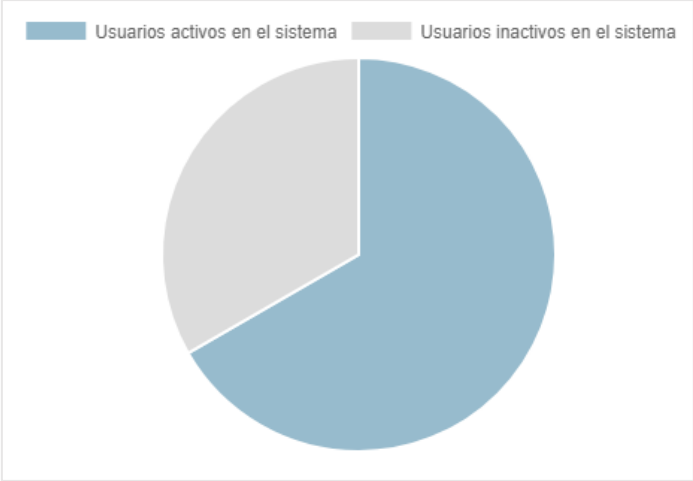

Figura 47. Estadística de usuarios activos e inactivos.

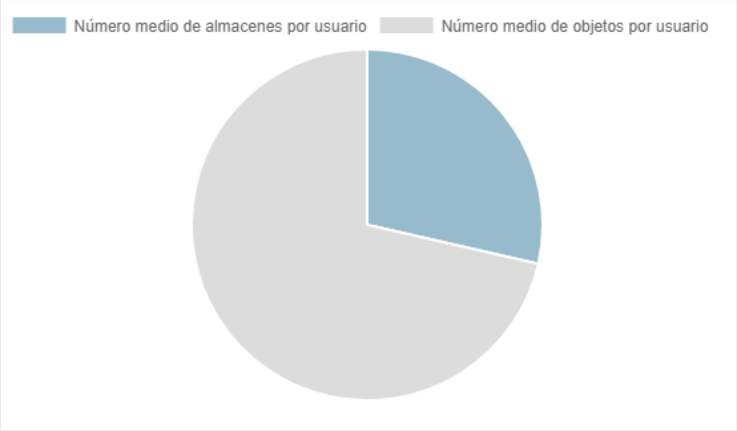

Figura 48. Estadística de objetos y almacenes por usuario.

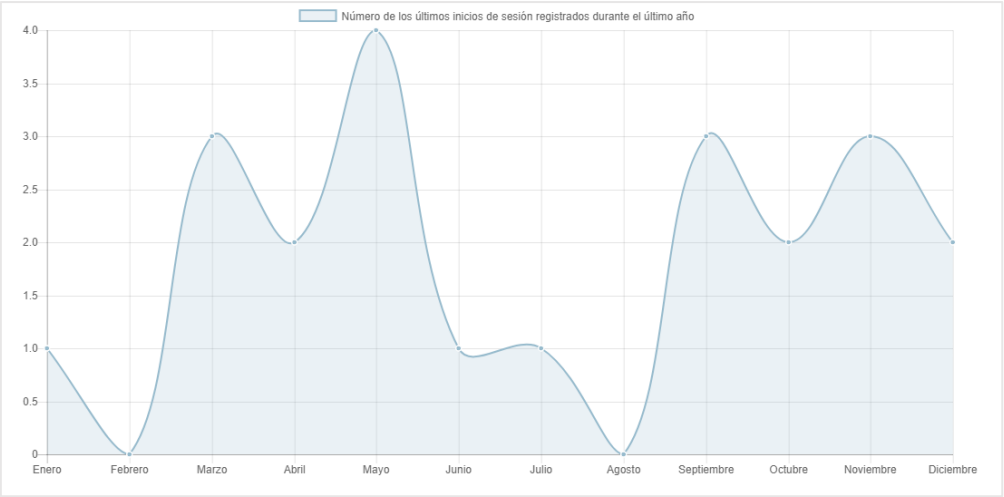

Figura 49. Estadística de inicios de sesión en el último año.

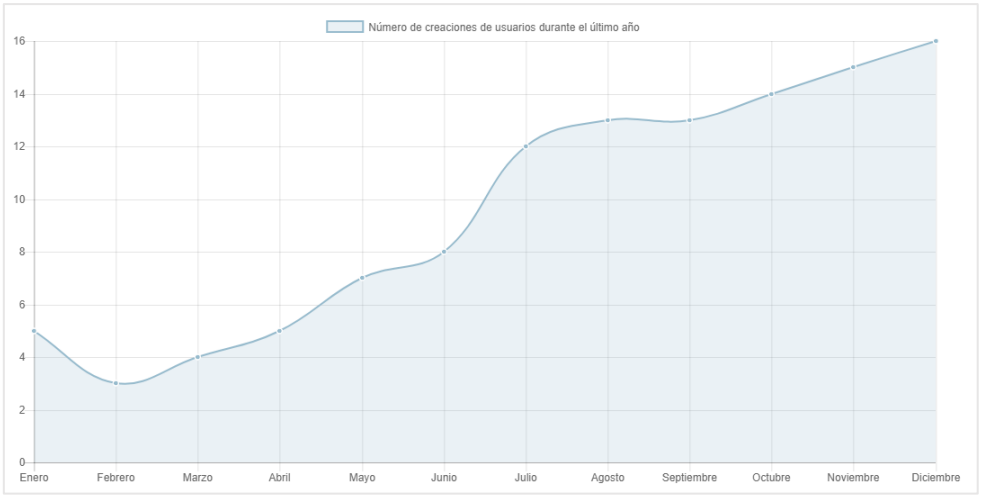

Figura 50. Estadística de cuentas de usuario creadas en el último año.

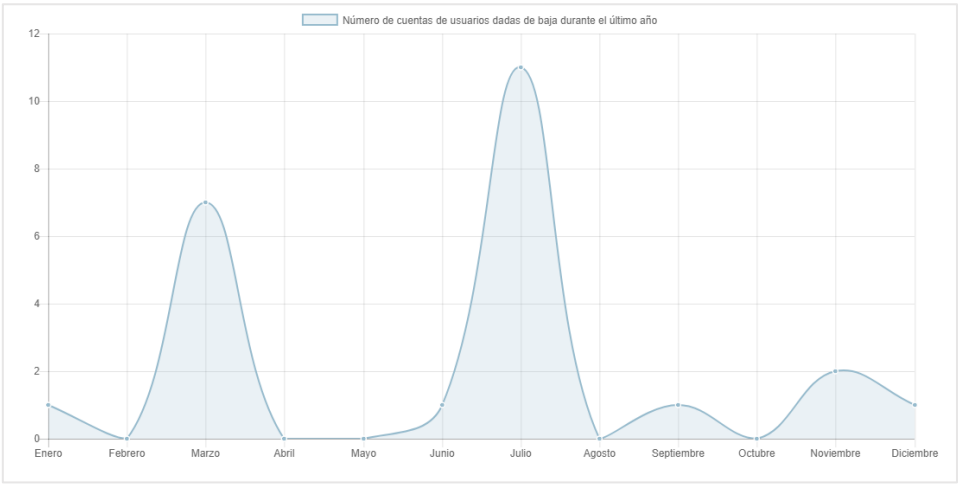

Figura 51. Estadística de cuentas de usuario borradas en el último año.

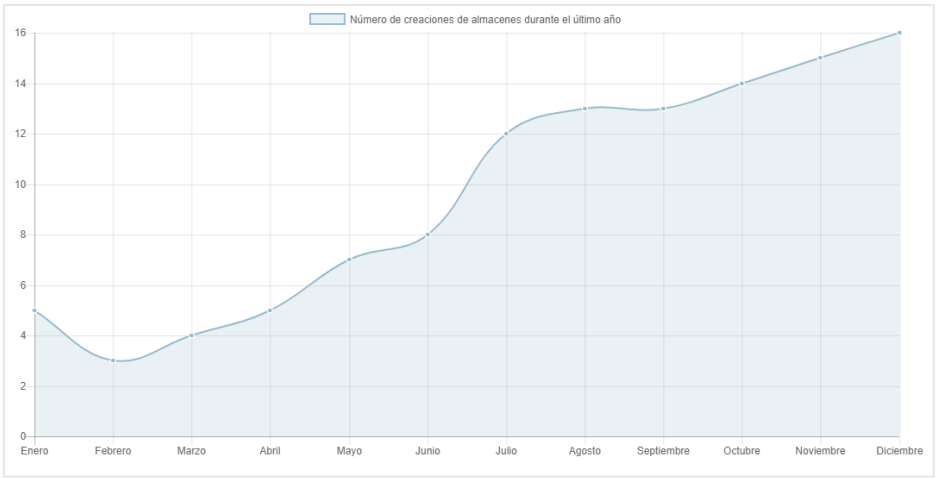

Figura 52. Estadística de almacenes creados en el último año.

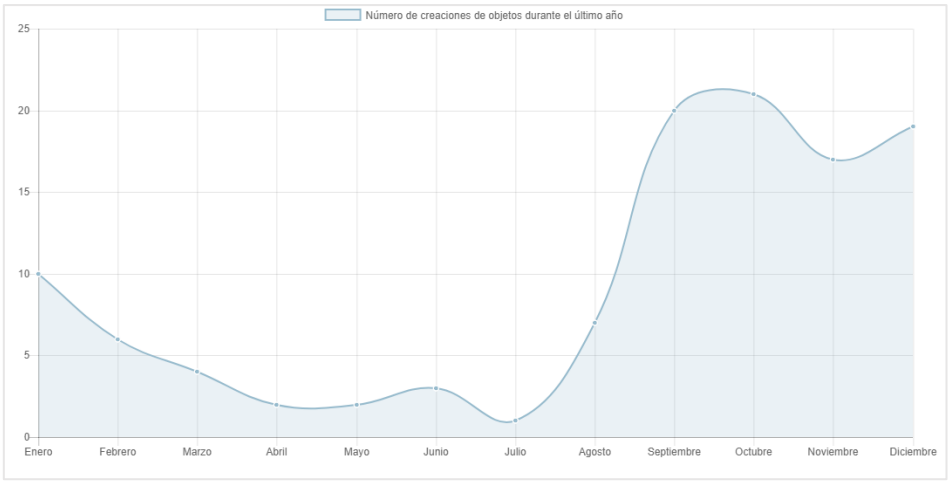

Figura 53. Estadística de objetos creados en el último año.

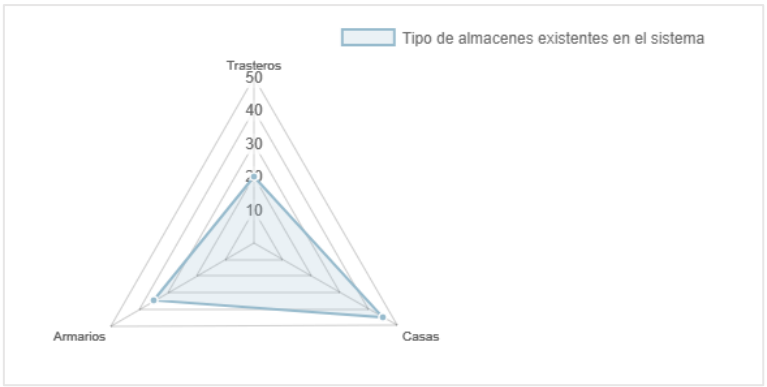

Figura 54. Estadística de almacenes según el tipo.

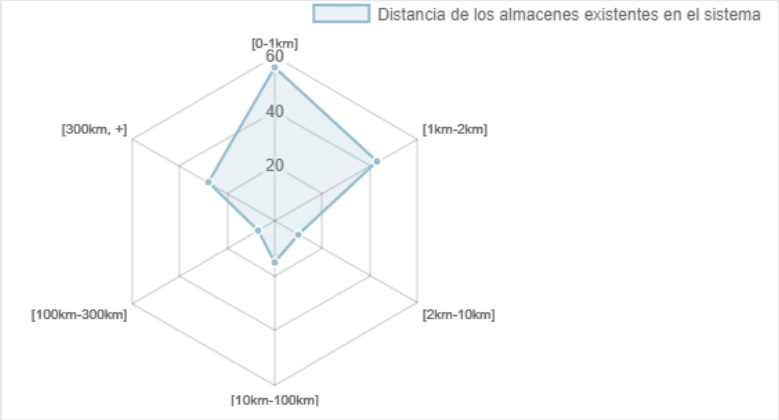

Figura 55. Estadística de almacenes según la distancia.

Estas estadísticas están diseñadas con los datos que más puedan interesar a los usuarios de tipo administrador. Este tipo de usuarios, tienen prioridades como el control del número y del estado de usuarios, y el resto de las entidades del sistema (véase objetos y almacenes). Por ello se considera que las estadísticas elegidas son relevantes y útiles.

Las imágenes de las estadísticas mostradas en este documento han sido alteradas con fin de que se puedan visualizar de una forma representativa de la realidad.

Anexo M: Instrucciones y despliegue del sistema

Para realizar el despliegue del sistema, asumiendo que el sistema anfitrión ya tiene instaladas las tecnologías de Node.js y MongoDB, hay que seguir los siguientes pasos en el orden especificado, situándonos de partida en la raíz del proyecto:

1. Abrir una terminal de comandos y ejecutar '**npm install'**. Sólo será necesario la primera vez que se despliegue el sistema para descargar todas las dependencias.
2. Lanzar un servicio de MongoDB con el siguiente comando en la terminal: '**sudo service mongod start**'. Si se prefiere, también puede crearse una carpeta 'db' en la raíz del directorio y ejecutar el comando '**mongod - -dbpath db'**.
3. Añadir un usuario administrador en la base de datos. Para ello, ejecutar el fichero initialize.js mediante el comando '**node initialize.js'**. El usuario administrador se registra con el correo 'admin@email.com' y la contraseña 'password', con lo que podrá iniciar sesión en la aplicación.
4. Ejecutar en otra terminal el comando '**npm start**' o, alternativamente, el comando **'node server.js**'. Esto hace que se lance el servidor y se despliegue el cliente de escritorio. El servidor se habrá desplegado cuando la consola indique que se encuentra escuchando en los puertos 8080 (HTTP) y 8443 (HTTPS).
    a. Para emplear la aplicación de escritorio, abrir una venta de un navegador (preferiblemente Mozilla Firefox o Google Chrome) e introducir la URL 'http://localhost:8080' o 'http://localhost:8443'.
    b. Para ejecutar el cliente de dispositivos móviles primero hay que situarse en la carpeta correspondiente (con la consola: **'cd public-ionic'**). A continuación, ejecutar el comando **'ionic serve'** para lanzar el servidor web. Esperar a que el servidor indique por consola que se ha inicializado, abrir un navegador web y acceder a la dirección 'http://localhost:8100'.

Por último, si se desea ejecutar los test se han de seguir las siguientes instrucciones (siempre con el servidor sin lanzar):

- Para ejecutar los test de Protractor en el proyecto hay que seguir las siguientes instrucciones:

1. Comprobar que se ha instalado bien la versión del framework al inicializar el proyecto ejecutando el comando '**node_modules/protractor/bin/protractor –version'**.
2. Lanzar el servidor de Selenium: '**node_modules/protractor/bin/webdriver-manager start'**. Esto imprimirá información por pantalla hasta que muestre que ya se ha inicializado (se puede ver el estado del servidor en 'http://localhost:4444/wd/hub').
3. Por último, descomentar la línea 7 del fichero './public/app.js' para agilizar el proceso y lanzar el comando '**node_modules/protractor/bin/protractor test/protractor/conf.js'.**

- Para ejecutar los test de Mocha en el proyecto, únicamente hay que ejecutar el comando '**npm test**' y se lanzarán automáticamente.

Anexo N: API RESTful del sistema

Todas las peticiones de la API RESTful del sistema pueden devolver un código de respuesta HTTP 401, 404, y 500. La primera indica que el token de la sesión ha caducado, es inválido, o es inexistente. La segunda que la petición está mal formada ya que se ha utilizado de forma incorrecta. La tercera que ha ocurrido un error interno en el servidor. A continuación, se puede ver el formato que siguen:

HTTP 401 | 404 | 500	{ "success": boolean, "message": string }

Además, todas las peticiones incluidas en las secciones **Administrador: Gestión de usuarios**, **Estadísticas de administrador**, y la operación "GET /users/" de **Gestión de cuentas de usuario** también pueden devolver un código HTTP 403 con el mismo formato mostrado anteriormente que indica que no se tiene los permisos adecuados para realizar dicha petición.

N.1 Administrador: Gestión de usuarios

PUT /admin/users/{email}	
Permite a un administrador modificar una información determinada de un usuario en concreto.	
Request Headers	{ "Authorization": "Bearer " + JSON Web Token }
Request Body	{ "name": string, "newEmail": string }
Responses	
HTTP 200	{ "success": boolean, "message": string }

PUT /admin/users/{email}/active	
Permite a un administrador reactivar a un usuario que previamente ha borrado su cuenta.	
Request Headers	{ "Authorization": "Bearer " + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ "success": boolean, "message": string }

N.2 Gestión de miembros de unidad familiar

POST /members/{email}/{name}	
Añade un miembro a la unidad familiar.	
Request Headers	{ "Authorization": "Bearer " + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ "success": boolean, "message": string }

PUT /members/{email}/{name}	
Modifica el nombre de un miembro de la unidad familiar.	
Request Headers	{ "Authorization": "Bearer " + JSON Web Token }
Request Body	{ "newName": string }
Responses	
HTTP 200	{ "success": boolean, "message": string }

DELETE /members/{email}/{name}	
Elimina un miembro de la unidad familiar.	
Request Headers	{ "Authorization": "Bearer " + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ "success": boolean, "message": string }

N.3 Gestión de cuentas de usuario

GET /users/	
Lista todos los usuarios del sistema (a excepción de usuarios administradores) con información a la que solo un usuario administrador puede acceder.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “users”: [{ “email”: string, “name”: string, “admin”: boolean, “isActive”: boolean, “members”: [string] }] }

POST /users/	
Crea una nueva cuenta de usuario.	
Request Headers	Ninguno
Request Body	{ “name”: string, “password”: string, “rePassword”: string, “email”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

PUT /users/{email}	
Permite cambiar la información de una cuenta de usuario.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “name”: string, “current”: string, “new”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

DELETE /users/{email}	
Elimina una cuenta de usuario.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “current”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

GET /login/	
Consigue un JSON Web Token que le permite iniciar sesión al usuario en el sistema.	
Request Headers	{ “Authorization”: “Basic “ + Base64.encode(user + ‘:’ + password) }
Request Body	Ninguno
Responses	
HTTP 200	{ “token”: { “email”: string, “name”: string, “admin”: boolean, “isActive”: boolean, “members”: [string] } }

N.4 Almacenes

GET /depots/{owner}	
Lista todos los almacenes pertenecientes a una unidad familiar.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depots”: [{ “_id”: string, “name”: string, “owner”: string, “location”: string, “type”: string, “distance”: string, “description”: string }] }

GET /depots/{owner}/{id}	
Obtiene toda la información correspondiente a un almacén determinado.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depot”: { “_id”: string, “name”: string, “owner”: string, “location”: string, “type”: string, “distance”: string, “description”: string } }

POST /depots/{owner}	
Crea un almacén con su correspondiente información y asociando la creación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “name”: string, “member”: string, “location”: string, “type”: string, “distance”: string, “description”: string }
Responses	
HTTP 200	{ “depot”: { “_id”: string, “name”: string, “owner”: string, “location”: string, “type”: string, “distance”: string, “description”: string } }

PUT /depots/{owner}/{id}	
Modifica un almacén con su correspondiente información y asociando la modificación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “name”: string, “member”: string, “location”: string, “type”: string, “distance”: string, “description”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

DELETE /depots/{owner}/{id}	
Elimina un almacén con su correspondiente información y asociando la eliminación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “member”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

N.5 Objetos de almacén

GET /depotOjects/{depot}	
Lista todos los objetos que pertenecen a un almacén de una unidad familiar.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depotObjects”: [{ “_id”: string, “name”: string, “owner”: string, “image”: string, “depot”: string, “guarantee”: string, “dateOfExpiry”: string, “description”: string }] }

GET /depotObjects/search/{owner}/{name}	
Busca todos los objetos pertenecientes a una cuenta de usuario que contengan la palabra {name} (la búsqueda no es sensible a las minúsculas o mayúsculas).	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }

GET /depotObjects/search/{owner}/{name}	
Request Body	Ninguno
Responses	
HTTP 200	{ “depotObjects”: [{ “_id”: string, “name”: string, “owner”: string, “image”: string, “depot”: string, “guarantee”: string, “dateOfExpiry”: string, “description”: string }] }

POST /depotObjects/{depot}	
Crea un objeto con su correspondiente información y asociando la creación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “owner”: string, “name”: string, “member”: string, “image”: string, “guarantee”: string, “dateOfExpiry”: string, “description”: string }
Responses	
HTTP 200	{ “depotObject”: { “_id”: string, “name”: string, “owner”: string, “image”: string, “depot”: string, “guarantee”: string, “dateOfExpiry”: string, “description”: string } }

PUT /depotObjects/{owner}/{name}	
Modifica un objeto con su correspondiente información y asociando la modificación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “owner”: string, “name”: string, “member”: string, “image”: string, “guarantee”: string, “dateOfExpiry”: string, “description”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

DELETE /depotObjects/{owner}/{name}	
Elimina un objeto con su correspondiente información y asociando la eliminación al miembro de la unidad familiar correspondiente.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	{ “owner”: string, “member”: string }
Responses	
HTTP 200	{ “success”: boolean, “message”: string }

N.6 Informes

GET /activities/{user}	
Lista todas las actividades que pertenecen a una cuenta familiar.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “activities”: [{ “owner”: string, “type”: string, “action”: string, “name”: string, “author”: string, “activityDate”: string }] }

GET /reports/{user}	
Lista todos los informes/recomendaciones relacionados con una cuenta familiar.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	

GET /reports/{user}	
HTTP 200	{ “reports”: [{ “owner”: string, “type”: string, “depotObject”: string, “reportDate”: string }] }

N.7 Estadísticas de administrador

GET /adminStats/totalUsers	
Devuelve el número de usuarios totales registrados en el sistema, incluidos usuarios con cuentas desactivadas, pero no los administradores.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “totalUsers”: number }

GET /adminStats/totalDepots	
Devuelve el número de almacenes totales registrados en el sistema.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “totalDepots”: number }

GET /adminStats/totalDepotObjects	
Devuelve el número de objetos totales registrados en el sistema.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “totalDepotObjects”: number }

GET /adminStats/usersStatus	
Devuelve el número de usuarios cuyas cuentas se encuentran actualmente activas e inactivas.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “activeUsers”: number, “inactiveUsers”: number }

GET /adminStats/depotPerUser	
Devuelve el número medio de almacenes totales creados en el sistema en función del número de usuarios totales registrados en el mismo.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depotsPerUser”: number }

GET /adminStats/depotObjectsPerUser	
Devuelve el número medio de objetos totales creados en el sistema en función del número de usuarios totales registrados en el mismo.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depotObjectsPerUser”: number }

GET /adminStats/lastLogins	
Devuelve el número de inicios de sesión de usuarios registrados en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “lastLogins”: number }

GET /adminStats/lastLogins	
Devuelve el número de inicios de sesión de usuarios registrados en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “lastLogins”: number }

GET /adminStats/lastRegistrations	
Devuelve el número de registros de cuentas de usuarios en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “lastRegistrations”: number }

GET /adminStats/lastDeactivations	
Devuelve el número de cuentas de usuario dadas de baja en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “lastDeactivations”: number }

GET /adminStats/creationDateDepots	
Devuelve el número de creaciones de almacenes en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “creationDateDepots”: number }

GET /adminStats/creationDateDepotObjects	
Devuelve el número de creaciones de objetos en el sistema durante el último año, agrupados por meses.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “creationDateDepotObjects”: number }

GET /adminStats/depotTypes	
Devuelve el número de almacenes creados en el sistema de cada tipo.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }

GET /adminStats/depotTypes	
Request Body	Ninguno
Responses	
HTTP 200	{ “storageRooms”: number, “houses”: number, “wardrobes”: number }

GET /adminStats/depotDistances	
Devuelve el número de almacenes creados en el sistema según su distancia.	
Request Headers	{ “Authorization”: “Bearer “ + JSON Web Token }
Request Body	Ninguno
Responses	
HTTP 200	{ “depotDistances”: [number] }