ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE LOS COMPUTADORES

APLICACIÓN MÓVIL PARA LA GESTIÓN DE LOS GASTOS EN PISOS COMPARTIDOS

MOBILE APPLICATION FOR MANAGEMENT OF EXPENDITURES IN SHARED FLATS

Realizado por
Don Álvaro Marjalizo Aguilera
Tutorizado por
Don Eduardo Guzmán de los Riscos
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre 2015

Fecha defensa:
El Secretario del Tribunal

Resumen: En esta memoria se describe el diseño y las pautas seguidas para la construcción de una aplicación móvil que permite la creación de grupos de personas para la gestión de los gastos, facturas y deudas comunitarias. Con un soporte Front-End en Android, un Back-End desarrollado en PHP y un almacenamiento de la información en una base de datos NoSQL, concretamente en MongoDB. El proyecto se ha realizado en grupo, separando los módulos Front-End y Back-End en dos proyectos distintos. En este proyecto desarrollamos el módulo Back-End. En él encontraremos el diseño para la construcción de la arquitectura REST y dar solución a la comunicación cliente y servidor de la aplicación. Además se ha añadido la componente PaaS (Platform as a Service) para acercar el desarrollo de este proyecto a un entorno de producción más real, afrontando así problemas reales. Al tratarse de un proyecto en equipo, el uso de metodologías ágiles cobra más importancia, por ello en este proyecto se ha hecho uso de la metodología Scrum.

Palabras claves: PHP, MongoDB, Paas, Back-End, OpenShist, Unix, Scrum, KanbanFlow, NoSQL, JSON, REST, Git, Scrum, KanBanFlow.

Abstract: In this document we can find the design and the guidelines followed in the construction of a mobile application enabling the creation of groups of people to management expenses, bills and debts in a community. With Android support in Front-End, Back-End module has been developed in PHP with information storage in a NoSQL database (more concretely, in MongoDB). The project was accomplished in group, separating the modules Front-End and Back-End in two different projects. In this project we have developed the Back-End module. This document contains the design for construction of REST architecture and the procedure used to communicate the client and server of the application. Moreover it has been added a PaaS component (Platform as a Service) to bring the development of the project to a more real production environment, able to face real problems. Since it is a team project, the use of agile methodologies is more important, thus in this project we have been used the Scrum methodology.

Keywords: PHP, MongoDB, Paas, Back-End, OpenShist, Unix, Scrum, KanbanFlow, NoSQL, JSON, REST, Git, Scrum, KanBanFlow.

Índice de Contenido

Índice de Figuras

1. Introducción

La situación actual que estamos viviendo en el mercado de las aplicaciones móviles podría ser comparada con la Fiebre del Oro que sufrió California en el siglo XIX. Son muchas las empresas grandes o pequeñas las que se están beneficiando de este auge. Y sí, con la evolución de los smartphones y la aparición de las tablets la vida que nos rodea nos parece mucho más cómoda.

La problemática que supone llevar al día las gestión del dinero, ya sea en un grupo de amigos, en la familia, en tu piso compartido,… ha sido un problema recurrente a lo largo de la historia del ser humano. En la actualidad estamos en pleno auge de los smartphones, tablets,… que nos facilitan un poco la vida con pequeñas aplicaciones para dar solución a problemas cotidianos como lo es este. En este proyecto se ha propuesto la creación de una aplicación móvil para plataformas Android apoyada en una base de datos NoSQL (las cuales difieren del modelo clásico del sistema de gestión de bases de datos relacionales) y un back-end desarrollado en PHP que permitirá a los usuarios controlar sus deudas entre sus grupos de gente conocida, dando lugar a solución cómoda y eficaz.

Este trabajo consta de dos partes bien diferenciadas y que hoy día tan familiarizados estamos con ellas en el ámbito tecnológico. El front-end (el módulo donde el usuario va a desarrollar sus funciones) y el back-end (el módulo que procesa las entradas desde el front-end y está oculto para el usuario). Se ha optado por dividir el trabajo en estas dos partes que han sido realizadas por dos personas distintas. En esta parte se desarrolla el módulo back-end que está realizado por Álvaro Marjalizo.

La idea del trabajo en equipo es una idea que existe desde el momento en que el ser humano comenzó a vivir en sociedades y requirió para ello la colaboración de todos los miembros de una comunidad. La importancia del trabajo en equipo surge entonces por el hecho de que mientras más personas se comprometan en la realización de una actividad, mejores y más efectivos

serán los resultados. Por ello pretendemos añadir, mi compañero y yo, el componente del trabajo en equipo, tan esencial hoy en día y que nos vamos a encontrar en cualquiera de las empresas que hoy compiten en nuestro país.

Pero el trabajo en equipo no es una tarea sencilla de llevar a cabo y si no se sigue una buena estrategia desde el principio los resultados finales en cuanto a calidad, tiempo y satisfacción del usuario final no serán los esperados al inicio del proyecto. Surge así el desarrollo ágil de software, que se refiere a métodos de ingeniería de software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones se consiguen mediante la colaboración de grupos auto-organizados. En los últimos años han aparecido una serie de métodos que están teniendo una gran aceptación en las empresas, uno de ellos es la metodología Scrum que seguimos en la realización de este proyecto.

Por último, para acercar aún más el desarrollo del proyecto a un entorno de producción, afrontando así problemas reales, se va a alojar el servidor de aplicaciones en una PaaS (Platform as a Service), concretamente la ofrecida por Redhat (Openshift).

1.1 Objetivos

Se debe construir una aplicación móvil que permita a los usuarios registrarse en ella cómodamente, si no lo han hecho anteriormente, haciendo uso del protocolo OAuth, y autenticarse para acceder a ella. Una vez han accedido a la aplicación los usuarios podrán ver y acceder en todo momento a los distintos grupos en los que esté añadido, así como la posibilidad de crear nuevos grupos. El usuario podrá realizar invitaciones a los demás usuarios, mediante su dirección de correo electrónico. Si el usuario invitado no ha hecho un registro en la plataforma, tendrá que pasar por este paso antes de unirse al grupo.

Dentro de cada grupo, los usuarios pueden crear las facturas que ellos deseen, añadiendo a ellas los usuarios del grupo que ellos quieran. En estas

facturas se indica la cantidad deudora que cada usuario que ha sido añadido debe pagarle al creador del grupo. Sólo el usuario deudor podrá marcar como pagada la cantidad deudora, una vez haya realizado el pago. Por último los usuarios podrán modificar tanto su información personal, como la información del grupo y la información de las facturas en todo momento.

Este proyecto ha sido desarrollado en grupo, separando los módulos Back-End y Front-End. El desarrollo del Front-End en dos Trabajos de Fin de Grado diferentes. El desarrollo del Front-End lo realizará el alumno Diego Ojeda García en otro proyecto aparte. El desarrollo del Back-End se describe en este proyecto. Al ser un proyecto desarrollado mediante la colaboración en equipo, la importancia de cumplir los plazos de entrega y construcción de los diferentes módulos cobra mayor importancia, puesto que el trabajo que cada uno realiza está fuertemente ligado al trabajo del compañero.

En la figura 1 observamos el esquema de la arquitectura del proyecto, donde se pueden ver diferenciadas ambas partes. Por un lado tendremos el Front-End que será el que posibilite la interactuación de los usuarios con la plataforma, además hará uso de la API de Google Plus para que el registro de los usuarios sea más cómodo. En el otro lado tenemos el módulo Back-End que será el encargado de procesar las peticiones de los usuarios y almacenarlos en una base de datos.

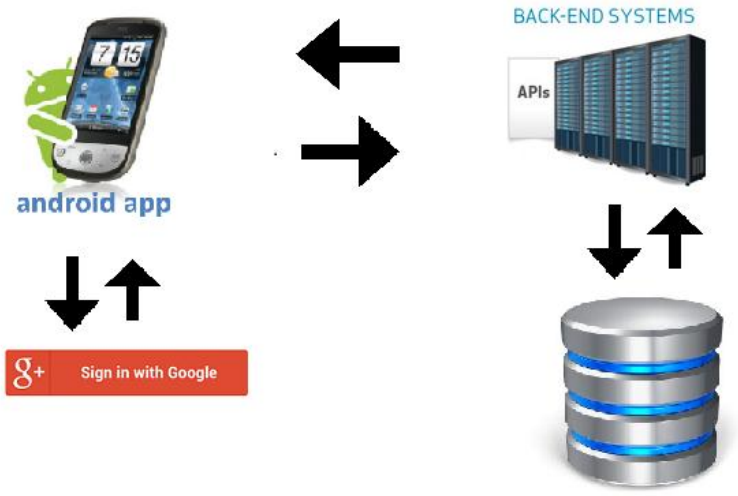

Figura 1: Arquitectura del Sistema

1.2 Características técnicas del proyecto

Los aspectos técnicos que van a caracterizar este TFG se enumeran a continuación:

Podemos comenzar con la utilización de la **arquitectura REST** para la comunicación entre el cliente y el servidor. En esta misión tendremos que ser capaces de publicar *recursos.* Un recurso se puede considerar como una entidad que representa un concepto de negocio que puede ser accedido públicamente. Cada recurso posee un identificador único y global que lo distingue de cualquier otro recurso. Dada una URI (el identificador único global) y mediante el protocolo HTTP deberemos operar sobre estos recursos. La operación a realizar se especifica mediante la acción HTTP adecuada, como veremos más adelante en el diseño del sistema.

La representación de estos recursos se especifican mediante los llamados tipos mime. El usar tipos mime estándar facilita la interoperabilidad. La mayoría de los tipos mime son estándares, como **JSON**, el cual deberemos emplear en la comunicación cliente-servidor.

Para el **almacenamiento de los datos** que son generados con la comunicación surgida entre el cliente y el servidor y que posteriormente son necesarios para la interoperabilidades del usuario con la aplicación se debe hacer uso de las bases de datos **NoSQL** que están en auge tras sus buenas respuestas en cuanto a rendimiento y sus propiedades en tiempo real**.** En este tipo de bases de datos, los datos no requieren estructuras fijadas como tablas, como ocurre en las entidades relacionales, normalmente no soportan operaciones JOIN ni garantizan ACID, pero suelen escalar bien horizontalmente.

El tiempo de cómputo generado en la gestión de las peticiones por parte del cliente en el lado del servidor deberá lograr buenos tiempos de respuesta en cuanto a **eficiencia**, permitiendo una velocidad cómoda de uso para el usuario en la aplicación. Se deberá hacer uso del lenguaje **PHP** en esta parte del proyecto. Velocidad, estabilidad, seguridad y simplicidad son algunas de las características que representan a unos de los lenguajes más populares en el lado del servidor.

Para la gestión de los diversos cambios que se realizan sobre las partes del proyecto se utilizará el control de versiones. Aunque un sistema de **control de versiones** puede realizarse de forma manual, hoy día es obligatorio disponer de herramientas que faciliten esta gestión. Estos sistemas facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas.

Aunque el proyecto está compuesto por dos partes bien diferenciadas, el hecho de que estén realizadas por dos personas diferentes hace que las etapas y fechas fijadas para la realización de las partes cobre mayor importancia. Por eso es imprescindible el uso de una de las metodologías ágiles de software existentes. **Scrum** es una de las más utilizadas hoy día en el sector empresarial, adopta una estrategia incremental y basa la calidad del resultado en mayor grado en el conocimiento táctico de las personas que componen el proyecto en vez de en la calidad de los procesos empleados. Por ello será de vital importancia crear y mantener una buena **comunicación e integración** de los componentes del proyecto entre las partes del equipo.

Por último para acerca el desarrollo del proyecto a un entorno de producción, afrontando así problemas reales, el back-end deberá estar alojado en un servidor externo para que las peticiones no solo sean locales. Para ello se usará una de las **PaaS** (Platform as Service) que hay disponibles y gratuitas. Una PaaS es una plataforma de software para las cuales la herramienta de desarrollo en si *misma* está alojada en la nube. Los desarrolladores pueden construir aplicaciones web sin tener que instalar ninguna herramienta en su computadora.

1.3 Antecedentes

En el siguiente capítulo describimos algunas de las aplicaciones similares que ya existían antes de la creación de *Appartment*. No entraremos en detalles sino más bien haremos una breve descripción de algunos de las apps que ya han sido utilizadas con anterioridad para situarnos más cerca del problema.

Nuestra aplicación está orientada a efectuar pagos entre una comunidad de gente conocida. Permite la creación de grupos de personas y cargar facturas que se hayan generado entre esos grupos de gente permitiendo llevar un control eficaz y un pago cómodo. A continuación se muestran otras ideas parecidas a la nuestra que ya están en el mercado:

1.3.1 Yaap Money

Una aplicación desarrollada por Yapp (alianza de CaixaBank, Banco Santander y Telefónica) que permite a sus usuarios enviar y recibir dinero de móvil a móvil con solo un mensaje, sin dar el número de cuenta o conocer los datos bancarios del receptor. Permite la transferencia con únicamente el número de teléfono del destinatario o su nombre en redes sociales como Facebook o Twitter.

El método para comenzar a utilizar el servicio es muy sencillo e intuitivo: se crea la cuenta, se asocia una tarjeta de crédito o de débito y solo con eso ya podría empezar a enviar dinero a cualquiera de los contactos de su agenda.

La aplicación funciona como intermediario, ya que recarga el dinero y a partir de ahí es el usuario el que lo distribuye según desea. Lo mismo sucede con el destinatario: puede seleccionar mantener la cantidad como saldo de Yaap recuperarlo en la correspondiente tarjeta asociada.

1.3.2 PagaMobil

Pagamobil es una aplicación hecha para realizar pagos para facturas. Lo que busca es la comodidad del usuario a la hora de afrontar el pago de una factura en un banco, con la pérdida de tiempo en colas que ello supone. Construida por una empresa desde México. De momento solo se puede pagar

las facturas para un determinado grupo de servicios (entre ellos Gas Natural, Cablevisión, Telmex …).

Para realizar los pagos el usuario debe hacer un registro de datos y de tarjeta de crédito. Tras ello con la cámara del móvil se leerá el código de barras y la app lo escanea. El pago se puede hacer vía tarjeta de crédito o con Paypal. Lama la atención su opción para pagar facturas aunque ya estén vencidas.

1.3.3 Momo Pocket

Momo Pocket es una alternativa real al uso de dinero en efectivo y de tarjetas u otros instrumentos de pago al por menor y comercio electrónico, tanto para clientes bancarizados como no bancarizados. Uno de los valores fundamentales es el abaratamiento de costes por transacción diferencial a los modelos existentes en el mercado; es decir, aplica una baja comisión de descuento para el comercio y gratuidad total para el cliente. Funcionamiento sencillo y la gratuidad, tanto para comerciantes como para clientes.

Desarrollada en Málaga, Momo Pocket funciona como un monedero electrónico. Necesitamos crear una cuenta en la web oficial del producto. Una vez tengamos la cuenta le añadimos saldo, con un mínimo de 10 euros, desde la propia aplicación o desde la web, con cargo a nuestra tarjeta de crédito. El saldo es reintegrable y lo podemos recuperar cuando queramos.

En cuanto al funcionamiento, es tan sencillo como abrir la aplicación, mostrar la ventana de pago al comerciante, que escaneará un código (de barras, QR o cadena de caracteres) y pago realizado. Momo Pocket cuenta con un mecanismo de seguridad, para evitar que alguien pueda gastar el saldo si se apodera de tu móvil: para mostrar la ventana de pago hay que introducir un PIN.

1.3.4 Square

Square es un sistema distinto de los vistos hasta aquí. Es un servicio de pagos (o cobros) enfocado a los pequeños comerciantes, una especie de TPV (aunque trasciende el ámbito de ese concepto), que emplea la tarjeta de crédito del cliente para realizar los pagos.

Square requiere de un dispositivo lector de tarjetas propio (esto representa un inconveniente frente a los productos mencionados en este artículo), que recoge la información y valida la transacción mediante la firma del comprador.

Para el cliente tiene el valor añadido, frente al pago con tarjeta tradicional, de que una vez validado el pago mediante la conexión de datos del móvil, se puede solicitar un comprobante de la operación mediante SMS o correo electrónico, que puede incluir una imagen del producto y la geolocalización de la tienda.

Square es un producto para el comerciante, que sigue un modelo propio del mercado de EEUU, y por ello parece ser poco atractivo para el mercado español, según han analizado en la publicación Pymes y Autónomos. No obstante puede tener sus ventajas para la venta ambulante, al poder realizar las operaciones vía 3G o WiFi.

2. Tecnologías y herramientas utilizadas

A continuación se expone brevemente las diferentes tecnologías que han sido empleadas en la construcción del back-end del proyecto.

2.1 PHP

Para el desarrollo del código del servidor se ha optado por unos de los lenguajes de programación de uso general más utilizados. PHP (Hypertext Pre-processor) se considera uno de los lenguajes más flexibles, potentes y de alto rendimiento conocidos hasta el día de hoy. Es un lenguaje que forma parte del software libre.

Para los que son principiantes en este lenguaje, tiene una curva de aprendizaje muy baja, su sintaxis es simple y cumple los estándares de la programación orientada a objetos. No son necesarios complejos entornos de desarrollo, son de rápida y fácil configuración. Además tiene un fácil despliegue a la hora de instalar los diferentes paquetes que integran PHP.

Pero sin duda una de las claves de PHP es que probablemente tenga una de las comunidades en Internet más grande con respecto a otros lenguajes. Ya que encontramos en esta comunidad soporte, documentación, componentes librerías y soluciones a casi cualquier duda que pueda surgir.

2.2 Mongo

Dentro de las bases de datos NoSQL, probablemente una de las más famosas sea MongoDB. Es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda datos en documentos. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Mongo DB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Además viene con una consola construida sobre JavaScript, por lo que las consultas se realizan utilizando este lenguaje.

Cualquier aplicación que necesite almacenar datos semi-estructurados puede usarlo. Es el caso de las típicas aplicaciones CRUD. Aunque no se necesita definir un esquema, es importante que diseñemos nuestra aplicación para seguir uno. Tendremos que analizar si necesitamos normalizar los datos, denormalizarlos o utilizar una aproximación híbrida. Es especialmente útil en entorno que requiera escalabilidad. Podremos conseguir un sistema que escale horizontalmente sin demasiados problemas.

No existen las transacciones, solo garantiza operaciones atómicas a nivel de documento, Tampoco existen los JOINS. Para consultar datos relacionados en mas de una colección tendremos que realizar mas de una consulta. Aunque dispone de las consultas de agregación y Map Reduce, estos métodos no llegan a la potencia de un sistema relacional.

2.3 Openshift

Es una “Plataforma como servicio” (PaaS) gratuita de Red Hat, cuyo objetivo es el despliegue de aplicaciones en la nube. Con ello la plataforma logra abstraer a los desarrolladores de todo lo relacionado con la

infraestructura. En ella podremos desplegar aplicaciones Java, Perl, PhP,… Además permite la instalación de un servidor de bases de datos como MySQL, Postgres o MongoDB. Al abrir una cuenta en Openshift, se generará una URL única para tu aplicación y un repositorio Git asociado.

La portabilidad de las aplicaciones entre entornos de tiempo de ejecución distintos permite una mayor libertad a la hora de desarrollar las aplicaciones, ya que OpenShift es íntegramente OpenSource y compatible con los estándares abiertos.

2.4 Unix

Es el sistema operativo utilizado para la realización del proyecto. Es un sistema operativo de tiempo compartido que controla los recursos de una computadora. Permite a sus usuarios correr sus programas. Dispone de un lenguaje de control programable llamado “Shell”, por lo que el sistema presenta comandos de usuario para iniciar y manipular procesos concurrentes. Además ofrece facilidades para la creación de programas y sistemas y el ambiente adecuado para las tareas de diseños de software. Unix garantiza un alto grado de portabilidad.

2.5 Git

Es un software de control de versiones, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Es un sistema distribuido SCM (Source Code Management), una herramienta que nos resuelve una serie de problemas para el trabajo de código fuente.

¿Qué nos aporta Git? Auditoría del código, nos permite saber en cada momento quién y cuándo ha tocado el código. Podemos volver hacia atrás de una forma rápida con cualquier cambio nuevo. Todas las estructuras internas

de datos están firmadas con SHA1, por lo que no se puede cambiar el código sin que nos enteremos y por supuesto mejora la capacidad para trabajar en equipo.

2.6 KanBan Flow

Es una aplicación de productividad gratuita que nos permite crear listas de tareas y compartirlas con nuestros compañeros de proyecto. Es la herramienta utilizada que nos facilitará el seguimiento para la realización de Scrum.

Utiliza el clásico gestor de tareas conocido como to-do-list. También presenta un método de organización en columnas. Se podría decir que es una suite de herramientas para mejorar la productividad. Para utilizar KanBan Flow basta con registrarse utilizando una cuenta de correo. La web está toda en inglés.

3. Especificación y análisis

En los capítulos siguientes se definirá técnicamente “cómo se debe comportar” el servidor que hemos preparado para procesar y dar respuesta a las peticiones clientes. La especificación de requisitos se apoya en los datos obtenidos durante el análisis, dando una descripción completa del comportamiento del sistema que se va a desarrollar; definiéndose los datos e información que el producto final tratará.

Se presentará en detalle el sistema mostrando cómo interviene el programa cliente en el sistema, cómo hace uso de él, cuáles son los componentes que integran el sistema y qué operaciones soportan estos componentes. Para ello se hará uso del lenguaje de modelado unificado, UML, como notación estándar para el análisis de requisitos y especificación del sistema. Las técnicas que incorpora UML son sencillas y dan una descripción del sistema que está más cerca de lo que el usuario espera obtener.

3.1 Tipología de las peticiones

Para la realización de este proyecto se ha utilizado un modelo de desarrollo incremental, consistente en la iteración de las fases de análisis, diseño, codificación y prueba para cada módulo independiente de la aplicación, uniéndose posteriormente.

A continuación se describe el formato con el que se van a tratar las peticiones provenientes del cliente. Diferenciaremos entre peticiones GET y POST, son los dos tipos de peticiones que se va a tratar para este proyecto:

- ***PETICIONES GET:***

Las peticiones GET se usarán para obtener información del servidor construido. Traer datos que están en el servidor, provenientes de la base de datos, independientemente de que para ello el cliente deba enviar alguna información adicional.

La estructura de la petición será la siguiente:

Dirección base + entidad + identificador + parámetros.

- *Dirección base*: Es la llamada a la url donde está alojado nuestro servidor, siempre será la misma. Es la siguiente:

  http://appartment-pruebamarja.rhcloud.com

- *Entidad*: Será la entidad a la que la petición quiere hacer referencia y obtener su información. Como sabemos una entidad es la representación de un objeto o concepto del mundo real que se describe en una base de datos. En los siguientes capítulos se describen las distintas entidad de las que el proyecto se compone

- *Identificador*: Este parámetro solo será necesario para aquellas peticiones que lo requieran. El identificador se refiere a un objeto en concreto dentro del mundo de objetos del que se compone una entidad.

- *Parámetros*: parámetros varios que necesita la petición, como pueden ser los id de sesión.

- ***PETICIONES POST:***

Las peticiones POST se usarán para enviar información desde el cliente para que sea procesada y actualice o agregue información en la base de datos del servidor.

La estructura de la petición será la siguiente:

Dirección base + entidad + identificador + id sesión + acción

- *Dirección base*: Como hemos descrito anteriormente es la llamada a la url donde está alojado nuestro servidor:

  http://appartment-pruebamarja.rhcloud.com

- *Entidad*: Será la entidad a la que la petición quiere realizar algunos de los tipos de manejo que nuestro servidor controla. Los veremos en el punto de acciones.

- *Identificador*: Este parámetro solo será necesario para aquellas peticiones que lo requieran. El identificador se refiere a un objeto en concreto dentro del mundo de objetos del que se compone una entidad.

- *Id sesión*: El identificador de la sesión para que el servidor pueda obtener información almacenada en la sesión por el usuario.

- *Acción*: Indica qué tipo de tratamiento requiere la petición POST. Es enviado como parámetro y no aparecerá en la url. Nuestro servidor controla 3 tipos diferentes, son los siguientes:

  - Insert: Será la inserción de un nuevo objeto dentro de una entidad especificada por la petición.

  - Update: Actualizará la información de un objeto determinado por la petición.

  - Delete: Borrará el objeto que especifique la petición.

3.2 Especificación de las entidades

En el siguiente apartado trataremos de abordar las diferentes entidades que componen la base de datos. Como hemos utilizado una base de datos no relacional no tendremos un esquema relacional, pero abordaremos la estructura de cada entidad. También se describe el por qué se ha dado solución al mantenimiento de la información con una base de datos NoSQL. Precisamente, empezamos tratando esto último.

Como ya se ha mencionado anteriorment, para este proyecto se ha optado por utilizar una base de datos NoSQL. En este caso hemos optado por MongoDB, una de las principales razones es sin duda que Openshift da soporte para ello, pero además MongoDB da una buena solución para las aplicaciones que necesite almacenar datos estructurados o semi-estructurados, es el caso de las aplicaciones CRUD (Create, Read, Update y Delete), como lo es esta, donde hemos visto en el anterior capítulo que el Read se hace en la petición Get y el Create, Update y Delete se hace en los POST. Además PHP (el lenguaje utilizado en el servidor) tiene un muy buen soporte y una magnifica estabilidad de la API para MongoDB, de muy fácil instalación. Como los datos vienen estructurados por parte del cliente, la inserción de ellos se hace muy cómodamente, a través del formato BSON, es el formato de datos usado para almacenamiento y transferencia de datos por MongoDB. Es una representación binaria de estructuras de datos y mapas. El nombre BSON está basado en el término JSON y significa Binary JSON.

Ya hemos dado buenas razones del porqué del uso de MongoDB para esta aplicación. A continuación nos adentramos en las distintas entidades que conforman el mundo de Appartment y las tablas que generan cada una de ellas.

- **Entidad Usuario:**

Esta entidad engloba las características de un usuario Appartment. La generación de esta entidad genera tres diferentes tablas: la del propio usuario, la tabla que contiene las características de los dispositivos del usuario y una

última tabla en la que establece relación de amistad con diferentes usuarios de Appartment. Vamos a ver cada una de las tablas a continuación:

- *User:* Esta colección contiene la información del usuario extraída al hacer login. Los elementos que la componen son:
    - _id: identificador que genera mongo al hacer insert del objeto.
    - gplus: identificador generado por Google Plus.
    - cdt: timestamp, que es una secuencia de caracteres que denotan la hora y la fecha en la cual ocurrió la inserción del objeto.
    - last_login: timestamp que denota la hora en la que el usuario hizo el último acceso a la aplicación.
    - session: identificador de la última sesión del usuario.
    - name: nombre del usuario.
    - lang: idioma del usuario. El que genera Google Plus.
    - email: dirección de correo del usuario.
    - picture: url donde está la imagen del usuario almacenada.

- *User.devices:* Esta colección contiene la información del dispositivo del usuario. El usuario podría tener varios documentos user.devices que pertenezcan a él, como tantas autenticaciones en la plataforma haya realizado con distintos dispositivos.
    - _id: identificador que genera mongo al hacer insert del objeto.
    - gplus: identificador de Google Plus que indica a qué usuario pertenece ese dispositivo.
    - hw_id: identificador único del dispositivo.
    - plataforma: tipo de plataforma del dispositivo.
    - dev_brand: marca del dispositivo.
    - os_version: número de versión de la plataforma del dispositivo.
    - dev_model: modelo del dispositivo.
    - lang: idioma del dispositivo.

- token: token del dispositivo para poder lanzar notificaciones.
- cdt: timestamp de la hora y la fecha en la cual ocurrió la inserción del objeto.
- last_longin: timestamp que denota la hora en la que el usuario hizo el último acceso a la aplicación.
- notif: flag que indica si a ese dispositivo se le envían notificaciones o no.

- *User.friends.:* Esta colección contiene los usuarios que son amigos del usuario propietario de la tabla.
  - _id: identificador que genera mongo al hacer insert del objeto.
  - gplus: identificador de Google Plus que indica a qué usuario pertenece esta tabla.
  - friends: array que contiene la lista de los identificadores Google Plus de los que el usuario es amigo.

❖ Grupo:

La entidad grupo representa las características para la asociatividad de más de un usuario en un conjunto. Esta asociatividad permite en el futuro realizar acciones que solo se pueden realizar dentro de un grupo. La colección que genera es la siguiente:

- *Group:* Esta colección contiene la información necesaria para mantener a una serie de usuarios en una comunidad.
  - _id: identificador que genera mongo al hacer insert del objeto.
  - cdt: timestamp de la hora y la fecha en la cual ocurrió la inserción del objeto.
  - creator: identificador Google Plus del usuario que inició el grupo.

- members: array que contiene los identificadores Google Plus de los usuarios que pertenecen al grupo.
- name: nombre del grupo.
- picture: url donde está la imagen del usuario almacenada.

❖ Recibo:

La entidad recibo reúne la información para que los usuarios puedan crear diferentes recibos dentro de un mismo grupo.

- *Receipt:* Esta colección contiene la información necesaria para la generación de recibos.
  - _id: identificador que genera mongo al hacer insert del objeto.
  - cdt: timestamp de la hora y la fecha en la cual ocurrió la inserción del objeto.
  - creator: identificador Google Plus del usuario que generó el recibo
  - debtors: array que contiene los identificadores Google Plus de los usuarios que adeudan dinero.
  - idgroup: identificador del grupo al que pertenece el recibo.
  - amount: cantidad que deben los deudores.
  - flag: boolean que indica si el recibo esta pagado por todos lo miembros o no.
  - name: nombre del recibo.

❖ Inicio:

La entidad inicio establece los parámetros inicales para que el usuario pueda realizar un login en la aplicación.

- *Access:* Esta colección controla los accesos de los usuarios. Registra las acciones de los usuarios.
    - _id: identificador que genera mongo al hacer insert del objeto.
    - sesión_id: identificador de la nueva sesión creada para el usuario.
    - cdt: Timestamp de la hora y la fecha en la cual ocurrió la inserción del objeto.
    - hw_id: identificador del dispositivo.
    - platform: plataforma desde la que se ha accedido.
    - os_version: número de versión de la plataforma del dispositivo.
    - app_version: número de la versión de la aplicación
    - ip: ip de acceso del usuario.
    - action : acción que se está realizando dentro de la aplicación.

3.3 Modelo dinámico

A continuación se describe cómo interactúan los elementos que componen el servidor. Para ello empleando el lenguaje UML utilizaremos un diagrama de actividades. Un diagrama de actividad es provechoso para entender el comportamiento a alto nivel de la ejecución del sistema, sin profundizar en los detalles internos de los mensajes.

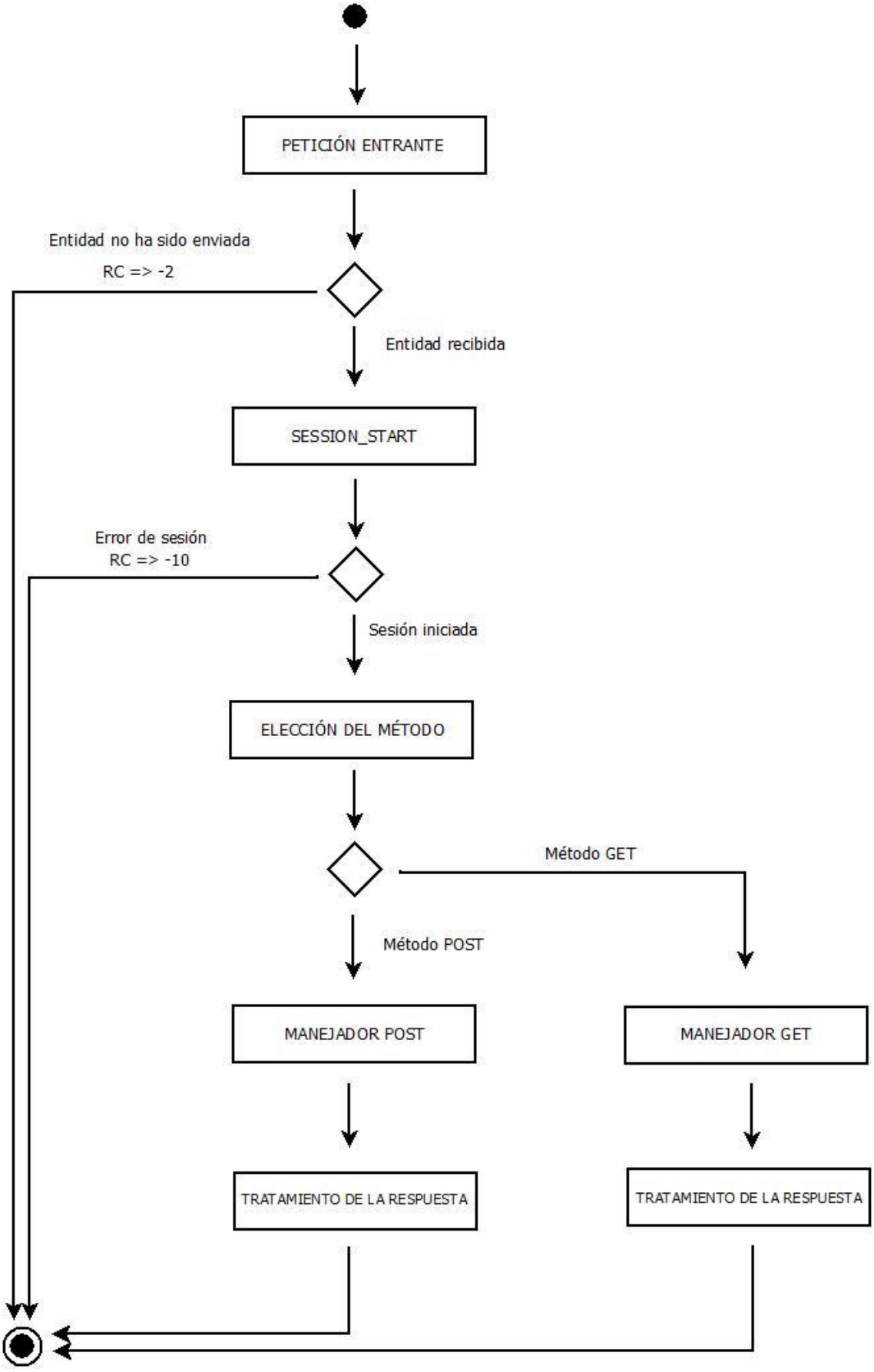

Figura 2: Diagrama general de la actividad del servidor

El diagrama simula la llegada de una petición al controlador frontal del servidor, éste será el encargado de no permitir la entrada a los siguientes módulos a las peticiones que vengan sin entidad, devolviendo en el mensaje de respuesta un código de error de -2. Si la petición contiene la entidad a la que se quiere hacer la petición el siguiente proceso será el de iniciar la sesión. Si en este proceso ocurriera algún fallo, el controlador devolvería en el mensaje un código -10 y finalizaría la petición.

Si todo ha ido correctamente en los procesos anteriores, el manejador frontal llevará, dependiendo de qué tipo haya sido la petición entrante, el flujo de información a los manejadores de peticiones Post y Get. Estos harán el manejo adecuado y devolverán la respuesta generada tras su proceso. Este proceso es detallado en capítulos posteriores.

4. Diseño del sistema

Una vez se ha realizado el análisis del sistema, se precisa decidir la forma de aproximarse al diseño. El diseño del sistema es la estrategia de alto nivel para resolver el problema y construir una solución. Este incluye decisiones acerca de la organización del sistema en subsistemas para comprender la mejora de la estructura del sistema, decisiones fundamentales conceptuales y de política que son las que constituyen un marco de trabajo para el diseño detallado del sistema.

El diseño del sistema es la primera fase de diseño en la cual se selecciona la aproximación básica para resolver el problema. Durante el diseño del sistema, se decide la estructura y el estilo global. La arquitectura del sistema es la organización global del mismo en componentes llamados subsistemas. Esta arquitectura del sistema representa los diferentes subsistemas o paquetes de los que constará el sistema. Lo más importante en este punto es definir correctamente la misión y los límites de ningún subsistema ya que llevaría a un sistema mal estructurado. Las dos siguientes secciones tendrán el contenido que se describe a continuación.

- Estructura del Sistema: donde se indicarán los módulos que forman el sistema, su función principal y las funciones principales asociadas al mismo.
- Diseño de las peticiones del sistema: En este apartado se realizará un diseño de las peticiones que forman el sistema, se indicarán los elementos y atributos que la forman y se indicará cual es el objetivo fundamental de cada una de ellas.

4.1 Estructura del sistema

La definición de la estructura del sistema es el primer y más importante paso dentro del mismo, ya que en base a ella se realiza todo el diseño.

Cada una de las estructuras o paquetes de funciones definidas abarca aspectos del sistema que comparten alguna propiedad en común (una funcionalidad similar, la misma ubicación física, préstamos de información, etc.). Una estructura no es más que un paquete o conjunto de datos de diferente tipo que representan la funcionalidad de un agente dentro del sistema. A cada uno de los datos o elementos almacenados dentro de una estructura se les denomina miembros.

En la figura 3 podemos ver la estructura del sistema, los diferentes módulos en los que se ha dividido el sistema completo. A continuación haremos una breve referencia de cada una de ellas.

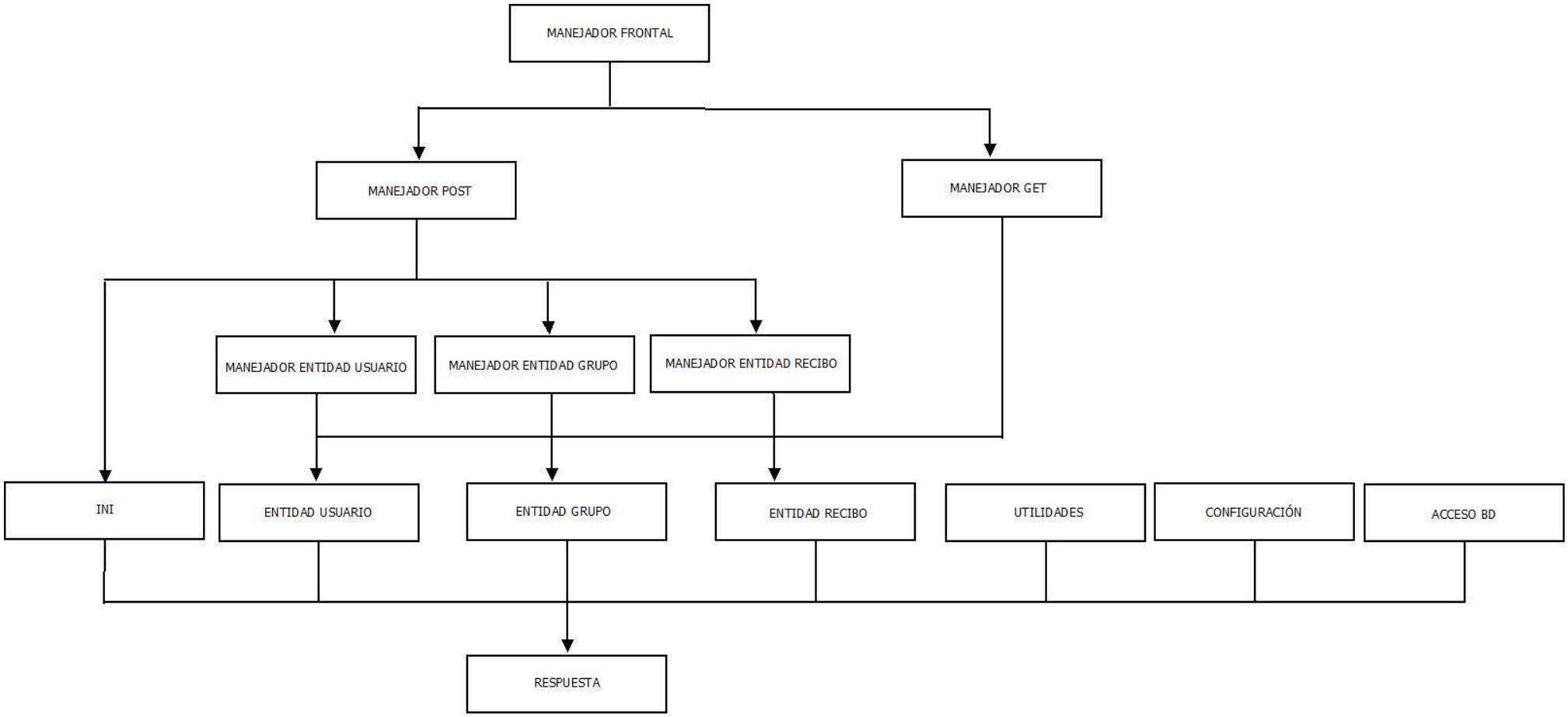

Figura 3: Estructura del Sistema

- **Manejador frontal:** es la puerta de entrada al servidor, todas las peticiones pasan por él. Es el encargado de dirigir las peticiones a los manejadores secundarios o rechazar una petición no válida.
- **Manejador POST:** recibe todas las peticiones de tipo POST, y se encarga de dirigir la petición a los manejadores de entidades. Cuando la petición es de tipo Ini la dirige directamente al módulo. Rechaza peticiones con entidades que no existen y peticiones que no incluyen acciones.
- **Manejador GET:** recibe todas las peticiones de tipo GET y las dirige directamente al módulo requerido. Como en el anterior manejador, rechaza peticiones con entidades no reconocidas.
- **Manejador Entidad Usuario:** recibe todas las peticiones POST para la entidad usuario y las dirige al módulo perteneciente. Rechaza acciones no reconocidas.
- **Manejador Entidad Grupo:** recibe todas las peticiones POST para la entidad grupo y las dirige al módulo perteneciente. Rechaza acciones no reconocidas.
- **Manejador Entidad Recibo:** recibe todas las peticiones POST para la entidad recibo y las dirige al módulo perteneciente. Rechaza acciones no reconocidas.
- **Ini:** es el módulo encargado de inicializar sesiones antes de que un usuario realice un login.
- **Entidad usuario:** módulo que engloba todas las peticiones que puede realizar un usuario.
- **Entidad grupo:** módulo que engloba todas las peticiones que se puede realizar en un grupo.
- **Entidad recibo:** módulo que engloba todas las peticiones que se puede realizar en un recibo.
- **Utilidades:** módulo auxiliar que engloba numerosas funciones que son necesarias para que las entidades puedan llevar a cabo la respuesta.
- **Configuración:** módulo con los archivos de la configuración para el acceso a la base de datos y el envío de emails.
- **Acceso a la BD:** módulo que reúne todas las funciones utilizadas por los módulos de entidades para el acceso a la base de datos.

4.2 Diseño de las peticiones

En este apartado se describen el conjunto de peticiones que forman el servidor. Las peticiones están agrupadas por entidades y se pueden extraer de la propia definición del problema. Para cada petición describiremos brevemente la función que realiza, indicaremos los parámetros que son necesarios en el envío, y mostraremos una respuesta a modo de ejemplo.

Comentar por último que en todas las peticiones será obligación enviar el identificador de la sesión, salvo en la petición de inicio, donde se crea este identificador.

4.2.1 Inicialización

Este es un bloque especial y diferente a los siguientes. Ni siquiera forma completamente una entidad, pero es un bloque esencial para la aplicación. Solo consta de una petición de tipo POST, como veremos a continuación. Permite inicializar a un usuario una sesión, sin ella el usuario nunca va a poder realizar ninguna de las peticiones que conforman el servidor. También realiza un insert en la tabla Access de la base de datos, indicando los parámetros iniciales del acceso del usuario.

Url: http://appartment-pruebamarja.rhcloud.com/ini/

Parámetros obligatorios:

- Hw_id : identificador del dispositivo con el que se está accediendo.

Parámetros opcionales:

- Lang: idioma del dispositivo.
- Platform: plataforma del dispositivo.
- Dev_brand: Marca del dispositivo.
- Dev_model: Modelo del dispositivo.
- Os_Version: Version de la plataforma.
- App_version: Versión de la aplicación.
- Token: Push token del dispositivo.

Respuesta:

{"rc":0,"session_id":"inv854en0t75t6sd6gt3arc742"}

4.2.2 Usuario

En la entidad Usuario encontramos las siguientes peticiones, las hemos calsificado según el tipo:

Peticiones GET:

- Get all friends

Esta petición devuelve todos los amigos del usuario.

Url:
http://appartmentpruebamarja.rhcloud.com/user?session_id=234434234324

Parámetros obligatorios:

- Ninguno

Respuesta:

{"friends":[{"gplus":"1111","name":"AlvaroMarjalizo"},{"gplus":"2222","name":"Lol
a Marjalizo"},{"gplus":"3333","name":"paco Marjalizo"}],"rc":0}

Peticiones POST

- Login

Realiza automáticamente una insercción o actualización tanto en la tabla user como en la tabla user.devices de la información personal del usuario. Además almacena esta información en la sesión. Utiliza la funcionalidad upsert de mongo para almacenar la información, upsert crea un nuevo documento cuando no realiza un match.

Url:
http://appartment-pruebamarja.rhcloud.com/user/?session_id=234434234324

Parámetros obligatorios:

- Action: login.
- gplus: identificador de google plus.
- email: dirección de correo del usuario.
- name: del usuario.
- lang: idioma que proporciona google plus.

Respuesta:

Estado de la inserción. Cualquier número distinto de 0 es que algo ha fallado, mirar la tabla de errores del capítulo siguiente.

{"rc":0}

- Update User

Permite al usuario actualizar su información personal.

Url:
http://appartment-pruebamarja.rhcloud.com/user/?session_id=234434234324

Parámetros obligatorios:

- Acción: update.

Respuesta:

{"rc":0}

4.2.3 Grupo

En la entidad Grupo encontramos las siguientes peticiones, como en el anterior apartado, también las hemos divido por tipo:

Peticiones GET:

- Group

Devuelve la información del grupo solicitado.

Url: http://appartment-pruebamarja.rhcloud.com/user/id_group?session_id=234434234324

Parámetros obligatorios:

- Identificador del grupo

Respuesta:

{"group":{"id":"55859d73192adaaed3d3902f","name":"adios","creator":"1111","members":[{"gplus":"1111","name":"Alvaro Marjalizo","email":"Amarjalizo89@gmail.com"},{"gplus":"2222","name":"Lola Marjalizo","email":"Amarjalizo89@gmail.com"},{"gplus":"3333","name":"paco Marjalizo","email":"lolaaa@gmail.com"}]},"rc":0}

- Groups

Devuelve la información de todos los grupos a los que pertenece el usuario que realiza la petición

Url: http://appartment-pruebamarja.rhcloud.com/group/?session_id=234434234324

Parámetros obligatorios:

- Ninguno

Respuesta:

{"groups":[{"id":"556ccc06e7a7ea954b8b4567","cdt":1433193478,"creator":1111,"name":"554a2532d839da6a398b4568","nmembers":1},{"id":"556d6920e7a7ea9e4b8b4569","cdt":1433233696,"creator":1111,"name":"Mi grupo","nmembers":1},"rc":0}

Peticiones POST

- Insert Group

Crea un nuevo grupo en la base de datos.

Url:
http://appartment-pruebamarja.rhcloud.com/group/?session_id=234434234324

Parámetros obligatorios:

- name : Nombre del grupo
- action: insert

Respuesta:

{"inserted":{"id":"55b2a635192adaaed3d39063","cdt":1437771317,"creator":1111,"name":"adios","nmembers":1},"rc":0}

- Update Group

Actualiza la información básica de un grupo

Url:http://appartmen-tpruebamarja.rhcloud.com/group/id_group?session_id=234434234324

Parámetros obligatorios:

- id_group : Identificador del grupo
- action: update.
- type : group

Respuesta:

{"rc":0}

- Add Member

Permite incorporar a un grupo ya creado un nuevo miembro o varios.

Url: http://appartment-pruebamarja.rhcloud.com/group/id_group?session_id=234434234324

Parámetros obligatorios:

- action: update.
- type: rmember
- idgroup: Identificador del grupo
- ids: Identificador/es del nuevo/s usuario/s

Respuesta:

{"rc":0}

- Delete Member

Permite a un usuario salir de un grupo. Solo el usuario en activo es el que puede darse de baja en un grupo. Ninguno de los demás usuarios podrá eliminarle. Si al salir un usuario del grupo este queda sin usuarios, el grupo es eliminado automáticamente.

Url: http://appartment-pruebamarja.rhcloud.com/group/id_group?session_id=234434234324

Parámetros obligatorios:

- Action: update.
- type: dmember
- idgroup: Identificador del grupo

Respuesta:

{"rc":0}

- Delete Group

Realiza una eliminación de un grupo creado. Esta función la puede realizar cualquier miembro del grupo.

Url: http://appartment-pruebamarja.rhcloud.com/group/id_group?session_id=234434234324

Parámetros obligatorios:

- action: delete.
- idgrup: Identificador del grupo

Respuesta:

{"rc":0}

- Invite

Este petición sirve para invitar a personas que aún no han interaccionado con la aplicación. Envía un correo electrónico a las personas que deseemos invitar con un link que será interpretado por el cliente para llevarle a la aplicación e introducirle en el grupo invitado. El envío del correo electrónico a cargo del servidor se realiza en segundo plano para no demorar la respuesta al cliente. Se ha utilizado la biblioteca PHPMailer para el envío de este.

Url: http://appartment-pruebamarja.rhcloud.com/group/?session_id=234434234324

Parámetros obligatorios:

- action: invite.
- emails: direcciones de correos de las personas a invitar.

Respuesta:

{"rc":0}

4.2.4 Recibo

En la entidad Recibo encontramos las siguientes peticiones:

Peticiones GET:

- Receipt

Devuelve la información de un recibo solicitado.

Url: http://appartment-pruebamarja.rhcloud.com/receipt/id_recibo?session_id=234434234324

Parámetros obligatorios:

- idgroup: Identificador del recibo

Respuesta:

{"receipt":{"id":"5585a202192adaaed3d39037","creator":{"gplus":"1111","name":"AlvaroMarjalizo","email":"Amarjalizo89@gmail.com"},"cdt":1434821122,"amount":"11","flag":0,"name":"Nuevonombreeee","users":[{"gplus":"1111","name":"Alvaro Marjalizo","email":"Amarjalizo89@gmail.com","paid":0}]},"rc":0}

- Group Receipt

Devuelve la información de los recibos pertenecientes a un grupo

Url:
http://appartmentpruebamarja.rhcloud.com/receipt?idgroup=asdda4234&session_id=234434234324

Parámetros obligatorios:

- idgroup: Identificador del grupo

Respuesta:

{"receipts":[{"id":"557d2eaaf3f896be29a5d0a1","creator":1111,"cdt":1434267306,"amount":"11","name":"Nuevonombreeee","flag":"0","numdebtors":1},{"id":"557d2eb4f3f896be29a5d0a3","creator":1111,"cdt":1434267316,"amount":"11","name":"Nuevo nombreeee","flag":"0","numdebtors":1},],"rc":0}

Peticiones POST

- Insert Receipt

Inserta un recibo en un grupo indicando qué miembros son los adeudores. Cualquier miembro de un grupo puede insertar un nuevo recibo

Url:
http://appartment-pruebamarja.rhcloud.com/receipt/?session_id=234434234324

Parámetros obligatorios:

- action: insert.
- debtors: Identificadores de los miembros deudores
- idgroup: Identificador del grupo.
- amount: Cantidad adeudadora.

- name: Nombre del recibo.

Respuesta:

{"inserted":{"id":"55b2b820192adaaed3d39074","cdt":1437775904,"creator":1111,"name":"Nuevoromper","amount":"11","flag":0,"idgroup":"5549d098d839da1dca8b4567","debtors":[{"gplus":"2222","name":"LolaMarjalizo","email":"Amarjalizo89@gmail.com"}]},"rc":0}

- Update Receipt`

Actualiza la información básica de un recibo.

Url: http://appartment-pruebamarja.rhcloud.com/receipt/idreceipt?session_id=234434234324

Parámetros obligatorios:

- action: update.
- yype: receipt.
- idreceipt: Identificador del recibo

Respuesta:

{"rc":0}

- Update Debtors

Actualiza el estado del pago en un recibo. Solo el propio usuario adeudor en un recibo puede actualizar su estado, una vez haya realizado el pago al solicitante.

Url: http://appartment-pruebamarja.rhcloud.com/receipt/idreceipt?session_id=234434234324

Parámetros obligatorios:

- action: login.
- type: debtors.
- idreceipt: Identificador del recibo

Respuesta:

{"rc":0}

- Delete Receipt

Borra un recibo ya creado. Cualquier usuario del grupo puede borrar el recibo.

Url: http://appartment-pruebamarja.rhcloud.com/receipt/idreceipt?session_id=234434234324

Parámetros obligatorios:

- action: delete.
- Idreceipt: Identificador del recibo.

Respuesta:

{"rc":0}

4.3 Tipología de los errores

Ante una acción que no ha continuado un curso normal, el servidor maneja una enumeración en su respuesta indicando qué tipo de error ha podido ocurrir en la comunicación. Siempre que devuelva 0 es que todo ha ido correctamente, en caso contrario se devolverá uno de los errores que se enumeran a continuación:

-1 Error en la base de datos
-2 No se ha enviado entidad
-3 No se ha enviado acción
-4 No se ha enviado session
-5 Fallo de parámetros
-6 Fallo al insertar el usuario
-7 Fallo al inserter el dispositivo
-8 Fallo en entidad
-9 Fallo de acción
-10 Session expired
-11 Fallo en la conexión a la base de datos
-12 Fallo en el upsert de access
-13 Fallo en el update de group
-14 Fallo al enviar el email
-15 Fallo en el add friend
-16 No se ha enviado tipo Get
-17 Tipo Get erróneo
-18 Fallo al insertar receipt
-19 Fallo al insertar hash
-20 Fallo en el find user
-21 Fallo en el update user
-22 Fallo en el find device
-23 Fallo en el upsert device
-24 Fallo en el insert group
-25 Fallo en el update group
-26 Fallo en el get group
-27 Fallo en el find hash
-28 Fallo en el delete group
-29 Fallo en el get friends
-30 Fallo en el get receipt
-31 Fallo en el get group receipts
-32 Fallo en el update debtors
-33 Fallo en el update receipt

-34 Fallo en el delete reciept
-35 Fallo en el get devices by user
-35 Fallo en el get receipt
-36 Fallo en el get user by id
-37 No está la id de la Url
-38 Fallo en el find de group
-39 Fallo en el flag de receipt

4.4 Integración de los módulos

La construcción e integración de las diferentes partes del proyecto se pueden dividir en cinco bloques distintos, esta parte del proyecto puede ser considerada la más delicada de todas puesto que es el punto de unión del módulo Front-End construida por mi compañero Diego y el módulo Back-End construida por mí. Cada bloque pertenece a un Sprint distinto. Los diferentes Sprints han tenido duraciones de tiempo variables, pero ninguno se ha alargado más allá de la semana. De cada uno de los bloques podemos destacar:

En el primer bloque tenemos la primera toma de contacto del proyecto, se definió qué es lo que queríamos construir y qué herramientas íbamos a utilizar. Además se prepararon los equipos personales y se montaron los diferentes entornos para poder trabajar y construir el proyecto. Por último se preparó la parte inicial del Back-End para que pudiera dar soporte a peticiones básicas.

Los siguientes tres bloques pertenecen a cada una de las entidades de las que se componente el proyecto. Siguen un orden cronológico intuitivo y necesario de seguir, comenzando la construcción de la entidad menos dependiente a la más dependiente. Primeramente se construyó la entidad Usuario, seguidamente la entidad Grupo y por último la entidad Recibo.

Por último y no menos importante encontramos el bloque Test. En una primera fase comprobamos la funcionalidad correcta de la aplicación y posteriormente dejamos que algunos usuarios elegidos por nosotros la probaran y dieran algunas opiniones sobre ella. Esto conllevó a posibles

ampliaciones y mejoras del proyecto que pueden ser abordadas en futuras fases.

En la figura 4 tenemos el área de trabajo de uno de los Sprints de los que se ha compuesto la construcción del proyecto. La figura muestra la construcción del módulo Grupo en el momento en el que se estaba construyendo la invitación de usuarios a un grupo.

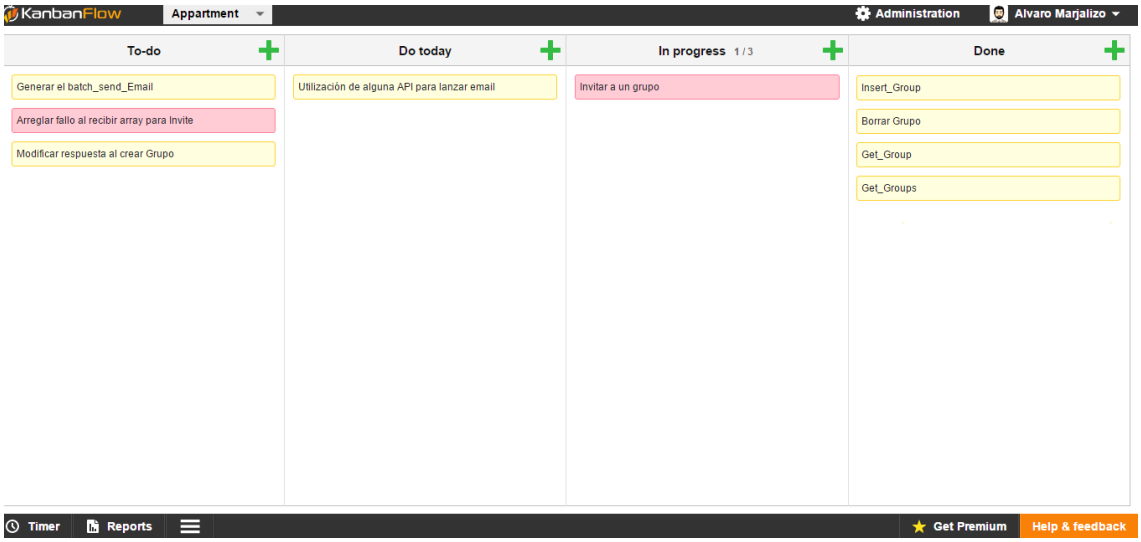

Figura 4: Scrum con KanBanFlow

5. Conclusiones y trabajos futuros

5.1 Objetivos cumplidos

Todos los objetivos propuestos en el inicio del proyecto han sido alcanzados en mayor o menor medida. Estos objetivos han contribuido a afianzar los conocimientos adquiridos y en algunos de ellos se ha adquirido nuevos conocimientos, como por ejemplo, alojar el servidor de aplicaciones en una PaaS, con el que se ha conseguido acercar el desarrollo del proyecto a un entorno real de producción.

El principal objetivo de este proyecto era poder dar un servicio Back-End a la aplicación móvil Appartment. Como hemos visto a lo largo del documento el servidor de aplicaciones responde a todas las necesidades propuestas por el cliente.

Para ello se requería que la comunicación entre el cliente y el servidor fuera mediante una arquitectura REST. Por ello con la creación de las entidades y los identificadores únicos para los recursos contribuidos por la base de datos, el servidor es capaz de publicar los recursos que el cliente precisa. Además esta comunicación debía usar una comunicación mime estándar, que se ha logrado con la representación de los datos en JSON.

Este último hecho y el haber utilizado Mongo como base de datos ha resultado que las operaciones con la propia base de datos hayan sido muy cómodas. Como ya hemos mencionado con anterioridad en este documento Mongo es una base de datos orientada a documentos. Con el uso de Mongo

abarcamos otro de los objetivos propuestos que era la utilización de una base de datos NoSQL.

Además se nos pedía unos tiempos en la generación de la respuesta hacia el cliente buenos. Con el uso de PHP hemos logrado una comunicación fluida y segura. Como ya hemos visto en la parte del diseño del proyecto, se han utilizado estructuras que trabajan en segundo plano, para que el tiempo de cómputo final sea óptimo.

A la hora de dar solución al empleo de una PaaS se ha recurrido a Openshift de Redhat con el que se ha dado una buena solución al objetivo establecido al comienzo del proyecto. Con ello el servidor está capacitado para dar servicio a la aplicación en cualquier parte del planeta. Con el uso de Openshift nos hemos ahorrado la parte tediosa de preparar e instalar todos los componentes necesarios para que el servidor trabaje, puesto como ya hemos visto en el documento, es la propia plataforma la que prepara, bajo diversos parámetros de configuración, los componentes que necesitas para dar servicio. Tan solo ha sido necesario montarnos un entorno similar localmente para que, una vez construido el servidor, trabaje en la PaaS de la misma manera que trabajaría localmente.

Por último y no por ello un objetivo de menor importancia, el componente de trabajo en equipo. La realización del proyecto en equipo más que un problema ha supuesto una ventaja, el trabajo con mi compañero ha sido muy cómodo y sobretodo que su trabajo dependa del mío ha influenciado en la velocidad de la construcción del servicio positivamente, no generando retrasos en los tiempos de entrega marcados.

5.2 Dificultades encontradas

No son muy remarcables las dificultades que he tenido a la hora de elaborar el back-end de appartment, puesto que no es la primera vez que me enfrento a ello, y casi todas las tecnologías utilizadas aquí ya he tenido

anteriormente un primer contacto con ellas. Aunque sí he tenido ciertos en los siguientes aspectos:

En los proyectos anteriores había trabajado, para el control de versiones, había utilizado SourceTree, un programa cliente para Git que de momento solo tiene plataforma para los sistemas OS X de Mac, aunque se espera que pronto ofrezcan soporte para Ubuntu. Sin embargo, en este proyecto es la primera vez que trabajo con un control de versiones en Ubuntu, y tras probar tres programas diferentes ninguno de ellos es comparable a Source Tree, bajo mi punto de vista. Incluso en algunos casos he tenido que clonar algunas de las veces el proyecto al generarse inconsistencias. El programa con el que mejor me he adaptado ha sido Gitg.

Por último también he encontrado los típicos problemas iniciales de no conocer la plataforma a la hora de enfrentarme con el Openshift de RedHat. Aunque tienen una buena documentación para principiantes, arrancar en un mundo desconocido siempre cuesta al principio. Pero una vez solventados los problemas iniciales, ha sido un acierto usarlo: fácil, cómodo y sobre todo muy útil.

5.3 Posibles ampliaciones

De las numerosas ampliaciones que este inicio de proyecto tiene, las más prioritarias podrían ser las siguientes.

En el apartado de invitación a la aplicación, cuando el servidor manda el correo electrónico a la persona o personas indicadas, el cliente no tiene en ningún momento constancia de que el correo ha sido entregado. El servidor no informa de este hecho, solo se limita en segundo plano a enviar el correo, sin detenerse en la respuesta del envío. Se podrían volcar los estados de envío de los correos electrónicos en la base de datos y realizar un manejador que se centrara en los erróneos y avisara al cliente del fallo.

Otra de las ampliaciones inminentes que el proyecto requiere es un sistema manejador de notificaciones para la aplicación. La aplicación carece aún de las tan importantes notificaciones que hoy en día existen para Android y que tan útiles son para el usuario, para ir informándole de sus estados en la aplicación.

Para acabar, habría que incluirle al proyecto el sistema de pago para que los clientes puedan realizar con dinero de verdad y saldar sus cuentas. Puesto que ahora en la versión beta solo se trabaja con dinero ficticio.

BIBLIOGRAFÍA

[1] "Getting Started with Openshift Online". Agosto, 2015. https://developers.openshift.com/en/getting-started-overview.html.

[2] "¿Qué es Openshift?". Agosto, 2015 http://www.sombrerorojo.com/que-es-openshift/.

[3] "El servicio REST". Agosto, 2015. https://eamodeorubio.wordpress.com/2010/07/26/servicios-web-2-%C2%BFque-es-rest/.

[4] "Conceptos sobre APIs REST". Agosto, 2015. http://asiermarques.com/2013/conceptos-sobre-apis-rest/.

[5] "¿Por qué PHP?". Agosto, 2015. http://programacion.net/articulo/por_que_elegir_php_143.

[6] "Motivos para trabajar con PHP". Agosto, 2015. http://www.lancetalent.com/blog/6-buenos-motivos-para-trabajar-con-php/.

[7] "Ventajas de las programación PHP". Agosto, 2015. http://www.staffcreativa.pe/blog/ventajas-programacion-php/.

[8] "Manual PHP". Agosto, 2015. https://secure.php.net/manual/es/index.php.

[9] "¿Mongo DB?". Agosto, 2015. http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no.

[10] "Manual MongoDB". Agosto, 2015. http://docs.mongodb.org/manual/.

[11] "Guía para Git". Agosto, 2015. http://rogerdudler.github.io/git-guide/index.es.html.

[12] "Scrum Methodology". Agosto, 2015. http://scrummethodology.com/.

[13] "Yaap Money". Agosto, 2015. https://money.yaap.com/.

[14] “Paga Mobil”. Agosto, 2015. https://www.pagamobil.com/.

[15] “Momo Pocket”. Agosto, 2015. https://www.momopocket.com/.

[16] “Square”. Agosto, 2015. https://squareup.com/global/es/register.