UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DESARROLLO DE UNA APLICACIÓN MÓVIL DE CONTROL Y REPORTES DE GASTOS

Por:
SUSANA CHARARA CHARARA

Realizado con la asesoría de:
Tutor Académico: PROF. XIOMARA CONTRERAS
Tutor Industrial: LIC. LUIS AUGUSTO PEÑA PEREIRA

INFORME DE PASANTÍA LARGA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero en Computación

Sartenejas, SEPTIEMBRE de 2016

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

DESARROLLO DE UNA APLICACIÓN MÓVIL DE CONTROL Y REPORTES DE GASTOS

Por:
SUSANA CHARARA CHARARA

Realizado con la asesoría de:
Tutor Académico: PROF. XIOMARA CONTRERAS
Tutor Industrial: LIC. LUIS AUGUSTO PEÑA PEREIRA

INFORME DE PASANTÍA LARGA
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero en Computación

Sartenejas, SEPTIEMBRE de 2016

Resumen

En este informe se describe el proceso de desarrollo de una aplicación móvil para el reporte de gastos, realizado para la compañía Digitalica Group C.A. De forma general, la aplicación permite al usuario registrar gastos e ingresos a una cuenta, asociarlos a categorías, así como capturar y guardar fotos de cada uno. Además, puede generar reportes y enviarlos a un servidor web, desarrollado igualmente en esta pasantía.

Se describe brevemente el marco de trabajo utilizado, Scrum, así como las tecnologías empleadas en el desarrollo tanto de la aplicación móvil como del servidor web.

Índice general

Índice de figuras

Lista de Símbolos y Abreviaturas

CEO: Chief Executive Officer (en español Director Ejecutivo)

COO: Chief Operations Officer (en español Director de Operaciones)

CTO: Chief Technology Officer (en español Director de Tecnología)

DAO: Data Access Object (en español Objeto de Acceso a Datos)

DVCS: Distributed Version Control System (en español Sistema de Control de Versiones Distribuido)

DVM: Dalvik Virtual Machine (en español OMáquina Virtual Dalvik)

HTTP: HyperText Transfer Protocol (en español Protocolo de Transferencia de Hipertexto)

IDE: Integrated Development Environment (en español Entorno de Desarrollo Integrado)

JSON: JavaScript Object Notation (en español Notación de Objetos de JavaScript)

JVM: Java Virtual Machine(en español OMáquina Virtual de Java)

MVC: Model View Controller (en español Modelo Vista Controlador)

MVP: Model View Presenter (en español Modelo Vista Presentador)

ORM: Object-Relational Mapping (en español Mapeo Objeto-Relacional)

PDF: Portable Document Format (en español Formato de Documento Portátil)

POJO: Plain Old Java Object (en español Objeto de Java Plano Antiguo)

SQL: Structured Query Language (en español Lenguaje de Consulta Estructurada)

Introducción

En la actualidad, muchas empresas le ofrecen a sus empleados el beneficio de realizar gastos personales y posteriormente iniciar un proceso de reembolso. Este proceso puede tomar mucho tiempo y puede resultar difícil si se hace manualmente, pues implica que el trabajador deba guardar todas las facturas o recibos de los gastos implicados. También dificulta la organización de estos registros por parte de la empresa, porque al mantener todos los recibos en físico, estos se pueden extraviar.

Este problema llevó a la conclusión de que es necesario agilizar todo el proceso, y dada la facilidad de acceso que se tiene hoy en día a un dispositivo móvil inteligente, se decidió crear una aplicación que permita mantener el registro de gastos.

Actualmente existen aplicaciones que sirven para llevar un registro de gastos. Sin embargo, estas aplicaciones no satisfacen las necesidades de la empresa por diversas razones:

- En primer lugar, la empresa realiza el reembolso de gastos en ciertos rubros y para esto se debe especificar a qué categoría pertenece cada gasto. Las aplicaciones ya existentes pueden limitar el proceso al no contar con la posibilidad de poder asociar un gasto a un rubro cubierto por la empresa.

- Se desea crear archivos de reportes de los gastos registrados por los trabajadores. Se desea que estos reportes puedan ser personalizados y presenten una estructura particular.

- Se desea mantener toda la información referente a estos reportes de manera centralizada de manera que se pueda acceder a la misma de una manera más fácil.
- Si el proceso de reembolso sufre alguna modificación, se desea contar con una aplicación que tome en cuenta los nuevos cambios.

Por las razones expuestas anteriormente, se tomó la decisión de crear una aplicación móvil que permita llevar un registro de gastos. Esta aplicación debe contar con las siguientes funcionalidades:

- Registrar gastos con su fecha, monto, una breve descripción, fotos y categoría a la que pertenecen
- Organizar y consultar gastos con criterios de búsqueda pre-establecidos
- Crear reportes personalizados con los gastos y enviar dichos reportes a un servidor
- Enviar reportes mediante archivos con formato PDF a los supervisores
- Controlar el estado de aprobación de los reportes

En este informe se describe el proceso de desarrollo de una aplicación móvil para el registro de gastos.

El informe se estructura de la siguiente manera: En el Capítulo 1 se describe de una forma general la empresa; en el Capítulo 2 se definen los conceptos teóricos estudiados para el desarrollo del proyecto; en el Capítulo 3 se mencionan las herramientas y tecnologías que facilitaron el desarrollo; en el Capítulo 4 se describe brevemente el marco de trabajo utilizado, Scrum; en el Capítulo 5 se describen todas las fases involucradas tanto en el diseño como el desarrollo de la solución; en el capítulo 6 se exponen las conclusiones y recomendaciones que surgieron luego de la investigación y desarrollo del proyecto; finalmente, se muestran las referencias bibliográficas consultadas.

Capítulo 1

Entorno Empresarial

1.1. Descripción de la empresa

Digitalica Group, C.A. es una empresa que se dedica al desarrollo de soluciones móviles, aplicaciones empresariales y *middleware*. Se caracteriza por estar a la vanguardia, utilizando tecnologías innovadoras [1]. Sus principales servicios son:

- Consultoría en el área de desarrollo de aplicaciones móviles: Durante los últimos años se ha brindado apoyo en el área de soluciones móviles a empresas del sector educativo.
- Desarrollo de *software* y aplicaciones móviles para clientes. Se han desarrollado aplicaciones para medios de comunicación nacionales e internacionales, así como para empresas del sector educativo.
- Ofrecer *software* como un servicio: Actualmente, los productos que ofrece la empresa están enfocados en el área de noticias, medios de comunicación y reporte de gastoss.

1.2. Misión

"Proveer soluciones y servicios de consultoría basados en tecnolías móviles, utilizando la mejores herramientas, metodologías y estándares de calidad en la industria" [1].

1.3. Visión

"Ser el líder en Latinoamérica en brindar soluciones innovadoras basadas en tecnologías móviles, con un equipo creativo con los conocimientos más sólidos en tecnologías de información para móviles, para cometir en el mercado global" [1].

1.4. Estructura organizacional

La estructura organizacional de la empresa está comprendida principalmente por la Junta de Directores, como se muestra en la figura 1.1.

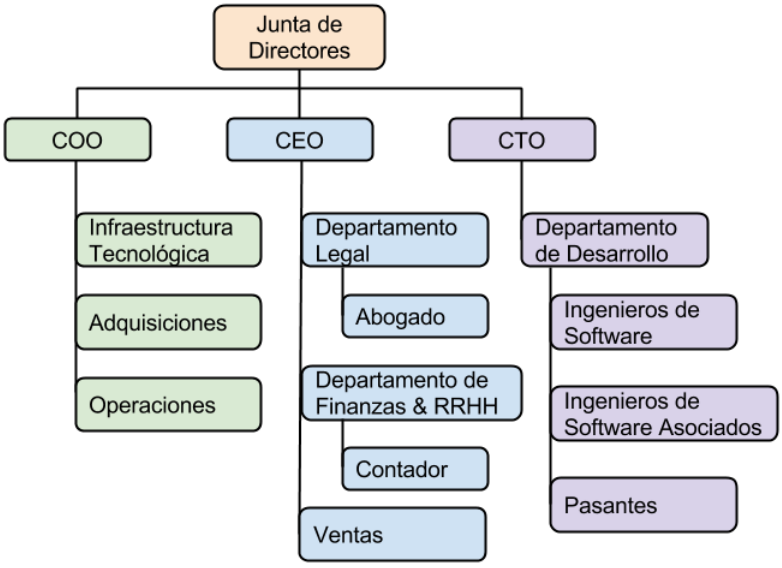

Figura 1.1: Estructura organizacional de la empresa

La Junta de Directores está compuesta por tres cargos: Director de Operaciones (COO por sus siglas en inglés), Director Ejecutivo (CEO por sus siglas en inglés) y Director de Tecnología (CTO por sus siglas en inglés).

El COO se encarga de gestionar las adquisiciones de la empresa y de mantener la infraestructura tecnológica.

El CEO se encarga de manejar la Dirección de Ventas, dirigir el Departamento Legal junto al abogado y dirigir el Departamento de Finanzas y Recursos Humanos junto al contador.

El CTO se encarga de dirigir el departamento de desarrollo,por lo que el equipo de desarrollo de *software* se comunica y le rinde cuentas a él.

1.5. Ubicación del pasante

Los pasantes se encuentran en el Departamento de Desarrollo, dirigido por el CTO, y donde se encuentan los especialistas de todas las áreas de desarrollo de *software*. Esto permite que los pasantes estén en constante contacto con personas que puedan asistirlos y solventar las dudas que puedan surgir durante el desarrollo.

Capítulo 2

Marco Teórico

En este capítulo se exponen los conceptos teóricos estudiados, tanto para entender las tecnologías utilizadas como para realizar el desarrollo de *software* requerido. Los conceptos estudiados formaron parte fundamental del proceso de diseño y desarrollo de la solución.

2.1. Patrones de diseño

Un patrón de diseño es una solución general y repetible a problemas que suelen presentarse en el proceso de diseño de software. Para que una solución pueda ser considerada como un patrón de diseño debe ser reutilizable, es decir, que se pueda aplicar a diferentes problemas de diseño en diferentes situaciones [2]. A continuación se describen los patrones utilizados en el diseño de la solución del proyecto.

2.1.1. *Singleton*

El *singleton* es un patrón de diseño que asegura la existencia de una sola instancia de la clase que lo implementa. Esto es útil cuando se necesita únicamente un objeto para coordinar las acciones en el sistema [3].

2.1.2. *Adapter*

Transforma la interfaz de una clase en otra interfaz que el cliente espera. Esto permite que una clase que no pueda utilizar la primera interfaz, sí pueda hacerlo a través de la otra [3].

2.1.3. *Facade*

Se utiliza para ofrecer una interfaz sencilla para operaciones complejas. Con este patrón se logra ofrecer una interfaz de alto nivel que hace que el sistema sea más fácil de usar para los clientes [3].

2.2. Patrones de arquitectura

Un patrón de arquitectura es una solución a problemas de arquitectura de *software*, ayudando a definir una estructura para sistemas de *software*. La diferencia entre los patrones de diseño y los de arquitectura es que estos últimos tienen un nivel de abstracción mayor. [4]

2.2.1. Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de *software* que divide una aplicación en tres partes que están interconectadas: los datos y la lógica de negocio (modelo), la interfaz de usuario (vista) y el módulo encargado de gestionar las comunicaciones (controlador). Esto se utiliza para separar la representación de la información de la forma en que dicha información es presentada al usuario [5].

2.2.2. Modelo Vista Presentador (MVP)

Es una derivación del patrón Modelo Vista Controlador (MVC) que se utiliza generalmente para construir interfaces de usuario [6]. En este patrón, la interacción entre el modelo

y la vista se logra únicamente a través del presentador, mientras que en el MVC la vista en ocasiones puede comunicarse directamente con el modelo. Otra diferencia con el patrón MVC es que en este último las peticiones son recibidas por el controlador. Éste, se comunica con el modelo para pedir los datos y luego se encarga de mostrar la vista adecuada. En el patrón MVP las peticiones son recibidas por la vista y delegadas al presentador, que es quien se comunica con el modelo para obtener los datos [7].

2.3. Arquitectura cliente-servidor

Es un modelo de arquitectura de sistema distribuido, donde existe un conjunto de procesos que proporcionan servicios (servidores) a otros procesos (clientes). Por lo general, el servidor no conoce la identidad ni el número de sus clientes, pero el cliente sí conoce la identidad del servidor [4]. Dentro de esta arquitectura existen cuatro tipos [8]:

- Cliente activo, servidor pasivo: El cliente procesa toda la información.
- Cliente pasivo, servidor pasivo: Tanto el cliente como el servidor pasan información.
- Cliente pasivo, servidor activo: El servidor procesa toda la información y el cliente presenta los datos
- Cliente activo, servidor activo: Tanto el servidor como el cliente procesan la información.

2.4. Servicio web

Es un conjunto de funcionalidades diseñadas para permitir la comunicación entre diferentes dispositivos a través de una red, utilizando un conjunto de protocolos y estándares [9].

2.5. HTTP (*Hypertext Transfer Protocol*)

Es el protocolo de comunicación que permite la transferencia de recursos en la web. Un recurso es cualquier información que puede ser identificada por un URL. Existen diversos métodos que permiten realizar peticiones con el protocolo HTTP, entre los que están GET y POST [10].

2.5.1. HTTP GET

Es el método más común de HTTP, que permite pedir datos de un recurso en específico [11].

2.5.2. HTTP POST

Es uno de los diferentes métodos de peticiones que soporta el protocolo HTTP, que permite enviar datos a un recurso, los cuales deben ser procesados. Se requiere que los datos sean enviados dentro del cuerpo del mensaje [11].

Para enviar estos datos dentro de una petición POST, existen diferentes tipos de codificación. Entre estos tipos está Multipart, que permite enviar mensajes con varias partes combinadas en un solo cuerpo [12].

2.6. POJO *Plain Old Java Object*

Es una instancia de una clase que no extiende ni implementa ninguna otra [13]. Sirve para promover el uso de clases simples que no dependen de ningún *framework* [14].

2.7. DAO *Data Access Object*

Es un objeto que provee una interfaz abstracta para realizar operaciones sobre una base de datos u otro mecanismo de persistencia. Esconde todos los detalles de implementación a sus clientes, por lo que la interfaz expuesta por un DAO no cambia [15].

2.8. *Manager*

Es una interfaz que se utiliza para actuar como puente entre la capa de persistencia (base de datos) y la capa web. Dentro del *manager* se incluye la lógica de negocios de la aplicación. Se suele implementar en conjunto con DAO's. [16]

Capítulo 3

Marco Tecnológico

En este capítulo se definen las diferentes herramientas utilizadas a lo largo de todo el proyecto. Dichas herramientas permitieron el desarrollo del *software* planteado como solución al problema.

3.1. Herramientas, entornos y lenguajes

3.1.1. Android

Es un sistema operativo de código abiertos basado en el sistema operativo Linux, utilizado principalmente para dispositivos móviles [17]. Es la plataforma para la cual se desarrolló la aplicación.

La plataforma de Android está basada en una arquitectura que tiene diferentes capas que van desde servicios de bajo nivel del sistema operativo, que interactúan directamente con el *hardware*, hasta las aplicaciones. Además, Android provee un kit de desarrollo de *software* (SDK por sus siglas en inglés), que permite crear aplicaciones [18].

La capa de más bajo nivel es la del kernel de Linux, como se puede observar en la figura

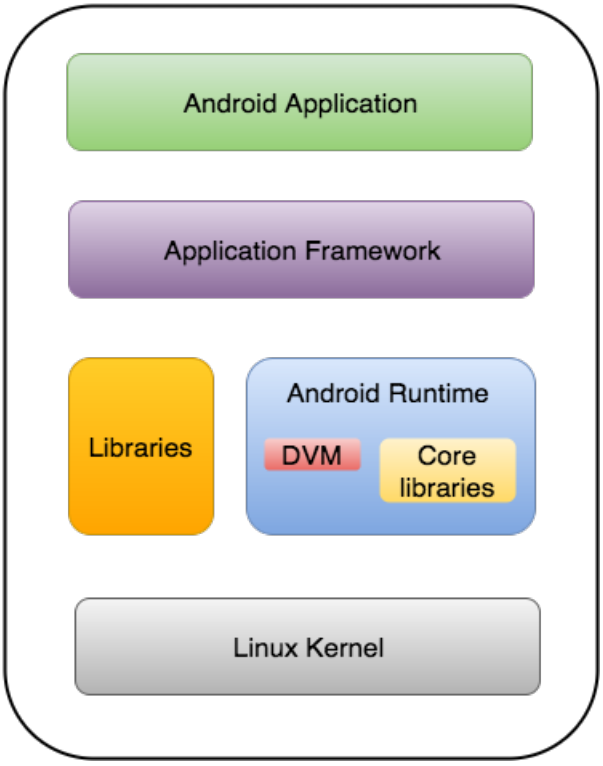

Figura 3.1: Arquitectura de la plataforma de Android

3.1. Provee un nivel de abstracción para la comunicación con el *hardware* del dispositivo, y contiene los *drivers* esenciales para el funcionamiento de dicho *hardware*. Además, provee servicios del sistema operativo, como el manejo de memoria y procesos [18].

Encima de esta capa se encuentra la de librerías, dentro de las cuales se encuentra SQLite, que permite guardar datos de las aplicaciones [18].

En conjunto con la librerías, está la sección de Android *Runtime* en el segundo nivel. Dentro de ella se encuentra un componente llamado Máquina Virtual de Dalvik (DVM por sus siglas en inglés), que es una máquina virtual de Java diseñada y optimizada especialmente para Android. Además de la DVM, provee un conjunto de librerías que permite desarrollar aplicaciones en Android utilizando el lenguaje de programación Java [18].

La tercera capa de abajo hacia arriba corresponde al *framework* de las aplicaciones. En ella se proveen servicios de alto nivel mediante clases de Java. Estos servicios pueden ser utilizados

dentro de las aplicaciones. Dentro de estos servicios destaca el Manejador de Actividades (*Activity Manager*), que se encarga de controlar todas las actividades de una aplicación. Una actividad (*activity*) es un componente que provee una pantalla con la cual el usuario final puede interactuar. Generalmente, a cada actividad corresponde una interfaz de usuario [18].

Por último, se encuentra la capa de aplicaciones, que es donde se encuentran instaladas las aplicaciones [18].

3.1.2. Java

Es un lenguaje de programación de alto nivel orientado a objetos, que puede correr virtualmente en cualquier computadora [19]. Es el lenguaje en el que están desarrolladas la mayoría de las aplicaciones de Android [20]. Se utilizó tanto para la programación de la aplicación móvil como del servidor.

3.1.3. MySQL

Es un sistema de manejo de base de datos relacional basado en el Lenguaje de Consulta Estructurado (SQL por sus siglas en inglés) [21]. Se utilizó para la gestión de la base de datos del servidor.

3.1.4. SQLite

Es un sistema de manejo de base de datos relacional basado en el Lenguaje de Consulta Estructurado (SQL por sus siglas en inglés) que forma parte del programa principal, en lugar de ser un proceso independiente como ocurre con los sitemas de gestión de base de datos cliente-servidor [22].

3.1.5. Maven

Es una herramienta que se permite compilar y manejar proyectos de *software* basados en Java. Se encarga de conectar las dependencias de los paquetes [23].

3.1.6. Jetty

Es un servidor HTTP que puede funcionar independientemente por sus propios medios, o que puede correr dentro de otra aplicación [24].

3.1.7. Android Studio

Es el entorno de desarrollo integrado (IDE por su siglas en inglés) oficial para el desarrollo de aplicaciones nativas para Android [25].

3.1.8. Eclipse

Es un entorno desarrollo integrado (IDE por sus siglas en inglés) utilizado principalmente para desarrollar aplicaciones en Java [26].

3.1.9. Git

Es un sistema de control de versiones distribuido (DVCS por sus siglas en inglés). Es una herramienta que permite gestionar las distintas versiones de un proyecto de *software* [27].

3.2. *Frameworks y librerías*

3.2.1. AppFuse

Es un *framework* utilizado para la creación de aplicaciones web en la máquina virtual de Java (JVM por sus siglas en inglés). Dentro de AppFuse, se pueden integrar otras tecnologías como Bootstrap, Maven, Hibernate, Spring, Tapestry, etc [28].

3.2.2. Hibernate

Es una herramienta de mapeo objeto-relacional (ORM por sus siglas en inglés) para Java que permite el mapeo de objetos de Java a bases de datos relacionales [29].

3.2.3. Spring

Es un *framework* para el desarrollo de aplicaciones en Java.

3.2.4. Tapestry

Es un *framework* para el desarrollo de aplicaciones web en Java, Groovy o Scala. [30].

3.2.5. Gson

Es una librería de Java que permite la conversión de objetos a JSON y viceversa [31] .

3.2.6. Retrofit

Es una librería de Android que permite realizar peticiones HTTP mediante una interfaz de Java [32].

Capítulo 4

Marco Metodológico

A continuación se describe el procedimiento seguido para el desarrollo del proyecto. Se decidió utilizar el marco de trabajo Scrum, dado que éste es utilizado por la empresa para el desarrollo de *software*.

4.1. Scrum

Scrum es un marco de trabajo que se basa en el desarrollo iterativo e incremental de un producto, en lugar del modelo clásico de planificación y ejecución completa [33]. Se caracteriza por ser una metodología ligera, fácil de entender y difícil de dominar, que permite entregar incrementos de producto potencialmente productivos [34].

4.1.1. Roles

En Scrum el desarrollo se realiza por uno o más equipos de trabajo dentro de los cuales existen tres roles: dueño del producto, facilitador de Scrum y el equipo de desarrollo [33].

4.1.1.1. Dueño del producto (*Product owner*)

Es el representante de los clientes. Dentro del equipo de Scrum, es el líder principal del producto y el responsable de decidir qué funcionalidades serán desarrolladas y la prioridad que tendrá cada una de ellas. Debe comunicar al resto de los involucrados en el proyecto una visión clara de lo que se quiere lograr. Tiene la obligación de asegurar que siempre se entregue un producto con el máximo de valor, por lo que debe colaborar con el resto del equipo para responder cualquier duda que surja [33]. Este rol fue asumido por el Ing. Chi Wang Zhong.

4.1.1.2. Facilitador de Scrum (*Scrum master*)

Actúa como facilitador tanto para el dueño del producto como para el equipo de desarrollo. Es el encargado de ayudar al resto del equipo a entender y cumplir con los principios y prácticas de Scrum. También tiene la responsabilidad de eliminar cualquier impedimento que el equipo no sea capaz de resolver y que afecte su productividad [33]. Este rol fue asumido por el Lic. Luis Augusto Peña.

4.1.1.3. Equipo de desarrollo

Es el encargado de desarrollar el producto. Es un equipo que está compuesto por arquitectos, programadores, probadores, administradores de base de datos, diseñadores de interfaces, entre otros. Son los responsables de diseñar, desarrollar y probar el producto [33]. El equipo de desarrollo estuvo integrado únicamente por la pasante Susana Charara.

4.1.2. Actividades

En Scrum, el trabajo se desarrolla en iteraciones de una duración máxima de un mes, llamadas ***sprints***. Al final de cada iteración, se debe haber desarrollado una parte del producto final, la cual debe ser completamente funcional. Dentro de cada iteración existe una

serie de eventos o actividades que se llevan a cabo: planeación de la iteración, ejecución de la iteración, reuniones diarias, revisión de la iteración y retrospectiva de la iteración [33].

4.1.2.1. Planeación de la iteración (*Sprint planning*)

Para determinar qué funcionalidades del producto final son las más importantes y próximas a desarrollar, el equipo de trabajo (dueño del producto, facilitador de Scrum y el equipo de desarrollo) realizan una reunión llamada *sprint planning* o planeación de la iteración [33].

Durante la reunión, el dueño del producto y el equipo de desarrollo establecen una meta que debe ser cumplida para el final de la iteración. De acuerdo a esta meta, el equipo de desarrollo decide de una manera realista qué incrementos del producto final pueden entregarse al terminar la iteración [33].

4.1.2.2. Ejecución de la iteración (*Sprint execution*)

Luego de la ejecución de la iteración, el equipo de desarrollo desarrolla todas las tareas acordadas en la reunión. Esto es lo que se conoce como la ejecución de la iteración [33].

4.1.2.3. Reunión diaria (*Daily scrum*)

Cada día dentro de la ejecución dela iteración , los miembros del equipo de desarrollo se reúnen durante un máximo de 15 minutos con el fin de informar qué se hizo el día anterior, qué se tiene planificado realizar el presente día y qué impedimentos se han presentado durante el desarrollo de su trabajo [33].

4.1.2.4. Revisión de la iteración (*Sprint review*)

Al final de cada iteración, ocurre un evento que se conoce como *sprint review* o revisión de la iteración. El objetivo de esta actividad es revisar el incremento y realizar las adaptaciones necesarias al prouducto [33].

4.1.2.5. Retrospectiva de la iteración (*Sprint retrospective*)

Esta actividad ocurre generalmente luego del *sprint review* y antes del próximo *sprint planning*. Es una oportunidad para que todo el equipo se reúna para discutir qué ha funcionado y qué no acerca de Scrum. Al final de la retrospectiva, el equipo de Scrum habrá discutido qué acciones deberán tomarse para mejorar la dinámica en las próximas iteraciones [35].

4.1.3. Artefactos

Dentro de Scrum existen dos herramientas o artefactos que permiten mantener un seguimiento del proyecto: el documento de historias de usuario y la lista de tareas de la iteración. [35].

4.1.3.1. Documento de historias de usuario (*Product backlog*)

Es una lista de los requerimientos funcionales del producto ordenados según su importancia. El dueño del producto es el responsable de definir qué elementos serán incluidos en esta lista y de colocarlos según su prioridad, de manera que los elementos de mayor valor o prioridad aparezcan al principio de la lista, y los de menos valor al final de la misma [35].

4.1.3.2. Lista de tareas de la iteración (*Sprint backlog*)

Es una lista donde se presenta un subconjunto de los elementos del *product backlog* divididos en tareas más pequeñas [35].

Capítulo 5

Desarrollo de la aplicación

En este capítulo se describe el trabajo realizado para el desarrollo de la aplicación. Este proceso se dividió en tres fases: investigación, diseño y desarrollo del sistema que incluye tanto la aplicación móvil como el componente servidor.

5.1. Investigación

El objetivo de esta primera fase era familiarizarse con el entorno de trabajo.

En primer lugar se estudiaron diversos patrones de diseño. Para esto, se investigó acerca de patrones ampliamente utilizados en el desarrollo de aplicaciones para Android y en la programación orientada a objetos en general. Esto permitió estructurar la aplicación de una manera más ordenada y entendible, de manera que se pueda extender fácilmente.

Luego de esto, se investigó acerca del desarrollo de aplicaciones para Android. Se aprendió a trabajar con el IDE Android Studio, así como los componentes y herramientas para el desarrollo bajo el sistema operativo.

Por último, se realizó una aplicación de prueba para poner en práctica algunos conceptos básicos en el desarrollo de aplicaciones para Android. Con esto se aprendió la estructura

general de una aplicación, así como el flujo de ejecución.

5.2. Análisis y diseño de la solución

En esta fase tuvo lugar dos etapas fundamentales para el desarrollo: análisis del problema y diseño de la solución.

Para comprender mejor el problema presentado, se realizó una reunión con el dueño del producto (*product owner*). En esta reunión se realizó el primer levantamiento y refinamiento de requerimientos. A partir de estos requerimientos, se diseñó una arquitectura base que diera solución al problema.

5.2.1. Análisis del problema

Se necesita una aplicación móvil que permita a los trabajadores de la compañía registrar gastos personales que posteriormente serán reembolsados. El reporte de estos gastos debe ser enviado a un servidor web como un archivo con formato PDF. También se requiere una aplicación web que permita aprobar o rechazar los reportes enviados.

La aplicación móvil debe permitir la creación de cuentas con diferentes monedas. Para cada una de estas cuentas, se pueden registrar gastos, a los cuales se puede asociar un monto, una fecha y una descripción. Además, se pueden tomar fotos de las facturas o recibos de dichos gastos.

Dado que la empresa realiza el reembolso de gastos en determinadas áreas, es necesario que estos gastos puedan asociarse a categorías que los identifiquen. Además, se debe permitir la creación de nuevas categorías en caso de que en algún momento la empresa decida incluir nuevos rubros.

También se quiere poder realizar consultas de los gastos registrados en un rango de fechas y generar un reporte de dichos gastos. Este reporte debe ser enviado a un servidor como un

archivo de formato PDF.

Por último, se quiere que los supervisores de la compañía puedan tener acceso a los reportes de gastos mensuales de los trabajadores. Los supervisores deben poder aprobar o rechazar dichos reportes. Además, el archivo PDF de cada reporte debe ser guardado en el servidor.

5.2.2. Diseño de la solución

Arquitectura base

Dados los requerimientos anteriores, se diseñó una arquitectura base para solucionar el problema. Se dividió el sistema según sus componentes: servidor web, dispositivos móviles y cliente web. Como se muestra en la figura 5.1, es una arquitectura cliente-servidor, donde el componente móvil y la aplicación web juegan el papel de cliente. En este caso, la aplicación móvil actúa como un cliente activo, pues es quien manejará los datos relevantes. El servidor solo recibirá esta información, sin poder modificarla. Esto corresponde a una arquitectura del tipo cliente activo-servidor pasivo, descrita en el capítulo 2.

Por medio de la aplicación móvil se podrá hacer el registro de los gastos personales. El servidor permitirá la revisión de los reportes de gastos, así como su aprobación o rechazo.

Toda la información relacionada a los gastos (fecha, monto, categorías, fotos) será creada y guardada localmente en cada dispositivo móvil. Cada dispositivo podrá enviar al servidor su reporte de gastos en forma de un archivo PDF. La comunicación entre la aplicación móvil y el servidor se hace por medio de servicios web que presta este último.

Estos reportes deberán ser guardados en la base de datos del servidor. Además, a través de una aplicación web, se podrá aprobar o rechazar los reportes recibidos.

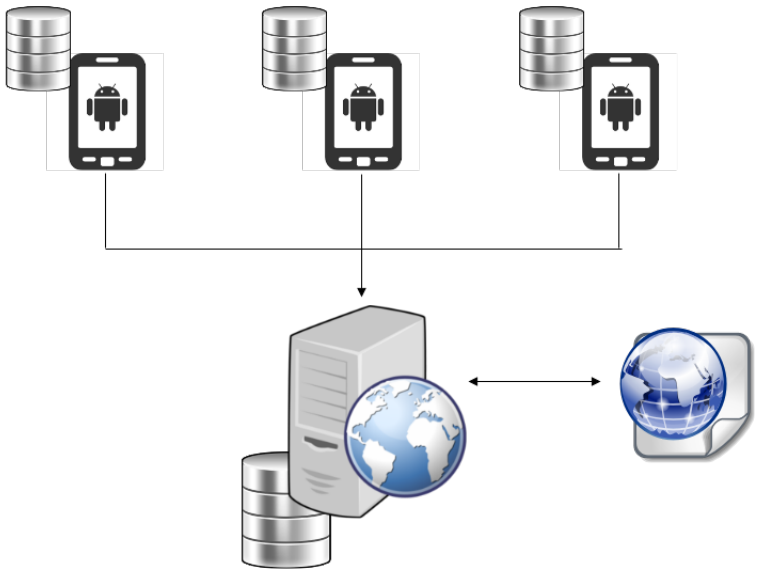

Figura 5.1: Arquitectura base de la solución

Arquitectura de *software*

Para los componentes mencionados anteriormente, se definieron los patrones de arquitectura de *software* sobre los cuales se crearon. Para el servidor se utilizó el patrón Modelo Vista Controlador (MVC) y para la aplicación móvil el Modelo Vista Presentador (MVP).

Tal como se explicó en el capítulo 2, MVC se usa para aislar la lógica de negocio de la interfaz de usuario. En este patrón, el modelo representa todos los datos de la aplicación y las reglas de negocio que se usan para manejar dichos datos.

Como se muestra en la figura 5.2, el modelo incluye tanto los POJO's, DAO's como la base de datos. La vista corresponde a todo lo que son los elementos de la interfaz de usuario que, en este caso, son las vistas de la aplicación web. El controlador se encarga de manejar las acciones del usuario, y se comunica con el modelo o con la vista según sea necesario. El *manager* definido en el capítulo 2 actúa como controlador.

Por su parte, MVP es una derivación del patrón MVC, como se observa en la figura 5.3. El modelo incluye tanto la base de datos como los POJO's, los cuales fueron utilizados para

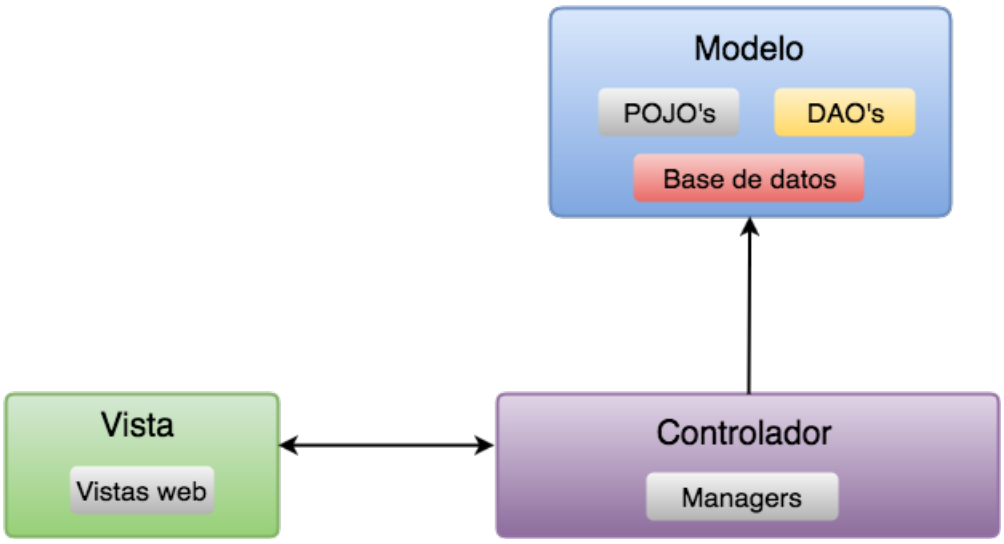

Figura 5.2: Modelo Vista Controlador

establecer una correspondencia entre objetos de la aplicación con registros de la base de datos. La vista está formada por las actividades (*activities*) definidos en el capítulo 2, y las interfaces de usuario.

En este caso, las acciones de usuario son recibidas por la vista. Luego, la vista se comunica con el presentador, que pide los datos al modelo. Al terminar de trabajo, el modelo notifica al presentador, y este último se encarga de modificar la vista a través de una interfaz.

5.2.2.1. Estructura de datos

Para dar lugar al desarrollo del sistema, fue necesario diseñar un modelo de datos que permita mantener la información necesaria en el dispositivo móvil, y otro para mantener información en el servidor.

En la figura 5.4 se puede observar el diagrama de clases que representa el modelo de datos de la aplicación móvil.

A continuación se describirán cada una de estas clases, así como sus atributos:

- ***Account***: representa un conjunto de registros de movimientos de dinero. Almacena

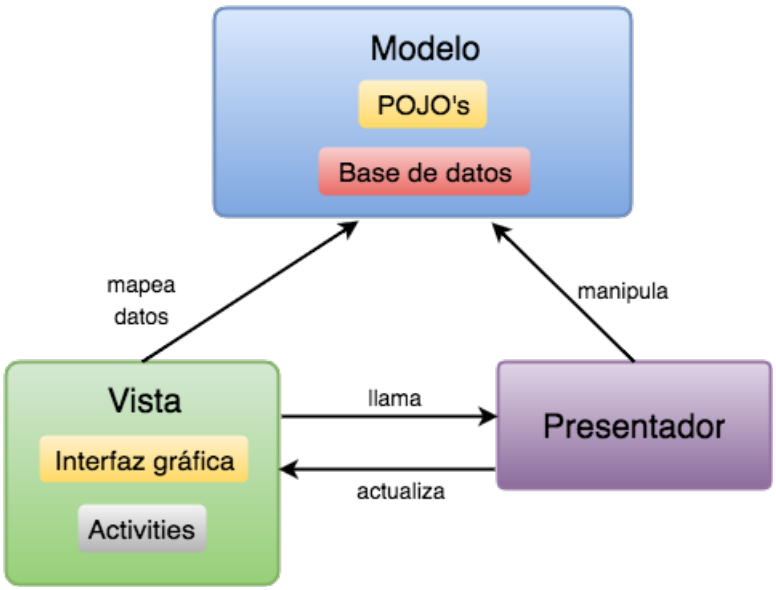

Figura 5.3: Modelo Vista Presentador

información de su nombre, estado (activa o archivada) y la moneda en la que estarán los montos de los registros dentro de la cuenta.

- ***Entry***: representa un registro dentro de una cuenta, y puede ser de dos tipos: ingreso o gasto. Almacena información del monto, fecha y descripción.

- ***Category***: representa las clases a las que puede pertenecer un *entry*. Puede ser de dos tipos: categoría de ingresos o categoría de gastos. Almacena información general (nombre) y un ícono que la representa.

- ***Photo***: representa las fotos asociadas a un *entry*. Almacena información del *entry* al que pertenece, y la ruta del archivo dentro del dispositivo móvil.

Por otra parte, en la figura 5.5 se puede observar la estructura de datos del servidor. En este caso, se tienen dos clases: *user* y *report*.

A continuación se describirán cada una de estas clases, así como sus atributos:

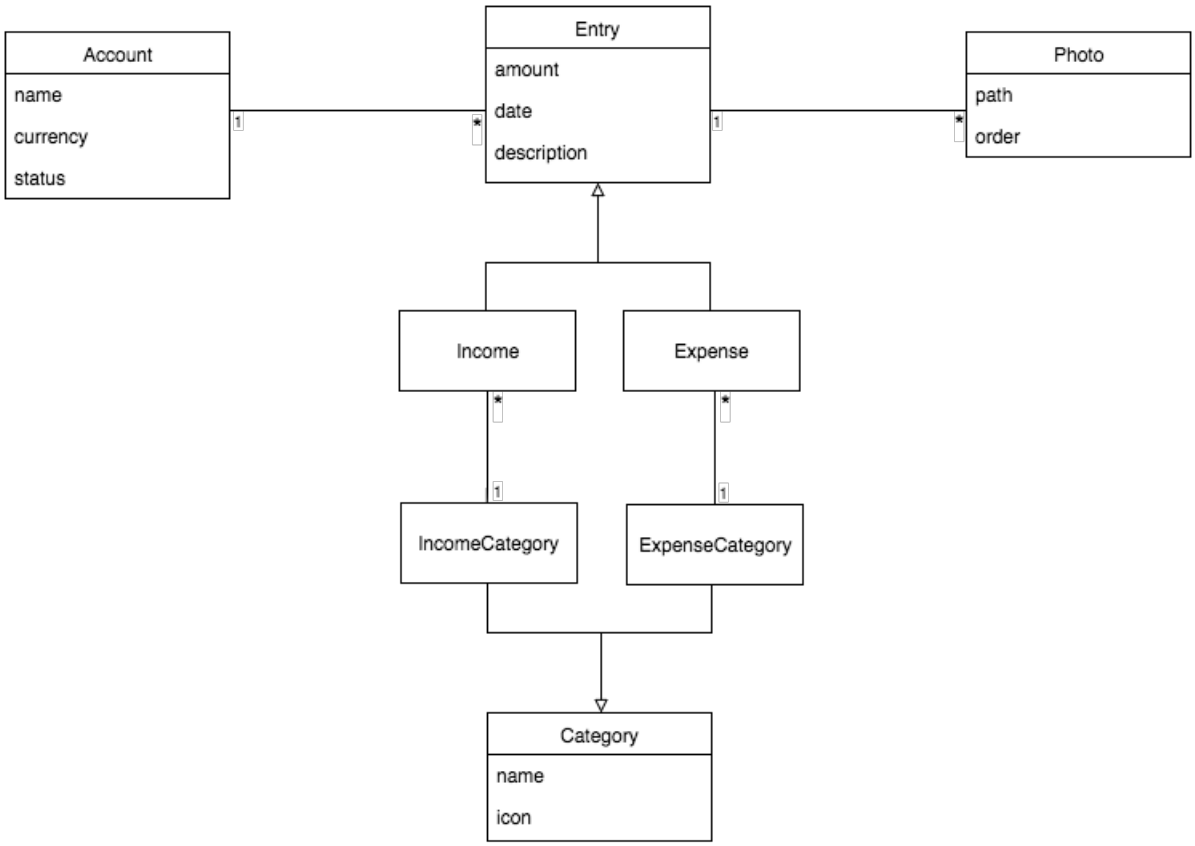

Figura 5.4: Diagrama de clases de la aplicación móvil

- ***User***: representa los usuarios registrados en el sistema. Almacena información del nombre completo, nombre de usuario, contraseña.

- ***Report***: representa un reporte enviado por un usuario, que contiene información de la suma total de *entries* pertenecientes a ciertas categorías. Almacena información del monto total de entries de cuatro categorías: comida (*food*), transporte (*transportation*), salud (*health*) y deporte (*fitness*). Además, también almacena la fecha en que se envió el reporte, su estado (aprobado, rechazado o sin revisar) y la fecha en que fue revisado. Por último, tiene la ruta del archivo PDF correspondiente al reporte.

Estos diagramas sirvieron como base para la creación del modelo tanto del servidor como de la aplicación móvil.

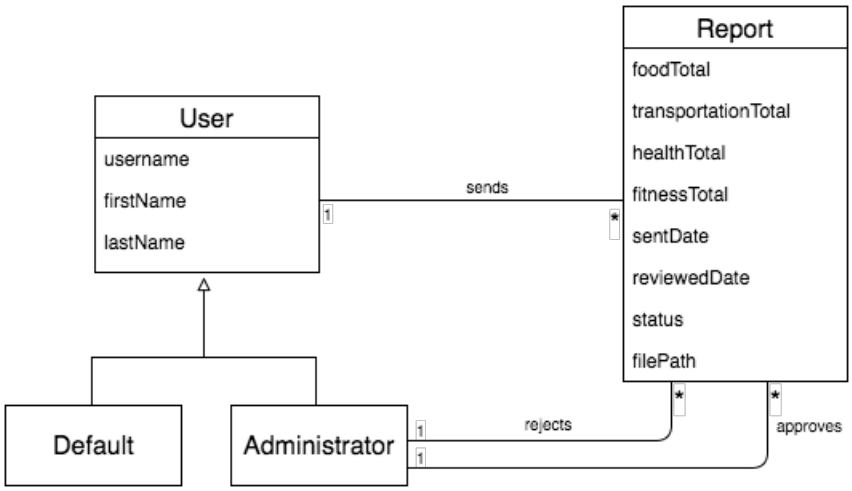

Figura 5.5: Diagrama de clases del servidor

5.3. Desarrollo

Durante esta fase se implementó progresivamente la versión alfa del prototipo funcional.

Se creó un componente móvil y un servidor para dar solución al problema de registro y control de gastos.

Para cumplir con los requerimientos del componente móvil, se creó una aplicación para el sistema operativo Android, y se trabajó con el IDE Android Studio.

El desarrollo de la aplicación implicó la creación de múltiples vistas (tanto interfaces de usuario como la lógica para manejar acciones de usuario), y de diversas consultas a la base de datos, muchas de las cuales realizaban modificaciones a la misma.

En primer lugar, se creó la interfaz de la vista principal de la aplicación, donde se muestra el total de ingresos, el total de gastos y el balance general.

Se implementó una vista para crear y editar gastos. Para estos gastos, se debe guardar su monto, fecha, descripción, fotos de recibos y categoría a la que pertenecen. Posteriormente, se decidió extender la funcionalidad de creación de gastos para permitir también el manejo de ingresos.

Para mostrar los gastos e ingresos guardados, se creó una vista para listarlos. En las listas

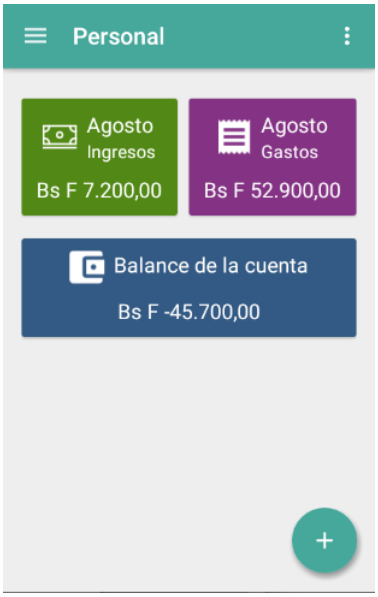

Figura 5.6: Interfaz principal de la aplicación

se muestran la descripción/categoría a la que pertenece cada entrada, su monto y su fecha. Dentro de cada lista, se permite ver los detalles de cada gasto/ingreso, editarlos y eliminarlos. Además, también se permite filtrar las listas por palabras claves.

También se crearon vistas para el manejo de categorías, que incluyen la creación, edición y eliminación de las mismas. Estas categorías pueden ser de dos tipos: categorías de gastos o categorías de ingresos.

Luego de la creación de las principales funcionalidades del sistema, mencionadas en los párrafos anteriores, se introdujo el concepto de c̈uentas: Hasta el momento, los gastos estaban asociados a una única cuenta por defecto. Por esta razón, también se creó una funcionalidad para agregar nuevas cuentas, y poder crear nuevos gastos dentro de dichas cuentas. Estas cuentas pueden ser de dos tipos: activas o archivadas; la navegación entre cuentas se hace únicamente con las cuentas activas.

Además, también se creó una vista para poder listar las cuentas creadas y guardadas en el dispositivo. Dentro de esta vista, se puede editar, eliminar o cambiar el estado de las cuentas

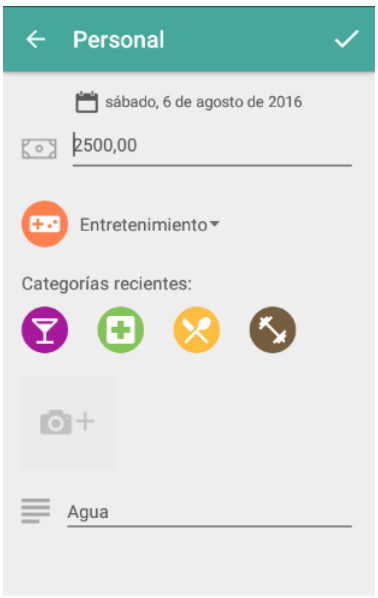

Figura 5.7: Interfaz para crear o editar un gasto/ingreso

(activa o archivada).

Como se explicó en la sección anterior, es necesario guardar toda esta información en la base de datos local de cada dispositivo. Para ello, se utilizó la librería SQLite.

Además, también se crearon dos vistas para reportes de gastos: una para reportes de balance general, y otra para reporte de gastos por categorías.

La vista de reporte de balance general muestra una lista de gastos e ingresos filtrados por fecha. La vista de reporte por categorías muestra una lista de categorías con la suma total de los gastos e ingresos de dichas categorías, filtradas por fecha.

Para mantener la lógica separada de la interfaz gráfica, se utilizó el patrón de arquitectura MVP o Modelo Vista Presentador, descrito en el capítulo 2. De esta manera, toda acción en la vista que requiera realizar operaciones sobre la base de datos, se realizó por medio de una capa intermedia: el presentador.

Para lograr esto, fue necesario en primer lugar implementar el modelo descrito en la figura 5.4. Se creó la base de datos con cada una de sus tablas, así como las clases (POJO's) que

Figura 5.8: Interfaz para listar gastos

permitieron dar un nivel de abstracción mayor para el mapeo con la base de datos.

También se crearon las vistas necesarias, lo que incluyó tanto las interfaces gráficas como la lógica necesaria para manejar las acciones del usuario (actividades). Para cada vista, se creó una clase que actuó como la capa de presentador, quien era invocado por la vista para realizar acciones sobre el modelo. La vista se encargó de la creación de los objetos correspondientes al modelo, para luego pedir alguna acción al presentador sobre dichos objetos. El presentador tenía la función de invocar al modelo para realizar los cambios necesarios en la base de datos.

Por otra parte, se creó un servidor al que se pueda enviar información desde la aplicación móvil.

Para permitir el acceso a la aplicación móvil, se creó un servicio web que permita hacer la autenticación del usuario. La aplicación envía al servidor un JSON con los datos del usuario (nombre de usuario y contraseña) por medio de una petición HTTP POST.

Se creó un servicio web que recibe un archivo PDF que contiene el reporte de gastos. Además del archivo, recibe la suma de gastos por cada categoría y el total de gastos. Estos

Figura 5.9: Interfaz para listar ingresos

datos se enviaron por medio de una petición HTTP Post Multipart, compuesta por un archivo y un JSON. Para la conversión de objetos de Java a JSON, y viceversa, se utilizó la librería GSON tanto en el servidor como en la aplicación móvil. Para persistir esta información, se utilizó Hibernate y MySQL como manejador de base de datos.

También se creó una aplicación web que permita aprobar o rechazar los reportes enviados al servidor (ver figuras 5.11 y 5.12). Además de esto, la herramienta utilizada para el desarrollo del servidor trae consigo algunas funcionalidades por defecto: iniciar sesión y el manejo de usuarios (agregar, modificar y eliminar usuarios).

El desarrollo del componente servidor se realizó con el IDE Eclipse. Se utilizó el *framework* AppFuse, que integra a su vez otros *frameworks* que facilitan el desarrollo web. En este caso, se trabajó en conjunto con Spring y Tapestry.

Se utilizó Spring para la implementación de los servicios web que permiten recibir datos desde la aplicación móvil. Estos servicios se implementaron utilizando el patrón *Facade*, mediante una clase *Manager*, la cual accede a los POJO's mediante un DAO (para más

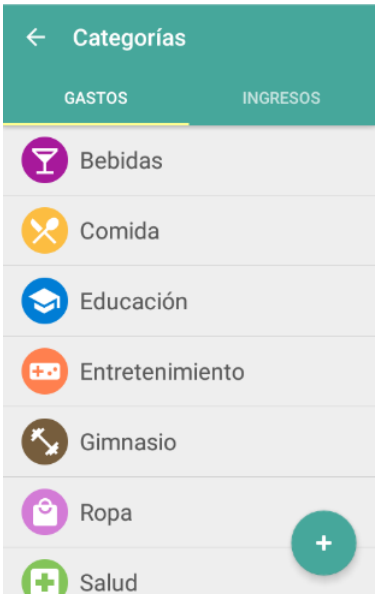

Figura 5.10: Interfaz para listar categorías de gastos

información, ver capítulo 2). Además, Spring también cuenta con el *framework* Hibernate para la persistencia de los datos, utilizando como manejador de base de datos MySQL.

Para crear la aplicación web, se utilizó Tapestry. Con Jetty se proveen los servicios, mediante el protocolo HTTP, que permiten la comunicación con la aplicación web y la aplicación móvil.

Por otra parte, a través de la librería Retrofit, la aplicación se comunica con el servidor web utlizando el protocolo HTTP. La comunicación con la base de datos del dispositivo y con los servicios web se realizan en la capa del presentador. Por su parte, en la vista se maneja toda la interfaz gráfica a través de la cual el usuario interactúa con la aplicación, y toda la lógica necesaria para la comunicación con la capa de modelo y el presentador.

El proyecto se desarrolló bajo el marco de trabajo de *Scrum*, descrito en el capítulo 1. Durante la ejecución de las iteraciones, se implementaron las funcionalidades planificadas, así como las pruebas necesarias para validarlas.

A continuación se presentarán las iteraciones (*sprints*), durante las cuales se desarrollaron

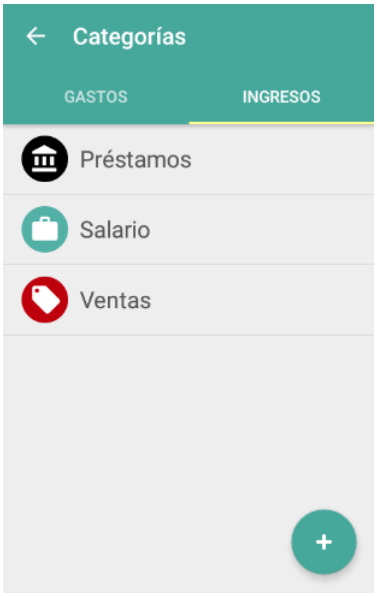

Figura 5.11: Interfaz para listar categorías de ingresos

las funcionalidades descritas en los párrafos anteriores. Para cada iteración, se mencionarán sus objetivos, resultados y una breve descripción de las actividades realizadas.

5.3.1. Iteración 1

5.3.1.1. Objetivos

- Mostrar información del balance de una cuenta
- Permitir la creación de un nuevo gasto

5.3.1.2. Resultados

- Se creó una vista para mostrar el total de ingresos,gastos y balance general de una cuenta.
- Se creó una vista para permitir la creación de un nuevo gasto.

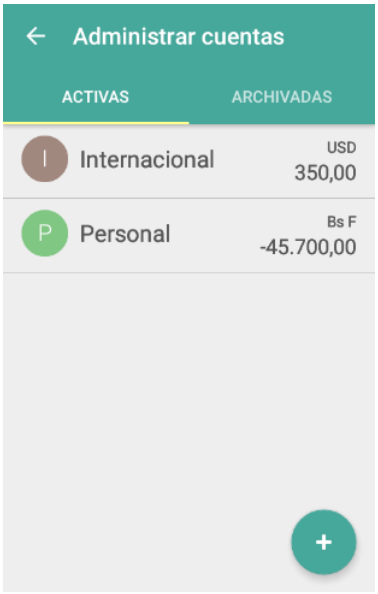

Figura 5.12: Interfaz para listar cuentas

5.3.1.3. Actividades

En la primera parte de la iteración se implementó el modelo de datos del dispositivo móvil, lo que implicó la creación de la base de datos con sus tablas, así como los POJO's correspondientes.

Se creó la vista principal de la aplicación móvil, donde se muestra el total de ingresos y el de gastos, así como el balance general. Para esto, fue necesario crear consultas a la base de datos que permitieran obtener el total de ingresos/gastos de una cuenta. También se realizó una consulta que permite obtener el balance total de una cuenta.

Por otro lado se implementó una vista que permite crear un nuevo gasto, con su fecha, monto y descripción.

5.3.2. Iteración 2

5.3.2.1. Objetivos

- Mostrar la lista de categorías

Figura 5.13: Interfaz para listar cuentas

- Permitir la creación de un nuevo ingreso
- Permitir guardar fotos de un ingreso/gasto
- Asociar un ingreso/gasto a una categoría

5.3.2.2. Resultados

- Se creó una vista para mostrar la lista de categorías a las que puede pertenecer un gasto.
- Se adaptó y reutilizó la vista existente para crear un gasto para permitir la creación de un ingreso
- Se agregó la funcionalidad para tomar fotos relacionadas a un ingreso/gasto
- Se agregó la funcionalidad para poder asociar un ingreso/gasto a una categoría existente

Figura 5.14: Interfaz para mostrar el balance general

5.3.2.3. Actividades

En esta iteración se creó la vista para mostrar las categorías presentes en el dispositivo.

Se creó una consulta para guardar en la base de datos una nueva categoría con su nombre y un ícono que la represente. Además, se creó una consulta para guardar en la base de datos una lista de categorías por defecto.

También se adaptó la vista existente para la creación de un gasto, de manera que se pueda reutilizar esta vista para la creación de un ingreso.

Por otra parte, se implementó la funcionalidad para tomar y guardar fotos asociadas a un ingreso/gasto. Para esto se tuvo que crear un método que permita abrir la aplicación de la cámara del dispositivo desde la aplicación móvil. Luego, se tuvo que adaptar la vista para la creación de un ingreso/gasto para mostrar las miniaturas de las fotos tomadas. Se puso un límite por parte de la empresa para que el máximo de fotos que se puedan guardar por ingreso/gasto sean 3.

Por último, se modificó la vista para la creación de un ingreso/gasto para que se permi-

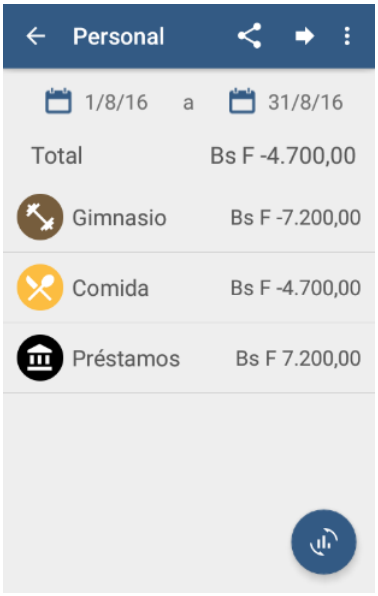

Figura 5.15: Interfaz para mostrar el balance por categorías

tiera asociar una categoría. Se tenía como requerimiento guardar las cuatro categorías que se utilizaron más recientemente. Para esto, se tuvo que crear los métodos necesarios para persistir en el dispositivo móvil las categorías más recientes. Dado que esta información no tiene una estructura compleja, se decidió utilizar un objeto que provee la plataforma de Android, llamado *SharedPreferences*, que permite guardar un conjunto de pares clave-valor. En este caso, se guardó una lista con las categorías más recientes.

5.3.3. Iteración 3

5.3.3.1. Objetivos

- Mostrar la lista de ingresos/gastos del mes actual
- Mostrar los detalles de los ingresos/gastos existentes
- Permitir editar y borrar un ingreso/gasto existente
- Implementar una calculadora para guardar el monto de un ingreso/gasto

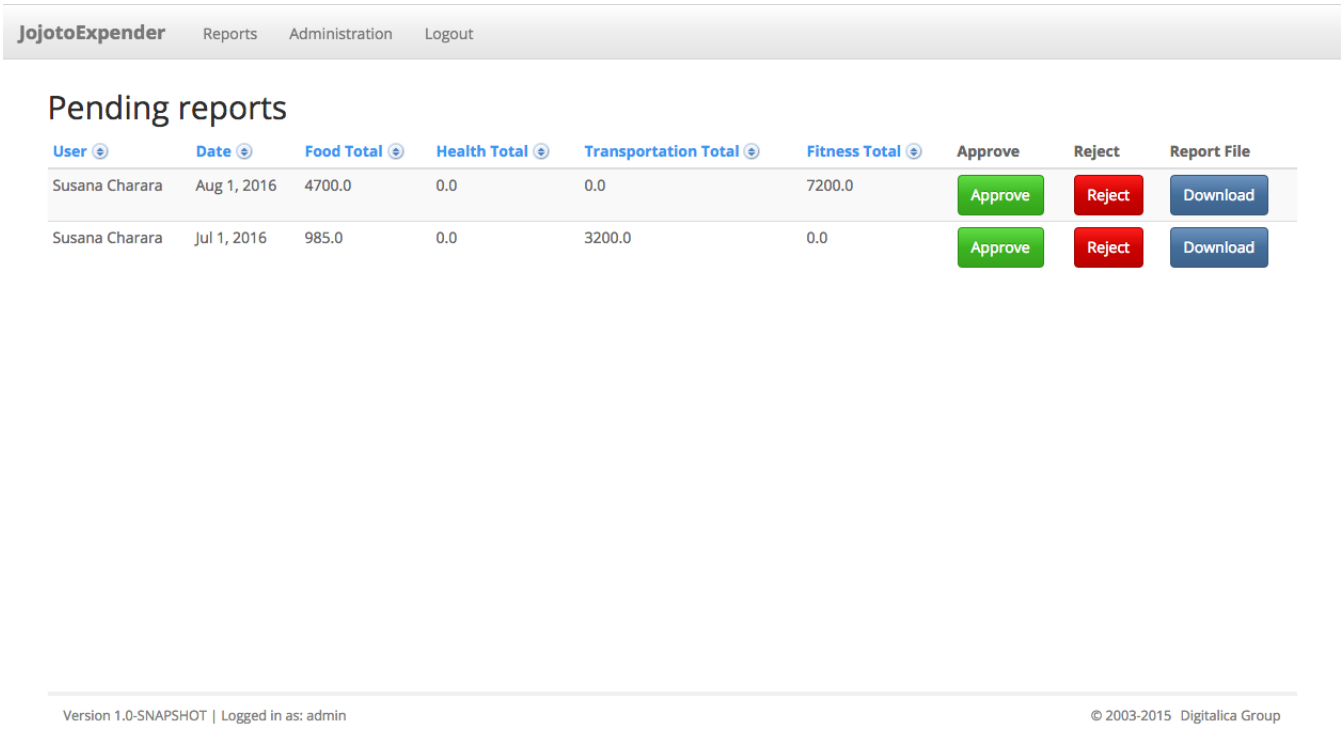

Figura 5.16: Interfaz para listar reportes pendientes de revisión

5.3.3.2. Resultados

- Se creó la vista para mostrar una lista con los gastos y otra con los ingresos del mes actual.
- Se agregó la funcionalidad para ver los detalles de un ingreso/gasto ya existente, y poder editarlo
- Se agregó la funcionalidad para eliminar ingresos/gastos
- Se creó la interfaz para usar la calculadora que permita ingresar el monto de un ingreso/gasto

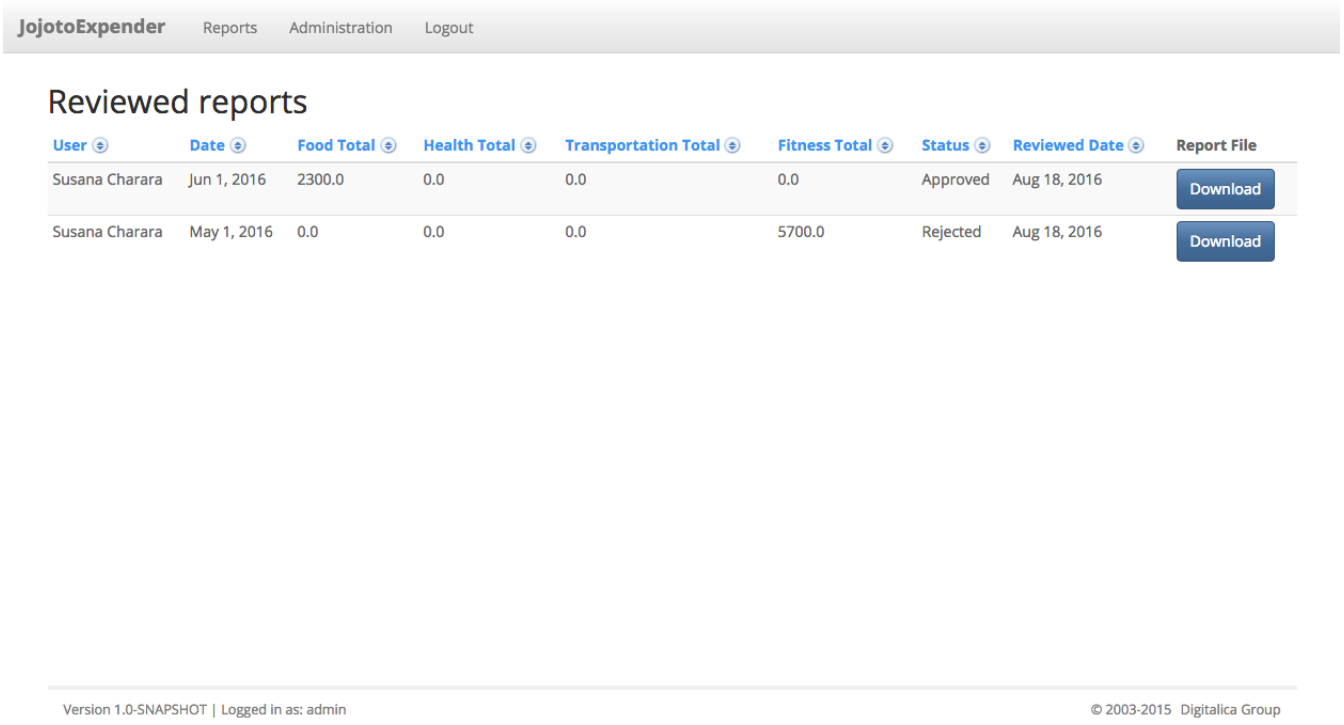

Figura 5.17: Interfaz para listar reportes aprobados/rechazados

5.3.3.3. Actividades

En primer lugar, se adaptó la vista principal de la aplicación para mostrar el total de ingresos y gastos únicamente durante el mes actual. Para esto, se tuvo que modificar la consulta a la base de datos para obtener el total de ingresos/gastos dado un rango de fecha.

Se implementó la funcionalidad para navegar a la lista de ingresos o de gastos del mes desde la vista principal de la aplicación. Para esto, se creó una vista general donde se pueda mostrar una lista de ingresos/gastos, con su descripción (o categoría, en caso de que no tenga descripción), su fecha y su monto.

Por otra parte, se creó la funcionalidad para eliminar ingresos/gastos existentes desde la vista mencionada en el párrafo anterior, lo que implicó realizar cambios a la vista y la creación de una consulta que permitiera eliminar de la base de datos un ingreso/gasto.

También se creó la funcionalidad para editar un ingreso/gasto existente. Para esto no fue

necesario crear una vista nueva, sino que se adaptó la que se tenía para la creación de un nuevo ingreso/gasto, de manera que se muestre la información relacionada (fecha, monto, categoría, descripción y fotos).

Por último, se tenía como requerimiento mostrar una calculadora para ingresar el monto de un ingreso/gasto, en lugar de un teclado numérico común. Por esta razón, se tuvo que modificar la vista de creación de un ingreso/gasto para incluir la nueva funcionalidad.

Para este último requerimiento fue necesario crear un teclado virtual personalizado, pues el teclado numérico del dispositivo no incluye los operadores matemáticos básicos. Se creó el formato del teclado, el cual incluye las teclas presentes en una calculadora básica. También se tuvo que manejar el uso de las teclas: detectar qué tecla fue presionada y realizar una acción en base a esta. Se utilizó una librería para evaluar fórmulas dada una cadena de caracteres.

Para la calculadora, se tenía que mostrar el símbolo decimal de acuerdo con el país para el cual está configurado el teléfono, ya que en algunos países se usa la coma (,) como separador y en otros el punto (.). Por esta razón, se tuvo que obtener el país de configuración del teléfono para decidir qué símbolo decimal mostrar.

5.3.4. Iteración 4

5.3.4.1. Objetivos

- Permitir el manejo de nuevas cuentas

5.3.4.2. Resultados

- Se creó una vista para crear una nueva cuenta
- Se implementó la funcionalidad para listar las cuentas del usuario
- Se implementó la funcionalidad para cambiar de cuenta

- Se implementó la funcionalidad para editar el nombre de una cuenta existente
- Se implementó la funcionalidad para eliminar cuentas existentes
- Se implementó la funcionalidad para cambiar el estado de una cuenta: activa o archivada
- Se implementó la funcionalidad para ver la lista de ingresos/gastos de una cuenta desde su creación hasta la fecha actual

5.3.4.3. Actividades

En la ejecución de las iteraciones anteriores se estuvo trabajando con una sola cuenta creada por defecto en la aplicación. Para esta iteración, se decidió agregar la funcionalidad para poder manejar nuevas cuentas.

Para esto, se creó la vista que permite agregar una nueva cuenta, con su nombre y la moneda en la que van a estar sus ingresos/gastos.

Se creó una vista para mostrar la lista de cuentas guardadas. Se implementó la funcionalidad para cambiar el estado de una cuenta: activa o archivada. Esto implicó modificar la vista de la lista de cuentas, así como la creación de una consulta que permita cambiar el estado de una cuenta. La lista de cuentas se muestra según su estado, es decir, se muestra una lista para la lista de cuentas activas y una para las cuentas archivadas.

Además, se implementaron las funcionalidades para editar el nombre de una cuenta y eliminar cuentas ya existentes. Para ello se crearon consultas a la base de datos que permitieran la modificación y eliminación de cuentas.

También se creó la funcionalidad para cambiar la cuenta que se está mostrando actualmente en el dispositivo. Para mostrar la lista de cuentas activas, de las cuales de puede escoger alguna para el cambio, se agregó una barra lateral (*sidebar*) a la vista principal de la aplicación.

Por último, se adapto la vista existente para mostrar la lista de ingresos/gastos, de manera que se pueda mostrar todos los ingresos/gastos (en una misma lista) asociados a la cuenta que se muestra actualmente.

5.3.5. Iteración 5

5.3.5.1. Objetivos

- Permitir el manejo de las fotos de un ingreso/gasto
- Permitir el manejo de las categorías

5.3.5.2. Resultados

- Se agregó la funcionalidad para ver las fotos de un ingreso/gasto
- Se agregó la funcionalidad para borrar las fotos de un ingreso/gasto
- Se agregó la funcionalidad para cambiar la ubicación en el dispositivo de las fotos tomadas de un ingreso/gasto
- Se agregó la funcionalidad para crear una categoría
- Se agregó la funcionalidad para eliminar una categoría existente
- Se agregó la funcionalidad para editar una categoría existentes

5.3.5.3. Actividades

La primera parte de la iteración se dedicó al manejo de las fotos. En primer lugar se creó la vista para poder ver en pantalla completa las fotos de un ingreso/gasto (esto se hace desde la vista de creación de un ingreso/gasto).

También se implementó la funcionalidad para eliminar una foto, para lo que se tuvo que crear una consulta a la base de datos para eliminar registros de la tabla correspondiente.

Por otra parte, se creo la vista de configuraciones de la aplicación. En esta vista se agregó una opción para cambiar la ubicación en el dispositivo en la que se guardan los archivos de las fotos: memoria interna o memoria extraíble.

La segunda parte del *sprint* se dedicó al manejo de las categorías. Se creó la vista para agregar una nueva categoría. Se modificó la vista existente para la creación de una nueva categoría, de manera que se soporte también la edición. Para estas funcionalidades fue necesario crear consultas a la base de datos que permitieran agregar, editar y eliminar categorías.

5.3.6. Iteración 6

5.3.6.1. Objetivos

- Mostrar un reporte con la lista de ingresos/gastos asociados a una cuenta, en un rango de fecha dado
- Mostrar un reporte con el balance total por categorías asociadas a una cuenta, en un rango de fecha dado
- Permitir el envío de los reportes en un archivo PDF a otras personas

5.3.6.2. Resultados

- Se implementó la funcionalidad para listar los ingresos/gastos de una cuenta en el rango de fecha ingresado por el usuario
- Se implementó la funcionalidad para listar el balance total de las categorías de una cuenta en el rango de fecha ingresado por el usuario

- Se creó la funcionalidad para compartir el reporte con la lista de gastos e ingresos, incluyendo las fotos, de la cuenta en el rango de fecha ingresado.
- Se creó la funcionalidad para compartir el reporte con la lista de gastos e ingresos, sin incluir las fotos, de la cuenta en el rango de fecha ingresado.
- Se creó la funcionalidad para compartir el reporte con el balance total de las categorías de la cuenta en el rango de fecha ingresado.

5.3.6.3. Actividades

Esta iteración se dedicó exclusivamente al manejo de los reportes. Se tenía como requerimiento el manejo de dos tipos de reportes: reporte de balance general y reporte por categorías. Para la generación de ambos reportes, se puede especificar una cuenta y un rango de fechas. El reporte de balance general incluye la lista de todos los ingresos/gastos de la cuenta escogida, dentro del rango de fechas establecido. El reporte por categorías incluye únicamente el monto total de todos los ingresos/gastos (es decir, la suma de ingresos y gastos) divididos por categorías, de la cuenta y rango de fechas escogidos.

Se creó una vista para mostrar el reporte de balance general, que muestra una lista con los ingresos/gastos asociados, sus montos, fechas y descripción (o categoría, en caso de que la descripción sea nula). También se creó una vista para el reporte por categoría, donde se muestra una lista con con el nombre de cada categoría y la suma total de los ingresos y gastos correspondientes.

Luego, se creó la funcionalidad para generar un archivo PDF con la información de los reportes descritos anteriormente. Para esto se usó una librería que provee la plataforma de Android para la creación de archivos PDF.

Se tenía como requerimiento que el usuario pueda escoger generar un archivo con el reporte de balance general incluyendo las fotos de los ingresos/gastos asociados, o sin incluirlas.

Luego se implementaron los métodos necesarios para crear un archivo PDF con el reporte del balance general. Esta funcionalidad luego se adaptó para permitir también la creación del reporte por categorías. Para la escritura del PDF, se tuvo que lidiar manualmente con la paginación.

Por último, se creó la funcionalidad para compartir este archivo PDF con otras personas a través de otras aplicaciones instaladas en el dispositivo móvil. Dentro de estas aplicaciones se incluyen las de correo electrónico y mensajería móvil que soporten el envío de archivos.

5.3.7. Iteración 7

5.3.7.1. Objetivos

- Crear un servicio web para permitir la autenticación de un usuario
- Crear un servicio web para recibir un reporte de gastos
- Crear una aplicación web para mostrar los reportes recibidos
- Permitir la aprobación o rechazo a través de la aplicación web de los reportes recibidos

5.3.7.2. Resultados

- Se diseñó e implementó el modelo de datos del servidor web
- Se creó un servicio al cual la aplicación móvil se puede conectar para la autenticación de usuarios
- Se creó un servicio a través del cual la aplicación móvil puede enviar reportes de gastos
- Se creó una aplicación web, a través de la cual se permite la creación de nuevos usuarios y el manejo de reportes (listar, aprobar y rechazar reportes recibidos)

5.3.7.3. Actividades

En esta iteración se creó el servidor web al cual la aplicación móvil puede enviar peticiones.

En primer lugar, se creó el proyecto en AppFuse, con Tapestry como *framework* para el desarrollo de la aplicación web. Utilizando Hibernate con MySQL, se crearon los POJO's y las tablas necesarias para implementar el modelo de datos mostrado en la figura 5.5.

Luego, se utilizó Spring para crear el servicio web que permitiera la autenticación de un usuario.

Una vez creado, se implementó en la aplicación móvil una funcionalidad que permite iniciar sesión en el dispositivo. Para esto fue necesario crear una vista y los métodos necesarios para conectarse al servicio web para la autenticación del usuario.

Por otra parte, se creó un servicio web para recibir un reporte de gastos. Era necesario enviar tanto el archivo PDF con la información de los gastos, como un resumen de los totales por las categorías de interés para la empresa. Por esta razón, la comunicación entre la aplicación móvil y el servidor para el envío de reportes se hizo a través de una petición HTTP POST Multipart, que contiene el archivo PDF y un JSON con el resumen mencionado anteriormente.

También se creó un módulo para el envío de correos electrónicos. Esto se hizo para notificar al supervisor cuando un nuevo reporte llega al servidor, así como para enviar el archivo PDF adjunto al correo.

Luego de esto se creó la página web a través de la cual un supervisor puede ver la lista de reportes recibidos, así como aprobarlos o rechazarlos. Para esto, se creó una vista y un controlador que permitan el manejo de las acciones de usuario.

Se creó una vista que muestre la lista de reportes pendientes por revisión (es decir, que aún no han sido aprobados ni rechazados). En esta vista se agregó una opción para poder descargar el archivo PDF de cada reporte, así como opciones para aprobarlo/rechazarlo.

Finalmente, se adaptó la vista mencionada anteriormente para poder mostrar una lista de los reportes ya revisados (es decir, que ya fueron aprobados o rechazados).

5.3.8. Iteración 8

Capítulo 6

Conclusiones y Recomendaciones

En este proyecto de pasantía, realizado para la empresa Digitalica Group C.A, se diseñó y desarrolló un prototipo funcional de una aplicación móvil para el registro de gastos e ingresos, así como un servidor web que se comunica con ella. La aplicación fue desarrollada para el sistema operativo Android. Con ella, se permite registrar ingresos y gastos, a los cuales se puede asociar un monto, fecha, descripción, fotos y categoría. El servidor web permite hacer la autenticación de usuarios, así como el envío de reportes compuestos por un conjunto de gastos.

El uso de Scrum como marco de trabajo permitió desarrollar el proyecto progresivamente, y tener una parte del producto funcional luego de cada iteración. Esto fue de gran ventaja porque le facilitaba al dueño del producto (*(*Product Owner)) evaluar constantemente las funcionalidades, verificar que cumplieran con las necesidades del producto y modificarlas de ser necesario. Además, las reuniones diarias (*Daily Scrum*) le permitía a todo el equipo de trabajo estar informado de la evolución del producto, además de ser una oportunidad en que el equipo de desarrollo (en este caso integrado únicamente por el pasante) pudiera solventar cualquier duda o impedimento que surgiera.

A partir de la experiencia que se obtuvo durante el desarrollo del proyecto, complemen-

tado con el aprendizaje adquirido a lo largo de toda la carrera universitaria, se identificaron distintas funcionalidades que pudieran ser añadidas o mejoradas.

Actualmente, toda la información es guardada en el dispositivo móvil. El servidor tiene únicamente información de los reportes que son enviados por la aplicación móvil, lo cual es un subconjunto muy limitado de todos los datos presentes en el dispositivo.

Por otra parte, una de las funcionalidades que tiene la aplicación es poder tomar fotos para cada ingreso/gasto. Estas fotos se hacen a través de la aplicación de cámara instalada por defecto en el dispositivo móvil, lo que implica que se está delegando en una aplicación externa la elección de la resolución en que se tomarán las fotos. Esto significa que la foto pudiera estar tomándose con una calidad muy baja o que, por el contrario, la resolución sea demasiado alta y para guardarla en el dispositivo se estaría gastando espacio más espacio del necesario. Por esta razón, se recomienda a la empresa capturar las fotos internamente, sin hacer uso de otra aplicación. Para lograr esto, Android ofrece un API que permite construir un *activity* para el uso personalizado de la cámara [36].

Para la creación del archivo PDF se presentaron diferentes dificultades. En primer lugar, las librerías existentes en Java para generar archivos con dicha extension hacen uso de otras librerías, las cuales no son soportadas directamente en la plataforma de Android. Por esta razón, en un principio se decidió utilizar una adaptación de una librería de Java para poder ser utilizada en Android. Sin embargo, no se encontró documentación suficiente que permitiera entender su uso. Por esta razón, se decidió utilizar una librería nativa de Android que facilita la creación de archivos PDF, pero que sólo está disponible para versiones de Android a partir de la 4.4. Esta decisión se apoyó en el hecho de que aproximadamente el 78,5 % de los usuarios de Android utilizan una versión mayor o igual a 4.4 [37].

Durante el proceso de investigación, se encontraron librerías que permiten la generación de archivos PDF para todas las versiones de Android. Sin embargo, se debe pagar para

obtener una licencia comercial. Si se piensa lanzar al mercado la aplicación, se recomienda a la empresa evaluar si dejar de ofrecer la funcionalidad a un porcentaje de usuarios generará mayores pérdidas que la adquisición de una licencia comercial de alguno de los productos que pueda solucionar el problema.

Bibliografía

[1] "Digitalica — the technology partner of your disruptive ideas." http://digitalicagroup.com/, consultado el 30 de julio de 2016.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Dessign Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1 ed.

[3] E. Freeman, E. Freeman, K. Sierra, and B. Bates, *Head First: Design Patterns*. O'Reilly, 1 ed., 2004.

[4] D. Garlan and M. Shaw, *An introduction to Software Engineer*. World Scientific, 1 ed., 1993.

[5] "Modelo-vista-controlador." https://msdn.microsoft.com/en-us/library/ff649643.aspx, consultado el 18 de agosto de 2016.

[6] "Mvc or mvp pattern - what's the difference?." http://www.codeproject.com/Articles/288928/Differences-between-MVC-and-MVP-for-Beginners, consultado el 18 de agosto de 2016.

[7] "Differences between mvc and mvp for beginners." http://www.codeproject.com/Articles/288928/Differences-between-MVC-and-MVP-for-Beginners, consultado el 10 de julio de 2016.

[8] "Tipos de arquitectura cliente servidor." https://www.ibiblio.org/pub/Linux/docs/LuCaS/Manuales-LuCAS/doc-curso-salamanca-LAMP/lamp-teoria-html/ch01s02.html, consultado el 2 de agosto de 2016.

[9] "Web service." https://en.wikipedia.org/wiki/Web_service, consultado el 2 de agosto de 2016.

[10] "Http made really easy." http://www.jmarshall.com/easy/http/#whatis, consultado el 15 de agosto de 2016.

[11] "Http methods get vs post." http://www.w3schools.com/tags/ref_httpmethods.asp, consultado el 15 de agosto de 2016.

[12] "Rfc1341(mime): 7 the multipart content type." https://www.w3.org/Protocols/rfc1341/7_2_Multipart.html, consultado el 2 de agosto de 2016.

[13] "Pojo." http://www.martinfowler.com/bliki/POJO.html, consultado el 17 de agosto de 2016.

[14] "Understanding pojo." https://spring.io/understanding/POJO, consultado el 17 de agosto de 2016.

[15] "Core j2ee patterns - data access object." http://www.oracle.com/technetwork/java/dataaccessobject-138824.html, consultado el 17 de agosto de 2016.

[16] "Services - appfuse - appfuse confluence." http://appfuse.org/display/APF/Services, consultado el 9 de agosto de 2016.

[17] "What is android os?." https://www.techopedia.com/definition/14873/android-os, consultado el 30 de julio de 2016.

[18] "Android architecture." http://www.tutorialspoint.com/android/android_architecture.htm, consultado el 16 de agosto de 2016.

[19] "What is java? definition and meaning." http://www.businessdictionary.com/definition/Java.html, consultado el 26 de julio de 2016.

[20] "Android software development." https://en.wikipedia.org/wiki/Android_software_development, consultado el 30 de julio de 2016.

[21] "What is mysql?." http://searchenterpriselinux.techtarget.com/definition/MySQL, consultado el 26 de julio de 2016.

[22] "Sqlite." https://es.wikipedia.org/wiki/SQLite, consultado el 30 de julio de 2016.

[23] "Maven - introduction." https://maven.apache.org/what-is-maven.html, consultado el 30 de julio de 2016.

[24] "Jetty." https://es.wikipedia.org/wiki/Jetty, consultado el 26 de julio de 2016.

[25] "Android studio." https://es.wikipedia.org/wiki/Android_Studio, consultado el 26 de julio de 2016.

[26] "Eclipse - the eclipse foundation open source community website." http://www.eclipse.org/home, consultado el 30 de julio de 2016.

[27] S. Chacon and B. Straub, *Pro Git: Everything you ned to know about Git.* Apress, 2 ed., 2014.

[28] "Home - appfuse - appfuse concluence." http://appfuse.org/display/APF/Home, consultado el 26 de julio de 2016.

[29] "Hibernate." https://es.wikipedia.org/wiki/Hibernate, consultado el 26 de julio de 2016.

[30] "Apache tapestry home page." http://tapestry.apache.org/index.html, consultado el 30 de julio de 2016.

[31] "Github - google/gson." https://github.com/google/gson, consultado el 26 de julio de 2016.

[32] "Retrofit." http://square.github.io/retrofit/, consultado el 30 de julio de 2016.

[33] K. Rubin, *Essential Scrum: A practical guide to the most popular agile process.* Addison-Wesley Professional, 1 ed., 2012.

[34] "Scrum basics." http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf, consultado el 7 de julio de 2016.

[35] "Artefactos en scrum: claves para una organización diaria." http://www.desarrolloweb.com/articulos/artefactos-scrum.html, consultado el 7 de julio de 2016.

[36] "Camera — android developers." https://developer.android.com/guide/topics/media/camera.html, consultado el 17 de agosto de 2016.

[37] "Dashboards — android developers." https://developer.android.com/about/dashboards/index.html?hl=es, consultado el 29 de julio de 2016.