Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación móvil para la gestión de tareas domésticas en familias con hijos

Trabajo Fin de Grado
Grado en Ingeniería Informática

Autor: Valls Ciscar, Carlos
Tutor: De Andrés Martínez, David
Curso 2017/2018

Resumen

Dado el reclamo actual de los dispositivos móviles en las generaciones más jóvenes y los nuevos formatos de familia, se ha visto la necesidad de desarrollar una herramienta de ayuda en la gestión del hogar. GeFa es una aplicación en el sistema operativo Android para la gestión de las tareas domésticas mediante el uso de dos roles: el supervisor y el supervisado. El supervisor son los padres, madres o tutores que se encargan de crear las tareas y asignarlas a los supervisados que son los/as hijos/as. Para que todos los miembros de la familia puedan acceder a las tareas y otras utilidades se han creado distintas versiones de la aplicación. Dependiendo del rol asignado a la hora del registro se tiene acceso a la versión de los padres, madres o tutores o la de los/as hijos/as. Toda la información se almacena en un servidor que hemos creado, que se encarga también del control de acceso al sistema y ofrece una forma segura de comunicación de los datos.

Palabras clave: Android, REST, Tareas domésticas, JHipster, Spring.

Abstract

Given the current demand of mobile devices in younger generations and new family formats, the need to develop a help tool in home management has been seen. GeFa is an application in the Android operating system for the management of domestic tasks through the use of two roles: the supervisor and the supervised. The supervisor is the parents or guardians who are responsible for creating the tasks and assign them to the supervised ones that are the children. Different versions of the application have been created so that all the members of the family can access the tasks and other utilities. Depending on the role assigned at the time of registration, is available the access to the parents' version or the children's version. All the information is stored in a server that we have created, which also controls the access to the system and offers a secure way of communicating the data.

Keywords: Android, REST, Housework, JHipster, Spring.

Índice de contenidos

Índice de Figuras

Índice de tablas

1. Introducción

El proyecto "Aplicación móvil para la gestión de tareas domésticas en familias con hijos" tiene como principal motivación mejorar y profundizar en los conocimientos de la plataforma Android y el aprendizaje de tecnologías Web y de servidores en Java.

La elección del sistema operativo móvil Android para el desarrollo de la aplicación se debe a que el mercado español de móviles con dicho sistema es del 92% [1], mucho más elevado que el resto de los sistemas operativos. Además, la utilización del lenguaje de programación Java para el desarrollo de la aplicación Android como para el desarrollo del servidor lo hacen buena elección.

La gestión de tareas o recordatorios fueron de las primeras aplicaciones móviles en salir al mercado. Aunque hay aplicaciones de tareas muy buenas en las tiendas, existe una falta de éstas con tareas compartidas o supervisadas. Aquí es donde queremos aportar un enfoque distinto, centrándonos en la gestión de tareas domésticas.

Hoy en día es cada vez más común encontrar a niños de menor edad poseedores de un teléfono móvil inteligente, por lo que intentar gestionar las tareas del hogar mediante tareas compartidas asignadas por los padres a los hijos usando una aplicación móvil puede resultar esencial. Para conseguir esto será necesario la implementación de dos aplicaciones móviles en Android: una para los padres y otra para los hijos; y también hará falta un servidor que almacene las tareas y gestione a los usuarios y la información a la que tienen acceso.

Por lo tanto, se desarrollará una aplicación, que llamaremos GeFa, para los padres que sea capaz de crear, asignar y modificar tareas; controlar el acceso a la información de la familia y que pueda crear pagos para motivar el cumplimiento de las tareas. La aplicación de los hijos se centrará en la visualización de las tareas que tiene pendientes de realizar, permitiéndoles marcar como completadas las tareas realizadas, aunque éstas serán confirmadas por los padres. El servidor se encargará de almacenar todas las tareas, pagos, usuarios y familias.

Contexto sociocultural

En un mundo en el cual el uso de teléfonos móviles no para de crecer, el desarrollo de aplicaciones para el mercado de *Smartphone* es imprescindible. España en particular es el quinto país, por detrás de Brasil, China, EEUU e Italia, que más tiempo pasa utilizando el teléfono. Además, el porcentaje de niños que usa y tiene un teléfono móvil no deja de aumentar. En 2014, el porcentaje de niños entre 9 y 10 años que poseía un teléfono era un 9%. Pero en cuanto aumentaba la edad pasábamos a un 49% para los de entre 11 y 12 años y un 81% para los niños entre 15 y 16 años. Tanto es el crecimiento de usuarios menores con teléfono móvil que el porcentaje global de menores poseedores de uno es igual al de los adultos [2] [3].

Con estos datos de uso se aprecia la necesidad de no sólo crear una aplicación para la gestión del hogar para los padres, sino también una para los hijos. Esto hará que éstos sean más independientes cuando los padres estén trabajando o no estén en casa y puedan realizar las tareas domesticas asignadas permitiendo a los padres comprobar la realización desde cualquier lugar.

Para que los hijos asuman responsabilidades y sean cada vez más autónomos, se recomienda ampliamente el uso de la tabla Montessori. Maria Tecla Artemisa Montessori fue la primera mujer de nacionalidad italiana que se graduó como doctora en Medicina, pero, es más conocida por el impacto que tuvo en la transformación de los

métodos pedagógicos en los principios del siglo XX. La mayoría de sus ideas hoy parecen indiscutibles, pero en su época, fueron innovaciones sustanciales que alzaron el debate en su momento. La idea principal de su método pedagógico era el uso del juego como principal actividad durante los primeros años de edad, y, progresivamente conforme aumentará la edad del niño asignarle tareas conforme a sus etapas sensibles.

Las etapas sensibles son los periodos en los cuales los niños pueden obtener una destreza con más facilidad. Los principales periodos son: el lenguaje, la coordinación, el agudizamiento de los sentidos y la mejora del comportamiento social. Además, también manifestaba la idea de los niños autónomos. “Ayúdame a hacerlos solo”, era la formula que resumía un modo de motivar a los niños y estimular sus ganas de aprender.

La tabla Montessori se compone mayoritariamente de tareas domésticas, de las cuales se responsabilizarán. Además, el cumplimiento de estas tareas ayudará al desarrollo de su pragmatismo, motricidad y experiencia sensorial, siguiendo los anteriormente citados periodos sensibles [4] [5] [6].

1. Objetivos

El objetivo principal de este proyecto es el desarrollo de una aplicación en el sistema operativo móvil Android para la gestión de las tareas domésticas con hijos. Para conseguir este objetivo habrá que cumplir otros esenciales: se realizará un servidor para el control de acceso a la información compartida, se implementarán aplicaciones diferenciadas entre el supervisor (Padres) y los supervisados (Hijos), la interfaz será sencilla y contendrá multitud de consejos e información sobre lo que supone realizar cada acción o rellenar cada campo, los datos se accederán a través de canales seguros y habrá un control de activaciones de los usuarios.

En lo referente al servidor, se implementará de forma que las peticiones y las respuestas tengan un formato estándar para que en un futuro se puedan realizar aplicaciones en otras plataformas. El acceso a los datos de este servidor se realizará usando llamadas REST a un API mediante HTTPS. Además, aunque el canal sea seguro, el acceso a la información de una determinada familia sólo podrá ser leída o modificada en caso de que un familiar haya activado al miembro que quiere acceder a los datos. Por lo tanto, para acceder al servicio, un usuario, tendrá que estar registrado en la plataforma y además registrado en una familia y activado en ella.

En cuanto a la interfaz de la aplicación, se crearán dos distintas en función de si el usuario es un padre o un hijo. Es decir, aunque la aplicación será la misma, se modificará el contenido dependiendo el tipo del usuario. Esto permitirá que tanto padres como hijos tengan acceso a las tareas, pero, los hijos tendrán una capacidad limitada de modificación; básicamente será un medio para ver las tareas y de poder marcarlas como completadas (aunque posteriormente tendrá que confirmar el cumplimiento de la misma un padre). Ambas interfaces contendrán mensajes y botones que ofrecerán ayuda en el uso de la aplicación. Por ejemplo, en la creación de una tarea, en la que habrá que rellenar muchos campos, se mostrarán comentarios en cada uno de ellos que indicarán qué hay que poner y qué finalidad tienen. Todo en conjunto hará más sencillo el uso de la aplicación en un mercado con una variedad de edades muy amplia.

2. Estructura del documento

En esta memoria se describirá el proceso de la implementación de la aplicación Android y parte de la implementación del servidor que da soporte. Comenzaremos por comentar en la sección segunda las herramientas de gestión de familias que existen actualmente en el mercado y extraeremos de ahí una priorización de objetivos a realizar. A continuación, en el apartado tercero, describiremos las aplicaciones, bibliotecas y *frameworks* más importantes que se han usado en el desarrollo explicando su uso en el proyecto.

Después, en la sección cuarta, pasaremos a hablar de la especificación del proyecto. Se detallará cada una de las capas en la que está compuesto el proyecto usando para ello casos de uso, representaciones de bases de datos y descripciones a través de tablas y bocetos. Luego, en el apartado quinto, procederemos a explicar la implementación del proyecto mediante esquemas, diagramas de flujo y a través de la exposición de algunas secciones importantes del código usado en la aplicación.

Para terminar, se mostrarán en la sección sexta los resultados de la implementación del proyecto mediante imágenes finales que se ajustarán con los casos de uso descritos en el apartado de especificación. Se cerrará la memoria plasmando en el apartado séptimo una serie de conclusiones a tener en cuenta en el proyecto.

2. Herramientas de gestión familiar existentes

Dentro de aplicaciones para la gestión de las familias existen distintas soluciones, desde las que proponen una solución a distintos aspectos de una familia, como Picniic, hasta las que se dedican exclusivamente a la gestión de tareas de los hijos. A continuación, se explicarán brevemente las distintas alternativas más relevantes, según nuestro criterio, que existen actualmente en el mercado.

Para que una aplicación de esta índole tenga éxito creemos que tiene que tener unas características y funcionalidades básicas para conseguirlo:

1. **Aplicación móvil**: con la cuota de mercado situándose en un 63% para teléfonos móviles y de un 37% para ordenadores personales (PCs) [7] realizar una aplicación móvil para ofrecer acceso a un servicio es indispensable.
2. **Gratuidad**: actualmente las aplicaciones que cosechan mayor triunfo son aquellas que tienen un precio de entrada cero y que se financian con compras *in-app* o con publicidad [8]. Dada la reticencia de la mayoría de los usuarios a pagar por una aplicación móvil, vemos que para alcanzar el éxito debemos tener la aplicación a precio cero.
3. **Versión web**: como extra a la aplicación móvil se puede realizar una versión web para aquellos que le resulte más fácil o cómoda esta opción, pero pensamos que es más necesaria una aplicación móvil.
4. **Actualizaciones**: mantener un servicio activo y con añadidos y mejoras para satisfacer al cliente final.
5. **Sistema de pagos**: para este tipo de gestión de tareas para niños/as será necesario un método de recompensa por realizar las tareas.
6. **Tareas asignables:** no queremos que sea simplemente un conjunto de listas de cosas que hacer. Lo recomendable sería que esas tareas fueran asignables a un determinado niño/a.
7. **Aplicación para los niños**: como se comentó en el contexto sociocultural (pág. 9) cada vez los niños/as poseen un teléfono móvil con una edad más temprana, por lo que ofrecerles una forma de acceder a su información sería un añadido a la plataforma.

1. Soluciones existentes.

OurHome

Página principal: http://ourhomeapp.com/

El servicio de OurHome se compone de una aplicación móvil y una web gratuitas para la gestión de tareas domésticas. Permite asignar tareas a los distintos miembros de la familia y a cambio reciben una recompensa en forma de puntos que luego pueden canjear en regalos. El inconveniente de esta aplicación es que el único sistema de pagos es a través de puntos, los cuales, cuando los niños llegan a una cierta edad dejan de apreciar y prefieren dinero.

Figura 1: Aplicación móvil de OurHome.

Picniic

Página principal: https://picniic.com/

Picniic es una aplicación móvil y web de pago para la gestión global de la casa de una familia. Permite crear calendarios, álbumes y listas compartidas, tiene un localizador para los niños con avisos de llegadas a un lugar, menús de comidas...
El inconveniente de esta aplicación es que no se pueden asignar las tareas a una persona determinada y no se recibe ningún tipo de recompensa a cambio de realizaras.

Figura 2: Aplicación web y móvil de Picniic.

Family Chores

Página principal: http://www.familychores.com/

Family Chores es una aplicación web para la gestión de tareas domésticas que permite asignar tareas a un niño/a en particular y fijar tanto puntos como dinero a las tareas. El inconveniente de esta solución es su falta de actualizaciones, de aplicaciones móviles y su interfaz anticuada.

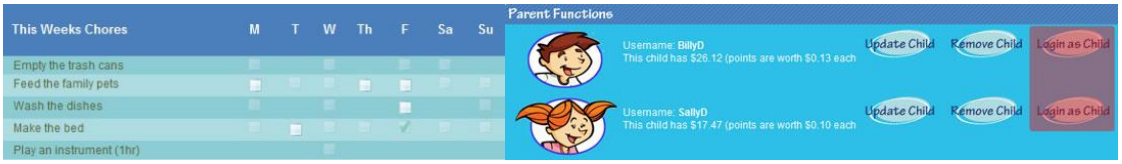

Figura 3: Web de Family Chores.

iAllowance

Página principal: http://www.jumpgapsoftware.com/allowance/index.html

iAllowance es una aplicación móvil de pago para la gestión de tareas domésticas. Permite asignar tareas a los distintos miembros de la familia y a cambio reciben una recompensa en forma de puntos o dinero. El inconveniente de esta aplicación es que sólo dispone de una versión en iOS, que no dispone de una versión de la aplicación para los niños y que la interfaz es esqueumorfista; es decir, el diseño de la aplicación intenta imitar la realidad siendo un estilo ya poco usado y anticuado.

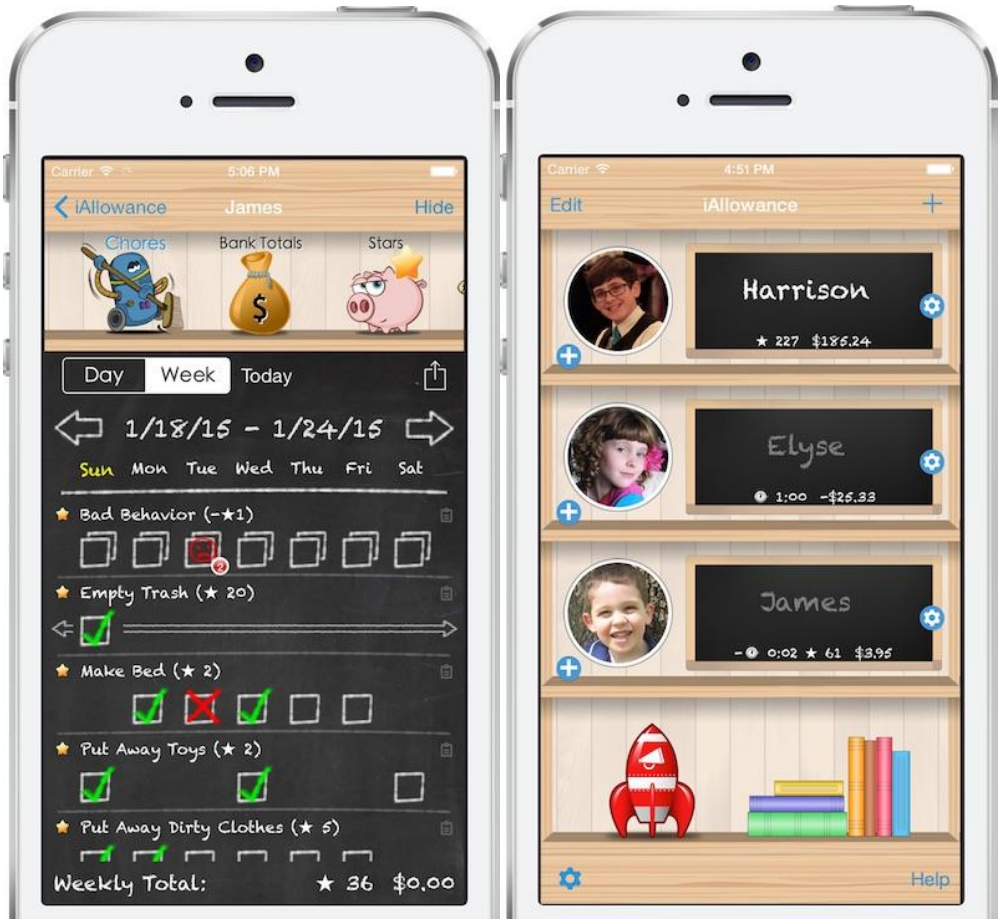

Figura 4: Aplicación móvil de iAllowance.

Comparación de las distintas soluciones

En la Tabla 1 se realiza una comparación de las aplicaciones y servicios descritos anteriormente teniendo en cuenta las funcionalidades que creemos que ha de tener nuestra aplicación.

Tabla 1: Comparación de las aplicaciones y servicios existentes.

Funcionalidad	Aplicación o servicio			
	OurHome	Picniic	Family Chores	iAllowance
Gratuita	Sí	No	Sí	No
Aplicación móvil	iOS y Android	iOS y Android	No	iOS
Versión web	Sí	Sí	Sí	No
Última actualización	Enero 2018	Mayo 2018	Abril 2013	Octubre 2017
Sistema de puntuación	Puntos	-	Puntos y dinero	Puntos y dinero
Tareas asignables	Sí	No	Sí	Sí
App/Web para los niños	Sí	Sí	Sí	No

Como se aprecia en la Tabla 1 muchas de las opciones que existen actualmente ofrecen una versión y una aplicación para móviles, pero sólo la mitad las ofrece gratuitamente. Con respecto a los sistemas de pagos, algunos ofrecen puntos y dinero, otros puntos o directamente no ofrece opción. Dado que ninguno de los servicios actuales brinda una solución completa, se ha decido realizar una propia en la que priorizaremos sus funcionalidades en el siguiente apartado.

2. Priorización de funcionalidades

Después de analizar las aplicaciones existentes en el mercado nos encontramos en la situación de esclarecer nuestras prioridades en el desarrollo de la aplicación. En la Figura 5 se muestra una representación usando el método MoSCoW para identificar estas prioridades. Éste trata de establecer los requisitos por orden de prioridad, donde los más importantes para el producto deben cumplirse primero para tener más posibilidades de éxito. El método MoSCoW es un acrónimo formado por las primeras letras del Inglés: Must, Should, Could y Won't [9].

Mo

- Gestión de tareas
  - Tareas asignables
  - Tareas con recompensas monetarias
- Aplicación en Android
  - Aplicación con versiones para padres e hijos
- Gratuita
- Acceso seguro a los datos

S

- Control de activaciones de los ususarios
- Ayuda y tutoriales integrados

Co

- Recompensación de las tareas mediante puntos y lista de deseos.
- Aplicación web
- Aplicación en iOS

W

- Almacenamiento local y acceso sin conexión
- Tareas compartidas
- Listas compartidas

Figura 5: Representación del método MoSCoW sobre prioridades para el desarrollo de la aplicación.

Las cualidades que tiene que implementar la aplicación final se dividen en cuatro partes. La primera tiene que ver con la gestión de tareas; éstas, tienen que ser asignables a los miembros de la familia y deben de tener una recompensa para motivar a los hijos/as. La segunda parte tiene que ver con la capa de presentación; por el momento solamente se implementará la aplicación sobre Android, ya que éste el sistema operativo móvil más usado en España y el uso del teléfono móvil es más elevado que el de ordenadores personales. Además, esta aplicación contará con una versión para que la usen los padres, madres o tutores y de una versión para los hijos/as. Aun siendo desarrollada sólo en Android, por el lado del servidor se implementará de tal forma que las llamadas puedan ser usadas por cualquier sistema (Web o iOS) que se implemente más adelante. La tercera parte presenta que la aplicación ha de ser gratuita. La cuarta parte tiene que ver con la seguridad de los datos que vamos a almacenar; ya que contaremos con información relacionada con los niños hay que cerciorarse de que el acceso y almacenamiento de estos datos es seguro.

En lo referente a las cualidades que debería implementar la aplicación final se dividen en dos segmentos. El primero es la gestión de activaciones de los miembros de la familia. Para que los miembros de la familia no tengan que crear usuarios y añadirlos a la familia; serán los nuevos usuarios los que se registren y pidan acceso a una familia determinada; no podrá acceder a los datos relativos a la familia hasta que alguno de los miembros (Padres) autoricen el acceso. La segunda parte tiene que ver con la

usabilidad de la aplicación. Para ello se deberían incluir tutoriales o ayudas no intrusivas para que faciliten al nuevo usuario el uso de la aplicación.

En lo relacionado a las cualidades que son deseables, es decir, que se implementarían si se dispusiera de más tiempo, se dividen en tres.
La primera parte tiene que ver con el sistema de recompensa a los hijos/as. Por el momento sólo se implementará la recompensa monetaria dado que tiene más alcance de uso con respecto a la edad de los hijos/as. Para llegar a un mercado un poco más joven se podrían añadir un sistema de recompensa basado en puntos canjeables a través de unas listas de deseos.
La segunda parte es la implementación de la aplicación en iOS para llegar a prácticamente la totalidad del mercado móvil.
La tercera parte es la implementación de una aplicación web para la gestión de la familia y las tareas.

En cuanto a los requisitos que están descartados de momento, pero que en un futuro podrían tenerse de nuevo en cuenta y ser reclasificados en una de las categorías anteriores, son: el almacenamiento local de la información de las tareas para ser accedido sin conexión a internet y la posibilidad de tener listas y tareas compartidas. El problema que puede ocasionar el almacenamiento local es que, a diferencia de la información en el servidor que puede ser modificada por distintos miembros al mismo tiempo sin alterar la cohesión, la coherencia de los datos podría verse afectada si la información fuera modificada por distintos miembros en estado desconectado. Por ello, y dada la complejidad que requeriría controlar estos cambios debidamente, se implementará en futuras versiones en caso de ser necesario. En cuanto a las tareas y listas compartidas decir que, aunque pueden resultar útiles, no son esenciales para obtener los objetivos de la aplicación.

3. Sistemas operativos móviles

Actualmente existen dos sistemas operativos móviles que tienen casi la totalidad del mercado mundial: Android 84.1% e iOS 14.8% [10]. En España, el uso de Android se eleva aún más, llegando a un 92% y esto hace imprescindible que la primera aplicación móvil que se desarrolle sea en esta plataforma [1]. A continuación explicaremos cómo está estructurado el sistema operativo y los puntos fuertes y débiles que podemos encontrar en él.

Android

La aparición del sistema operativo móvil Android en 2008 y su rápido crecimiento dio acceso a un nuevo mercado de aplicaciones por explotar.

El sistema funciona sobre un núcleo basado en Linux que se encarga de funcionalidades subyacentes, como la generación de subprocesos, la seguridad, la gestión de memoria, energía, pantalla, etc.
La capa de abstracción de hardware (HAL) ofrece interfaces estándares para que sean explotadas por el dispositivo usando el framework de Java de nivel más alto.
En las primeras versiones de Android se utilizaba una máquina virtual Java propia llamada Dalvik que se sustituyó a partir de la versión 5 por ART, la cual se encarga de compilar en tiempo de instalación la aplicación para que esté optimizada para el dispositivo, de recolectar los datos no utilizados (*Garbage collector*) y de mejorar la compatibilidad en la depuración de las aplicaciones. Tanto la capa HAL como el *runtime* ART se basan en código escrito en C o C++ por lo que requieren las librerías nativas que soportan la ejecución.

El conjunto de funciones del sistema operativo Android está disponible mediante un API escrito en Java. Esta API se encarga de lo necesario para crear aplicaciones de Android simplificando la reutilización de componentes del sistema y servicios centrales y modulares. En la figura Figura 1 se muestra la pila de software Android en la que se aprecian todas las capas descritas anteriormente [11].

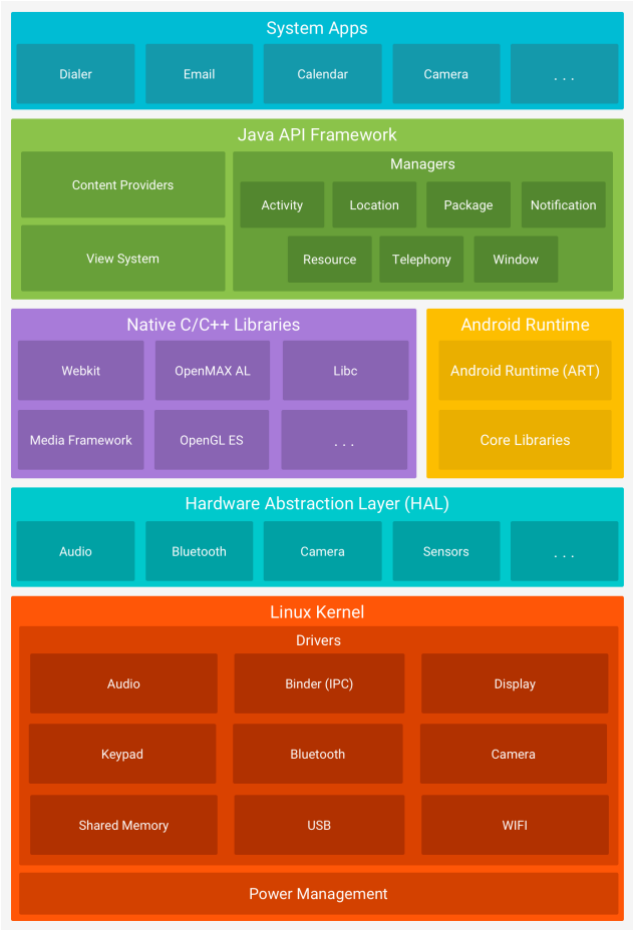

Figura 6: Pila de software de Android [11]

Las aplicaciones en Android se basan en el uso de *Activities*, que se trata de un componente que contienen una interfaz con la que los usuarios pueden interactuar. Cada *Activity* tiene un ciclo de vida que hay que controlar: creación, pausa, destrucción, reanudación... Éste se realiza mediante el uso de *callbacks* que se implementan en la clase que la contiene. En versiones posteriores de Android se introdujeron Fragmentos (*Fragments*) que permitía fraccionar la *Activity* en pedazos para que fuesen reutilizables. Cada fragmento representa un comportamiento o una parte de la interfaz de usuario en una *Activity*, combinando varios o sustituyéndolos se pueden crear aplicaciones más dinámicas y rápidas sin necesidad de lanzar nuevas *Activities* [11].

Como puntos fuertes del sistema operativo Android podemos destacar los siguientes:

1. **Elección de teléfono móvil:** Android distribuye su sistema operativo a través de licencias con fabricantes de teléfonos, por lo que existen gran variedad de teléfonos con distintos precios y características usando un mismo sistema operativo a distintos precios. Aunque también puede ser un punto débil del ecosistema.

2. **Android Market o Google play:** dado el bajo precio de publicar las aplicaciones en la tienda y la gran cantidad de usuarios que tienen acceso a ésta. Aunque también permite la instalación de aplicaciones de cualquier parte.
3. **Sistema Abierto:** Android permite editar la apariencia de su interfaz, por parte de los fabricantes y de terceros, e instalar aplicaciones de los fabricantes en los teléfonos. Esto supone una libertad a la hora de personalizar la interfaz de tu dispositivo, pero puede verse afectado por la falta de optimización de las interfaces o un problema de seguridad y privacidad en el caso de las aplicaciones instaladas por defecto.

Como puntos débiles podemos destacar los siguientes:

1. **Fragmentación:** aunque Android saque actualizaciones de su sistema operativo o parches de seguridad, estos sólo llegan en pocas circunstancias o con mucho retraso a los usuarios finales, dado que estas actualizaciones se tienen que realizar a través del fabricante del teléfono móvil. Dada esta fragmentación de versiones dentro del mismo sistema hace que los desarrolladores tengan que dedicar más tiempo en desarrollar las aplicaciones para una determinada versión o tener que prescindir de una funcionalidad por la falta de usuarios en la versión de Android compatible. En la Figura 7 se muestra la distribución de las distintas versiones de Android y su evolución con respecto al tiempo.
2. **Problemas de seguridad:** Android siempre estuvo diseñado para permitir a los usuarios cargar software de fuentes que no son de confianza, por lo que el modelo de seguridad debe ser "condensado" en el teléfono y no puede depender de procesos de revisión externos. Cada aplicación en Android se asigna a sí misma un UID durante la instalación, y la aplicación del usuario no tiene derechos fuera del acceso a su directorio principal, la capacidad de ejecutarse y escribir en la pantalla [12]. La instalación de cualquier aplicación sin las revisiones que se realizan en la Play Store o en la App Store hace que la posibilidad de infectar el Smartphone sea mayor.

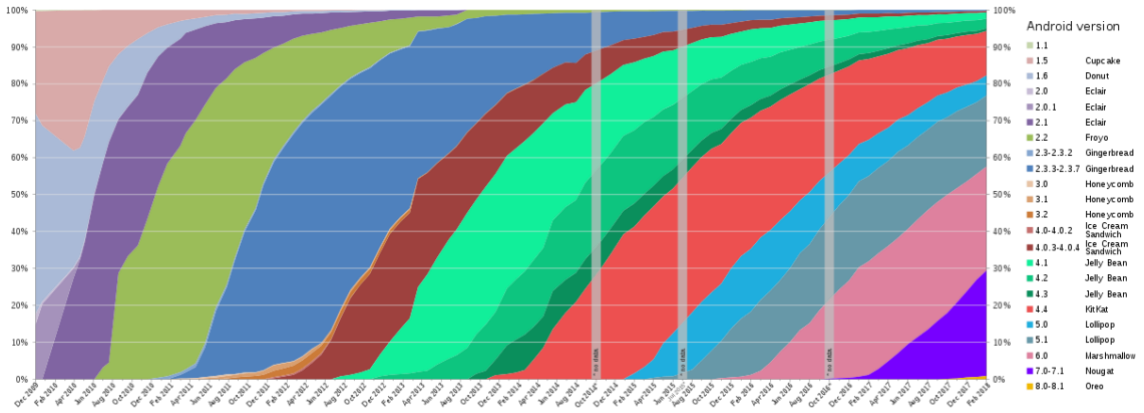

Figura 7: Distribuciones globales de las distintas versiones de Android [13].

3. Entorno de desarrollo

Para la creación de este proyecto se han utilizado diferentes aplicaciones, bibliotecas y *frameworks* indispensables para su realización. En la presente sección se explicará brevemente su utilidad y uso.

1. IDEs y aplicaciones

Durante el desarrollo del proyecto se han usado distintas aplicaciones y entornos de desarrollo integrado (IDE) para el desarrollo de distintas facetas del trabajo. A continuación, se explicarán brevemente los programas empleados y su utilización.

IntelliJ IDEA

Para el desarrollo del servidor que usará la aplicación móvil se ha dispuesto de este entorno de desarrollo integrado (IDE), dadas las facilidades que aporta cuando se emplean los *frameworks* en la implementación, los cuales se exponen más adelante. IntelliJ IDEA tiene dos ediciones:

1. **Community Edition:** gratuita y de código abierto en la cual se basa el IDE que ofrece Google para el desarrollo de las aplicaciones, Android Studio.
2. **Ultimate Edition:** una versión de pago que ofrece soporte para los *f*rameworks usados en el proyecto. Además, tiene integrada una gestión de bases de datos de gran interés que permite construir todo el proyecto usando una única aplicación.

En este caso se ha empleado la versión de pago usando una licencia de estudiante gratuita que ofrecen por ser miembro de una universidad.

Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android, que como se ha dicho anteriormente, se basa en IntelliJ IDEA. Además del editor de códigos y las herramientas para desarrolladores de IntelliJ, Android Studio ofrece aún más funciones que aumentan la productividad durante la compilación de apps para Android [14]. Entre ellas se incluye un gestor de máquinas virtuales que emulan móviles Android para realizar pruebas, un editor de traducciones de *strings*...

Postman

Se trata de una aplicación para el testeo de APIs. Se ha utilizado para probar el correcto funcionamiento de las llamadas al API creadas en el servidor antes de implementar en la aplicación móvil el soporte para ellas. Hablamos de un software con distintas modalidades de pagos, siendo la gratuita más que suficiente para este proyecto. Asimismo, se ha usado para realizar pruebas de estrés con la versión final del servidor por tal de determinar el correcto funcionamiento ante situaciones adversas.

Gitlab

Para el control de versiones del software se ha utilizado GIT ya que sus funcionalidades y su versatilidad lo hacen esencial para cualquier desarrollo software. Todo el código, documentos y utilidades están alojados en un repositorio en Gitlab. Se ha usado ésta por ofrecer un gran abanico de opciones y ser completamente compatible con los IDEs utilizados. Además, cuenta con repositorios privados al contrario de otras opciones más empleadas como Github.

Pencil

Pencil es una aplicación diseñada con el propósito de proporcionar una herramienta de creación de prototipos de GUI gratuita y de código abierto [15]. Su principal uso ha sido la creación de bocetos de las interfaces en Android y la creación de diagramas de flujo y de casos de uso para la aplicación Android y el servidor.

2. Bibliotecas y *frameworks*

Para la realización del proyecto se ha dispuesto de distintas bibliotecas y *frameworks* que ofrecen diferentes soluciones, desde el arranque de las aplicaciones hasta las actualizaciones de la base de datos. En este apartado se muestran los más usados e importantes en el proyecto.

Spring

Spring es un *framework* para el desarrollo de aplicaciones que se encarga de resolver dependencias en tiempo de compilación que permite construir sistemas desacoplados. En los inicios Spring se componía de solamente de las inyecciones de dependencias, pero, hoy en día se compone de distintos proyectos que permiten la construcción de una aplicación de nivel empresarial. De entre los distintos proyectos se han usado los siguientes:

1. **Spring Boot:** se utiliza para configurar la aplicación en la compilación y en la ejecución de la misma. Entre otras cosas, permite incrustar el código en servidores Tomcat o Undertow, resuelve dependencias de bibliotecas propias y de terceros y proporciona funciones para la métrica y controles de estado cuando la aplicación se encuentra en producción [16]. En nuestro caso, se ha utilizado para el desarrollo y el arranque del servidor de la aplicación.
2. **Spring Data:** Spring Data se encarga de proporcionar un modelo de programación basado en el modelo Spring que sea coherente con el acceso a los datos, sin perder los rasgos especiales del almacén de datos subyacente. Facilita el uso de tecnologías de acceso a datos, bases de datos relacionales y no relacionales [17]. Se ha empleado para acceder a la base de datos MySQL integrada en el servidor. Como se puede apreciar en la Figura 8, el uso de Spring Data es muy simple y potente, permitiendo un acceso rápido y seguro a los datos. Éste se realiza simplemente escribiendo una "versión" de SQL en un método Java con los parámetros de los métodos como variables de búsqueda. Tal como se muestra en la Figura 8, se ve el uso de Spring Data en un repositorio que representa el acceso a la tabla de *Parent*. El acceso a los datos se puede realizar como se ha comentado antes con la "versión" de SQL o mediante anotaciones, "@Query" y "@Param", en los métodos. En estas anotaciones se pude escribir directamente el SQL que se quiere ejecutar al llamar al método inmediatamente inferior.

```java
/**
 * Spring Data JPA repository for the Parent entity.
 */
@SuppressWarnings("unused")
@Repository
public interface ParentRepository extends JpaRepository<Parent, Long> {

    @Query(value = "SELECT p FROM Parent p WHERE p.family.id = :familyID")
    List<Parent> findAllByFamilyID(@Param("familyID") Long familyID);

    Parent findOneParentByUserIdEquals(Long userId);

    List<Parent> findAllByFamilyIdEqualsAndActiveStatusEquals(Long family_id, ActiveStatus activeStatus);

    Parent findOneParentByIdEquals(Long parentId);
}
```

Figura 8: Ejemplo de uso de Spring Data JPA

3. **Spring Security:** como la aplicación contiene información relacionada con niños, la seguridad es muy importante. Para garantizar dicha seguridad en el acceso a los datos se ha utilizado Spring Security. Se encarga de proporcionar autenticación y autorización para aplicaciones Java. El punto fuerte del *framework* es la facilidad con la que se puede ampliar y configurar para cumplir con los requisitos de la aplicación en desarrollo [18].
4. **Spring MVC REST:** forma parte del *framework* de Spring, el cual se encarga de facilitar la creación de llamadas REST mediante anotaciones Java. Se ha utilizado para crear las llamadas al servidor que realiza la aplicación Android.

Angular 5

Para la creación de la administración en la web de las entidades del servidor se ha usado Angular 5. Angular es un *framework* que facilita la creación de aplicaciones en la web, combinado plantillas declarativas, inyección de dependencia, herramientas de extremo a extremo y mejores prácticas integradas para resolver los problemas durante el desarrollo [19]. Para ello utiliza TypeScript, una extensión de JavaScript, que agrega tipos, clases y módulos opcionales en él. Cuando se compila TypeScript se traduce a JavaScript legible y basado en los estándares, pudiendo ser usado en cualquier navegador [20].

Jackson Project

Jackson es un conjunto de herramientas de procesamiento de datos para Java (y la plataforma JVM), incluida la biblioteca de generador / analizador sintáctico JSON, la biblioteca de enlace de datos (POJO y JSON) y los módulos de formato de datos adicionales para procesar datos codificados en texto plano [21]. Se ha empleado tanto en el servidor como en la aplicación Android, para pasar objetos Java a través de internet entre ambos.

3. JHipster

JHipster es una parte esencial en la creación del servidor. Se trata de un generador de código para desarrollar aplicaciones web modernas y *Microservices*. Para la creación del código, el generador realiza una serie de preguntas (Figura 9) para configurar la aplicación [22]. Entre ellas podemos destacar:

1. **"¿Qué tipo de aplicación te gustaría crear?":** permite escoger entre una aplicación monolítica, dos tipos diferentes de *Microservices* y un servidor de autenticación UAA.
2. **"¿Qué tipo de autenticación le gustaría usar?":** permite escoger entre JWT, HTTP *Session Authentication* y OAuth.
3. **"¿Qué tipo de base de datos te gustaría usar?":** se puede seleccionar distintos tipos de bases de datos, tanto SQL como no-SQL. Entre las distintas opciones destacamos: MySQL, PostgreSQL, MSSQL, Oracle, MongoDB, Cassandra o Couchbase.
4. **"¿Qué *framework* le gustaría usar para el cliente?":** aquí se puede seleccionar el *framework* que va a ser usado en el cliente. Tienes dos opciones: Angular 5+ o React.

Para este proyecto se ha empleado la siguiente configuración:

1. **Tipo de aplicación:** monolítica.
2. **Tipo de autenticación:** JWT.
3. **Tipo de base de datos:** SQL.
4. **Base de datos en desarrollo y producción:** MySQL.
5. **Framework en el cliente:** Angular 5.
6. **Sistema de construcción de código:** Gradle.

Figura 9: Preguntas del generador de código JHipster

JHipster se encarga de poner a punto un servidor escrito en Java, en base a la configuración previa, usando Spring y los *frameworks* explicados anteriormente. Por otra parte, JHipster prepara la aplicación para entrar en producción, habilitando la posibilidad de desplegar la aplicación en múltiples proveedores, como AWS, Docker o Heroku, y también permite generar un JAR ejecutable o un contenedor Docker para poder ser desplegado en un servidor cualquiera. En este proyecto, el servidor se ha desplegado en Heroku dado que ofrecían una opción gratuita de alojamiento sin necesidad de asignarle un dominio propio. Al servidor se accede tras este enlace: https://gefa.herokuapp.com/ desde un navegador para entrar a la configuración y a la supervisión. La aplicación Android hace uso de ese servidor para realizar peticiones al API y disponer de información que almacena.

JHipster Domain Languaje (JDL)

JHipster proporciona un lenguaje que permite configurar su dominio, describir todas las entidades que confirman una aplicación y sus relaciones en un solo archivo. También se pueden configurar algunos aspectos internos del servidor, como el uso de *Services* mediante interfaz Java o el uso de *Mapstruct* para la conversión entre objetos Java, y de la aplicación web, como paginaciones, scrolls infinitos o búsquedas usando *elasticshearch*.

Una vez creado el archivo de configuración se debe ejecutar un comando del generador JHipster que lee el documento y genera las clases, tablas, interfaces y demás componentes necesarios para satisfacer lo especificado.

JSON Web Token (JWT)

JSON Web Token es el método de autenticación utilizado en el servidor. Se trata de un estándar abierto que define una forma compacta y autónoma para transmitir de un modo seguro información entre dos partes mediante un objeto JSON. Esta información puede ser verificada y es confidencial, pues está firmada digitalmente. Los JWT se pueden firmar usando una clave secreta (con el algoritmo HMAC) o con un par de claves públicas / privadas usando RSA.

El uso más común de JWT es la autenticación. Una vez que el usuario haya iniciado sesión, cada solicitud posterior incluirá el JWT en las cabeceras HTTP, lo que le permitirá acceder a las rutas, servicios y recursos permitidos con ese *token*. Se emplea ampliamente hoy en día, debido a su pequeña sobrecarga y su capacidad de ser integrado fácilmente en diferentes dominios [23].

En la Figura 10 se muestra un ejemplo de un *token* utilizado en el proyecto. El *token* JWT se compone de 3 partes: la primera especifica el algoritmo de cifrado y el tipo de *token*, la segunda parte es la carga del mensaje y la última es la firma de verificación de éste. En la muestra se puede visualizar que se ha usado el algoritmo HS512 y que el nombre del usuario es 'padre1' (con los roles asignados de *Parent* y *User*) y la fecha de expiración del *token*. Estos datos se verificarán en el servidor usando la clave secreta almacenada en él.

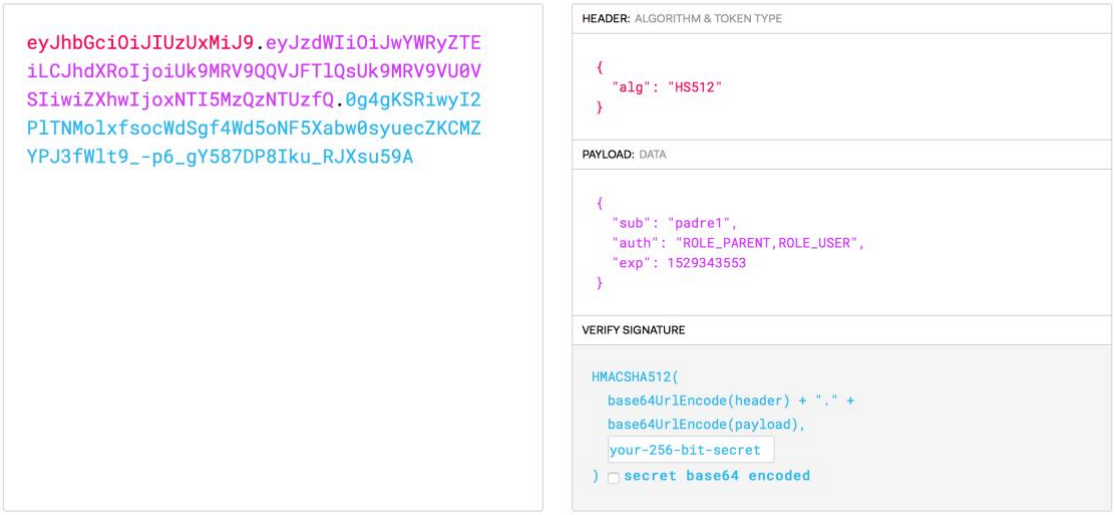

Figura 10: Ejemplo de token JWT

4. Especificación

GeFa se ha desarrollado usando un modelo en capas, la cual, es una técnica de software que divide la aplicación en distintas capas, normalmente 3, que solo se comunican con la capa inmediatamente superior o inferior. De esta forma, el conjunto de la aplicación se encuentra desacoplada siendo posible modificaciones en las capas o sustituirlas sin afectar al resto [24]. El modelo en capas más utilizado es el modelo a tres capas, el mismo que hemos usado para desarrollar este sistema. Este modelo se compone de los siguientes niveles:

1. **Capa de presentación:** es la interfaz que el usuario final ve y usa.
2. **Capa o lógica de negocio:** en ésta se define todas las reglas que se deben cumplir para la correcta ejecución del programa.
3. **Capa de datos:** es la encargada de realizar transacciones con bases de datos y con otros sistemas para obtener o introducir información al sistema.

Dada esta estructura desacoplada, el desarrollo de la aplicación en Android es solamente una capa de presentación, estando la capa de negocio y de datos en el servidor. Con esto, el desarrollo a futuro de una aplicación web o iOS será mucho más rápida y ágil.

Una vez especificado la estructura del sistema, podemos considerar a los siguientes actores involucrados:

1. **El usuario de la aplicación.** Este a su vez de divide en dos: Padres e hijos.
2. **El sistema de gestión.**
3. **El sistema de bases de datos.**

A continuación, se expondrán cada una de las capas que componen el sistema GeFa: Lógica de negocio (Capa de negocio), Capa de presentación y Capa de persistencia (Capa de datos).

1. Lógica de negocio

Para ilustrar el funcionamiento de la lógica de negocio se van a utilizar diagramas de casos de usos y su consecuente tabla explicativa. Dado que la aplicación abarca una cantidad moderada de casos de uso, se ha optado por mostrar solamente los que consideramos más representativos de la aplicación.

1. **Caso de uso 01:** Crear una familia.
2. **Caso de uso 02:** Unirse a una familia.
3. **Caso de uso 03:** Crear una tarea.
4. **Caso de uso 04:** Crear un pago.
5. **Caso de uso 05:** Marcar una tarea como pendiente de confirmación.
6. **Caso de uso 06:** Marcar una tarea como completada.
7. **Caso de uso 07:** Activar un usuario.

En la Figura 11 se muestran los casos de uso de la aplicación GeFa que realizan los actores usuarios de la aplicación.

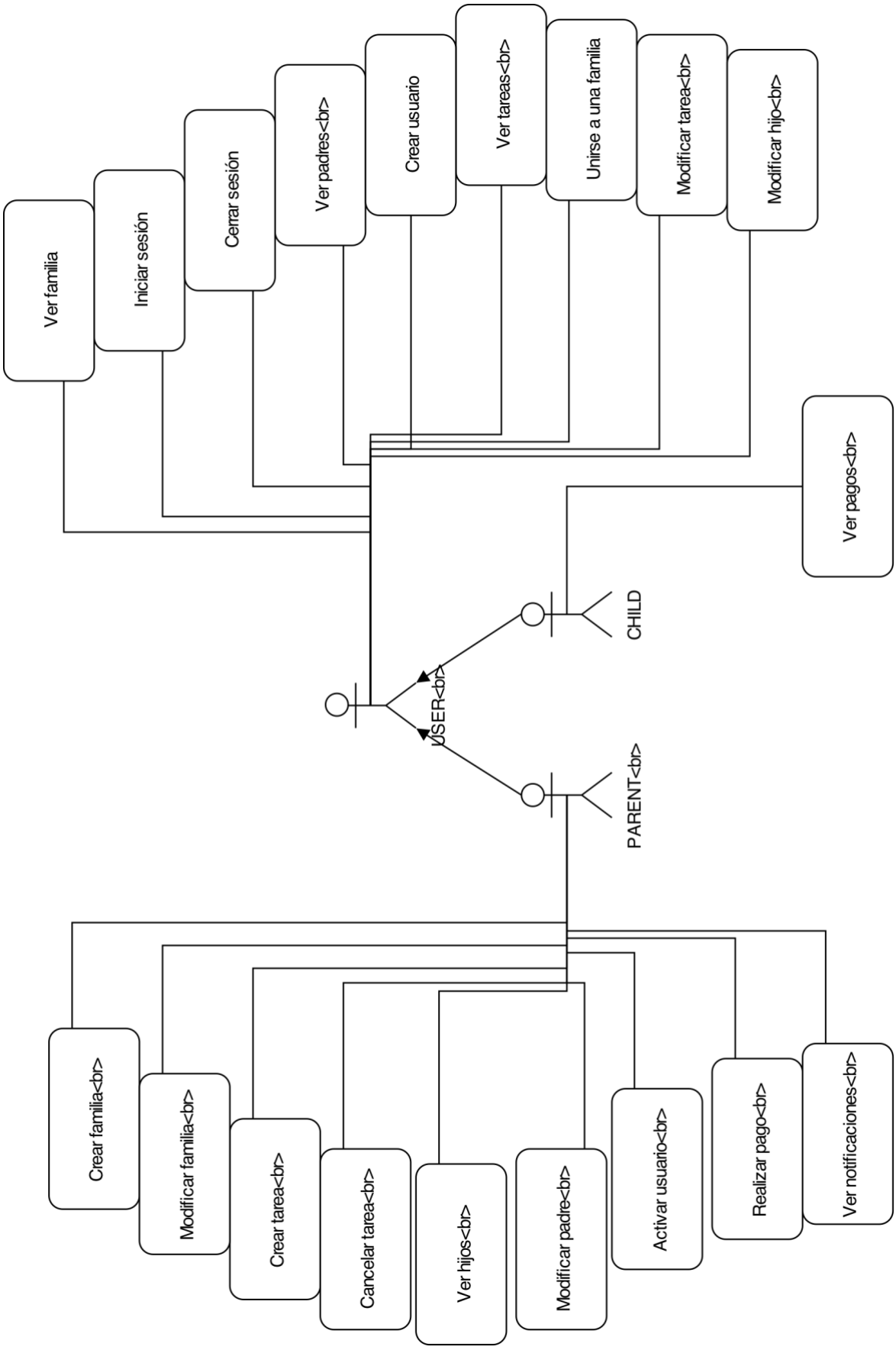

Figura 11: Casos de uso de la aplicación GeFa.

Crear una familia

Tabla 2: Caso de uso 1, Crear una familia.

CU-01	**Crear una Familia**	
Descripción	Este caso de uso corresponde a la acción de crear una familia, iniciada por un Padre, Madre o Tutor. Esta acción se lleva a cabo después de que el usuario se registre en el servicio, dando la opción de crear una familia o unirse a una existente.	
Precondición	Para poder crear una familia el usuario tiene que estar registrado y autenticado como Padre, Madre o Tutor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa sobre el botón de crear una familia nueva.
	2	El sistema de gestión solicita los datos de la nueva familia: nombre, día de la semana que se realizan los pagos y la frecuencia de éstos. En el apartado de la capa de persistencia se detallan cada uno de los campos.
	3	El usuario pulsa el botón de guardar una familia, se envían los datos al sistema de gestión y se guarda en la base de datos.
Resultado	Como resultado de este caso de uso se crea una familia con un único miembro, el creador, activado. Esto quiere decir que es el único autorizado para activar nuevos usuarios, para crear tareas y realizar pagos.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	Si el campo de nombre no se ha rellenado se le notifica al usuario, los otros campos adquieren un valor por defecto.

Unirse a una familia

Tabla 3: Caso de uso 2, Unirse a una familia.

CU-02	**Unirse a una familia**	
Descripción	Este caso de uso corresponde a la acción de unirse a una familia, iniciada por un usuario, ya sea de tipo *Parent* o *Child*. Esta acción se lleva a cabo después de que el usuario se registre en el servicio, dando la opción de crear una familia o unirse a una existente.	
Precondición	Para poder unirse a una familia el usuario tiene que estar registrado y autenticado.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa sobre el botón de crear una familia nueva.
	2	El sistema de gestión solicita el identificador único de la familia. En la propia ventana se le indica como puede obtener el identificador.
	3	El usuario pulsa el botón de unirse a una familia, se envían los datos al sistema de gestión y se guarda en la base de datos.
Resultado	Como resultado de este caso de uso, el usuario queda unido a una familia, pero, en estado pendiente de aprobación. Hasta que un usuario autorizado no active a éste, no podrá interactuar con la familia (ni ver ni crear datos en relación).	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	Si el campo del identificador no se ha rellenado se le notifica al usuario y se paraliza la activación

Crear una tarea

Tabla 4: Caso de uso 3, Crear tarea.

CU-03	**Crear una Tarea**	
Descripción	Este caso de uso corresponde a la acción de crear una tarea, iniciada por un usuario de tipo *Parent*.	
Precondición	Para poder crear una tarea el usuario ha de ser de tipo *Parent*, estar unido y activado en la familia, y que exista, al menos, un hijo también activado.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario pulsa sobre el botón de crear una nueva tarea para un hijo determinado.
	2	El sistema de gestión solicita los datos de la nueva tarea: nombre, descripción, icono, pago, penalización, si es obligatoria u opcional, entre otras. En el apartado de la capa de persistencia se detallan cada uno de los campos.
	3	El usuario pulsa el botón de guardar una nueva tarea, se envían los datos al sistema de gestión y se guarda en la base de datos.
Resultado	Como resultado de este caso de uso, se ha creado una tarea en estado *PENDING*, que, tanto el padre como el hijo, al que ha asignado la tarea, pueden ver en su lista de tareas. En el caso del padre para poder comprobar la evolución de las tareas del hijo y, en el caso del hijo, para ver las tareas pendientes.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	En la creación de la tarea existen algunos campos con la obligación de rellenarlos. En caso de no rellenarse, se le notificará al usuario para que los complete.

Crear un pago

Tabla 5: Caso de uso 4, Crear un pago.

CU-04	Crear un Pago	
Descripción	Este caso de uso corresponde a la acción de crear un pago, iniciada por un usuario de tipo *Parent*.	
Precondición	Para poder crear un pago el usuario ha de ser de tipo *Parent*, estar unido y activado en la familia, y que exista, al menos, un hijo también activado y con tareas completadas.	
Secuencia normal	**Paso**	**Acción**
	1	En el apartado para un hijo determinado, el padre pulsa sobre el botón de crear un pago nuevo.
	2	El sistema de gestión le muestra las tareas realizadas por el hijo y la cuantía final de la paga.
	3	El padre selecciona las tareas por las que quiere que se realice el pago. Por defecto el sistema le marca todas las tareas pendientes de pago. Estas tareas son las que tienen el estado de *COMPLETED*. En el apartado de la capa de persistencia se detallan cada uno de los campos y los estados.
	4	El usuario pulsa el botón de crear un nuevo pago, se envían los datos al sistema de gestión y se guarda en la base de datos.
Resultado	Como resultado de este caso de uso, desaparecen de las listas de tareas del padre y del hijo aquellas por las que se ha efectuado el pago. Además, el hijo/a tendrá disponible el detalle del mismo en su correspondiente apartado en la aplicación.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	Si el hijo/a no tiene ninguna tarea completada, se cancela la creación del pago.
	2	Si el padre deselecciona todas las tareas completadas del pago, se bloquea la realización y se le indica que no puede realizarse un pago sin tareas.

Marcar una tarea como pendiente de confirmación

Tabla 6: Caso de uso 5, Marcar una tarea como pendiente de confirmación.

CU-05	**Marcar una tarea como pendiente de confirmación**	
Descripción	Este caso de uso corresponde a la acción de marcar una tarea como pendiente de confirmación, iniciada por un usuario de tipo *Child*.	
Precondición	Para poder cambiar el estado de una tarea ha de existir una tarea que tenga el estado *PENDING*.	
Secuencia normal	**Paso**	**Acción**
	1	En la lista de tareas, un hijo/a pulsa sobre el botón de tarea completada.
	2	El sistema de gestión se encarga de cambiar el estado de la tarea en la base de datos.
Resultado	Como resultado de este caso de uso, la tarea en cuestión pasa a estar en el estado de pendiente de confirmación que será revisado por un padre.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	-

Marcar una tarea como completada

Tabla 7: Caso de uso 6, Marcar una tarea como completada.

CU-06	**Marcar una tarea como completada**	
Descripción	Este caso de uso corresponde a la acción de marcar una tarea como completada, iniciada por un usuario de tipo *Parent*. Esta acción hará que una tarea pase a estado *COMPLETED* y esté disponible para formar parte de un pago.	
Precondición	Para poder cambiar el estado de una tarea ha de existir una tarea que tenga el estado de pendiente de confirmación.	
Secuencia normal	**Paso**	**Acción y Consecuencia**
	1	En la lista de tareas de un hijo/a, el padre, pulsa sobre el botón de tarea completada.
	2	El sistema de gestión se encarga de cambiar el estado de la tarea en la base de datos.
Resultado	Como resultado de este caso de uso, la tarea en cuestión pasa a estar en el estado *COMPLETED* para que esté disponible para formar parte de un pago futuro.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	-

Activar un usuario

Tabla 8: Caso de uso 7, Activar un usuario.

CU-07	Activar un usuario	
Descripción	Este caso de uso corresponde a la acción de activar un usuario, ya sea de tipo *Parent* o *Child*.	
Precondición	Para poder activar a un usuario, el usuario actual ha de ser de tipo *Parent* y estar unido y activado en la familia.	
Secuencia normal	**Paso**	**Acción**
	1	En la pantalla principal el padre, madre o tutor verá que hay una notificación en la cual dirá que existen usuarios que quieren unirse a la familia. El padre, madre o tutor pulsará el botón para ir a la pantalla de notificaciones de la familia.
	2	El padre, madre o tutor ahora verá un listado de los usuarios que quieren unirse. Pulsará sobre aquel que quiera autorizar.
	3	El sistema de gestión se encarga de cambiar el estado de activación del usuario seleccionado en la base de datos.
Resultado	Como resultado de este caso de uso, el usuario seleccionado pasará a estar activado. Esto quiere decir que a partir de ese momento el usuario puede hacer *login* dentro de la familia y tendrá distintas acciones y accesos dependiendo si es de tipo *Parent* o *Child*.	
Excepciones	**Nº**	**Acción y Consecuencia**
	1	-

2. Capa de presentación

Como se ha explicado anteriormente, la forma desacoplada de la lógica de negocio, la capa de persistencia y la capa de presentación hacen que se puedan realizar adaptaciones a otras plataformas solamente desarrollando la capa de presentación. Esto nos permitirá equipararnos a otras soluciones actuales, mostradas en el estado del arte, creando una aplicación web y una aplicación en la plataforma iOS en un futuro. Para este proyecto se ha marcado como prioritario desarrollar una aplicación en Android. Para ello se expondrá el diseño de la capa de presentación de la aplicación GeFa en Android utilizando bocetos, asociados a los casos de uso expuestos en el apartado anterior, que idean el funcionamiento de la interfaz final de la aplicación.

Crear una familia

En la Figura 12 se muestra un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 1 (pág. 31) para la creación de una familia. La interfaz constará de tres entradas de datos y un botón de acción. En lo referente a las entradas de datos, habrá un campo de texto para introducir el nombre de la familia y dos *spinners* (*Dropdown* o *ChoiceBox*) para seleccionar el día de paga y el periodo de pagos, que son enumerados, en la familia. Por último, habrá un botón que accionará la creación de la familia enviando los datos al servidor, el cual, almacenará la nueva familia en la base de datos.

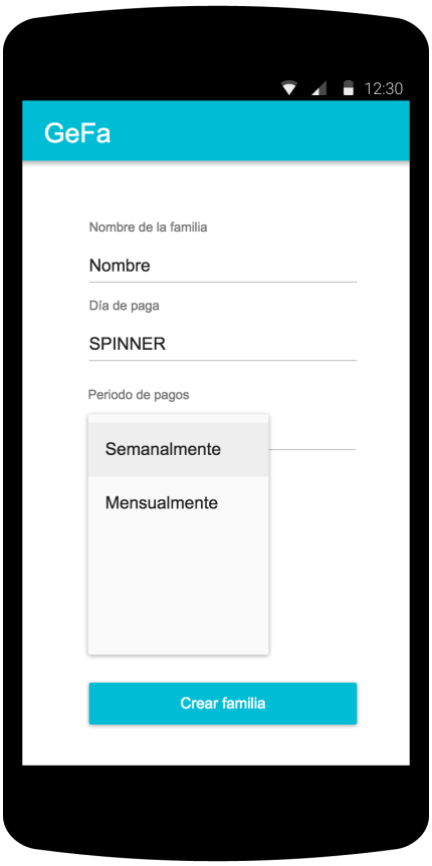

Figura 12: Boceto para crear una familia.

Unirse a una familia

En la Figura 13 se expone el boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 2 (pág. 32) para unirse a una familia existente. La interfaz constará de un texto explicativo, una entrada de datos y un botón de acción. El texto explicativo expondrá la necesidad de una familia para la asignación de la familia y revelará la forma de obtener el identificador de la misma dentro de la aplicación de un usuario ya activado. En lo referente a la entrada de datos, habrá un campo de texto para introducir el identificador. Finalmente, habrá un botón que accionará la unión a una familia enviando los datos al servidor, el cual almacenará los cambios en el usuario para unirse a ella en la base de datos. Una vez confirmados los cambios en la base de datos, se pasará a una nueva ventana de espera para el usuario que acaba de unirse, con un botón para intentar hacer *login* en la aplicación.

Figura 13: Boceto para unirse a una familia.

Crear una tarea

En la Figura 14 se muestra un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 3 (pág. 33) para la creación de una tarea. La interfaz constará de varias entradas de datos y distintos botones de acción. En lo referente a las entradas de datos, habrán los campos de texto necesarios para introducir la información referente a la tarea (nombre, notas, pagos, penalizaciones...) y un selector exclusivo que distinguirá los dos tipos de tarea posibles (Obligatoria u Opcional). También aparecerán dos botones que abrirán unas ventanas flotantes modales (*dialogs*) para la selección de un icono y una fecha límite para la tarea.
Por último, habrá un botón que accionará la creación de la tarea enviando los datos al servidor, el cual almacenará la nueva tarea en la base de datos pudiendo ser ésta ya mostrada en las listas del hijo/hija a la que ha sido asignada.

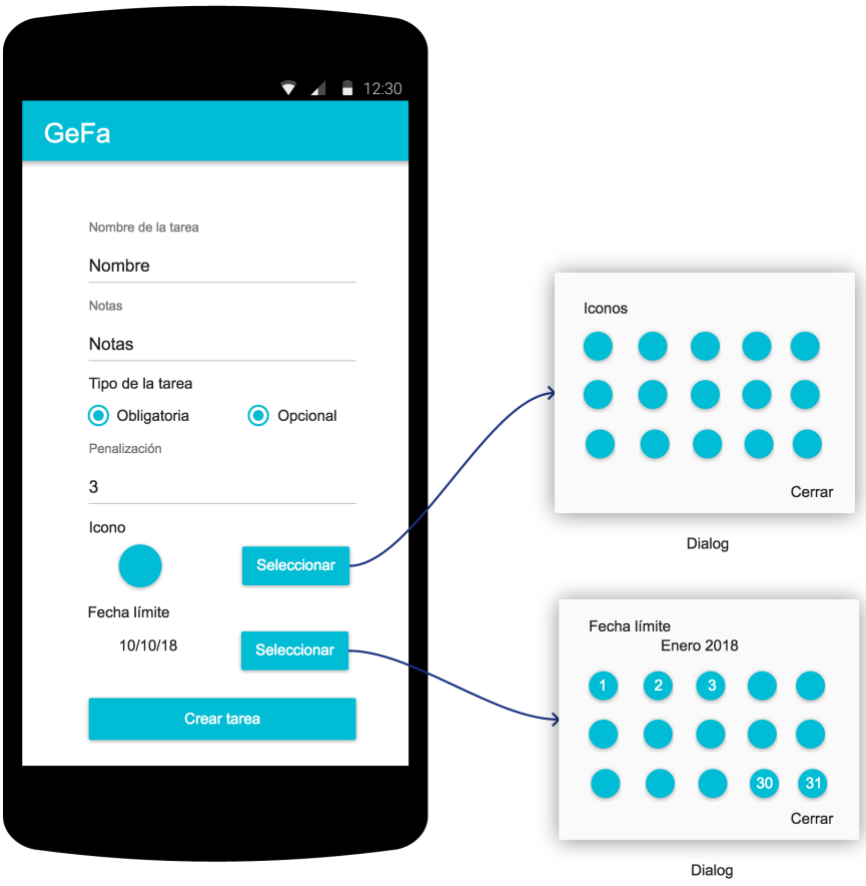

Figura 14: Boceto para crear una tarea.

Crear un pago

En la Figura 15 se expone un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 4 (pág. 34) para la creación de un pago. La interfaz constará de una lista de tareas, de un resumen de las cantidades de pago y de un botón de confirmación. En lo referente a la lista de tareas, aparecerán todas las que se han completado por parte del hijo, con una caja de verificación para poder seleccionar si formará parte del pago. También aparecerá un resumen del pago final que se dividirá en la paga base y los extras obtenidos de las tareas opcionales.
Finalmente, habrá un botón que accionará la creación del pago enviando los datos al servidor, el cual almacenará el nuevo pago en la base de datos pudiendo ser éste ya visto en las listas de pagos del hijo/hija.

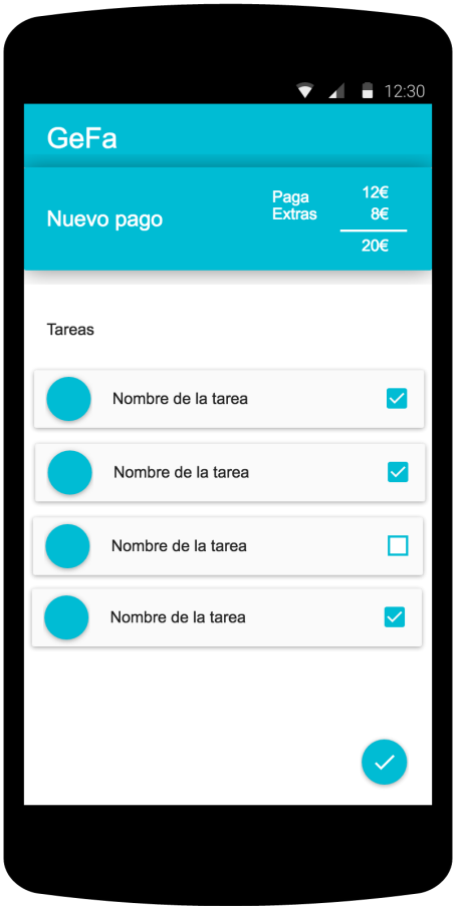

Figura 15: Boceto para crear un pago.

Marcar una tarea como pendiente de confirmación

En la Figura 16 se expone un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 5 (pág. 35) para marcar una tarea como pendiente de confirmación por parte de un padre, madre o tutor. La interfaz parte del listado de tareas de un hijo/hija. El cual estará formado por una cabecera que mostrará el porcentaje de progreso actual divido en tareas completadas, pendientes de confirmación y pendientes de completar. El cuerpo principal de la interfaz lo forma una lista de tareas dividida por secciones según su estado. Cada una de ellas en la lista estará compuesta por el icono, el nombre de la tarea y de una caja de verificación para accionar el cambio de estado de la misma. Cuando sea accionado, se mostrará un *dialog* informando de que se está actualizando la tarea. Cuando termine se ocultará el *dialog* y se actualizará la lista, mostrándola ahora la tarea pendiente de confirmación en su sección correspondiente.
Como resultado de este caso de uso, se activará la posibilidad de realizar el caso de uso 6 (Tabla 7)sobre esta tarea.

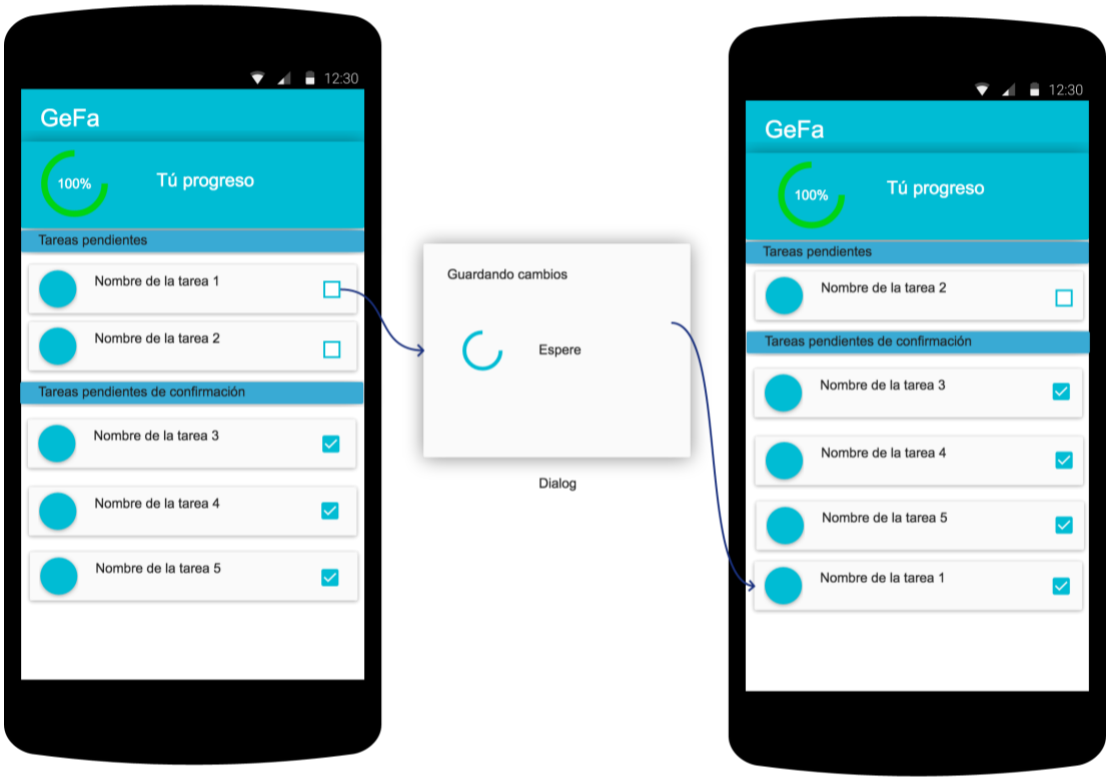

Figura 16: Boceto para marcar una tarea como pendiente de confirmación.

Marcar una tarea como completada

En la Figura 17 se expone un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 6 (pág. 36) para marcar una tarea como completada. La interfaz parte del listado de tareas de un hijo/a que puede visualizar el padre, madre o tutor. El cual, estará formado por una cabecera que mostrará la imagen de usuario del hijo/a y de un título indicado quién es (en los casos que no se agregue una imagen de perfil es esencial para diferenciar las ventanas). El cuerpo principal de la interfaz lo forma una lista de tareas divididas por secciones según su estado. Cada una en la lista estará compuesta por el icono, el nombre de la tarea y de un botón que accionará un menú con opciones sobre la misma. Cuando sea accionado, se mostrará un *dialog* en la que se visualizarán tres botones de acción distintos y otro para cancelar la acción. Estos botones servirán para cancelar, editar o marcar como completada una tarea. Cuando se accione el botón de marcar como completada, se ocultará el *dialog* y se actualizará la lista mostrándola ahora completada en su sección correspondiente en la lista.
Como resultado de este caso de uso, se activará la posibilidad de realizar el caso de uso 4 (Tabla 5) sobre esta tarea.

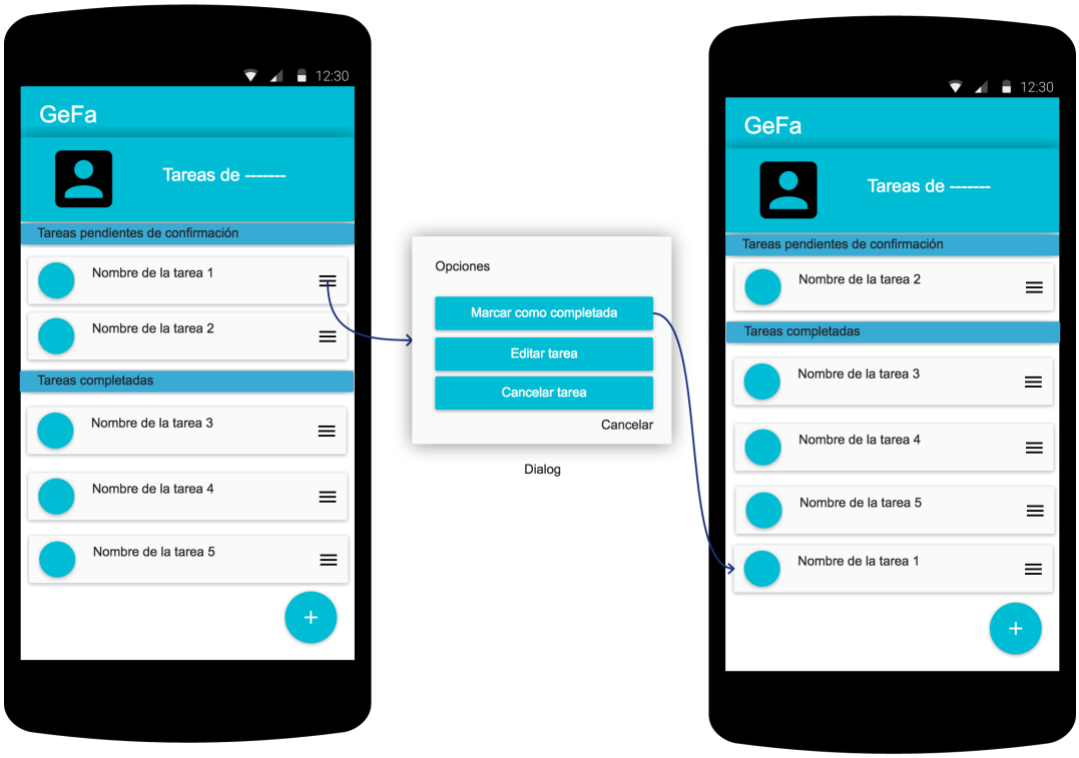

Figura 17: Boceto para marcar una tarea como completada.

Activar un usuario

En la Figura 18 se expone un boceto de la interfaz en la aplicación Android para llevar a cabo el Caso de uso 7 (pág. 37) para activar un usuario. La interfaz parte del detalle de familia de un padre, madre o tutor, la cual estará formado por una cabecera que mostrará la información de ésta (nombre, día de paga y periodo de pagos). El cuerpo principal de la interfaz lo forma una lista de notificaciones. Cada una de las notificaciones en la lista estará compuesta por el nombre descriptivo de la alerta y de un botón que accionará un *dialog*. Cuando sea accionado, se mostrará un *dialog* en la que se mostrará la acción que se va a realizar y las consecuencias de ésta. En este caso, la consecuencia será que el usuario activado podrá editar los datos relacionados con la familia. Cuando se accione el botón activar, se ocultará el *dialog* y se actualizará la lista mostrando las notificaciones restantes.
Como resultado de este caso de uso, se activará el usuario seleccionado pudiendo realizar ya distintos casos de uso sobre la familia.

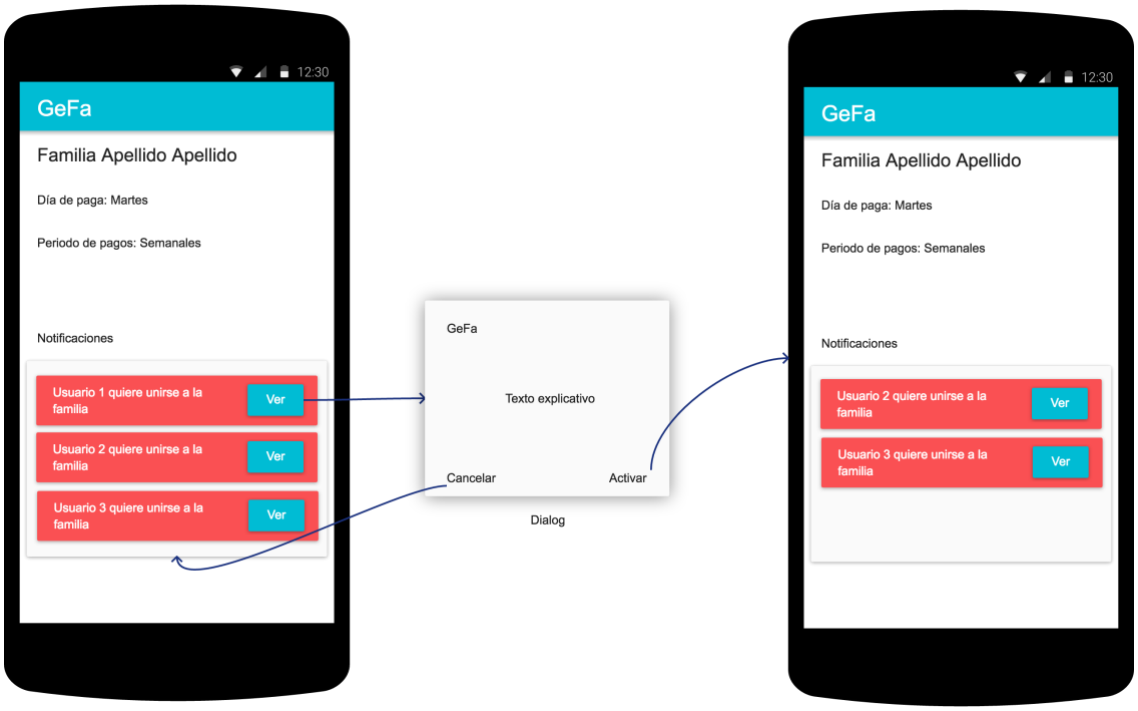

Figura 18: Boceto para activar a un usuario.

3. Capa de persistencia

En lo que respecta a la capa de persistencia, en este proyecto se ha utilizado un sistema de base de datos relacional SQL usando como sistema de gestión MySQL. Para la creación de la base de datos se ha usado el lenguaje JDL (pág. 26), proporcionado por JHipster, dado que permite crear con gran facilidad entidades, sus restricciones y sus relaciones, además de poder configurar su comportamiento y gestión en el servidor.

Como se aprecia en la Figura 19, la creación de una entidad se realiza de manera sencilla indicando el nombre y los campos que tiene con sus tipos de datos y sus restricciones (obligatoriedad, tamaño mínimo, tamaño máximo...). También permite la creación de enumerados que se incluirán como *enums* Java en el código del servidor, y que en la base de datos se guardarán como *Strings*.

Las entidades se convertirán en Objetos Java y, dependiendo de las relaciones entre entidades, el objeto puede tener una referencia a otro o tener una lista de éstos. Las restricciones de las variables se comprobarán a la hora de la inserción en la base de datos.

En la Figura 20 se muestra el resultado final de las entidades con sus relaciones y enumerados creados con JDL en un diagrama. Las entidades se identifican con un único título que indica su nombre. Para diferenciarlos de los enumerados, estos tienen un segundo título que indica que es un *enum* (<<enumeration>>). La entidad *User* que no tiene campos, pero sí relaciones, la crea automáticamente JHipster al cargar el modelo. Esta entidad representa a un usuario registrado que se comentará más adelante es este documento.

```
entity Task{
    name String required,
    notes String,
    icon String,
    taskPayment Float min(0),
    taskPenalty Float min(0),
    requiredType RequiredType,
    creationDate ZonedDateTime,
    status TaskStatus,
    completedDate ZonedDateTime,
    deadlineDate ZonedDateTime,
    cancelledDate ZonedDateTime,
    last_modified_date ZonedDateTime
    paymentStatus PaymentStatus,
    taskRepeat TaskRepeat
}

enum TaskRepeat {
    ONCE, DAILY, WEEKLY, MONTHLY
}

enum TaskStatus{
    COMPLETED, NOT_COMPLETED,
    REQUESTCONFIRMATION,
    CANCELLED, PENDING
}

enum RequiredType{
    REQUIRED, OPTIONAL
}

enum PaymentStatus{
    PAID, PENDING, TASK_NOT_COMPLETED
}
```

Figura 19: Ejemplo de creación de una entidad en JDL.

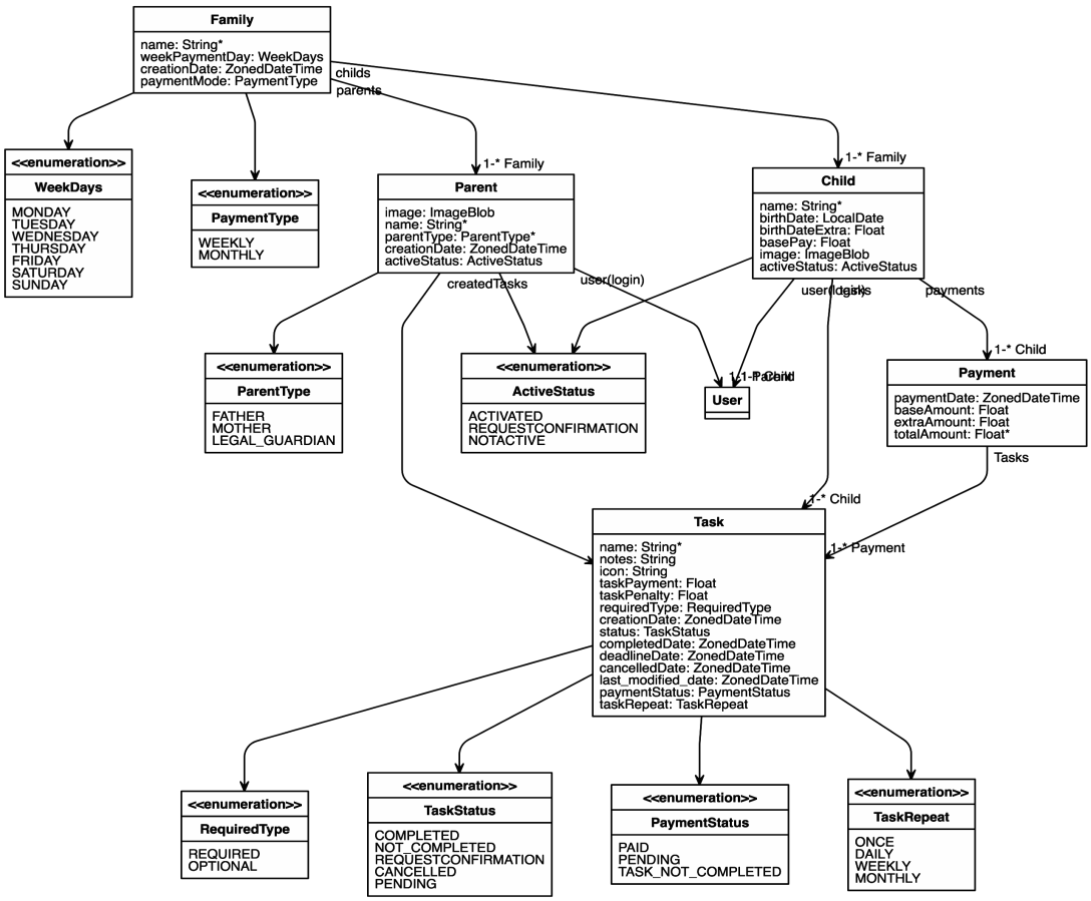

Figura 20: Modelo de entidades final creado con el lenguaje JDL.

Cuando se introduce el modelo de entidades en el generador de código JHipster se encarga de generar el código necesario para las siguientes funciones:

1. Creación de los objetos en Java y DTOs para trasmitirlos usando JSON.
2. Implementación de *Mappers,* usando Mapstruct, para hacer la conversión entre objetos Java y DTOs.
3. Implementación de llamadas y servicios CRUD sobre la base de datos, así como el acceso mediante un API usando Spring.
4. Creación de una web de edición CRUD para las entidades.
5. Creación de tablas extra en la base de datos para gestionar el acceso de usuarios y sus roles, usando Spring Security (pág. 24).
6. Creación de tablas extra para auditorias y para la gestión de actualizaciones de bases de datos, usando Liquibase.

Como resultado final se han creado las tablas y las relaciones entre ellas como se muestra en la Figura 21, que son las que realmente están generadas en la base de datos SQL.

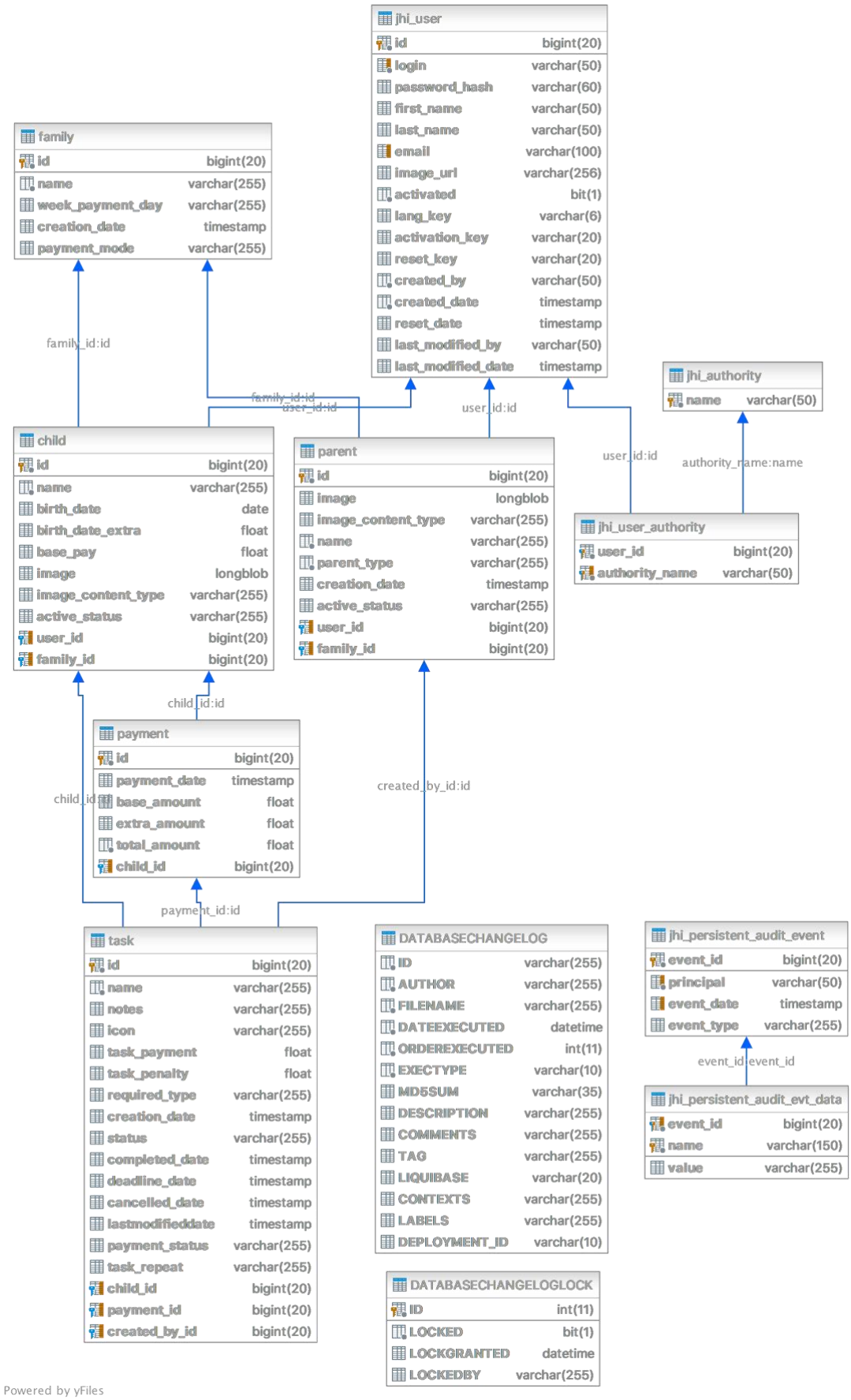

Figura 21: Diseño final de la base de datos.

A continuación, se describirán las tablas que conforman la base de datos, la función de los principales campos que almacenan y sus relaciones.

Familia

Esta tabla representa a una familia. En la Tabla 9 se muestran los campos que la forman. Además, forma la relación que une a los padres con los hijos, teniendo ésta una relación uno a muchos con *Child* y *Parent*.

Tabla 9: Descripción de la tabla Family.

Family	
Id	Identificador único de la familia.
Name	Nombre que identifica a la familia. Usado para mostrar un texto en forma: “Perteneces a la familia x”
Week payment day	Se trata de un campo de tipo enumerado que representa del día de la semana en la que se realizan los pagos. Usado para poder enviar notificaciones en futuras versiones.
Creation date	Fecha de creación de la familia, solamente informativo.
Payment mode	Campo de tipo enumerado que representa en que periodos se realizan la paga: semanalmente o mensualmente.

Padres

Esta tabla representa a un padre, una madre o un tutor legal. En la Tabla 10 se describen los principales campos que la constituyen. También tiene una relación uno a uno con JHI_USER, el cual forma el usuario con el acceso a la plataforma y se deja a la tabla *Parent* como una especificación de un usuario general.

Tabla 10: Descripción de la tabla Parent.

Parent	
Id	Identificador único del padre, madre o tutor.
Name	Nombre del padre, madre o tutor. Se utiliza para mostrarlo en distintas partes de la aplicación.
Image	Imagen de perfil del padre, madre o tutor. Se utiliza tanto en la versión de la aplicación para los padres como la de los hijos.
Parent type	Campo de tipo enumerado que identifica si eres padre, madre o tutor, De momento el campo es solamente informativo.
Creation date	Fecha de creación del padre, madre o tutor, únicamente informativo.
Active status	Campo de tipo enumerado que se utiliza para garantizar el acceso a una familia y los datos relacionados con ella.

Hijos

Esta tabla representa a un/a hijo/a. En la Tabla 11 se describen los principales campos que la constituyen. Además, ésta tiene unas relaciones con otras tablas:

1. Relación uno a uno con JHI_USER, el cual forma el usuario con el acceso a la plataforma y se deja a la tabla *Child* como una especificación de un usuario general.
2. Relación uno a muchos con *Task* que conformas un conjunto de tareas que tiene un usuario de tipo *Child*.
3. Relación uno a muchos con *Payments* que relacionan los pagos con un hijo/a.

Tabla 11: Descripción de la tabla Child.

Child	
Id	Identificador único del hijo/a.
Name	Nombre del hijo/a. Se utiliza para mostrarlo en distintas partes de la aplicación.
Birthday	Fecha de cumpleaños del hijo/a. Se utiliza para poder pagar un extra por ser el cumpleaños. Este aspecto es opcional por parte de los padres.
Birthday extra	En caso de querer incluir el extra por ser el cumpleaños del hijo/a se asigna una cantidad que se almacena en este campo.
Base pay	Paga base del hijo/a.
Image	Imagen de perfil del hijo/a. Se utiliza tanto en la versión de la aplicación para los padres como la de los hijos.
Active status	Campo de tipo enumerado que se utiliza para garantizar el acceso a una familia y los datos relacionados con ella.

Tareas

Esta tabla representa una tarea asignada a un/a hijo/a. En la Tabla 12 se describen los principales campos que la forman. Asimismo, esta tabla tiene unas relaciones con otras:

1. Relación con la tabla *Parent* que representa el padre, madre o tutor que creó la tarea.
2. Relación con la tabla *Child* que representa el/la hijo/a con la que está asignada la tarea.
3. Relación con la tabla *Payment* que representa si esta tarea forma parte de un pago realizado.

Tabla 12: Descripción de la tabla Task.

Task	
Id	Identificador único de la tarea.
Name	Nombre corto de la tarea. Se usa en los listados de éstas como titulo principal.
Notes	En el caso de que se necesite más explicación para una tarea se pueden añadir notas extra que se pueden visualizar al tocar alguna de ellas en la aplicación.
Icon	Nombre de un icono asociado a una tarea. Útil para poder diferenciarlas en las listas.
Required type	Campo de tipo enumerado que indica si una tarea es obligatoria (tiene que ser cumplida para obtener la paga) u opcional.
Task penalty	En el caso de ser una tarea obligatoria se puede asignar una cantidad a descontar de la paga por el no cumplimiento de ésta.
Task payment	En el caso de ser una tarea opcional se tiene que indicar el pago extra que se asigna a la paga.
Creation date	Fecha de creación de la tarea.
Status	Estado de cumplimiento de la tarea. Se trata de un campo de tipo enumerado que indica si la tarea ha sido completada, si esta pendiente de confirmación, si no se ha completado o si ha sido cancelada.
Completed date	Fecha de cumplimiento de la tarea. Depende del estado de la misma.
Deadline date	Fecha de límite del cumplimiento de la tarea. Depende del estado de la misma.
Cancelled date	Fecha de cancelación de la tarea. Depende del estado de la misma.
Last modification date	Fecha de última modificación.
Payment status	Campo de tipo enumerado que indica si esta tarea forma parte de un pago. Usado al realizar un pago de forma que sea más sencillo el acceso a las tareas pendientes de uno.
Task repeat	Campo de tipo enumerado que indica en qué periodos se repite la tares. Puede ser una única vez, cada día, cada semana o cada mes. Se usará en futuras actualizaciones.

Pagos

Esta tabla representa un pago realizado a un/a hijo/a mediante una relación con la tabla *Child*. En la Tabla 13 se describen los principales campos que la constituyen. Además de la relación con el/la hijo/a, también tiene una relación con la tabla *Task* que representan el conjunto de tareas que forman el pago.

Tabla 13: Descripción de la tabla Payment.

Payment	
Id	Identificador único del pago.
Payment date	Fecha de realización del pago.
Base amount	Paga base pagada. Dado que la paga base del/la hijo/a puede cambiar, aquí se almacena la que en el momento del pago tenía el/la hijo/a.
Extra amount	Paga extra por las tareas opcionales.
Total amount	Suma de la paga extra y la paga base.

Usuarios y seguridad

Para la seguridad y el control de acceso se ha usado Spring Security. Éste, necesita de un usuario con roles para garantizar el acceso a los servicios y para controlar a qué tiene acceso cada usuario. Las tablas JHI_USER y JHI_Authority, cuyos campos están descritos en la Tabla 14 y en la Tabla 15 respectivamente, son las que guardan la información de los usuarios básicos, que luego se especifican como padres o hijos, así como los roles que tienen asignados con una relación muchos a muchos con la tabla JHI_Authorty. Esta tabla cuenta con unas filas de la tabla estáticas en las que se guardan los distintos roles que hay en la plataforma. En nuestro caso existen los distintos roles:

1. **ROLE_ADMIN:** rol utilizado para el administrador del sistema, cuenta con privilegios para administrar usuarios, entre otros.
2. **ROLE_USER:** cualquier usuario de la plataforma tiene este rol asignado, incluido el administrador. Es un rol básico que autoriza el acceso al API del sistema.
3. **ROLE_PARENT:** este rol identifica, de entre todos los usuarios, los que son padres, madres o tutores, pudiendo acceder a determinados datos, modificaciones o funciones que un hijo no puede acceder.
4. **ROLE_CHILD:** este rol identifica, de entre todos los usuarios, aquellos que son hijos o hijas, restringiendo así el acceso a determinados datos o funciones que sólo pueden realizar los padres. Un ejemplo sería la creación de tareas o de pagos.

Tabla 14: Descripción de la tabla JHI_USER.

JHI_USER	
Id	Identificador único del usuario.
Login	*Login* (nombre de usuario) para acceder a la plataforma. Como en otros servicios web se usa el correo electrónico.
Password hash	Aquí se guarda el hash de la contraseña del usuario.
First name	Nombre del usuario.
Last name	Apellidos del usuario.
Email	Correo electrónico del usuario.
Activated	Estado de activación de usuario al registrarse. Para simplificar el registro en el sistema, dado que se van a registrar niños, se ha suprimido la función de activación por correo electrónico.
Lang key	Código que identifica el lenguaje preferido por el usuario. Usado en la versión web.
Activation / reset key	Usado para activar al usuario por una clave por correo electrónico y para resetear al usuario.
Created by	Usuario que creo al usuario (administrador, anónimo...).
Created date	Fecha de la creación del usuario.
Last modification date	Última actualización del usuario.

Tabla 15: Descripción de la tabla JHI_Authority.

JHI_Authority	
Name	Nombre del rol de autorización.

Actualización de la base de datos y auditorias.

Además de estas tablas que conforman la parte del contenido principal de la gestión de tareas de familias con hijos, existen una serie de tablas que se usan para las actualizaciones de la base de datos y para almacenar datos para auditorias:

1. **DATABASECHANGELOG y DATACHANGELOGLOCK:** estas tablas se encargan de mantener una consistencia ante los cambios en la base de datos. Almacenan los cambios realizados sobre la misma, como añadir columnas o

tablas, así como *scripts* de actualización de columnas. Usando el estado guardado en DATACHANGELOGLOCK se comprueba durante el arranque si se ha realizado algún cambio en la estructura, de ser así, se realizarán los cambios en la base de datos y se iniciará el servidor. Estas actualizaciones las realiza usando Liquibase.

2. **Jhi_persistent_audit_event y jhi_perisistent_audit_evt_data:** estas tablas almacenan los datos relacionados con el acceso a la plataforma. La primera guarda todos los accesos (*login*) al sistema, la segunda guarda exclusivamente los intentos fallidos de *login* y las excepciones que se realizan al acceder a datos no autorizados.

5. Implementación

En esta sección nos centraremos en presentar los aspectos más relevantes del proceso de implementación en diferentes aspectos del proyecto. Comenzaremos explicando cómo están estructurados en la red los servicios de GeFa y a continuación se expondrán aspectos y decisiones importantes que se han tomado en el desarrollo de la aplicación en Android.

1. Estructura de red

GeFa está desplegada siguiendo la estructura de red que se muestra en la Figura 22. En ella podemos apreciar una separación entre las interfaces de acceso, el servidor y la base de datos; usándose un modelo cliente-servidor pensando a futuro. Así será posible acceder a la información de manera transparente estando en entornos multiplataforma. Por lo tanto, se divide en 3 partes:

1. **Aplicación Android y gestión web:** ambas se comunican con el servidor mediante internet accediendo a los datos a través del API.
2. **Servidor:** el servidor está alojado usando la plataforma Heroku, ya que éste ofrece una versión gratuita de sus servicios. A su vez, Heroku usa los servicios de Amazon Web Services (AWS) para proporcionar los suyos.
3. **Base de datos:** la base de datos también está alojada con Heroku usando de igual forma los servicios de AWS. Heroku ofrece varios servicios que se pueden integrar para controlarla. En este proyecto se ha usado JawsDB con MySQL que ofrece bases de datos como servicio, lo que ha permitido tenerla desacoplada.

Aunque el servidor y el sistema de gestión de la base de datos están alojados usando la misma plataforma, ambos están completamente desacoplados permitiendo así el traslado posterior a otras ubicaciones o, por si fuera necesario, cambiar la base de datos.

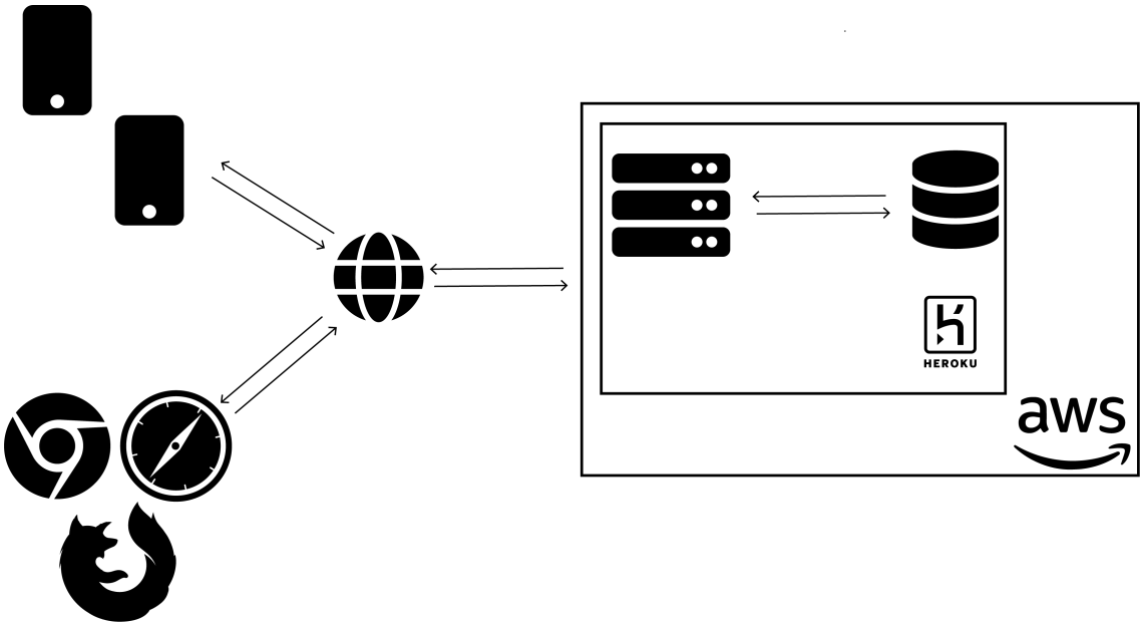

Figura 22: Estructura de la red de conexiones.

2. Material Design

Para el diseño de la aplicación GeFa se han usado los principios de diseño Material Design que proporciona el conglomerado de Alphabet Inc. en el que se incluye Android. Las metas de Material Design son: crear un lenguaje visual con los principios clásicos y el buen diseño, la unificación de la experiencia de usuario a través de distintas plataformas, proporcionar una base flexible para la personalización y mantener unos principios de diseño inspirados en el mundo físico y sus texturas [25].

Aunque son muchos los aspectos en los que Material Design hace hincapié, nosotros nos centraremos en los más importantes usados en la aplicación.

1. **Responsivo:** los diseños se adaptan y reaccionan a la entrada del usuario, el dispositivo y los elementos de la pantalla. Mediante el uso de contenedores responsivos que proporciona Android se puede conseguir que toda la interfaz se adapte a las circunstancias.
2. **Márgenes:** se han usado márgenes consistentes en todas las interfaces para dotar un aspecto de continuidad en el contenido mejorando así la experiencia de usuario.
3. **Elementos de interacción:** Material Design define que todos aquellos elementos de interacción que proporcionan entradas han de ser un al menos 48 x 48 dip (density-independent pixel) con al menos 8 dip de espacios entre ellos para equilibrar la densidad de información. Estas medidas se han utilizado en todos los elementos de las interfaces.
4. **Botón de acción flotante:** debe de tratarse de uno de los elementos más destacables en la pantalla. Debe de usarse el color principal o el secundario para que destaque sobre el resto de elementos. Se han utilizado en diversas pantallas de la aplicación; por ejemplo, para que los padres puedan crear una tarea con facilidad.
5. **Barras de menú superiores:** para destacar la barra superior de menús se ha de utilizar el color primario. Para enfatizar la diferencia entre las barras de aplicación y otras superficies, se debe usar un color secundario en componentes cercanos como el botón de acción flotante. Estas barras de menú se han utilizado en todas las interfaces salvo en la de carga, en la que sólo sale un símbolo de carga del color primario.
6. **Colores:** Material Design concreta el uso de dos tipos de colores: el primario y el secundario.
    1. **Color primario:** es el color que se muestra con mayor frecuencia en las interfaces y componentes. Se pueden utilizar variantes oscuras y claras para acentuar elementos o usar un color secundario.
    2. **Color secundario:** para proporcionar más formas de acentuar y distinguir elementos en la interfaz se puede añadir un color secundario. Este color es completamente opcional y se debe usar con moderación.

    En GeFa se han usas variantes de iluminación de un color principal para realzar los elementos empleados en todas las interfaces.
7. **Navegación:** se definen tres tipos de navegación entre las pantallas: la navegación lateral (*lateral navigation*), la de hacia delante (*forward navigation*) y la inversa (*reverse navigation*). En GeFa se ha usado la navegación lateral, que se refiere al movimiento entre pantallas en el mismo nivel de jerarquía (secciones diferenciadas), mediante una barra de navegación *drawer* que proporciona Android.
8. **Formas, tamaños e iconos:** para mantener una relación con el tamaño de los iconos se establecen una serie de medidas para mantener proporciones visuales consistentes en todos los iconos de la aplicación. También se define el estilo de los iconos para que sean simples y modernos reduciéndolos a su forma

mínima para que expresen sus características esenciales. Todo esto se aplica tanto a los iconos interiores como al de la aplicación. En cuanto a las formas, se recomienda el uso de figuras básicas (círculos, cuadrados, ortogonales...) para que se unifiquen con los iconos del sistema Android.

3. Usos y flujos de los procesos en la aplicación Android

La aplicación GeFa funciona mediante la asignación y comprobación de las tareas realizadas. Primero, al activar a un/a hijo/a, es necesario indicar cuál es la paga que le corresponde. Después, el/la padre, madre o tutor puede ya crear y asignar las tareas a un/a hijo/a. Cuando el/la hijo/a completa la tarea, lo marca así en la aplicación quedando en un estado de pendiente de confirmación. A continuación, un padre, madre o tutor debe confirmar que esas tareas se han realizado. Una vez hecho esto, esas tareas pueden formar parte de un pago. El pago de un hijo se compone de dos partes. La primera parte corresponde a la paga base que tiene asignada, que puede verse reducida en el caso de no cumplir con las tareas obligatorias que tienen asignadas una penalización. La segunda parte atañe al pago extra que se puede obtener al realizar tareas opcionales que tienen un extra por completarla. Cuando los padres quieren realizar un pago, sólo tienen que ir a la sección del hijo y pulsar sobre crear el pago. El sistema de gestión lo creará mostrando todas las tareas completadas y contabilizando la paga final. En el caso de que no se quiera pagar una determinada tarea en ese momento, los padres, podrán seleccionar las tareas que quieren que formen parte del pago y cuáles no, dejando las no seleccionadas para uno posterior. Una vez completado el hijo podrá verlo en su sección de pagos.

Registro e inicio de sesión en GeFa

Durante la implementación de GeFa, el registro de los usuarios ha recibido diversos cambios hasta llegar a la versión final que se muestra en la Figura 23.

La primera idea del registro de los usuarios que se descartó era que solamente se pudieran registrar los padres. Una vez lo estuvieran podrían crear y registrar a los hijos, hijas y otros miembros que conformasen la familia. El inconveniente de este modo de registro es que se cedería todo el control de unos usuarios a otros, permitiendo acceder y modificar datos personales como podría ser la contraseña de acceso. Por estos y otros motivos, se optó por usar una familia como contendedor de usuarios mediante activaciones para el acceso.

Otro punto del registro que se desechó fue la activación en el sistema mediante la comprobación del correo electrónico. Aunque la funcionalidad está implementada y es funcional, se hizo un 'bypass' en el servidor para que los registros realizados desde la aplicación Android estuvieran activados por defecto. Se realizó este cambio para simplificar el registro en el sistema, dado que para poder usar la aplicación también se tenia que ser activado por la familia y podría suponer una traba para captar usuarios. De igual modo, el mercado al que está enfocado la aplicación abarca usuarios con multitud de edades muy diferenciadas, por lo que realizar un registro y acceso lo más simple posible era la mejor opción.

En la Figura 23 se muestra un diagrama de flujo sobre el inicio y el registro de la aplicación GeFa. Al iniciarse, se hacen una serie de comprobaciones para configurar e iniciar correctamente la aplicación, como pruebas de conexión a internet y al servidor, si es la primera vez que se inicia la aplicación y si hay almacenados datos sobre preferencias en el móvil. En el caso de que el usuario ya hubiera iniciado sesión, se obtendrían los datos de las credenciales que se habrían almacenado y se descargarían los datos relevantes al usuario en la aplicación. En el caso de ser un usuario de tipo *Child* se descargaría sus tareas, y en el caso de ser de tipo *Parent* se descargaría un resumen de cada uno de los hijos y otros datos relevantes. En ambos casos, al terminar

la obtención de los datos, se mostraría la interfaz principal de la aplicación dependiendo del tipo de usuario.

En el caso de ser la primera vez que se usa la aplicación en ese dispositivo, aparece la ventana de *login* donde se podrá iniciar sesión en el caso de tener una cuenta y, para en el caso de no tenerla, también aparece un botón para registrarse en la plataforma. Si se acciona el botón de registro, aparece una ventana donde se le explica al usuario el funcionamiento de GeFa. Se expone que la aplicación funciona mediante la interacción entre usuarios de tipo padres (supervisores) y de tipo hijos (supervisados). Para continuar con el registro se muestran dos botones, uno para registrarse como un hijo/a y otro para registrarse como padre, madre o tutor. A continuación aparece un formulario, similares en los dos tipos, para registrarse en la aplicación. Cuando confirman los datos para registrarse en el caso de ser de tipo *Child* se debe indicar el número de identificación de la familia para unirse a ella, y en el caso de ser de tipo *Parent* se ofrecen dos opciones: crear una familia o unirse a una ya existente. En el caso de querer crear una familia deberá proporcionar los datos necesarios para cumplimentar el formulario mostrado y una vez completado aparecerá la pantalla principal de la aplicación.

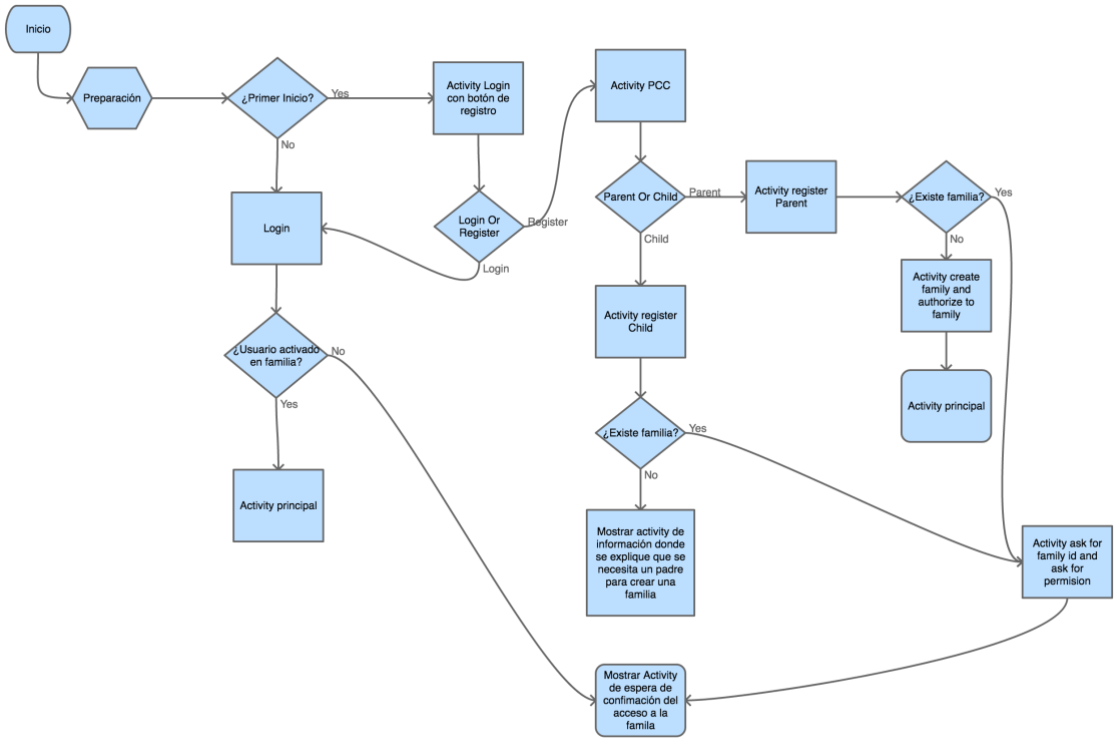

Figura 23: Diagrama de flujo del registro e inicio de sesión en GeFa

Vista principal de la aplicación

Los usuarios de tipo *Child* tienen como vista principal un listado de las tareas que comprenden dos periodos de pagos. Esto quiere decir que tienen una lista con las tareas sin completar, pendientes de confirmación, completadas y canceladas. Todas ellas están separadas por secciones según su tipo y ordenadas de la forma enunciada anteriormente. Conjuntamente, cuenta con una cabecera que le indica mediante un gráfico el porcentaje de cumplimiento que tiene el usuario. Esto resultará útil para percibir de un vistazo el progreso actual. De igual modo cuenta con un menú

desplegable lateral que le permitirá acceder al resto de vistas de la aplicación. Éstas son: Inicio (pagina principal), Familia, Pagos, Padres, Sobre nosotros, Cuenta y Cerrar sesión.

Los usuarios de tipo *Parent* tienen como vista principal un listado de hijos con un gráfico en anillo que muestra el progreso actual de cada uno de ellos. Esto resultará útil para percibir de un vistazo el cumplimiento actual de las tareas actual. Cuando se toque alguno de los hijos se mostrará la interfaz de detalle del hijo seleccionado. También cuenta con un menú desplegable lateral que le permitirá acceder al resto de vistas de la aplicación. Éstas son: Inicio (pagina principal), Familia, Hijos, Padres, Sobre nosotros, Cuenta y Cerrar sesión. En la Figura 24 se muestra un diagrama que indica las posibles interfaces a las que se pueden acceder a través de la vista principal de un *Parent*.

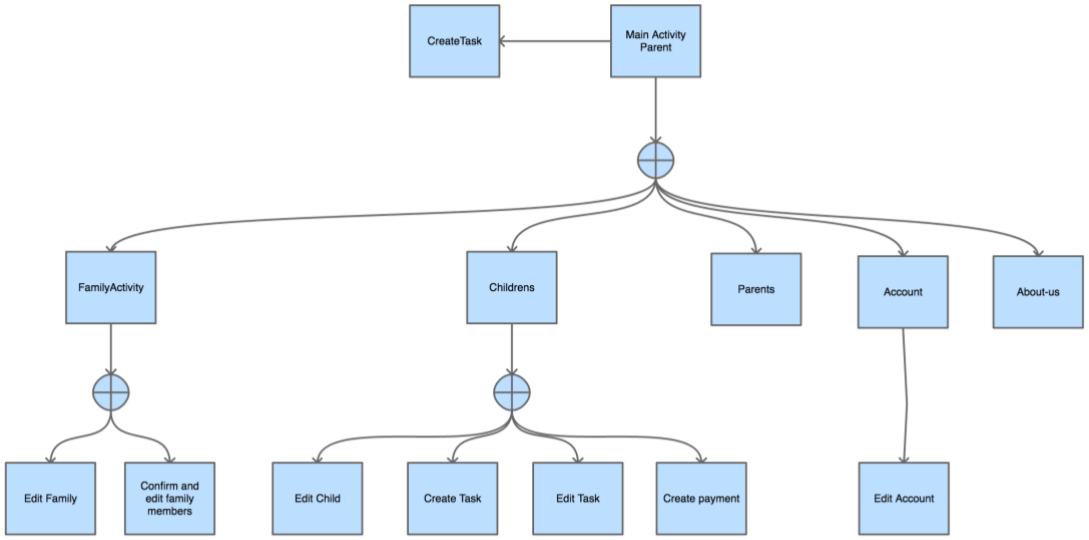

Figura 24: Diagrama de las acciones accesibles desde la Activity principal de los padres.

4. Estructura interna de las Interfaces

Para la implementación de las interfaces de la aplicación se han usado distintos componentes que ofrece Android para poder tener una aplicación modular y reutilizable, dado que se trata de dos aplicaciones que comparten algunas vistas y acciones. Esto ha permitido desarrollarlas de forma más ágil y rápida. Además, todas las actividades heredan de una Actividad básica, que engloba métodos y utilidades comunes, lo cual se explicará más adelante en este trabajo. El uso de la Actividad básica permite realizar funciones y métodos que tienen en común, de manera independiente a la actividad que lo hereda, permitiendo así el uso compartido de fragmentos.

Actividades principales

Ambas aplicaciones se basan en una *Activity* principal, que es la que aparece cuando iniciamos la aplicación, cuyo único contenido es un contenedor de tipo *fragment*. Esto permite mostrar cualquier interfaz en ese contenedor sin tener que iniciar una *Activity* nueva. Estos fragmentos se intercambian usando un panel lateral de navegación que permanece oculto hasta que el usuario desliza el dedo desde el lado izquierdo de la pantalla o pulsa el botón de apertura del menú.

5. Base Activity

Todas las *Activities* de la aplicación heredan de esta *Activity*, que a su vez hereda de *AppCompatActivity* permitiendo el uso de barra de acciones. La finalidad de esta BaseActivity es proporcionar funciones y métodos comunes a todas las actividades que la heredan y facilitar una interfaz a los *fragments* que se usan en distintas Activities.

Funciones y métodos comunes

Muchas *Activities* utilizan acciones y configuraciones comunes. Por eso, se ha decidido implementar esas en una sola actividad para así dejar el resto de código más limpio. Las acciones y configuraciones son las siguientes:

1. **Obtener el *core* de la aplicación:** se obtiene el *core* de la aplicación Android. Se explica en el siguiente punto, en la página 63.
2. **Habilitar la navegación hacia atrás:** para habilitar la navegación hacia atrás hace falta indicarlo en cada una de las actividades en su creación, además de necesitar que se sobrescriban dos métodos de la clase *Activity* (onSupportNavigateUp y onBackPressed).
3. **Ocultar el teclado:** muy útil cuando se está rellenado un formulario y se quiere mostrar un *dialog* indicando el progreso de la acción.
4. **Mostrar mensajes (*dialogs*):** para mostrar mensajes modales en la aplicación hace falta construir un *AlertDialog* con una serie de parámetros dependiendo de la forma en la que se quiere crear. Para estas distintas opciones se han ideado diferentes llamadas con múltiples parámetros. Por ejemplo, hay llamadas para crear mensajes de error, de información, de confirmación y una completamente personalizable.
5. **Mostrar *dialogs* de progreso:** parecidas al anterior sólo que éstas no se pueden cerrar hasta que la acción en segundo plano termine. Son creadas junto una barra de progreso tanto determinista como indeterminista; se usan normalmente cuando se hacen llamas al API.
6. **Crear *logs* o registros:** para crear *logs* en Android hace falta usar una librería propia del sistema que requiere unos parámetros adicionales al mensaje a mostrar. La llamada ideada simplifica la creación de registros pidiendo únicamente el mensaje que se quiere mostrar.
7. **Cambiar el título de la *Activity*:** dado que las aplicaciones se basan principalmente en una sola *Activity* que intercambia su contenido, es preciso cambiar el título de la *Activity* principal para indicar al usuario dónde se encuentra. Dada la fragmentación en Android y la discontinuidad de las llamadas a estas funciones con distintas versiones, es preciso realizar varias peticiones a distintos servicios capturando las posibles excepciones para poder efectuar el cambio del título.
8. **Crear *Toast* o *Snackbar*:** parecidos a los *dialogs* de mensajes sólo que éstos desaparecen pasado un tiempo y se muestran de manera no intrusiva en la zona inferior de la interfaz. La diferencia entre ambos es que el *Snackbar* puede contener un botón de acción dentro del mensaje.

Interacción con *fragments*

Como ya se ha comentado, se hace uso de los mismos *fragments* en distintas *Activities*. Para que estos *fragments* puedan interactuar con el resto de la aplicación y la *Activity* que lo contiene, es preciso obtenerla mediante una llamada interna que devuelve un objeto de tipo *Activity*. Para poder usar los métodos comunes y acceder también a los datos y llamadas al API que contiene el *core*, es necesaria hacer una conversión al tipo BaseActivity mediante un *casting* en el lenguaje Java. Usando esta *activity* común en todas, nos aseguramos de que no habrá problemas al hacer la trasformación dado que todas heredan de un mismo objeto.

6. Core

El inicio de la aplicación se realiza a través de una clase llamada GeFaApplication que extiende de *Application* que permite la configuración inicial de ésta. En este apartado se crea una instancia única o *singleton* de una clase que se ha llamado GeFaCore, en la que están implementados distintos métodos relacionados con la finalidad de la aplicación, así como de albergar datos compartidos entre *Activities*. Después se inicia la *Activity* principal que será lo primero que vea el usuario cuando inicie la aplicación.
El acceso al *core* se puede realizar desde cualquier *Activity* haciendo uso de la llamada implementada en BaseActivity, que a su vez, realiza una llamada a la clase GeFaApplication que tiene almacenada la referencia al *singleton* del *core*.

Llamadas al API

Todas las peticiones para obtener los datos del servidor se realizan usando el *core* como intermediario, mediante llamadas que tienen como máximo número de parámetros dos:

1. **Datos a enviar o referencias a obtener:** cuando se realizan llamadas GET en servicios REST y quieres obtener unos datos determinados se necesita la referencia a éstos. En el caso de realizar llamadas PUT o POST es necesario poner en el cuerpo de misma el objeto a crear o almacenar.
2. ***Listener*:** cuando el servidor conteste con los resultados obtenidos se llamará a este *listener* dentro del ámbito de la *Activity*.

Además, al realizar estas llamadas el *core* también puede incluir un *listener* propio para que se ejecute después de obtener los datos por tal de almacenarlos y organizarlos. Así, se podrá acceder a ellos posteriormente sin volver a realizar la llamada al servidor.

Información compartida entre Actividades y *Fragments*

Cuando el *core* ejecuta su propio *listener* después de que el servidor responda, guarda una copia de los datos obtenidos. Esto es útil dado que cuando se reemplazan los *fragments* de la *Activity* principal, se eliminan todos los datos relacionados con él. Así se pueden conseguir los datos obtenidos anteriormente comprobando si el *core* tiene una copia, y en caso contrario volver a pedirlo al servidor.
También es importante tener estos datos almacenados en el *core,* dado que distintas actividades y fragmentos hacen uso de la misma información, y al realizar múltiples llamadas al servidor para obtener la misma información es ineficiente y empeora la usabilidad de la aplicación.

7. Llamadas al API mediante REST

GeFa se basa en el almacenamiento de la información en la nube. Puesto que se realizan una gran cantidad de llamadas al API usando REST, fue necesario la implementación de distintas clases que albergaran todo lo necesario para realizar estas llamadas de manera simple y segura. Para ello se ideó una clase que alberga utilidades para la creación de rutas y cabeceras de las peticiones HTTP, además de una clase que ejecuta esa llamada de manera que no interfiera con el correcto refresco de la interfaz de usuario. Esto se realiza usando tareas asíncronas que provee Android.

Tarea asíncrona polimórfica

La tarea que realiza todas las llamadas al API se llama 'RestTask'. Esta clase extiende de 'AsyncTask' que permite la ejecución de código fuera del hilo de ejecución principal de la Actividad que hace el refresco de la interfaz; por lo tanto, la interfaz no se congelará mientras se espera la respuesta del servidor.

'RestTask' permite realizar los métodos CRUD (create, read, update y delete) sobre cualquier DTO (data transfer object). Para ello, se basa en el polimorfismo del lenguaje Java permitiendo así reutilizar la misma clase con distintos objetos. Para realizar el acceso al API la tarea requiere de los siguientes parámetros o argumentos:

1. **Mensaje:** se trata del objeto u objetos que se que quieren enviar como carga de datos al servidor en caso de ser una llamada PUT o POST.
2. **Método REST:** enumerado que indica si es una llamada GET, POST, PUT o DELETE.
3. **Autorización:** algunas llamadas al servidor requieren que tengas iniciada una sesión. Para realizar esta autenticación se usa JWT (pág. 27), que se basa en un *token* identificativo de usuario.
4. ***Listener* de la *Activity*:** es el *listener* que se ejecutará en la *Activity* cuando se obtenga el resultado de la consulta.
5. ***Listener* del *core*:** es el *listener* que se ejecutará en el *core* cuando se obtenga el resultado de la consulta.
6. **El tipo de datos a recibir:** se trata de un enumerado que indica si se va a recibir datos y si se trata de un objeto o un *array* de objetos. Este argumento podría obviarse haciendo intentos de prueba-error, pero se ha optado por incluirlo y así evitar excepciones y comprobaciones que puedan ralentizar la ejecución.
7. **El nombre de la Clase que representa el objeto Java a recibir**: para que Jackson (pág. 25) realice la conversión de JSON a objeto Java necesita la clase a la que se quiere efectuar el mapeo.

Todos los *listeners* que se usan en la tarea parten de la interfaz 'RestListener'. Se trata de una interfaz polimórfica constituida por tres métodos. Dos de ellos se ejecutan cuando la obtención de los datos del servidor ha sido un éxito. La diferencia entre ambos es que uno tiene como parámetros del método un único objeto y el otro una lista enlazada de objetos. En caso de que ocurra un error en la obtención, se ejecuta el método del 'RestListener' 'OnError'. Éste, tiene como parámetro un objeto de tipo 'RESTError' que es la representación que manda el servidor en caso de error.

8. Adaptador de listas en Android con múltiples objetos

La información principal de la aplicación son las tareas, por lo que la forma de mostrarlas y organizarlas es una parte muy importante de la App. Para mostrar las tareas se ha usado un contenedor *ListView* que permite mostrar un listado de *Views* o vistas en un formato lista. La forma común de usar este contenedor es mediante un único tipo objeto que almacena los datos a mostrar, pero para organizar mejor la vista se ha optado por usar varias que separan las tareas de forma que queden más claras.

Principalmente hay dos tipos de vistas diferentes: separadores y tareas. Los separadores son una vista sencilla con una única etiqueta que muestra el tipo de las tareas que contiene su sección. Las tareas, en cambio, son unas vistas algo más complejas; depende del estado de la misma. Por lo que, dependiendo de las tareas existentes, puede haber 4 vistas diferentes para éstas y las vistas que separan los diferentes estados de las tareas.

Para adaptar el comportamiento del contenedor *ListView* hace falta sobrescribir unos métodos que indican el tipo de objeto que está en una determinada posición de la lista y la cantidad de tipos diferentes que hay en la misma. En la Figura 25 se muestra un ejemplo del código del adaptador de la lista para la obtención de un objeto Java que se representa en una posición de ésta. En el método se comprueba cada una de las cantidades de cada tipo de tareas además del separador para obtener el objeto correcto

que hay en cada lugar. A la hora de cargar cada vista se llama a este método para obtener el objeto que se va a representar, para saber el tipo se llama a otro método que retorna el tipo para poder hacer un *casting* Java sin que se disparen excepciones que afecten al rendimiento de la aplicación.

```java
@Override
public Object getItem(int i) {
    int request_confirmation_tasks_count = this.request_confirmation_tasks.size();
    int pending_tasks_count = this.pending_tasks.size();
    int not_completed_tasks_count = this.not_completed_tasks.size();
    int completed_tasks_count = this.completed_tasks.size();
    int tasksBefore = 0;

    if (request_confirmation_tasks_count > 0) {
        if (request_confirmation_tasks_count + 1 > i && i >= 0) {
            if (i == 0) {
                return this.separatorsNames[0];
            } else {
                return this.request_confirmation_tasks.get(i - 1);
            }
        } else {
            tasksBefore += request_confirmation_tasks_count + 1;
        }
    }
    if (pending_tasks_count > 0) {
        if (tasksBefore + pending_tasks_count + 1 > i && i >= tasksBefore) {
            if (i == tasksBefore) {
                return this.separatorsNames[1];
            } else {
                return this.pending_tasks.get(i - tasksBefore - 1);
            }
        } else {
            tasksBefore += pending_tasks_count + 1;
        }
    }
    if (not_completed_tasks_count > 0) {
        if (tasksBefore + not_completed_tasks_count + 1 > i && i >= tasksBefore) {
            if (i == tasksBefore) {
                return  this.separatorsNames[2];
            } else {
                return this.not_completed_tasks.get(i - tasksBefore - 1);
            }
        } else {
            tasksBefore += not_completed_tasks_count + 1;
        }
    }
    if (completed_tasks_count > 0) {
        if (tasksBefore + completed_tasks_count + 1 > i && i >= tasksBefore) {
            if (i == tasksBefore) {
                return  this.separatorsNames[3];
            } else {
                return this.completed_tasks.get(i - tasksBefore - 1);
            }
        }
    }
    return null;
}
```

Figura 25: Método para la obtención del objeto que se muestra en una posición de una ListView.

6. Resultados

En esta sección se presentará en un primer apartado la aplicación final en Android mediante capturas de la propia aplicación que representan los casos de uso ya descritos (pág. 29). Para mostrar el buen funcionamiento del servidor implementado se han realizado una serie de pruebas que se comentarán en el segundo apartado.

Paleta de colores

Las guías de estilo de Android recomiendan el uso de tres colores principales para usar en una aplicación (pág. 58). Estos tres colores se usan para destacar ciertos elementos visuales y deben permitir la correcta visualización de un texto escrito en color blanco o negro como recomendación. En nuestro caso, hemos decidido usar los colores que se presentan en la Figura 26 y se ha usado el color blanco como el color del texto debido que se lee mejor que el texto negro. Los distintos usos de colores que recomiendan son los siguientes:

1. **Color *Primary*:** usado para el color de fondo de algunas partes de interfaces (cabeceras, títulos de *dialogs*…) y como color de realce de textos.
2. **Color *Primary Dark*:** usado para marcar diferencias entre secciones de las interfaces.
3. **Color *Accent*:** color de realce o énfasis usado mayoritariamente en los botones, *checkboxes* y otros elementos de entrada de las interfaces.

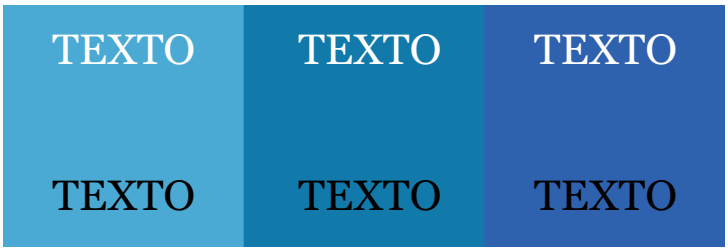

Figura 26: Paleta de colores de GeFa

Como se comentó en la sección sobre Material Design (pág. 58), los colores usados parten del color principal o *primary* hacia tonalidades más oscuras para así proporcionar mayor contraste o realce en las distintas interfaces.

Icono de la aplicación

El icono de la aplicación diseñado es el que aparece en la Figura 27. En ella se usan los colores descritos en el apartado anterior para mantener un estilo común en todos los aspectos de la plataforma. En el centro aparece el nombre de la aplicación “GeFa” (Gestión Familiar) y a su alrededor una serie de semicírculos, con los colores de la paleta, que simbolizan la unión de los miembros de la familia. El diseño se ha realizado siguiendo las guías de diseño indicadas en Material Design (pág. 58).

Figura 27: Icono de la aplicación GeFa

1. Aplicación Android

En el presente apartado se mostrarán capturas de la versión final de la aplicación Android. Muchas de ellas representan los casos de uso (pág. 29) descritos en apartados anteriores y son la implementación final de ciertos bocetos también explicados (pág. 38). Otras capturas constituyen partes de la aplicación que son imprescindibles para el uso de la misma y que se deben comentar para entender su funcionamiento.

Registro en la plataforma

Durante el registro en la plataforma se le hace escoger al usuario entre dos modalidades: hijo/a o como un padre, madre o tutor. Se ha incluido una explicación a la hora de escoger una de las opciones para comprender el funcionamiento de la aplicación.

> *"GeFa es una aplicación para gestionar las tareas del hogar mediante la asignación de tareas y recompensas a cada uno de los hijos. Para el funcionamiento de la aplicación es necesario al menos un adulto para que asigne tareas y supervise el cumplimiento de éstas. Ahora debes escoger cómo iniciar la aplicación."*

En la Figura 28 se muestra la interfaz final de la selección del tipo de usuario durante el registro. En ella se aprecia la explicación antes comentada y las dos opciones a escoger mediante unos botones.

Figura 28: Interfaz para la selección del tipo de usuario durante el registro en la plataforma.

Interfaz principal del usuario de tipo *Parent*

La interfaz inicial que visualiza un usuario de tipo *Parent* es la que se muestra en la primera captura de la Figura 29. En ella se lista los hijos o hijas que están en la familia indicando el nombre, la paga y un resumen de las tareas realizadas mediante un gráfico en forma de anillo. Este gráfico se divide en 3 partes: en color verde las tareas completadas, en amarillas las que están pendientes de confirmación por parte de los padres y en color rojo las no completadas. En el centro del gráfico se muestra el porcentaje completado y cuantas tareas a completado del total.

En la parte superior izquierda hay un botón que activa el menú lateral izquierdo que aparece en la segunda captura. Desde este menú se puede ir a todas las secciones de la aplicación: Inicio, Familia, Hijos, Padres, Sobre nosotros, Cuenta y Cerrar sesión. Conjuntamente aparece el nombre del usuario, su imagen de perfil y el nombre de la familia a la que pertenece. Algunas de estas secciones se comentarán más adelante.

El botón flotante de la parte inferior derecha permite accionar de forma rápida la creación de una tarea. Al pulsar sobre él, se activa una ventana flotante modal (*dialog*) que muestra el listado de hijos con la imagen de perfil y el nombre que aparece en la tercera captura. El padre, madre o tutor seleccionará de la lista el hijo al que quiere asignar una tarea, lo que le llevará a las interfaces que se muestran en la Figura 32.

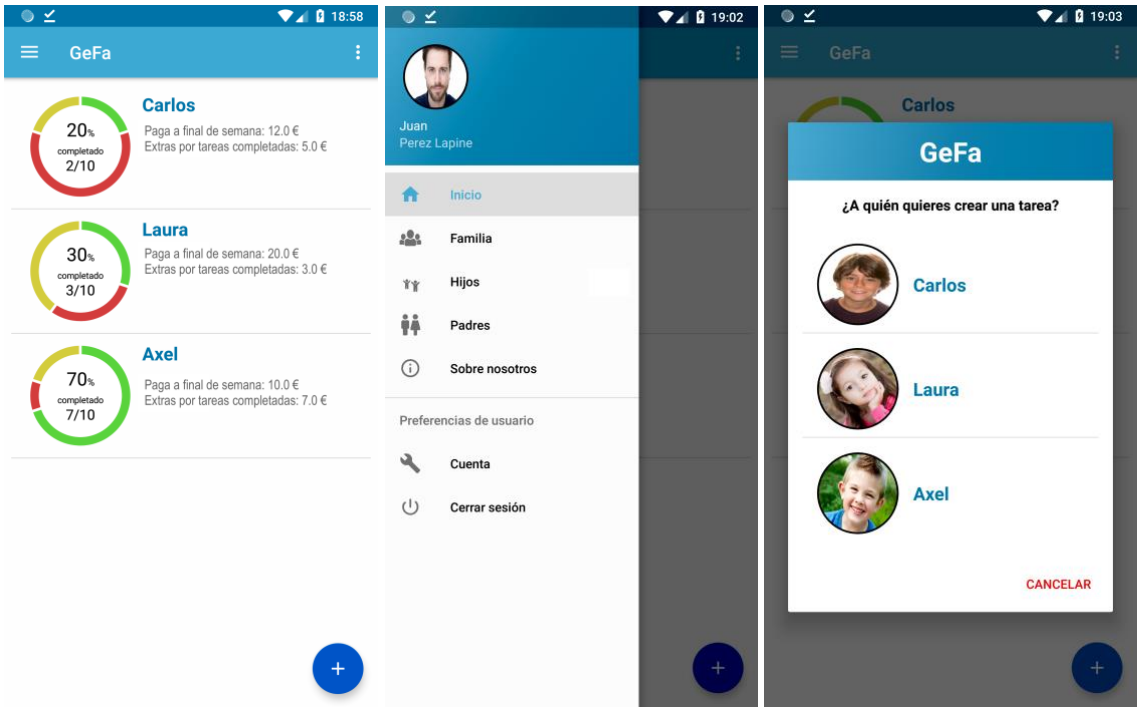

Figura 29: Interfaz principal del usuario tipo Parent.

Interfaz principal del usuario de tipo *Child*

La interfaz inicial que visualiza un usuario de tipo *Child* es la que se muestra en la primara captura de la Figura 30. La interfaz de divide en dos partes: la cabecera y el cuerpo. En la cabecera de la interfaz aparece el progreso de las tareas asignadas. Este gráfico se divide en 3 partes: en color verde las tareas completadas, en amarillas las que están pendientes de confirmación por parte de los padres y en color rojo las no completadas. En el centro del gráfico se muestra el porcentaje completado y cuántas tareas ha completado del total. Se trata pues del mismo gráfico que visualiza el usuario

de tipo *Parent*. Además, aparece la información sobre la paga que recibirá a final del periodo de pagos.

En el cuerpo de la interfaz se muestra un listado de tareas separadas en secciones dependiendo del estado de la misma. La primera sección es la de las tareas pendientes de confirmación por parte de un padre, la segunda son las tareas que tiene pendientes por completar y en la última las tareas completadas. Dependiendo del estado, el *checkbox* de la derecha de cada tarea esta deshabilitado o activado. Independientemente del estado de la tarea, al pulsar sobre alguna de ellas se mostrará un *dialog* en el cual se mostrará información adicional sobre la tarea. En la tercera captura de la Figura 33 se muestra un ejemplo de los datos mostrados. Asimismo, en el lado izquierdo de cada tarea, aparece un indicativo en color que distingue cuáles de las tareas son opciones (verdes) y cuales obligatorias (azules).

En la parte superior izquierda hay un botón que activa el menú lateral izquierdo que aparece en la segunda captura. Desde este menú se puede ir a todas las secciones de la aplicación: Inicio, Familia, Pagos, Padres, Sobre nosotros, Cuenta y Cerrar sesión. Conjuntamente aparece el nombre del usuario, su imagen de perfil y el nombre de la familia a la que pertenece. Algunas de estas secciones se comentarán más adelante.

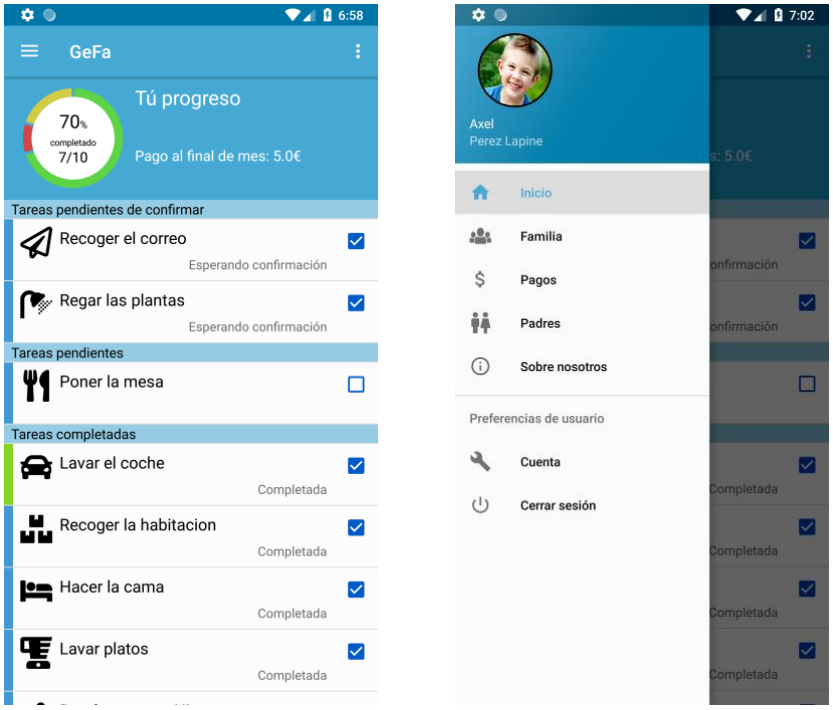

Figura 30: Interfaz principal del usuario tipo Child.

Crear una familia o unirse a una familia

Durante el registro en la plataforma se le indica al usuario del tipo *Parent* el siguiente mensaje:

> *"Para registrarte en la app es necesario que exista una familia asociada a ti. ¿Existe ya esa familia o quieres crear una nueva?"*

Dependiendo de la elección del usuario se mostrará una de las dos interfaces de la Figura 31. Si el usuario de tipo *Parent* indica que no existe ninguna se mostrará la de la izquierda y en caso contrario la de la derecha. Para los usuarios de tipo *Child* siempre se les muestra la interfaz de la derecha, por lo tanto, para que un hijo pueda usar la aplicación ha de estar registrada una familia por un padre, madre o tutor.

Para la creación de una familia se muestra la primera interfaz de la Figura 31 como se acaba de comentar. En ella se piden solamente 3 datos: el nombre de la familia, el día de paga y el periodo de pagos. En la captura se ven unos pequeños interrogantes azules con un signo de interrogación '?'. Estos botones abren distintos *dialog* con información relevante para cada campo; éstos son una de las prioridades descritas en el método MoSCoW de la Figura 5. Por ejemplo, en la selección del periodo de los pagos se muestra como información adicional el siguiente mensaje:

> *"Aquí eliges si los pagos se realizarán semanalmente o mensualmente."*

Se trata de un mensaje muy sencillo pero que resultará de gran utilidad para muchos de los usuarios que requieran un poco de ayuda.

En el caso de unirse a una familia existente se muestra la interfaz de la derecha de la Figura 31. En ella se muestra un texto que describe la aplicación GeFa y la forma de encontrar el único campo que se requiere para unirse: el identificador único de la familia. El mensaje que muestra la aplicación es el siguiente:

> *"Bienvenido a GeFa, para poder acceder a una familia debes indicarnos el número de referencia que tiene la familia. Este número se encuentra en el apartado de 'Familia' dentro del menú lateral de la aplicación. Cualquier miembro de la familia puede verlo, pregunta a cualquiera de ellos."*

Una vez escrito el identificador y pulsado el botón de unirse, el usuario quedará unido a la familia, pero en estado de espera. En el cual, no podrá acceder a los datos de la familia hasta que uno de los miembros apruebe el acceso.

Figura 31: Interfaz para la creación y la unión a una familia.

Crear una tarea

La función principal de la aplicación es la creación de tareas, para ello se ha diseñado la interfaz que aparece en la Figura 32. En la primera captura se muestran los campos que los padres han de rellenar para la creación de una tarea. Todos ellos cuentan con la ayuda descrita en el apartado anterior (pág. 70). Aparte de la información básica como lo son el nombre, notas o la repetición de las tareas, hay otros campos que requieren un poco más de atención.

El primero es el campo del tipo de familia. En caso de ser una tarea obligatoria el padre podrá incluir si la tarea tiene una penalización en caso de no cumplirla. Si selecciona el tipo opcional, el campo de penalización cambia por el de pago extra por completar la tarea.

El segundo campo es el de la fecha límite, el cual se introduce mediante un calendario embebido en un *dialog* que se muestra en la segunda captura. Esta fecha se mostrará en la información adicional de la tarea al pulsar en ella.

Por último, el campo del icono de la tarea se introduce mediante un *dialog* en el que se muestran un listado de iconos (118) incluidos en la aplicación. El icono seleccionado se mostrará siempre que se muestra una tarea, ya que resultará muy útil para poder identificar rápidamente cada una de ellas.

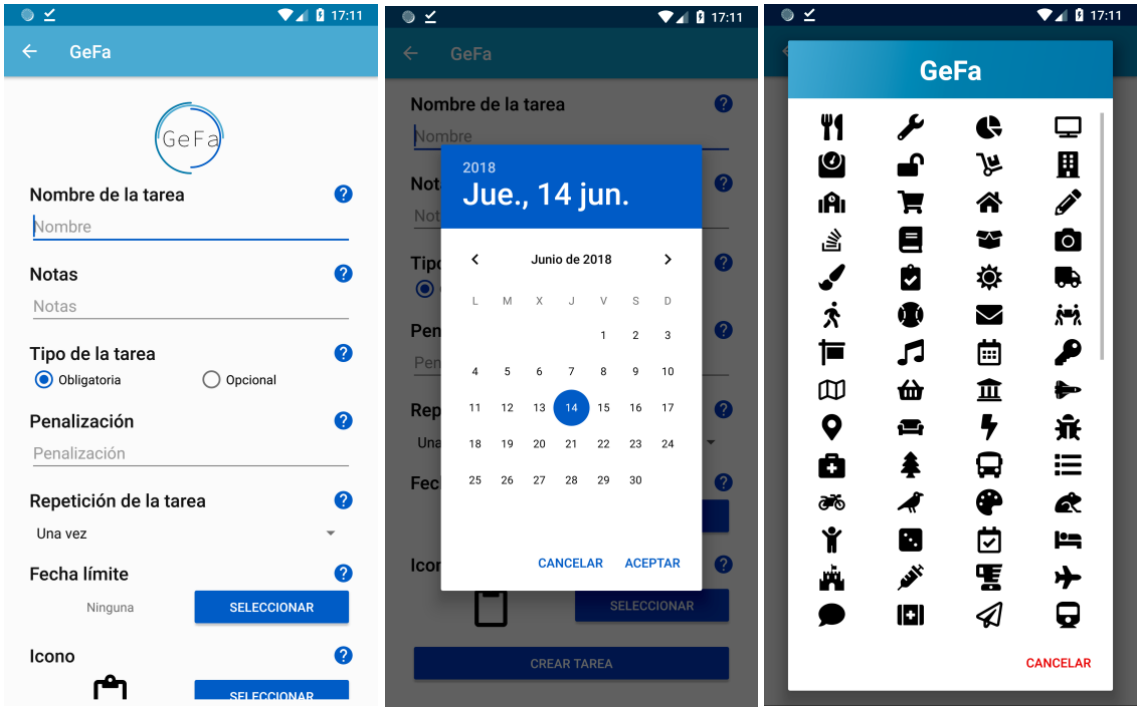

Figura 32: Interfaces para la creación de una tarea.

Interfaz de detalle de un hijo en la aplicación de los padres

Para que un padre, madre o tutor vea la información detallada de uno de los hijos, se ha creado las interfaces que se muestran en la Figura 33. Se puede llegar a esta interfaz mediante distintos sitios: desde el menú lateral y pulsando sobre uno de los hijos en la lista de la interfaz principal, entre otros.

En la primera captura se muestra la información relevante de uno de los hijos o hijas. En la cabecera aparece la imagen de perfil, el nombre del usuario y la paga que tiene asignada. Bajo la cabecera aparece un listado de tareas separadas en secciones dependiendo de su estado. La primera sección es la de las tareas pendientes de confirmación, la segunda son las tareas que tiene pendientes por completar y en la última las tareas completadas. Todas tienen un botón que hace aparecer un *dialog* con opciones a realizar sobre la tares: marcar como completada, modificar tarea y cancelar tarea. En el caso de pulsar sobre una de las tareas de la lista aparecerá el *dialog* mostrado en la tercera captura, en la cual se ve toda la información relacionada con la tarea: icono, nombre, notas, tipo, pago o penalización y fecha límite. También aparece un botón flotante de la parte inferior derecha que abre la interfaz para la creación de una tarea (Figura 32).

Si el padre, madre o tutor pulsa sobre el botón del dólar ($) de la parte superior derecha, se abrirá la interfaz que se muestra en la segunda captura de a Figura 33. Esta interfaz sirve para crear un pago al hijo o hija. Muestra en la parte superior el pago final dividido en la paga base y en los pagos extra por tareas opcionales. Bajo esto aparece un listado con las tareas que ha completado el hijo o hija, permitiendo mediante un *checkbox* la inclusión o exclusión de una tarea en el pago. Al pulsar el botón del *tick* se confirmará el pago y las tareas seleccionadas desaparecerán de la primera vista.

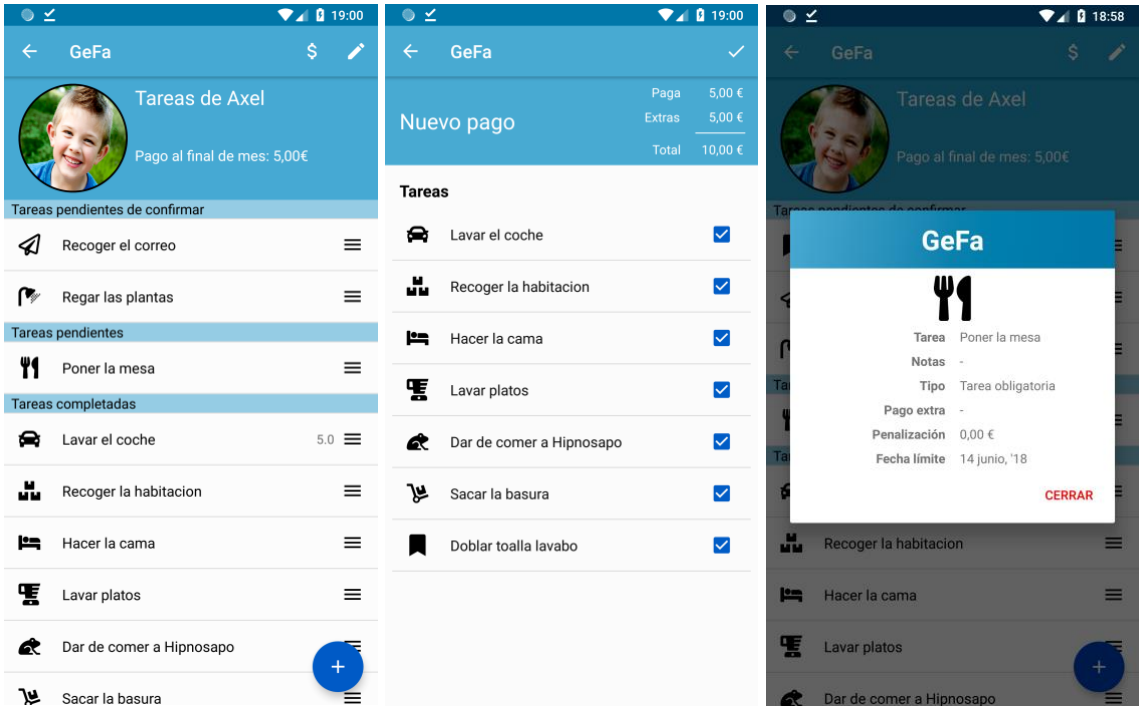

Figura 33: Interfaces de detalle de un hijo.

Activación de usuarios

En anteriores secciones se ha indicado la importancia del control de acceso a los distintos miembros de la familia. Cuando un usuario se incorpora en la familia se marca como "Pendiente de confirmación" hasta que otro usuario lo activa, permitiendo entonces el acceso a la información. Estas activaciones solamente las puede realizar un usuario de tipo *Parent*. Para realizar esta acción se ha creado la interfaz que muestra en la Figura 34. Esta interfaz muestra la información de la familia y permite modificar el día de paga y el periodo de pagos. Además, aparece una sección de notificaciones donde se muestran los usuarios que se quieren unir a la familia. Esta interfaz se muestra tanto para los padres como para los hijos, pero los hijos no pueden ver las notificaciones.

Las notificaciones de la familia se muestran mediante un listado de usuarios que quieren unirse con el nombre y un botón para ver más información. Al pulsar sobre el botón se muestra el *dialog* que aparece en la segunda captura con el siguiente mensaje:

> *"¿Quieres activar al usuario? Esto permitirá que pueda acceder y modificar la familia y las tareas."*

Si pulsa sobre activar, se actualizará el usuario permitiéndole el acceso total a la familia. A continuación, se actualizará la interfaz quitando la notificación como aparece en la tercera captura.

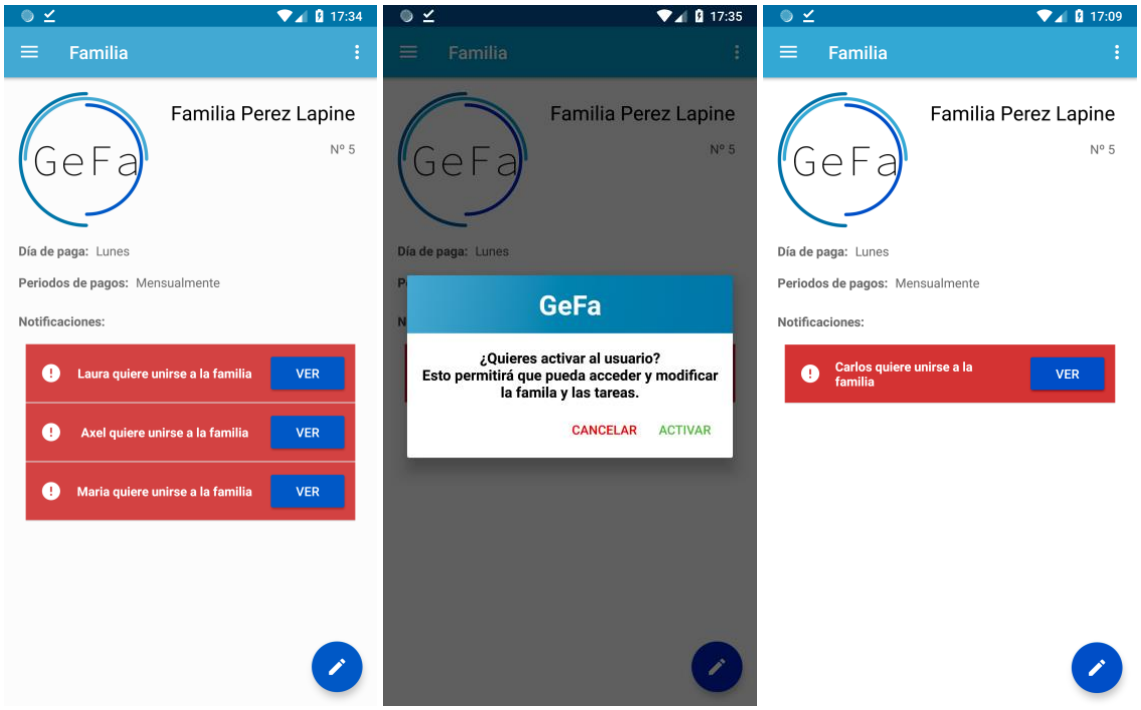

Figura 34: Interfaz para la activación de los usuarios.

2. Consultas y test de estrés en el servidor

Para comprobar el correcto funcionamiento del servidor implementado, cuando se realizan llamadas recurrentes o cuando se encuentre con carga excesiva, se han realizado distintas pruebas que comentaremos en las siguientes secciones.

Consultas a llamadas comunes

Para comprobar el coste temporal del acceso a los datos se han realizado distintas medidas usando las utilidades de testeo que ofrece la aplicación Postman (pág. 23). En la Figura 35 se muestran dos gráficos con cuatro de llamadas comunes que se realizan al servidor. El primer gráfico muestra la relación del coste temporal con respecto al tiempo (múltiples accesos secuenciales) y la segunda gráfica compara el tamaño de los paquetes recibidos por las distintas llamadas. Todas las llamadas se realizaron al servidor ya desplegado en Heroku.

La primera llamada trata de una petición para obtener las tareas actuales (entre dos periodos de pagos) de un hijo o hija. Se realiza constantemente esta llamada por parte de los hijos al acceder y al actualizarse los datos de la aplicación. Con esta petición se obtiene un paquete de tamaño muy reducido (3,98 KB) ya que se trata solamente de un listado de objetos de tipo tarea en JSON. La media de duración de esta petición es de 158 ms.

La segunda se trata de una petición para descargar el listado de los padres de una familia. Se realiza la llamada conjuntamente con otras peticiones para la obtención de información general de la familia. En esta petición se obtiene la información de un padre, incluida la foto de perfil; el tamaño del paquete es de casi 1 megabyte. La media de duración de esta llamada es de 473 ms.

La tercera llamada se realiza para identificarse en la plataforma mediante el usuario y la contraseña. Con ella se obtiene el *token* JWT (pág. 27) de unos 200 Bytes de tamaño para realizar las futuras peticiones que requieren autorización. La media de duración de esta petición es de 272 ms.

La última es la petición usada por los padres al iniciar la aplicación. Con ella se obtienen multitud de datos, entre ellos encontramos un resumen de las tareas de cada hijo, información de la familia, los datos de los hijos (incluidas las fotos de perfil) … De ahí el tamaño del paquete recibido de ~1,1MB. La media de duración de esta llamada es de 502 ms.

Como se aprecia en la Figura 35 el coste temporal más alto de las peticiones se obtiene en la primera iteración de la llamada realizada. Posteriormente se ve ligeramente reducido el tiempo de acceso debido a los aciertos en la *cache* configurada durante la creación del servidor; siendo, dependiendo del caso, de hasta un 98,8% de aciertos, dejando gran parte del tiempo de acceso a la latencia de red.

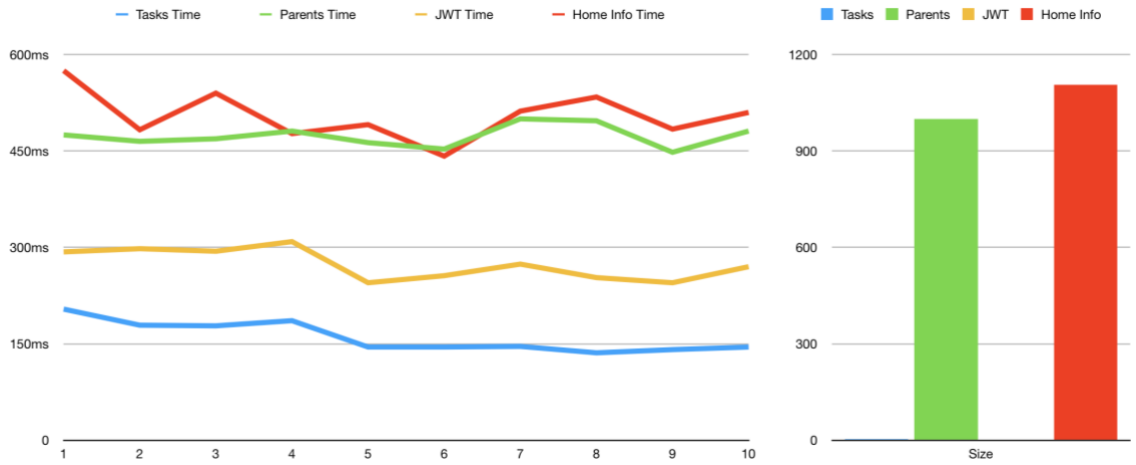

Figura 35: Gráfica del coste temporal de consultas al API comunes.

Pruebas de estrés

Al realizar estas pruebas de estrés no se pretende comprobar la capacidad computacional del servidor, ya que la versión gratuita del alojamiento tiene grandes limitaciones tanto a nivel de procesador como de memoria principal (512 MB). Lo que se quiere juzgar es que la capacidad de respuesta sea constante y la ayuda que ofrece la incorporación de la *cache*. También hemos de comentar que durante la realización de los experimentos se obtuvieron avisos de la plataforma Heroku indicando que se estaba sobrepasando el límite de la cuota gratuita, lo que supuso una penalización para la capacidad computacional.

Se han llevado a cabo pruebas de estrés realizando más de 100 llamadas concurrentes mediante las herramientas que proporciona Postman (pág. 23). Comentaremos únicamente el experimento con la primera llamada explicada en el apartado anterior. La media de acceso que se obtuvo fue de 143,63 ms con picos de hasta 214 ms y con una anomalía de 1142 ms (1,1 s). Se ha supuesto que esta anomalía se ha debido al capado que realizó Heroku, pues en el resto de las pruebas los resultados fueron constantes teniendo solamente ligeros picos.

Con los datos obtenidos, mediante las distintas pruebas, podemos asumir que el servidor se comportará de manera adecuada ante situaciones de carga intensiva, manteniendo un servicio activo y constante.

7. Conclusiones

A lo largo de este proyecto nos hemos propuesto la implementación de un sistema de gestión de tareas domésticas mediante el desarrollo de una aplicación en la plataforma Android. Dado que es cada vez más común que los/as niños/as posean un teléfono móvil personal a una temprana edad y que el sistema operativo Android supone la mayoría de la cuota en el mercado español, la implementación en esta plataforma pareció evidentemente imprescindible.

Para realizar la aplicación se han tenido que completar distintos objetivos que satisfacen el conjunto del servicio. La creación de un servidor usando JHipster y Spring dio la posibilidad de implementar una manera modular, ampliable, segura, estandarizada y rápida de acceso a la información y el control de usuarios. Para establecer los canales seguros de comunicación se desplegó el servicio en la plataforma AWS (Amazon Web Services), un servicio ampliamente utilizado y con años de experiencia contando con el respaldo de grandes empresas y corporaciones, que mediante el uso de protocolos de cifrados estándares como HTTPS, facilitó la incorporación de esta seguridad en nuestro servicio.

Para que el acceso a la información dentro de la plataforma fuera seguro se implemento un sistema de roles de acceso mediante el uso de Spring Security. Esto hizo que el acceso a determinados datos fuese autorizado o no de manera simple y desacoplada del código relativo al servicio. También se implemento un segundo control de los usuarios mediante una activación referente a una familia, haciendo así más segura nuestra plataforma. Aunque el descarte de la funcionalidad de la activación del correo fuese descartado para mejorar la experiencia de usuario, no hizo que supusiera una brecha en la seguridad del sistema.

Siguiendo las pautas de Material Design, se creó una aplicación en la plataforma Android con vistas diferentes dependiendo del usuario que iniciase sesión. Usando materiales, código e interfaces compartidas entre las dos vistas, se pudo crear más rápidamente ambas implementaciones. También se añadieron consejos y pistas para que los usuarios entendiesen qué se realizaba al hacer una acción determinada y el por qué de la necesidad de los datos requeridos en distintos ámbitos de la aplicación.

Aunque la aplicación cumple con los objetivos marcados en el inicio del proyecto, ahora podemos pensar en funcionalidades y cambios futuros que se podrían integrar en este nuevo servicio. Algunas de las posibilidades serían implementar la aplicación a distintas plataformas, añadir nuevas capacidades o incluso modificar la estructura interna del servidor para proporcionar un mejor servicio. Éstas y otras funcionalidades y mejoras se comentarán en las siguientes secciones.

1. Nuevas funcionalidades

Como se comentó en las priorizaciones de las funcionalidades de la aplicación, una de las funcionalidades que se excluían en la primera versión de la aplicación fue la realización de tareas y listas compartidas. La inclusión de éstas podría convertir la aplicación en algo más que un gestor de tareas: un gestor familiar.

La inclusión de tareas compartidas se usaría para que dos o más hijos compartieran una tarea doméstica y que así aprendieran a colaborar unos con otros, mejorando así su comunicación y la colaboración en grupo. Además, también se incluiría la posibilidad de asociar a un padre, madre o tutor a la tarea para que pueda ayudarles en la

realización de ésta. Así podría cerciorarse de que la tarea está siendo completada por todos los miembros que la tienen asignada.

La introducción de listas compartidas extendería el ámbito de uso de GeFa hacia una aplicación de gestión familiar. Las tareas compartidas podrían usarse como listas de compras, pagos, recordatorios, contraseñas, etc. Podrían asignarse qué miembros de la familia pueden verlas y cuáles editarlas, para así tener un buen control de ellas. La inclusión de estas listas también permitiría facilitar la implementación de las tareas con recompensas a través de puntos canjeables. Se añadiría una lista predefinida en la que asignar las recompensas con su respectivo coste en puntos. Con esto se conseguiría acceder a un mercado con niños y niñas con una menor edad en la que se aprecia más el concepto de regalos que la noción del dinero.

Otra de los objetivos futuros es la creación de GeFa para iOS. Aunque el mercado en España de iOS es de alrededor de un 6,4% (2016) [1], en otros países de Europa y en Norteamérica la situación en completamente contraria. Por ello, la implementación de GeFa en este sistema operativo sería crucial para que la aplicación accediera a un mercado mucho mayor en determinados lugares. Para acceder rápidamente a todos los mercados implementando únicamente una aplicación, se podría priorizar la aplicación web, así, también nos nivelaríamos con otras aplicaciones con servicios similares. Creando una aplicación web permitiría que tantos en PCs como en móviles se pudiera acceder independientemente del sistema operativo. Con esto se tendría más tiempo para poder desarrollar la aplicación nativa en iOS, y, también se podrían crean aplicaciones en Windows, Mac OS o Linux embebiendo la web en una aplicación Java con pocas líneas de código.

2. Escalabilidad futura

Aunque la existencia de GeFa es muy temprana, merece la pena pararse un momento a pensar a futuro. Actualmente todo el servidor que soporta la aplicación se ha implementado usando un modelo monolítico, esto quiere decir que todos los servicios que ofrece se ejecutan en una misma aplicación y servidor. Una forma de desacoplar el servidor seria usar una aplicación con una arquitectura de micro-servicios (*Microservices*).

La arquitectura de micro-servicios divide todas las entidades en servicios separados e independientes, permitiendo el desacople del servidor. Esto permite incluso tener lenguajes de programación distintos en cada entidad y que utilicen diferentes tecnologías de almacenamiento de datos. Actualmente GeFa no necesita este tipo de tecnologías, pero, si en un futuro se quiere expandir la aplicación a más campos de uso, puede ser una opción para poder desarrollar la aplicación solamente añadiendo más nodos al conjunto de servicios.

8. Bibliografía

[1] J. Motyka, «El Android Libre,» El Español, 11 Mayo 2016. [En línea]. Available: https://elandroidelibre.elespanol.com/2016/05/android-cuota-mercado-europa-espana-marzo.html. [Último acceso: 11 Junio 2018].

[2] R. Rodríguez, «El Confidencial,» El Confidencial, 26 Mayo 2017. [En línea]. Available: https://www.elconfidencial.com/tecnologia/2017-05-26/movil-uso-exceso-espana-salud-enganchados-smartphone_1389117/. [Último acceso: 8 Junio 2018].

[3] A. LENHART, «Teens and Mobile Phones Over the Past Five Years: Pew Internet Looks Back,» 19 Agosto 2009. [En línea]. Available: http://www.pewinternet.org/2009/08/19/teens-and-mobile-phones-over-the-past-five-years-pew-internet-looks-back/. [Último acceso: 8 Junio 2018].

[4] Colaboradores de Wikipedia, «Maria Montessori,» Wikipedia, La enciclopedia libre., 15 Junio 2018. [En línea]. Available: https://es.wikipedia.org/w/index.php?title=Maria_Montessori&oldid=108873891. [Último acceso: 15 Junio 2018].

[5] M. J. ROLDÁN, «Tareas del hogar para niños según la edad,» Etapa Infantil, 4 Junio 2016. [En línea]. Available: https://www.etapainfantil.com/tareas-hogar-ninos-segun-edad. [Último acceso: 15 Junio 2018].

[6] J. Delgado, «Tareas para promover la autonomía infantil en el hogar siguiendo el método Montessori,» Etapa Infantil, 13 Diciembre 2017. [En línea]. Available: https://www.etapainfantil.com/tareas-promover-autonomia-infantil-hogar-metodo-montessori. [Último acceso: 15 Junio 2018].

[7] E. Enge, «Mobile vs Desktop Usage in 2018: Mobile takes the lead,» Stone Temple, 27 Abril 2018. [En línea]. Available: https://www.stonetemple.com/mobile-vs-desktop-usage-study/. [Último acceso: 10 Mayo 2018].

[8] M. Kearl, «Braze,» Braze, 28 Julio 2016. [En línea]. Available: https://www.braze.com/blog/in-app-purchase-stats/. [Último acceso: 10 Mayo 2018].

[9] P. Mulder, «MoSCoW Method,» ToolsHero, 1 Agosto 2017. [En línea]. Available: https://www.toolshero.com/project-management/moscow-method/. [Último acceso: 12 Mayo 2018].

[10] Gartner, «Gartner Says Worldwide Sales of Smartphones Grew 9 Percent in First Quarter of 2017,» Gartner, Egham, U.K, 2017.

[11] Android developers, «Arquitectura de la plataforma,» Google, 25 Abril 2018. [En línea]. Available: https://developer.android.com/guide/platform/.

[12] J. Burns, «DEVELOPING SECURE MOBILE APPLICATIONS FOR ANDROID,» iSEC PARTNERS, 2008.

[13] Wikipedia contributors, «Android version history,» Wikipedia, The Free Encyclopedia., 29 Mayo 2018. [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Android_version_history&oldid=843258992.

[14] Google Developers, «Conoce Android Studio,» 25 Abril 2018. [En línea]. Available: https://developer.android.com/studio/intro/.

[15] Pencil Project, «Pencil,» Evolus, 2008. [En línea]. Available: pencil.evolus.vn.

[16] Pivotal Software, «Spring Boot,» Pivotal Software, 2018. [En línea]. Available: https://spring.io/projects/spring-boot.

[17] Pivotal Software, «Spring Data,» Pivotal Software, 2018. [En línea]. Available: https://spring.io/projects/spring-data.

[18] Pivotal Software, «Spring Security,» Pivotal Software, 2018. [En línea]. Available: https://projects.spring.io/spring-security/.

[19] Google, Inc., «Angular,» Google, Inc., 2018. [En línea]. Available: https://angular.io/docs.

[20] Microsoft, «TypeScript,» Microsoft, 2018. [En línea]. Available: https://www.typescriptlang.org/index.html.

[21] Jackson Project Contibutors, «FasterXML/jackson - Main Portal page for the Jackson project,» 28 Marzo 2019. [En línea]. Available: https://github.com/FasterXML/jackson.

[22] JHipster, «JHipster - Technology stack,» JHipster, 2018. [En línea]. Available: https://www.jhipster.tech/tech-stack/.

[23] Internet Engineering Task Force (IETF), «JSON Web Tokens Introduction,» Mayo 2015. [En línea]. Available: https://jwt.io/introduction/. [Último acceso: 10 Junio 2018].

[24] R. J. V. Del Valle y J. P. M. Granados, «Programación en Capas.,» Universidad de Costa Rica, Ciencias de Computación e Informática, Costa Rica, 2007.

[25] Google Developers, «Material Design,» Google, 2018. [En línea]. Available: https://material.io. [Último acceso: 13 Junio 2018].