Universidad de las Ciencias Informáticas

Facultad 1

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

“Tienda de aplicaciones Android”

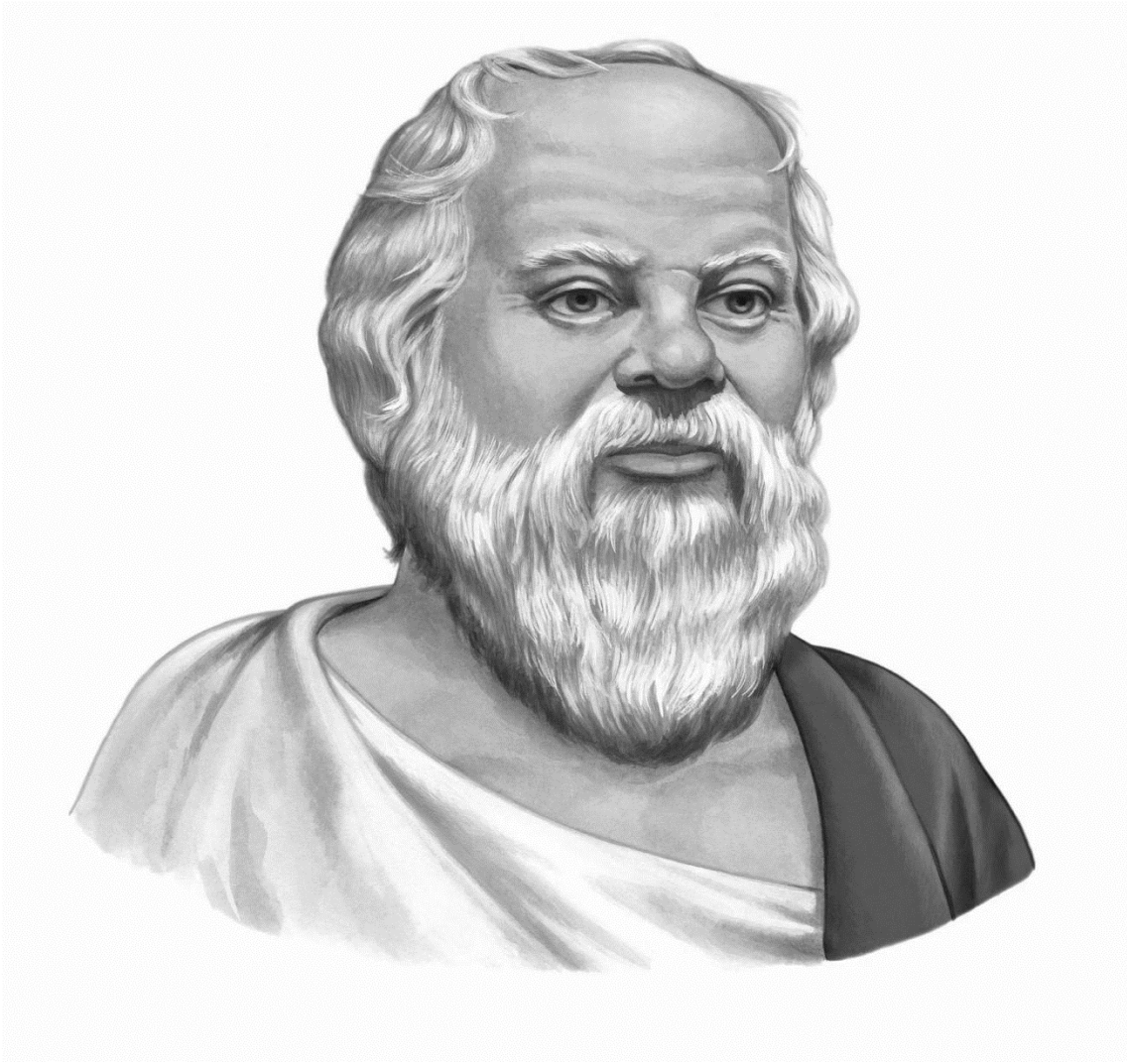
Autor: Michael Domenech Nodarse

Tutores: Ing. Nurisel Palma Pérez
Ing. David Hiram Hernandez Peña
Ing. Leonardo Quesada Cruz

La Habana, junio 2017

“Año 59 de la Revolución”

Frase



"La verdadera sabiduría está en saber que no se sabe nada".

Sócrates

Declaración del autor

Declaro por este medio que yo Michael Domenech Nodarse, con carnet de identidad 93012606625 soy autor principal del trabajo titulado “Tienda de aplicaciones Android” y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2017.

Michael Domenech Nodarse

_____

Firma de Autor

Ing. Nurisel Palma Pérez

_____

Firma de Tutora

Ing. David Hiram Hernandez Peña

_____

Firma de Tutor

Ing. Leonardo Quesada Cruz

_____

Firma de Tutor

Datos de Contacto

Tutora: Ing. Inst. Nurisel Palma Pérez

Graduada de Ingeniera en Ciencias Informáticas en el 2013 en la Universidad de las Ciencias Informáticas (UCI). Posee la categoría docente de Instructor y forma parte del departamento Servicios Integrales en Migración Asesoría y Sistemas (SIMAYS) donde se desempeña como desarrolladora de la Herramienta para la Migración y Administración de Servicios Telemáticos (HMAST). Posee publicaciones y participación en diferentes eventos como el II Congreso Internacional de Ingeniería Informática y Sistemas de Información, en la Conferencia Científica de la UCI en sus dos ediciones y el III Congreso Multidisciplinario de Ciencias Aplicadas en Latinoamérica. Correo electrónico: npalma@uci.cu

Tutor: Ing. David Hiram Hernandez Peña

Graduado de Ingeniero en Ciencias Informáticas en el 2015 en la Universidad de las Ciencias Informáticas (UCI). Forma parte del Laboratorio de Desarrollo de Aplicaciones Android (DroidLab) en el Centro de Soluciones Libres como desarrollador. Correo electrónico: dhpena@uci.cu

Tutor: Ing. Leonardo Quesada Cruz

Graduado de Ingeniero en Ciencias Informáticas en el 2016 en la Universidad de las Ciencias Informáticas(UCI). Forma parte del Laboratorio de Desarrollo de Aplicaciones Android (DroidLab) en el Centro de Soluciones Libres como desarrollador. Correo electrónico: lquesada@uci.cu

Dedicatoria

A toda mi familia por guiarme a ser la persona que hoy soy, en especial a mi abuela y a mi mamá.

Agradecimientos

A mi abuelo Ernesto, a mi abuela Rosa, a mi padre Angel por siempre estar a mi lado aunque no estén aquí físicamente y a mi otro padre Jose porque sin él no estuviera aquí hoy.

A mi mamá, mi abuela, a mis tíos, a mis primos y a toda mi familia; gracias por su apoyo.

A mis tutores Leo y David por los regaños apoyadores, por defenderme y guiarme en la realización de esta tesis y sobre todo a Nuri por prepararse junto conmigo como si fuéramos para una guerra (ella sabe) y por aguantarme en ocasiones de 8 de la mañana hasta las 5 de la tarde y cuestionándole a veces hasta cuando iba al baño porque quería que me revisara algo, por la comprensión, por los regaños gritados (aunque no literalmente) y por el apoyo como si fuera una madre, muchas gracias.

A mi hermana Aide porque si no la menciono lo que me espera es candela, gracias por tu amistad y por estar a mi lado durante estos cinco años.

A mis amigas Stephany, Rachel y Zuleimys por aguantarme y quererme, las quiero.

A toda mi aula por todos estos años compartidos, les deseo lo mejor.

A Rene y Asdrubal por ser mis amigos, por las noches de aburrimiento compartidas, por las fiestas, los consejos y por todo lo que aún nos espera.

A mis amigos Frank, Dariel, Manchón, Lourdes y Hansel por los momentos en que teníamos que soportarnos estudiando para las pruebas finales o jugando cualquier bobería.

A mi amigo David por seguir juntos en el mismo camino desde la secundaria y que si no fuera tan cabezón se estuviera graduando también en este año.

A Pedro por su gran amistad, por venir junto conmigo a estudiar en la UCI cuando era mi sueño desde que tenía 11 años y después fue el suyo cuando lo convencí de venir a esta escuela, por sentarse a mi lado durante 3 años en el tecnológico explicándole todo lo que no entendía principalmente cuando se estresaba con la profe de matemática, pero hoy te agradezco yo a ti por todas esas asignaturas en las que me enseñabas todo el contenido para la prueba a veces hasta un día antes o incluso horas, porque sabía que tenía a mi hermano de armas a mi lado, aún cuando perdimos el interés por el estudio, tú siempre de alguna forma me hacías estudiar, por eso te dedico el 60% de todas mi notas y gracias a ti también hoy estoy aquí.

Y a todas mis amistades que hoy se encuentran presentes, gracias por estar aquí.

Resumen

La presente investigación tuvo como objetivo desarrollar una tienda virtual que gestione aplicaciones móviles de sistema operativo Android, garantizando la publicación y descarga para la comunidad en la Universidad de las Ciencias Informáticas. Se empleó el método teórico Analítico-Sintético para el análisis de bibliografías existentes sobre las plataformas web de tiendas virtuales y la síntesis del contenido necesario para la ejecución de la investigación. Además, se utilizó el método empírico Observación para analizar el comportamiento de las tiendas virtuales con mayor cantidad de aplicaciones existentes en la web. Para el desarrollo se definieron como tecnologías a usar: la herramienta Visual Paradigm para el Lenguaje Unificado de Modelado, el entorno de desarrollo integrado Visual Studio, para el desarrollo del *back-end* Node.js, para el desarrollo del *front-end* AngularJS, diferentes lenguajes de programación como JavaScript, el sistema gestor de base de datos MySQL y como sistema de control de versiones Git. El proceso de desarrollo de software estuvo guiado por la metodología Proceso Unificado Ágil en su versión para la Universidad de las Ciencias Informáticas. Finalmente se obtuvo una tienda que permite publicar y descargar aplicaciones de sistema operativo Android desarrolladas en la universidad. La tienda contribuye además a fomentar el interés de desarrollo de las mismas por parte de la comunidad universitaria.

Palabras clave: Android, DroidLab, tecnología móvil, tienda virtual.

Índice

Índice

Índice

Índice

Índice de Imágenes

Índice

Índice de Tablas

Introducción

En la actualidad el desarrollo de las Tecnologías de la Información y las Comunicaciones (TIC) es un factor importante en el avance de la sociedad. La tecnología móvil se encuentra en constante auge dentro de esta rama trayendo beneficios significativos (Itunews, 2015). Está dirigida a satisfacer las necesidades y gustos, por tanto, en el desarrollo de aplicaciones se ha producido un rápido aumento para este mercado tan demandante. Actualmente, la tecnología móvil ha cubierto la mayoría de las áreas de servicio de comunicaciones y entretenimiento. Enfocándose en las aplicaciones se genera un mercado cautivador de esta tecnología, a los miles de usuarios que día a día adoptan el uso de servicios, tal como lo es, el envío de mensajes de texto y multimedia. En los últimos años la actualización de la información de las redes sociales y el uso de aplicaciones ha ido en aumento (Tilves, 2015).

El sistema operativo más usado en este tipo de aplicaciones es Android, sistema que está basado en Linux. De acuerdo con un estudio llevado a cabo el 81.7% de la base actual de usuarios de teléfonos inteligentes usan Android y su mayor competidor iOS[1], ocupa 17.9% del mercado (Gartner, 2017).

Las aplicaciones Android para los teléfonos inteligentes pueden ser descargadas de las tiendas virtuales o tiendas online ya sea por compra o gratis. Todo desarrollador puede subir sus aplicaciones a estos sitios web mientras cumplan con los requisitos que requieren las aplicaciones. Cuba no cuenta con una tienda de aplicaciones propia a la cual acceder, y en las existentes como *Google Play* o *Apple AppStore*, varias de sus aplicaciones son por pago y en el país no se tienen implementadas las pasarelas de pago necesarias, por lo que se hace imposible descargar las mismas. Por causa del bloqueo Cuba no puede usar la compañía de transferencia de dinero PayPal, por tanto, los desarrolladores de aplicaciones del país se ven limitados a que sus aplicaciones no sean conocidas. Ya que, para subir aplicaciones a las tiendas mencionadas y otras, se debe crear una cuenta en la misma como desarrollador, y a su vez depositar una suma de dinero. Debido a que la compañía PayPal restringe toda transacción desde y hacia Cuba, no es posible usar estas tiendas. Además, el código de estos sitios no está liberado pues pertenecen a organizaciones privativas y una de las políticas en Cuba es fomentar el uso de software libre y aplicaciones de código abierto como vía para alcanzar la soberanía tecnológica.

[1] iOs: es el sistema operativo diseñado por Apple para sus productos.

Introducción

Teniendo en cuenta este panorama, la Universidad de las Ciencias Informáticas (UCI) no queda exenta del desarrollo de aplicaciones en esta esfera, por lo que surge el Laboratorio de Android (DroidLab). Este laboratorio es una nueva línea de desarrollo e investigación impulsada por el Centro de Software Libre (CESOL) con el fin de promover y potenciar el desarrollo de aplicaciones utilizando el sistema operativo Android. No solo en este centro se desarrollan aplicaciones y la universidad no tiene un medio a través del cual se puedan publicar aplicaciones, descargarlas y así fomentar el interés de desarrollo de las mismas por parte de la comunidad universitaria. Como consecuencia, los usuarios en la universidad están desactualizados en cuanto a los productos e innovaciones que se desarrollan en la UCI.

Partiendo de esta situación se plantea el siguiente **problema de la investigación:** ¿Cómo gestionar aplicaciones de sistema operativo Android desarrolladas en la UCI, de forma que se facilite su publicación y descarga para la comunidad universitaria?

Se establece como **objeto de estudio** las tiendas virtuales de aplicaciones móviles, enmarcado en el **campo de acción** las tiendas virtuales de aplicaciones móviles de sistema operativo Android.

Para dar solución al problema de investigación planteado se define como **objetivo general** desarrollar una tienda virtual que gestione aplicaciones móviles de sistema operativo Android, garantizando la publicación y descarga para la comunidad universitaria.

Para darle cumplimiento al objetivo general planteado, se han proyectado los siguientes **objetivos específicos:**

1. Caracterizar las tiendas virtuales de aplicaciones móviles de sistema operativo Android.
2. Diseñar la tienda virtual para gestionar aplicaciones de sistema operativo Android de forma que se facilite su publicación y descarga para la comunidad universitaria.
3. Implementar las funcionalidades de la tienda virtual para gestionar aplicaciones de sistema operativo Android de forma que se facilite su publicación y descarga para la comunidad universitaria.
4. Probar las funcionalidades de la tienda virtual.

Como materialización y contribución al cumplimiento de los mismos, las **tareas de investigación** planificadas son**:**

1. Caracterización de las plataformas web de tiendas virtuales de aplicaciones móviles de sistema operativo Android.
2. Definición de los requisitos funcionales y no funcionales a implementar.
3. Especificación de las Historias de Usuario.
4. Descripción de la arquitectura.
5. Implementación de la tienda virtual.
6. Diseño y aplicación de los casos de prueba a la tienda virtual implementada.

Para el cumplimiento del objetivo general planteado se utilizarán los siguientes métodos de investigación:

Métodos Teóricos:

1. **Analítico-Sintético**: este método se utiliza para realizar el análisis y estudio de las bibliografías existentes sobre las plataformas web de tiendas virtuales y lograr obtener de manera sintetizada el contenido necesario y suficiente para la ejecución de la investigación.

Métodos Empíricos:

1. **Observación**: este método se utilizó para analizar el comportamiento de las tiendas virtuales con mayor cantidad de aplicaciones existentes en la web.

Para un mejor entendimiento de la realización de este trabajo se muestra un breve resumen de cada uno de sus capítulos.

Capítulo 1. Fundamentación teórica: en este capítulo se aborda todo lo referente a la fundamentación teórica de la investigación de tiendas virtuales de aplicaciones móviles de sistema operativo Android, se realiza el estudio del arte de sitios web similares. Para un mejor entendimiento se detallan los conceptos asociados al tema y componentes generales de una tienda virtual. Se adopta la metodología de desarrollo a emplear, se selecciona el lenguaje de programación que deberá usarse, así como la herramienta y lenguaje para el modelado. Queda plasmado el entorno de desarrollo integrado, el sistema gestor de base de datos que será usado y el sistema de control de versiones. Además de las herramientas para el desarrollo del servidor y las interfaces de la tienda virtual.

Capítulo 2. Análisis y diseño de la tienda virtual: en este capítulo se describe la propuesta de solución y se ofrecen detalles de los principales aspectos relacionados con su diseño. Se definen las funcionalidades que deberá tener la tienda virtual y el diseño de la arquitectura que tendrá la misma para una mejor

organización de sus componentes. Se definen los patrones de diseño usados, así como el diagrama de clases del diseño para un mejor entendimiento de cómo funciona la tienda virtual de aplicaciones.

Capítulo 3. Implementación y pruebas: en este capítulo se abordan las fases de implementación y pruebas, realizadas a la tienda virtual. Se comentan los tipos de pruebas realizados para evaluar el producto final, en este caso, las pruebas internas y las pruebas de aceptación. Además, se elabora un diagrama de despliegue que posibilita conocer el nivel de interacción de la tienda virtual con otros componentes.

Capítulo 1: Fundamentación teórica

En el presente capítulo se realiza el estudio de tiendas virtuales de aplicaciones móviles de sistema operativo Android. Además, se definen las herramientas que serán utilizadas en el desarrollo para la solución.

1.1. Conceptos asociados al dominio del problema

Para una mejor comprensión del tema se definen los siguientes conceptos, que están relacionados a las tiendas virtuales de aplicaciones.

Aplicación móvil

Aplicación informática diseñada para ser ejecutada en dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS, BlackBerry OS, Windows Phone, entre otros (Cuello y Vittone, 2015).

Tienda virtual o en línea

Se puede describir como una plataforma de comercio que se vale de un sitio web para realizar sus ventas y transacciones. Por lo general, las compras en una tienda virtual se pagan con tarjeta de crédito en el mismo sitio web y luego los productos son enviados por correo. Sin embargo, se pueden utilizar otros medios de pago como transferencias bancarias, cupones de pago, entre otros. En la mayoría de los casos la tienda virtual suele requerir que los usuarios se registren ingresando sus datos antes de poder realizar una compra (Headways Media, 2015).

Las tiendas virtuales deben cumplir con ciertos estándares o incorporar ciertos componentes para hacer más sencilla la navegación del usuario en el sitio.

1.2. Componentes de una tienda virtual de aplicaciones

Las tiendas virtuales incorporan ciertos elementos básicos de diseño para llamar la atención de quien visita estas páginas.

Diseño

La tienda virtual debe estar decorada con los propios contenidos de tal manera que las aplicaciones sean las que destaquen por encima del diseño. Combinar colores para crear una visión cálida y agradable. Los

menús de navegación de las diferentes páginas que conforman la tienda virtual deben seguir un orden establecido.

Usabilidad

El catálogo de aplicaciones debe estar visible desde el primer momento. Fácil acceso a las aplicaciones mediante una clara navegación por categorías y subcategorías. El proceso de la compra debe estar enfocado a la aplicación de una forma clara y rápida. Disponer de un potente buscador que ofrezca la posibilidad de acceder al catálogo de aplicaciones por distintos criterios como (precio, fecha u orden alfabético). Facilitar el acceso a apartados de servicio de la tienda como información de contacto, forma de comprar, condiciones generales, etc.

Accesibilidad

El catálogo de aplicaciones debe ser accesible por categorías comerciales. Es importante la correcta clasificación de las aplicaciones utilizando categorías como novedades, aplicaciones más vendidas, aplicaciones en oferta, etc. En todos los casos se tendría que escoger el tamaño del texto óptimo y destacando aquella información más relevante: nombre de la aplicación, botón Comprar, precio, etc. Además de hacer uso correcto del etiquetado de las imágenes favoreciendo su indexación.

Catálogo de aplicaciones

El catálogo de aplicaciones y servicios es la carta de presentación a los clientes. Se debe prestar atención y cuidado a la hora de seleccionar qué aplicaciones y servicios se van a ofrecer, y cómo se van a mostrar y destacar.

Carrito de la compra

La cesta o carrito de la compra es un elemento indispensable en la tienda virtual. Este elemento debe ofrecer la posibilidad de añadir, eliminar o modificar las aplicaciones que durante la navegación se han ido seleccionando e incorporando. Se debe dar la posibilidad, que el cliente pueda visualizar de una forma clara: las referencias compradas especificando la cantidad, los gastos de envío, impuestos aplicables de forma directa e importe total del pedido.

Mecanismos de promoción y ofertas

Uno de los factores más importantes que atraen a los clientes de una tienda virtual e influyen en la decisión de compra es el precio. El precio de las aplicaciones debe estar siempre bien visible.

Motor de búsqueda

Con la idea de facilitar al cliente encontrar las aplicaciones y servicios, es indispensable disponer de un potente motor de búsqueda o buscador integrado que permita la búsqueda de aplicaciones por diversos criterios.

Registro y área de usuario

El registro de clientes es un aspecto muy importante de una tienda virtual, tanto para que los clientes realicen sus pedidos, como a la hora de poder llevar a cabo acciones posteriores de fidelización y captación de nuevos clientes partiendo de una base de datos de clientes extensa. En la zona de registro de un comercio electrónico los clientes deben poder consultar, al menos: datos personales, estado del pedido realizado, datos de envío y facturación (Padilla, 2013).

Una vez detallados los componentes de una tienda virtual, posteriormente se realiza un estudio de algunas de las existentes.

1.3. Estudio de tiendas virtuales de aplicaciones

En este epígrafe se muestra un estudio de las tiendas virtuales de aplicaciones móviles de sistema operativo Android. Para la selección de estas, el autor se basó en las estadísticas que se muestran en la Figura 1 Esta gráfica contiene las tiendas virtuales de aplicaciones móviles más usadas en el mundo, ya sea por su sistema operativo o por la cantidad de aplicaciones que posee.

Para el estudio en la presente investigación, se descartan tres de estas tiendas ya que se basan en un sistema operativo diferente a Android, estas son: Apple, Windows Phone y BlackBerry World. A continuación, se describen Google Play y Amazon. Además, se incluye en el estudio Aptoide y F-Droid porque a pesar de ser menos usadas, cumplen con la política de CESOL de emplear aplicaciones de código abierto, a diferencia de las tiendas mencionadas anteriormente que son software privativo.

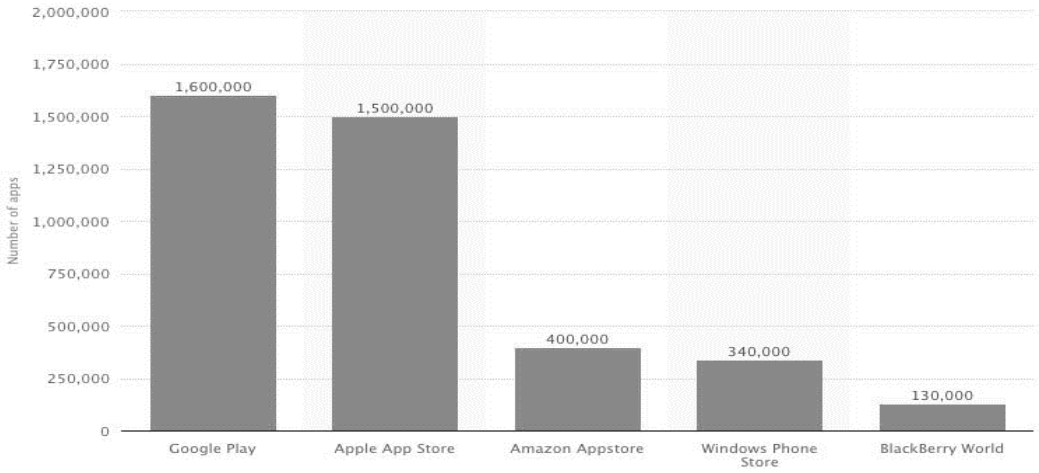


Figura 1: Gráfica comparativa de tiendas de aplicaciones (Xatakamovil, 2015).

Google Play

Se crea en marzo de 2012 como una evolución de Android Market con el objetivo de integrar sus distintas aplicaciones en una sola plataforma: Google Play Store. Está basada en la tecnología "en nube" o *cloud computing*[2] de modo que el usuario tiene siempre disponible sus archivos ya que el almacenamiento no se hace físicamente en el dispositivo sino en servidores especialmente diseñados para ello (Tecnología Qode, 2013). Recomienda las aplicaciones más descargadas. Ofrece más de 30 categorías, y dentro de cada una las aplicaciones se clasifican según una combinación de particularidades, reseñas, descargas, país y otros factores. La búsqueda permite a los usuarios encontrar una aplicación rápidamente (Google, 2016). No puede ser tomada como solución porque la manera de comprar sus aplicaciones no es posible para los residentes en Cuba, debido a que no se poseen las mismas pasarelas de pago.

Amazon

Fue lanzada para Android el 22 marzo de 2011. Ofrece una aplicación de pago de forma gratuita cada día. De esta forma, todos los usuarios de la aplicación tienen un excelente incentivo para volver a ella de manera periódica. Las aplicaciones descargadas en la Amazon dejan de funcionar si se desinstala el cliente para móvil de la tienda virtual (Kincaid, 2011). No se puede usar como solución porque la manera de comprar

[2] Cloud computing: computación en la nube, por su traducción en español.

sus aplicaciones no es posible para los residentes en Cuba, debido a que no se poseen las mismas pasarelas de pago.

Aptoide

La función principal de esta tienda virtual es ofrecer aplicaciones Android gratuitas. Permite a los usuarios registrados agregar su propio repositorio de aplicaciones y a la vez acceder a los ya existentes. Incluye la interacción con redes sociales como Facebook y Twitter. Permite mostrar las aplicaciones más votadas. Es una alternativa que libera su código fuente (Aptoide, 2016). Tiene un problema de seguridad ya que se suben muchas aplicaciones maliciosas que se hacen pasar por versiones gratuitas de aplicaciones de pago (Xataca Android, 2015).

F-Droid

Funciona de manera similar a la tienda Google Play Store, pero solo ofrece aplicaciones libres de pago y de código abierto. No utiliza sistema de autenticación. Es una alternativa que libera su código fuente. También incluye herramientas completas de creación y lanzamiento para administrar el proceso de convertir el código fuente de la aplicación en versiones publicadas (F-Droid, 2016).

1.3.1 Valoración del estudio de las tiendas de aplicaciones

Partiendo del análisis realizado sobre las tiendas de aplicaciones y la comparación que se muestra en la Tabla 1, no puede emplearse ninguna de estas plataformas como solución al problema de la investigación.

Tabla 1: Tabla comparativa de las tiendas virtuales estudiadas.

Tienda virtual	Código	Sistema de pago
Google Play	Privativo	Transacción
Amazon	Privativo	Transacción
Aptoide	Libre	Libre de pago
F-Droid	Libre	Libre de pago

Dos de estas tiendas son las que poseen mayor cantidad de aplicaciones (Xatakamovil, 2015), se basan en software privativo y los desarrolladores cubanos no pueden subir aplicaciones debido a las restricciones del bloqueo y las condiciones impuestas por la compañía de transferencia monetaria PayPal. Además, que no se pueden descargar las aplicaciones que son de pago debido a que este se realiza mediante transacción monetaria. También, hay que tener en cuenta que uno de los objetivos de la UCI es utilizar herramientas de software libre y desarrollar aplicaciones de código abierto.

En el caso de las tiendas de código abierto, F-droid tiene el inconveniente de no presentar un sistema de autenticación. Por otra parte, Aptoide presenta un problema de seguridad, ya que se suben muchas aplicaciones maliciosas al sitio que se hacen pasar por versiones gratuitas de aplicaciones de pago. Además, en sus respectivos servidores utilizan para el manejo de la información ficheros en formato Lenguaje de Marcas Extensible (XML, del inglés *eXtensible Markup Language*), lo que trae consigo que se haga más engorrosa la transformación, lectura y escritura de los datos a medida que aumenta el tamaño de estos ficheros, ralentizando el funcionamiento (Quesada, 2016). Si se publicaran las aplicaciones desarrolladas en la universidad en alguna de estas tiendas, no se podría tener control de lo que se hagan con ellas posteriormente. Por tanto, se hace necesario garantizar la seguridad de las aplicaciones desarrolladas en la UCI. Como ninguna de las tiendas mencionadas puede administrarse directamente, se descartan. Se decide desarrollar una tienda virtual de aplicaciones móviles de sistema operativo Android que funcione como tienda virtual con sistema de pago por puntos, ya que el objetivo no es descargar aplicaciones mediante transacciones monetarias debido a que es una tienda para la comunidad universitaria.

Sin embargo, pueden tomarse en cuenta elementos referentes al diseño de interfaz de usuario para el desarrollo de la solución. Se identificaron funcionalidades comunes en estos sitios: autenticar usuario, subir aplicación, descargar aplicación, realizar búsquedas, filtrar el listado de aplicaciones por categorías y ordenarlo en cuanto a varios criterios. Posteriormente se definen la metodología de desarrollo de software y herramientas a emplear para el desarrollo de la misma.

1.4. Metodología de desarrollo de software

Una metodología de desarrollo en ingeniería de software, es un conjunto de herramientas, técnicas, procedimientos y soporte documental encaminados a estructurar, planificar y controlar el proceso de

desarrollo de forma organizada y lógica, que tienen como objetivo apoyar a los desarrolladores en la creación de un nuevo software (Kaisler, 2005).

En la presente investigación se emplea Proceso Unificado Ágil (AUP, del inglés *Agil Unified Process*), que es una versión simplificada de la metodología de desarrollo Proceso Unificado Racional (RUP, del inglés *Rational Unified Process*). Describe de una manera fácil y simple de entender la forma de desarrollar aplicaciones de software de negocio empleando técnicas ágiles y conceptos que se mantienen válidos en RUP (Ambler, 2014).

La UCI desarrolló una versión de la metodología de desarrollo de software AUP, con el fin de crear una metodología que se adapte al ciclo de vida definido por la actividad productiva de la universidad. Esta versión decide mantener para el ciclo de vida de los proyectos la fase de Inicio, pero modificando el objetivo de la misma, y se unifican las restantes fases de la metodología de desarrollo de software AUP en una sola, denominada Ejecución, y agregándose también una nueva fase denominada Cierre (Rodríguez Sánchez, 2015).

Como se tiene un negocio bien definido de lo que se va a desarrollar se escoge el escenario cuatro en el que se modela el sistema con Historias de Usuario (HU), ya que el cliente es el jefe del centro, siempre se encuentra al tanto de los detalles de los requisitos lo que facilita la implementación, prueba y validación de los mismos. Además, no es un proyecto muy extenso lo que facilita que las HU no contengan tanto contenido informativo.

1.5. Herramientas y tecnologías

El proceso de desarrollo de software se apoya en el uso de diferentes lenguajes y herramientas, las cuales, unidas a la metodología seleccionada, conforman el ambiente de desarrollo de un sistema. A continuación, se detallan las seleccionadas en la presente investigación.

1.5.1. Lenguaje de modelado

Los lenguajes de modelado utilizan modelos visuales y diagramas que realizan esa representación de manera precisa, entendible y económica, lo que facilita su uso en las herramientas de Ingeniería de Software Asistida por Computadora (CASE, del inglés *Computer Aided Software Engineering*) (Zapata, 2009).

UML v2.5

Lenguaje Unificado de Modelado (UML, del inglés *Unified Modeling Language*), es un lenguaje de modelado de sistemas de software. Abarca la estructura de las aplicaciones, su comportamiento y arquitectura, procesos de negocio y datos. Incluye todo el proceso de desarrollo de software (Unified Modeling Language, 2015). Utilizado para especificar, construir, visualizar y documentar los artefactos de un sistema de software orientado a objetos (Object Management Group, 2015). Este lenguaje de modelado se utilizó en la realización de los artefactos generados para el desarrollo de todo el software, en el cual se realizó una documentación detallada de su proceso.

1.5.2. Herramienta de modelado

En las últimas décadas se ha trabajado en el área de desarrollo de sistemas, para encontrar técnicas que permitan incrementar la productividad y el control de calidad en cualquier proceso de elaboración de software. A continuación, se verán algunas de las características más importantes de esta herramienta CASE.

Visual Paradigm v14.0

Propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Soporta, entre otros, el lenguaje de modelado UML. Permite el análisis, diseño, codificación, prueba y despliegue de aplicaciones. Está disponible para Windows y Linux (Visual Paradigm, 2016). Entre sus características se encuentra la capacidad de realizar ingeniería directa e inversa, generación de código, generación de bases de datos, la incorporación de varios idiomas y una licencia gratuita y comercial, así como un diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.

Se selecciona Visual Paradigm en su versión 14.0 para el modelado de diagramas porque es una herramienta multiplataforma, permite la interoperabilidad con otras aplicaciones. Permite aumentar la calidad del desarrollo de un software, soporta su ciclo de vida completo y está enfocado al negocio. Es una herramienta fácil de utilizar y presenta soporte para aplicaciones web.

1.5.3. Entorno de desarrollo integrado

(IDE, del inglés *Integrated Development Environment*), es un programa compuesto por un conjunto de herramientas para que el programador las utilice. Puede dedicarse en exclusiva a un solo lenguaje de

programación o bien, puede utilizarse para varios. El entorno de desarrollo es imprescindible en la producción de un software.

Visual Studio

Se erige sobre un editor de código/interfaz que soporta los más variados lenguajes (desde HTML5 + JQuery, a C++ para dispositivos embebidos, pasando por Phyton con Django, o XMAL para Windows Phone), proporcionando un completo intellisense [3] predictivo y múltiples herramientas de refactorización y aceleración de la codificación. La inclusión de pruebas de rendimiento, y del análisis estático del código, redondean un módulo que orienta al desarrollador hacia las mejores prácticas de codificación y de técnicas avanzadas de programación. La documentación arquitectónica permite modelar en UML toda la estructura del proyecto, incluso generando código desde los diagramas, navegar por la vista de clases, verificar las referencias circulares, etc. Permite construir las aplicaciones para todos los dispositivos, plataformas y sistemas operativos soportados, además, de realizar decenas de operaciones y validaciones de depuración que permiten encontrar los fallos de manera fácil y sencilla (Genbetadev, 2013).

Puede conectarse contra una base de datos Lenguaje de Consulta Estructurada (SQL, del inglés *Structured Query Language*), comparar los esquemas, comparar los datos, lanzar querys; crear un Identificador Único Global (GUID, del inglés *Globally Unique Identifier*), ofuscar y analizar código, obtener la ejecución detallada de procesos y optimizar y configurar el propio IDE, son algunas de las decenas de herramientas que incluye Visual Studio. En las encuestas que se hacen a los programadores en todo el mundo, Visual Studio fue la herramienta de entorno de desarrollador más popular para desarrolladores web, de escritorio y científicos de datos (Stack Overflow, 2017).

1.5.4. Framework de desarrollo del front-end

AngularJS v2.0

Para la creación de las vistas o el *front-end*[4] se escoge AngularJS en su versión 2. Es un proyecto de código abierto, realizado en JavaScript que contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para llevarlas a cabo. Es un framework de desarrollo, en este caso sobre el lenguaje JavaScript con programación del lado del cliente. Ofrece

[3] Intellisense: es la aplicación de autocompletar de Visual Studio.
[4] Front-end: interfaz, por su traducción al español.

herramientas para que los desarrolladores sean capaces de usar el código JavaScript para mejorar el Lenguaje de Marcas de Hipertexto (HTML, del inglés *HyperText Markup Language*). AngularJS tiene como objetivo desacoplar la manipulación del Modelo de Objetos del Documento (DOM, del inglés *Document Objet Model*) de la lógica de aplicación, desacoplar el lado cliente del lado servidor en una aplicación y proveer estructura para el desarrollo de una aplicación (DesarrolloWeb.com, 2014). Los valores de las variables de JavaScript se pueden configurar manualmente, o ser recuperados de los recursos Notación de Objetos de JavaScript (JSON, del inglés *JavaScript Object Notation*) estáticos o dinámicos. Está construido en torno a la creencia de que la programación declarativa es la que debe utilizarse para generar interfaces de usuario y enlazar componentes de software (Angular, 2017).

Este framework adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de una *data binding*[5] bidireccional que permite la sincronización automática de modelos y vistas. Como resultado, pone menos énfasis en la manipulación del DOM y mejora la testeabilidad y el rendimiento (Atraura.com, 2016). AngularJS sigue el patrón MVVM (*Model View View-Model*) de ingeniería de software y alienta la articulación flexible entre la presentación, datos y componentes lógicos. El modelo de vista de MVVM es un convertidor de valor, lo que significa que el modelo de vista es responsable de convertir los objetos de datos del modelo de tal manera que los objetos sean administrados y presentados fácilmente. A este respecto, el modelo de vista es más modelo que vista y maneja la mayoría si no toda la lógica de visualización de la vista (Smith, 2009).

Provee un sistema de módulos propio, también trae su propio sistema de plantillas, provee servicios propios para traer datos vía JavaScript asíncrono y XML (AJAX, del inglés *Asynchronous JavaScript And XML*), sin necesidad de librerías externas, se pueden construir directivas para extender el HTML de las aplicaciones web. Entre los frameworks, librerías y otras tecnologías, AngularJS es la segunda herramienta más utilizada por los desarrolladores (Stack Overflow, 2017).

1.5.5. Tecnología de desarrollo del back-end

Node.js v6.10

[5] Data binding: enlace de datos por su traducción en español. Es un mecanismo mediante el cual se pueden enlazar los elementos de la interfaz de usuario con los objetos que contienen la información a mostrar.

La creación de la capa de servicios Transferencia de Estado Representacional (REST, del inglés *Representational State Transfer*) que funcionará en el *back-end*[6], se desarrollará con la herramienta Node.js en su versión 6.10. Node.js es un intérprete JavaScript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneas en una sola máquina física. Aunque es un programa de servidor, el producto base de Node.js no es como Apache o Tomcat (IBM, 2011).

Un servicio web que proporcione una Interfaz de Programación de Aplicaciones (API, del inglés *Aplication Programming Interface*) REST[7] toma algunos parámetros, los interpreta, arma una respuesta y descarga esa respuesta de vuelta al usuario. Lo cual es una situación ideal para esta herramienta, porque puede construirse para que maneje decenas de miles de conexiones. Tampoco requiere una gran cantidad de lógica y básicamente solo busca valores de una base de datos y los reúne como una respuesta. Como la solicitud entrante y la respuesta, son una pequeña cantidad de texto, el volumen de tráfico no es alto (IBM, 2011).

Cambia la forma en que se realiza una conexión con el servidor. En lugar de generar un nuevo hilo para cada conexión (y de asignarle la memoria acompañante), cada conexión dispara una ejecución de evento dentro del proceso del motor de Node.js. Además, nunca se quedará en punto muerto, porque no se permiten bloqueos y porque no se bloquea directamente para llamadas de Entrada/Salida. Un servidor que ejecute Node.js puede soportar decenas de miles de conexiones concurrentes (IBM, 2011). Entre los frameworks, librerías y otras tecnologías, Node.js es la herramienta más utilizada por los desarrolladores (Stack Overflow, 2017).

1.5.6. Lenguajes de programación

Los lenguajes de programación son un conjunto de reglas, herramientas y condiciones que permiten crear programas o aplicaciones dentro de un ordenador. Permiten además ordenar distintas acciones a la computadora en un lenguaje entendible por esta. Cada lenguaje posee su parte sintáctica y semántica, que

[6] Back-end: extremo trasero, por su traducción al español.

[7] REST: utiliza el Protocolo de Transferencia de Hipertexto (HTTP, del inglés *Hypertext Transfer Protocol*), se define un Identificador de Recursos Uniforme (URI, del inglés *Uniform Resource Identifier*) en el cual mediante los verbos disponibles en HTTP (GET, POST, PUT, DELETE) se consulta, se crea, se modifica y se elimina la información.

no son más que el conjunto de reglas acerca de cómo se deben escribir las instrucciones y de qué forma (Bonata, 2006). A continuación se describen los tres lenguajes utilizados.

JavaScript

Lenguaje de scripts desarrollado por Netscape para incrementar las funcionalidades del lenguaje HTML. El navegador del lado cliente se encarga de interpretar las sentencias JavaScript contenidas en una página HTML y ejecutarlas adecuadamente. Es un lenguaje orientado a eventos, se pueden desarrollar scripts que ejecuten acciones en respuesta a estos eventos. Es similar a Java, aunque no es orientado a objetos y no dispone de herencia, sigue el paradigma de programación basada en prototipos, ya que las nuevas clases se generan clonando las clases base y extendiendo su funcionalidad (Pérez, 2007). Es usado en el sitio tanto en las interfaces que se desarrollarán con AngularJS como en la capa de servicios REST que será creada con Node.js, que usa principalmente este lenguaje.

HTML 5

Contiene un conjunto más amplio de tecnologías que permite a los sitios web y a las aplicaciones ser más diversas y de gran alcance. Permite describir con mayor precisión cuál es su contenido, comunicarse con el servidor de formas nuevas e innovadoras. Permite a las páginas web almacenar datos localmente en el lado del cliente y operar sin conexión de manera más eficiente. Proporciona una mayor optimización de la velocidad y un mejor uso del hardware. Permite embeber dentro de su código scripts escritos en otros lenguajes como Preprocesador de Hipertexto (PHP, del inglés *Hypertext Preprocessor*) y JavaScript (Mozilla Developer, 2016). Este lenguaje se emplea en el desarrollo del sitio pues está compuesto por una serie de etiquetas que el navegador interpreta y da forma en la pantalla.

CSS 3

Hojas de Estilo en Cascada (CSS, del inglés *Cascading Style Sheets*) es un lenguaje creado para controlar y definir el aspecto o presentación de los documentos electrónicos definidos con HTML y XML. CSS es la mejor forma de separar los contenidos y su presentación, es imprescindible para crear páginas web complejas. Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes. Es utilizado para definir el aspecto de todos los contenidos, es decir, el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos y la tabulación con la que se muestran los elementos de una lista

(Librosweb, 2016). Este lenguaje se utiliza en el sitio pues es muy importante para organizar la información mostrada como la presentación de colores, tipos y tamaños de letra, etc.

1.5.7. Sistema gestor de base de datos

Sistema Gestor de Base de Datos (SGBD, del inglés *DataBase Management System*), es un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta (Ríos, 2009). Un SGBD permite definir los datos a distintos niveles de abstracción y manipularlos, garantizando la seguridad e integridad de los mismos.

MySQL v5.7

Es un SGBD relacional, multiusuario, de código abierto, se ofrece bajo la Licencia Pública General (GPL, del inglés *General Public License*), para cualquier uso compatible con esta licencia. Proporciona un servidor de base de datos SQL muy rápido (Gilfillian, 2010). La principal función de este sistema es la velocidad y la robustez, para las columnas soporta gran cantidad de tipos de datos. Funciona sobre múltiples plataformas y sistemas operativos. Presenta un excelente nivel de seguridad en los datos y es fácil de configurar e instalar. Por su implementación multihilo, tiene un sistema flexible de gestión de usuarios y contraseñas. Es la base de datos más usada por los programadores (Stack Overflow, 2017).

Se selecciona MySQL en su versión 5.7, debido a que posee gestión de usuarios y contraseñas para mantener un alto nivel de seguridad en los datos, soporta gran cantidad de tipos de datos para las columnas usadas en la base de datos. Es multiplataforma pues es aplicable a los sistemas operativos Windows y Linux. Es un gestor de bases de datos seguro debido a que las transacciones que se realizan requieren de autenticación, de esta forma se garantiza que solo las personas autorizadas realicen esta acción (MySql, 2017).

1.5.8. Sistema de control de versiones

Un sistema de control de versiones es una herramienta que registra todos los cambios hechos en uno o más proyectos, guardando así versiones del producto en todas sus fases del desarrollo. Las versiones son como fotografías que registran su estado en ese momento del tiempo y se van guardando a medida que se hacen modificaciones al código fuente (Hipertextual, 2014).

Git v2.13

Este sistema de control de versiones pertenece al movimiento de software libre utilizando GPL, fue desarrollado inicialmente por Linus Torvalds, aunque en la actualidad la comunidad de desarrollo es mucho mayor, es compatible con los sistemas operativos basados en Unix, con Posix, con Windows y con Mac Os X. Es un sistema de control de versiones distribuido lo cual lo hace más flexible para los desarrollos en las comunidades. Proporciona a cada desarrollador una copia local y los cambios son enviados de un usuario a otro como si este fuera el repositorio. Estos cambios son importados como ramas de desarrollo adicionales y pueden ser mezcladas con la línea de desarrollo local. Los repositorios pueden ser fácilmente accedidos mediante el protocolo Git. Con el uso de Capa de Sockets Seguros (SSL, del inglés *Secure Sockets Layer*) para la autenticación y la seguridad o simplemente HTTP se puede publicar el repositorio local en cualquier lugar sin ninguna configuración inicial del servidor web.

Git soporta una rápida y conveniente división en ramas de desarrollo, con las funcionalidades para la mezcla de las mismas; además de incluir poderosas herramientas para visualizar y navegar por un historial de desarrollo no lineal. Este sistema es muy rápido y escalable incluso cuando se trabaja con grandes proyectos y largos historiales. Es generalmente más rápido que la mayoría de los otros sistemas de control de versiones. Además, utiliza un eficiente formato de almacenamiento de paquetes para revisiones de gran tamaño lo que lo sitúa a la cabeza de cualquier otro sistema de control de versiones de código abierto (Git, 2017). Se escoge Git en su versión 2.13 para el control de versiones porque no es necesario depender de una conexión de red permanente, otorga seguridad en caso de fallo o pérdida de los repositorios principales, tanto el repositorio como las copias de trabajo individuales incluyen el historial completo y proporciona rápida transmisión de los cambios (Git, 2017).

1.6. Conclusiones del capítulo

De las tiendas virtuales estudiadas se identificaron funcionalidades a tener en cuenta en el proceso de desarrollo, como son: autenticar, descargar y publicar. Para la creación de la tienda virtual, se decidió emplear la metodología de desarrollo de software AUP-UCI en su cuarto escenario además de las herramientas: Node.js para la realización de la capa de servicios, AngularJS para la creación de las interfaces del sitio, Visual Studio como entorno de desarrollo integrado, MySQL como sistema gestor de base de datos, Visual Paradigm para la modelación y Git como sistema de control de versiones.

Capítulo 2: Análisis y diseño de la tienda virtual

En el presente capítulo se definirán los requisitos funcionales y no funcionales, así como las Historias de Usuario correspondientes. Se muestran además los diagramas de clases del diseño para representar la estructura del sistema. Se describen la arquitectura y los patrones de diseño a utilizar en la implementación. Además, se describe de forma general el proceso del funcionamiento del sistema.

2.1. Propuesta de solución

El desarrollo del presente trabajo propone como solución un sitio web. Dicho sitio funcionará como tienda virtual de aplicaciones de sistema operativo Android. En este sitio se podrá listar la información de las aplicaciones y realizar búsquedas, además de filtrar y ordenar sobre este listado. Se garantizará la publicación de ficheros (.apk), previamente revisados, así como la descarga de los mismos. Para garantizar un mejor funcionamiento del sitio se creará una capa de servicios web, una API REST, que contendrá todos los servicios a consumir por el usuario. El sitio contará con un sistema de pago por puntos que consiste en otorgarle al usuario una cantidad de puntos predefinidos. A su vez las aplicaciones costarán una cierta cantidad de puntos definidos por el administrador. Cuando el usuario descarga aplicaciones se le resta de su puntuación total la cantidad correspondiente que representa la aplicación. Cada 15 días los puntos se reiniciarán para que los usuarios puedan descargar nuevamente aplicaciones. Además, el usuario puede subir aplicaciones desarrolladas por él mismo que funcionen correctamente, lo cual será verificado por el administrador.

2.2. Especificación de los requisitos del sistema

Los requisitos determinan lo que hará el sistema y definen las restricciones de su operación e implementación. Se pueden clasificar en: funcionales y no funcionales (Jacobson, y otros, 2004). Partiendo de la descripción de clases representadas en el Modelo de Dominio y las necesidades del cliente, fueron determinados los requisitos funcionales y no funcionales del sistema.

2.2.1. Especificación de requisitos funcionales

Los requisitos funcionales definen las acciones que debe realizar el sistema, son capacidades o condiciones que debe cumplir, cómo debe comportarse en situaciones específicas. En algunos casos también pueden plantear explícitamente qué no debe hacer el sistema (Sommerville, 2011). A continuación, en la Tabla 2 se muestra el listado de requisitos funcionales con los que debe cumplir la tienda virtual:

Tabla 2: Especificación de requisitos funcionales.

Número	Nombre	Descripción	Prioridad	Complejidad
RF1	Autenticar usuario	El sitio debe permitir al usuario autenticarse en la tienda.	Alta	Alta
RF2	Subir aplicación	El sitio debe permitir al usuario subir aplicaciones en la tienda.	Alta	Media
RF3	Analizar aplicación subida por el usuario	El sitio debe permitir al administrador analizar una aplicación subida para saber si se publica o elimina.	Alta	Media
RF4	Descargar aplicación	El sitio debe permitir al usuario descargar una aplicación de la tienda.	Alta	Media
RF5	Editar aplicación	El sitio debe permitir al administrador editar los datos referentes a una aplicación.	Alta	Alta
RF6	Calificar aplicación	El sitio permite al usuario a partir de un calificador de cinco estrellas, votar por la aplicación.	Alta	Alta
RF7	Listar categorías de la tienda	El sitio debe mostrar las categorías existentes en la tienda.	Alta	Baja
RF8	Listar aplicaciones de la tienda	El sitio debe mostrar al usuario el listado de las aplicaciones disponibles en la tienda.	Alta	Baja

RF9	Eliminar aplicación	El sitio debe permitir al administrador eliminar una aplicación de la tienda.	Media	Media
RF10	Mostrar información de una aplicación de la tienda	El sitio debe mostrar los datos referentes a una aplicación almacenada en la tienda.	Media	Alta
R11	Buscar aplicación	El sitio permite buscar aplicaciones a partir de un criterio de búsqueda introducido por el usuario.	Media	Media
RF12	Adicionar categoría	El sitio debe permitir al administrador adicionar una nueva categoría.	Media	Media
RF13	Eliminar categoría	El sitio debe permitir al administrador eliminar cualquier categoría.	Media	Media
RF14	Editar categoría	El sitio debe permitir al administrador editar la información de una categoría.	Media	Media
RF15	Filtrar listado de aplicaciones por categoría	El sitio debe permitir al usuario filtrar el listado de aplicaciones a partir de categorías.	Baja	Media
RF16	Notificar sobre nuevas publicaciones	El sitio debe notificar al usuario sobre la existencia de nuevas aplicaciones en la tienda.	Baja	Media

RF17	Listar aplicaciones más votadas	El sitio debe mostrar al usuario el listado de las aplicaciones más votadas en la tienda.	Baja	Baja
RF18	Listar aplicaciones más populares	El sitio debe mostrar al usuario el listado de las aplicaciones más populares en la tienda.	Baja	Baja
RF19	Listar los usuarios de la tienda	El sitio debe permitir al administrador listar los usuarios que se hayan conectado a la tienda.	Baja	Baja
RF20	Editar usuario	El sitio debe permitir al administrador cambiar la cantidad de puntos de un usuario y otorgarle permisos de administrador.	Baja	Baja
RF21	Buscar usuario	El sitio debe permitir al administrador buscar un usuario.	Baja	Baja

2.2.2. Especificación de requisitos no funcionales

Usabilidad

El sistema podrá ser usado por todo aquel personal que posea un nivel básico en computación y en la navegación por sitios web.

Software

PC Cliente:

Los navegadores web a utilizar para acceder a la tienda virtual son:

Navegador web Mozilla Firefox a partir de la versión 51.0

Navegador web Chrome a partir de la versión 49.0

Navegador web Opera a partir de la versión 43.0

Navegador web Safari a partir de la versión 10.0

Navegador web Microsoft Edge versión 14.0 o superior

Seguridad

Identificar al usuario antes que este realice cualquier acción en el sitio, para ello se usa el mecanismo de Json Web Token. Asegurar que las funciones del sistema sean visibles según el nivel de acceso que tenga el usuario autenticado.

2.3. Historias de Usuario

Las HU constituyen una forma de administración de requisitos sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las mismas son escritas utilizando el lenguaje común del usuario. Son empleadas en las metodologías de desarrollo ágiles para la especificación de requisitos (Cohn, 2008). Se tomó la decisión de adoptar el escenario que emplea las Historias de Usuario para encapsular los requisitos funcionales.

Las HU solamente proporcionarán los detalles sobre la complejidad y cuánto tiempo conllevará implementarlas. El nivel de detalle de las HU debe ser el mínimo posible, permitiendo hacerse una ligera idea de cuánto costará implementar el sistema. Los programadores estiman el esfuerzo asociado y las dependencias entre ellas. Para planificar el trabajo desde el punto de vista técnico, las HU son divididas en tareas para las cuales también se realiza una estimación.

Tabla 3: Historia de Usuario "Autenticar usuario".

Número: 1	**Nombre de la Historia de Usuario:** Autenticar usuario
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 15 horas

Descripción: Los usuarios que pertenecen al dominio UCI se pueden autenticar en el sitio. Existen dos roles, uno administrador y otro cliente. Para autenticarse deben llenar los siguientes campos:

- **Nombre de usuario:** (Obligatorio. Campo de texto. Permite los caracteres: (a-z, A-Z, 0-9) y los restantes caracteres.)
- **Contraseña:** (Obligatorio. Campo de texto. Permite los caracteres: (a-z, A-Z, 0-9) y los restantes caracteres.)

Observaciones:

1. Si el usuario introduce la información de forma correcta, automáticamente entra a la tienda de aplicaciones.
2. Si el usuario introduce la información de forma incorrecta, el sistema emite un mensaje notificando el error.
3. Si el usuario introduce la información dejando campos obligatorios vacíos, el sistema emite un mensaje indicándole que los campos obligatorios deben llenarse.
4. Si el usuario no pertenece al dominio UCI no puede acceder a la tienda de aplicaciones y el sistema emite una notificación.

Prototipo de interfaz:

Figura 2: Interfaz "Autenticar usuario".

Tabla 4: Historia de Usuario "Subir aplicación".

Número: 2	**Nombre de la Historia de Usuario:** Subir aplicación	
Programador: Michael Domenech Nodarse		**Iteración Asignada:** Primera iteración
Prioridad: Alta		**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A		**Tiempo Real:** 20 horas
Descripción: Cuando un usuario con el rol de cliente se autentica puede subir aplicaciones a la tienda. Para ello es obligatorio dar clic en el símbolo adicionar que se encuentra en la parte superior de la tienda. Se muestra una ventana en la cual, al dar clic en el botón de subir, el usuario puede buscar la ubicación donde se encuentra la aplicación.		
Observaciones: 1. Si el usuario sube la aplicación de forma correcta, el sistema emite un mensaje notificando		

que se subió correctamente.

2. Si el usuario intenta subir un archivo que no tenga la extensión apk, el sistema emite un mensaje notificando el error.

Prototipo de interfaz:

Figura 3: Interfaz "Subir aplicación".

Tabla 5: Historia de Usuario "Analizar aplicación subida por el usuario".

Número: 3	**Nombre de la Historia de Usuario:** Analizar aplicación subida por el usuario
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 20 horas

Descripción: Cuando un usuario con el rol de administrador se autentica puede editar los datos referentes a una aplicación. Este usuario analiza las aplicaciones que no están publicadas para determinar si se publican o se eliminan. Para esto se debe modificar una de las opciones siguientes:

- Publicar: (Opcional. CheckBox. Para decidir si se publica una aplicación.)
- Eliminar: (Opcional. Se muestra en todas las aplicaciones.)

Observaciones:

1. Si el usuario verifica que la aplicación funciona correctamente, publica la misma.
2. Si el usuario verifica que la aplicación no funciona correctamente, elimina la misma.

Prototipo de interfaz:

Figura 4: Interfaz "Analizar aplicación subida por el usuario".

Tabla 6: Historia de Usuario "Descargar aplicación".

Número: 4	**Nombre de la Historia de Usuario:** Descargar aplicación	
Programador: Michael Domenech Nodarse		**Iteración Asignada:** Primera iteración
Prioridad: Alta		**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A		**Tiempo Real:** 20 horas
Descripción: Cuando un usuario con el rol de cliente se autentica y entra a la tienda, puede descargar las aplicaciones que se muestran en la misma. Para ello es obligatorio dar clic en el botón de descarga que se muestra en cada una de las aplicaciones.		
Observaciones: 1. Si el usuario posee una cantidad de puntos mayor o igual que los puntos asignados a la aplicación seleccionada, posteriormente se abre una ventana de navegador de archivos para que elija la ubicación donde va a guardarla. 2. Si el usuario posee una cantidad de puntos menor que los puntos asignados a la aplicación seleccionada, se muestra un mensaje notificando que no se puede realizar la descarga.		
Prototipo de interfaz:		

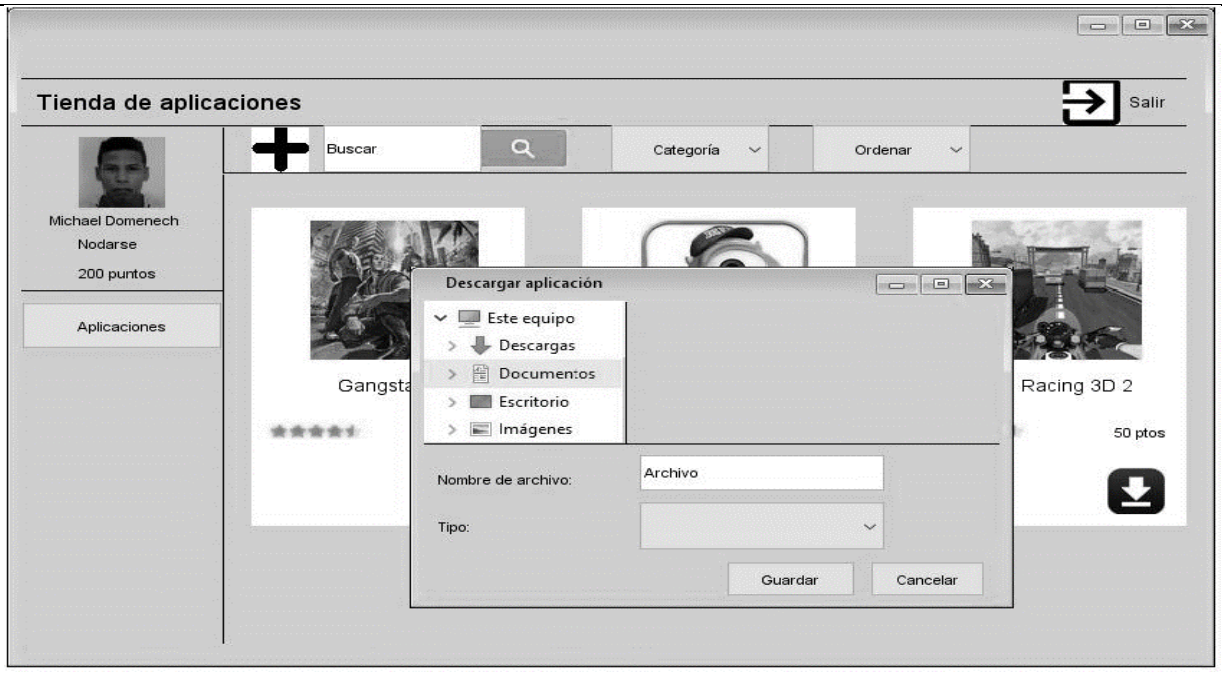


Figura 5: Interfaz "Descargar aplicación".

Tabla 7: Historia de Usuario "Editar aplicación".

Número: 5	**Nombre de la Historia de Usuario:** Editar aplicación
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 20 horas
Descripción: Cuando un usuario con el rol de administrador se autentica puede editar los datos referentes a una aplicación. Este usuario puede cambiar la cantidad de puntos asignados a una aplicación, así como su categoría, además de decidir si se publica en la tienda. Para ello debe modificar los campos siguientes:	

- Categoría: (Obligatorio. ComboBox. Se muestra la lista de categorías.)
- Cantidad de puntos: (Obligatorio. Campo de texto. Solo se admiten números entre 1 y 999.)
- Publicar: (Opcional. CheckBox. Para decidir si se publica una aplicación.)

Observaciones:

1. Si el usuario introduce la información de forma correcta, el sistema edita la información de la aplicación.
2. Si el usuario introduce la información dejando los campos categoría y cantidad de puntos, vacíos, el sistema emite un mensaje indicándole que los campos obligatorios deben llenarse.
3. Si el usuario introduce una cantidad de puntos incorrecta, se muestra una notificación de error.

Prototipo de interfaz:

Figura 6: Interfaz "Editar aplicación".

2.4. Diseño

El papel del diseño en el ciclo de vida de un software es facilitar la comprensión de su funcionamiento y proveer una representación o modelo del mismo con el propósito de definirlo con los suficientes detalles como para permitir su realización física. El modelo de diseño provee una representación arquitectónica del software que sirva de punto de partida para las tareas de implementación, dando al traste con los requisitos del sistema.

2.4.1. Descripción de la arquitectura

La arquitectura del software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos, el contexto en el que se implantarán, los principios que orientan su diseño y evolución (Buzz, 2015). En el presente trabajo se emplean diferentes estilos arquitectónicos. Para la realización de este trabajo se hace uso del estilo arquitectónico REST ya que la capa de servicios se dividirá en recursos, debido a las funcionalidades y al estado que poseen. A su vez se evidencia el estilo arquitectónico Presentación Desacoplada pues se separan las interfaces del flujo de datos de los servicios creados en una API REST. Las interfaces creadas en AngularJS se estructuran mediante el patrón arquitectónico MVVM por la forma en que se maneja la actualización de los datos mediante el *two way data binding*[8]. Estas interfaces consumen los servicios API REST creados en Node.js los cuales se estructuran mediante el patrón arquitectónico MVC en el cual los *handlers*, quienes poseen toda la información a devolver, se encargan de controlar todo el flujo de datos.

Estilo arquitectónico REST

El estilo REST es una abstracción de los elementos arquitectónicos dentro de un sistema hipermedia distribuido. REST ignora los detalles de la implementación de componentes y la sintaxis del protocolo para centrarse en los roles de los componentes, las restricciones sobre su interacción con otros componentes y su interpretación de elementos de datos significativos. Abarca las restricciones fundamentales sobre componentes, conectores y datos que definen la base de la arquitectura Web y, por tanto, la esencia de su comportamiento como una aplicación basada en red (Fielding, 2000).

Estilo arquitectónico Presentación Desacoplada

[8] Two way data binding: enlace de datos de dos vías, por su traducción al español.

El estilo arquitectónico Presentación Desacoplada indica cómo debe realizarse el manejo de las acciones del usuario, la manipulación de la interfaz y los datos de la aplicación. Este estilo separa los componentes de la interfaz del flujo de datos y de la manipulación (De la Torre y otros, 2010). A continuación, se muestra la figura 7 donde se muestran los componentes separados y la relaciones que existen entre los mismos.

Figura 7: Componentes del estilo arquitectónico Presentación Desacoplada.

Patrón arquitectónico MVC

La arquitectura Modelo-Vista-Controlador (MVC), tiene estilo de Presentación Desacoplada ya que la aplicación estará basada en la interacción de un cliente con un servidor a través de la web y esta arquitectura es ampliamente usada para este propósito, además de que es empleada cada vez más en la implementación de aplicaciones Android. Esta arquitectura separa presentación e interacción de los datos del sistema. El sistema se estructura en tres componentes lógicos que interactúan entre sí. El componente Modelo maneja los datos asociados al sistema y las operaciones asociadas a esos datos. El componente Vista define y gestiona cómo se presentan esos datos al usuario. El componente Controlador dirige la interacción del usuario y pasa estas interacciones a Vista y Modelo (Sommerville, 2011).

Permite que los datos cambien de manera independiente de su representación y viceversa. Soporta en diferentes formas la presentación de los mismos datos, y los cambios en una representación se muestran en todos ellos. A continuación, se muestra la figura 8 donde se muestran los paquetes en el directorio del *back-end* y la relaciones que existen entre los mismos.

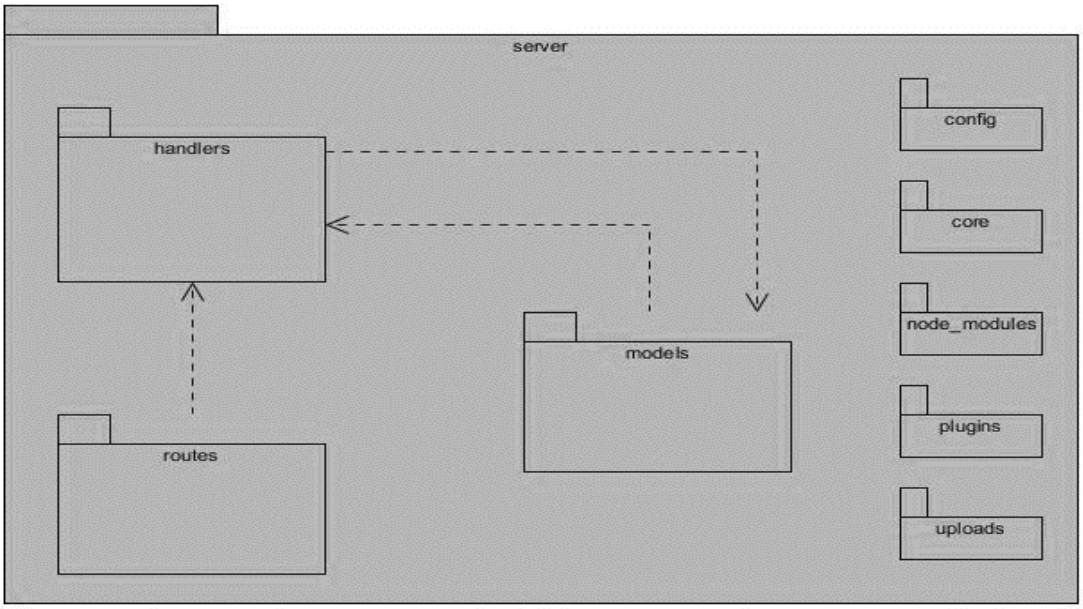


Figura 8: Diagrama de paquetes que representa la estructura de directorios del back-end.

Handlers: Es la carpeta que se encarga de manejar las clases que contienen la información a devolver cuando el usuario realice una petición. En general controla el flujo de la capa de servicios.

Routes: Es el que se encarga de mostrar las acciones a realizar y enrutar las peticiones hacia los Handlers.

Models: Es en esta carpeta donde se encuentran los módulos que se encargan de convertir la información almacenada en la base de datos a clases para poder mostrar la información.

Patrón arquitectónico MVVM

La arquitectura Model View ViewModel (MVVM) tiene como función separar el modelo (model) y la vista (view) introduciendo una capa abstracta entre ellos, un “modelo de la vista” o “ViewModel”. La vista y el modelo de la vista son instanciados normalmente por la aplicación contenedora. La vista guarda una referencia al ViewModel. Este expone comandos y entidades enlazables a los que la vista puede enlazarse. Las interacciones del usuario con la vista disparan comandos contra el ViewModel de forma análoga, las actualizaciones en el ViewModel serán propagadas a la vista de forma automática mediante enlace de datos. MVVM tiene como objetivo que los datos trasladados a la vista se puedan presentar y gestionar de la manera más sencilla. Es el ViewModel quien expone los datos desde el modelo y, en este sentido, el ViewModel es más un modelo que una vista. Además, gestiona la lógica de visualización de la vista por lo que esta, desde

este punto de vista, es más una vista que un modelo (De la Torre y otros, 2010). A continuación, se muestra la figura 9 donde se muestran los paquetes en el directorio del *front-end* y la relaciones que existen entre los mismos.

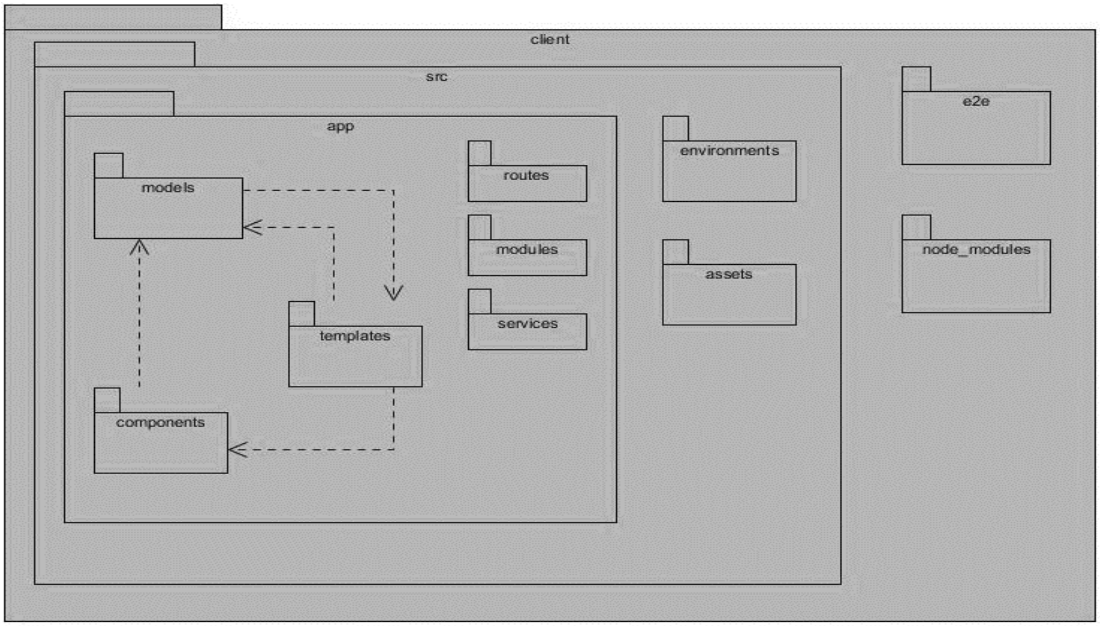


Figura 9: Diagrama de paquetes que representa la estructura de directorios del front-end.

App: Todos los componentes de AngularJS, plantillas, estilos, imágenes y cualquier otra cosa se guardan en esta carpeta.

Environments: Esta carpeta contiene un archivo para cada uno de los entornos de destino, cada uno de los cuales exporta variables de configuración simples. Los archivos se reemplazan al momento de crear la aplicación. Incluso puede usar algunos servicios de simulación.

Assets: En esta carpeta es donde pueden ponerse las imágenes y cualquier otra cosa que se copiará al por mayor cuando se construye el proyecto.

2.5. Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces de una aplicación. Es una representación concreta de lo que se debe implementar y son parte de la estética del sistema representándolo mediante clases y sus relaciones (Booch y otros, 2009).

A continuación, se muestran algunos diagramas de clases de la tienda como: autenticar usuario, descargar aplicación, subir aplicación, editar aplicación y calificar aplicación, los cuales muestran la estructura del sistema.

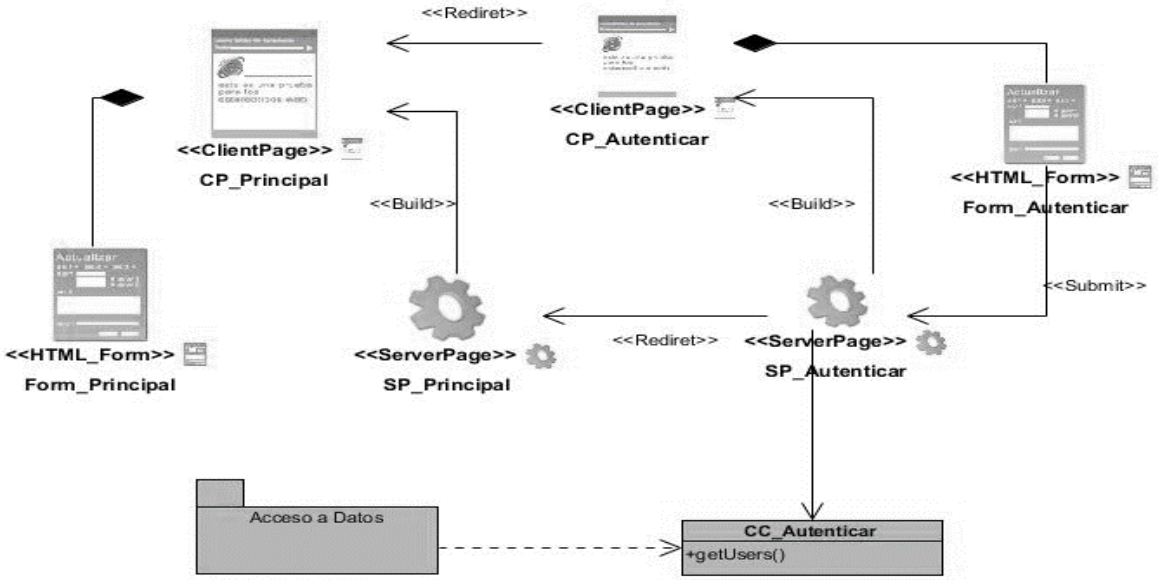


Figura 10: Diagrama de clases del diseño. Autenticar usuario.

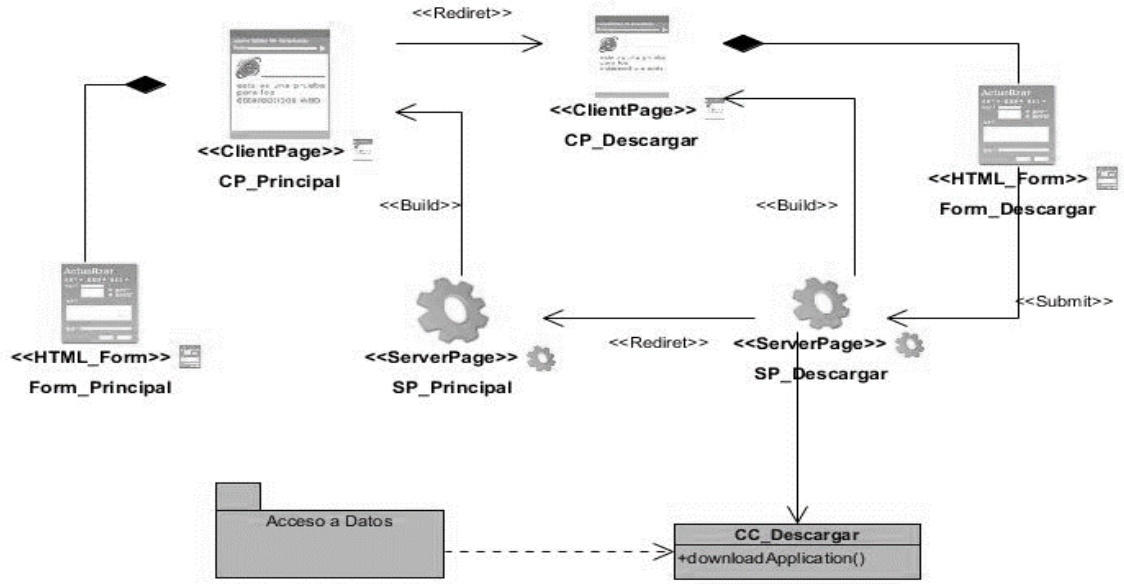


Figura 11: Diagrama de clases del diseño. Descargar aplicación.

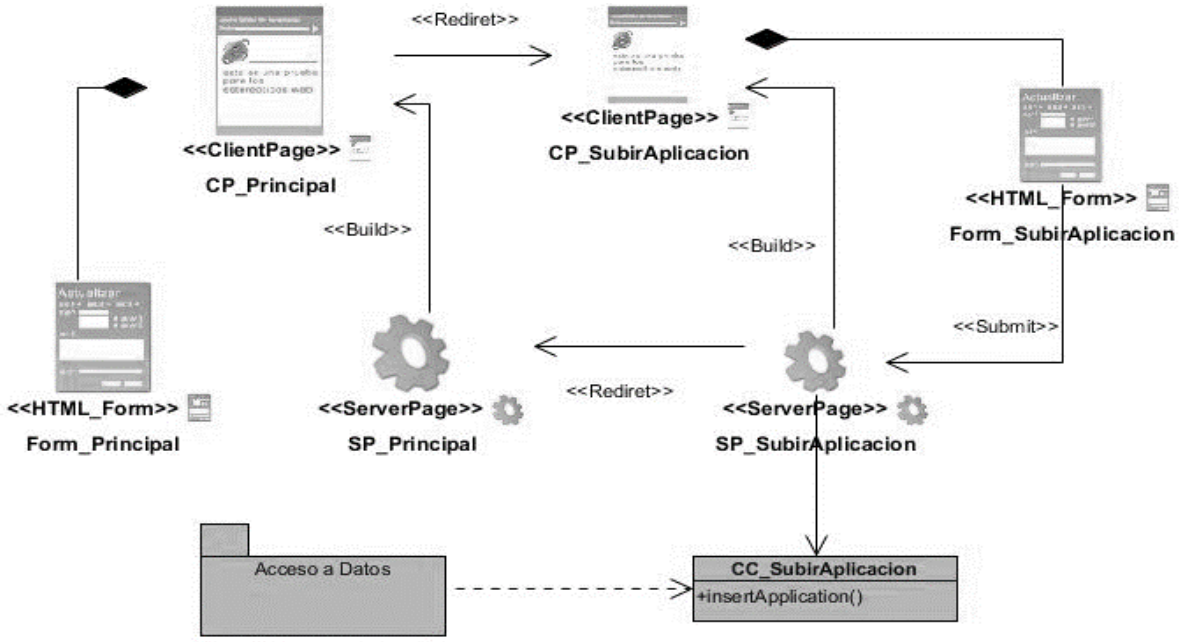


Figura 12: Diagrama de clases del diseño. Subir aplicación.

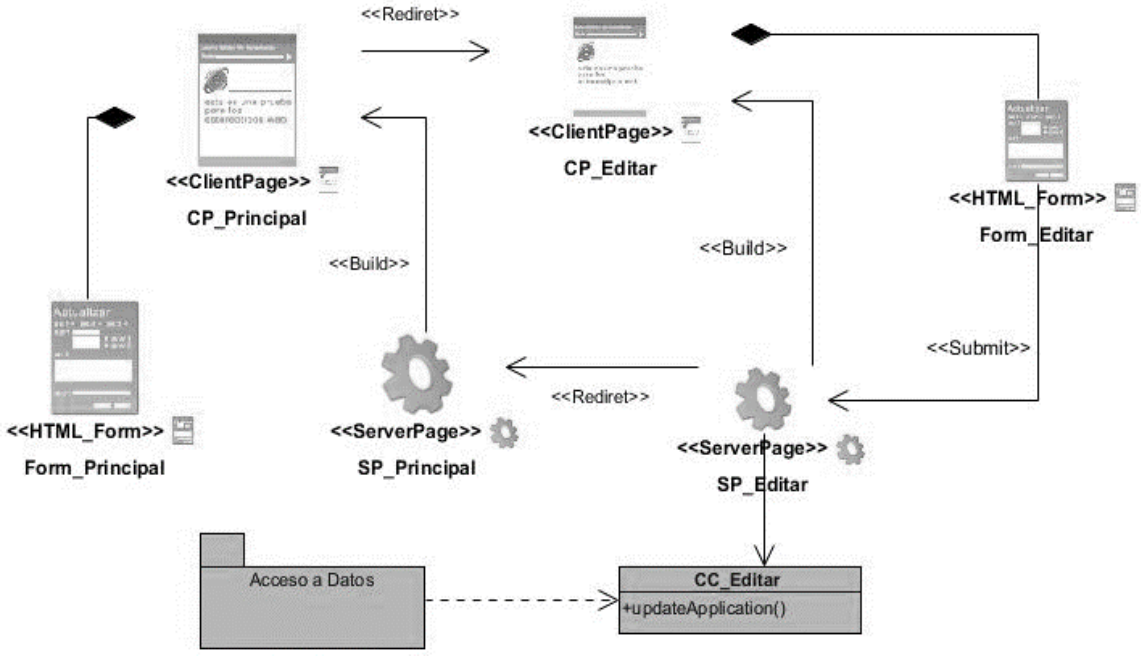


Figura 13: Diagrama de clases del diseño. Editar aplicación.

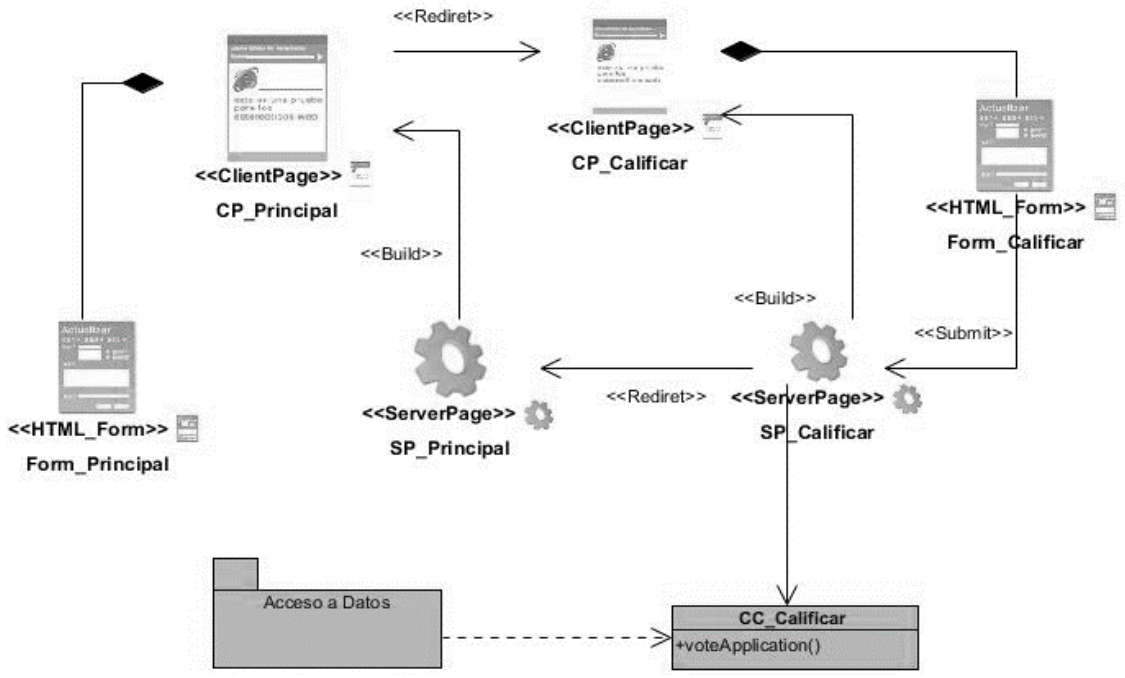


Figura 14: Diagrama de clases del diseño. Calificar aplicación.

2.6. Patrones de diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Representan una descripción de las clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. El uso de patrones posibilita estandarizar el modo en que se realiza el diseño y proporciona reusabilidad, extensibilidad y mantenimiento del código (Garlan, 1994). Los patrones de diseño se caracterizan por:

1. Representar soluciones técnicas a problemas concretos.
2. Propiciar la reutilización.
3. Representar problemas frecuentes.

2.6.1. Patrones GRASP

Los Patrones Generales de Software para Asignación de Responsabilidades (GRASP, del inglés *Object-Oriented Design General Responsibility Assignment Software Patterns*) describen los principios

fundamentales de la asignación de responsabilidades a objetos (Larman, 2004). El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar un software de manera eficaz.

Creador: se aplica para la asignación de responsabilidades a las clases relacionadas con la creación de objetos, de forma tal que una instancia de un objeto solo pueda ser creada por el objeto que contiene la información necesaria para ello (Larman, 2004). Ejemplo del uso de este patrón se evidencia en el método registerHandlers en el cual se crea una instancia del registro de los handlers.

```
const registerHandlers = (server, handlersdir) => {
    _.map(getFilesDir(handlersdir), (handlerdir) => {
        const classFunction = require(handlerdir);
        let handlerClass = new classFunction(server);
        const handlerName = camelCase(path.basename(handlerdir).replace(/.js/, ''));
        server.handler(handlerName, handlerClass.dispatch.bind(handlerClass));
    });
};
```

Controlador: un controlador es un objeto de interfaz no destinada al usuario que se encarga de manejar un evento del sistema (Larman, 2004). Ejemplo del uso de este patrón se evidencia en la clase ApplicationHandler que se encarga de atender las peticiones al recurso de las aplicaciones.

```
class ApplicationHandler {}
```

Experto: se usa más que cualquier otro al asignar responsabilidades, es un principio básico que suele utilizarse en el diseño orientado a objetos (Larman, 2004). Consiste en la asignación de una responsabilidad a la clase que cuenta con la información necesaria para llevarla a cabo. El uso de este patrón da pie a un bajo acoplamiento y una alta cohesión, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento. El cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. Ejemplo del uso de este patrón se evidencia en la clase CategoryHandler que se encarga de manejar toda la información de las categorías.

```
class CategoryHandler {}
```

Bajo Acoplamiento: el acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras (Larman, 2004). El bajo acoplamiento soporta el diseño de clases más independientes y

reutilizables, lo cual reduce el impacto de los cambios y acrecienta la oportunidad de una mayor productividad. Ejemplo del uso de este patrón se evidencia en las clases CategoryHandler, ApplicationHandler, SecurityHandler, UserHandler donde se minimizan las relaciones de estas con el resto de las clases.

Alta Cohesión: en la perspectiva del diseño orientado a objetos, la cohesión (o más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, clases con responsabilidades moderadas en un área funcional que colaboran con las otras para llevar a cabo las tareas (Larman, 2004). Ejemplo del uso de este patrón se evidencia en la clase ApplicationHandler pues en el método saveApplicationFiles se utiliza la función module.exports.getApkInfo del módulo apkInfo.

2.6.2. Patrones GOF

Los patrones (GOF, del inglés *Gang of Four*) son alternativas de solución a problemas conocidos, pero son mucho más específicas las situaciones en las que se aplican. Se clasifican en creacionales, estructurales y de comportamiento (Gamma et al., 1994). A continuación, se muestran los utilizados en el sistema.

Decorador: añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad. Es un patrón de tipo estructural (Gamma et al., 1994). Ejemplo del uso de este patrón se evidencia en la función appendAuthMessages que recorre todos los routers que requieran autenticación del usuario y/o del administrador y agrega los mensajes correspondientes.

```
const appendAuthMessages = (routes) => {
    routes.forEach((route) => {
        if(route.config.hasOwnProperty('auth') == false|| route.config.auth != false){
            route.config.notes.push('__Este recurso requiere autenticacion.__');
            let isRestrictedRouter =
config.get("/application/adminroutes").indexOf(route.config.id) != -1 ;
            if(isRestrictedRouter)
                route.config.notes.push('__Este recurso requiere usuario
administrador.__');
        }
```

```
    });
    return routes;
};
```

Singleton: está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella. Es un patrón de tipo creacional (Gamma et al., 1994). Ejemplo del uso de este patrón se evidencia en la clase soapService donde en la siguiente función se verifica si hay una única instancia de la clase, si no existe se crea.

```
SoapService.getInstance = function(){
    if(global.soapService_instance === undefined)
        global.soapService_instance = new SoapService();
    return global.soapService_instance;
};
```

2.7. Conclusiones parciales

Se evidenció la propuesta de solución para la problemática planteada, detallada a través de los requisitos funcionales y no funcionales, las Historias de Usuario y los diagramas de clases del diseño. Se identificaron 21 requisitos funcionales, 3 requisitos no funcionales y 21 Historias de Usuario en la cual se realizó la descripción de los requisitos funcionales. La utilización de los estilos arquitectónicos y los patrones de diseño GRASP y GOF contribuyó al diseño de la aplicación, proporcionando una estructura para la misma y posibilitando el empleo de buenas prácticas de programación y la reutilización de código.

Capítulo 3: Implementación y pruebas

En este capítulo se evidencian las pruebas realizadas, así como los resultados arrojados por las mismas. Se abordan las pautas de codificación utilizadas y se muestra el diagrama de despliegue.

3.1. Modelo de implementación

El modelo de implementación está comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se pueden encontrar datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. Un modelo de implementación muestra las dependencias entre las partes del código del sistema. (Pressman, 2009).

3.2. Estándares de codificación

Los estándares de codificación permiten un mejor entendimiento del código por parte de todos los miembros del equipo de desarrollo y en consecuencia hacen que el código sea fácil de mantener. El uso de estándares de codificación trae consigo beneficios. Facilita el mantenimiento de una aplicación, permite que cualquier programador entienda y pueda mantener la aplicación y mejora la legibilidad del código, al mismo tiempo que permite su rápida comprensión (Microsoft, 2016). Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Al comenzar un proyecto de software, se debe establecer un estándar de codificación para asegurar que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpora código fuente previo, o bien cuando se realiza el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente (Bruegge, 2008).

3.2.1. Estándares de codificación utilizados

- Se empleará el idioma inglés para la codificación de la solución.
- Los nombres de funciones y variables deben empezar siempre con una letra minúscula utilizando la forma "camelCase".
- Los ficheros creados por Node.js tienen que concluir con la extensión .js, y no deben incluir signos de puntuación excepto – (guión medio) o _ (guión bajo).

- Indentación: Consiste en insertar espacios en blanco o tabuladores en determinadas líneas de código para facilitar su comprensión.
- Las líneas tienen siempre no más de 80 caracteres.
- Estructuras de control:
    - Debe existir un espacio entre el comando que define la estructura (if, while, for, etc.) y el paréntesis de apertura.
    - La llave de apertura ({) se situará en la misma línea que la definición de la estructura, separada por un espacio.
    - Usar siempre las llaves {} aún en los casos en que no sea obligatorio su uso.

3.3. Diagrama de despliegue

Un diagrama de despliegue muestra las relaciones físicas que se establecen entre componentes de software y hardware dentro de un sistema de cómputo determinado. Se modela a través de un conjunto de nodos y las relaciones existentes entre ellos. Cada nodo en este tipo de diagrama representa un tipo de unidad computacional, en la mayoría de los casos de tipo hardware (Fowler y Scott, 2002).

En este caso el sistema debe disponer de computadoras clientes que presenten un navegador web para conectarse al sistema hospedado en la capa de servicios utilizando el protocolo HTTP. Un servidor que contiene los servicios y hospeda el código fuente del sitio web, además, que les brinda a los usuarios las interfaces de la misma para realizar los procesos definidos por cada uno de los roles del sistema. Esta estación se comunica con el servidor de base de datos, realizando la comunicación mediante el servicio TCP/IP. Un servidor para bases de datos que es el encargado del almacenamiento de los datos del sistema. Un servidor de directorio Protocolo Ligero/Simplificado de Acceso a Directorios (LDAP, del inglés *Lightweight Directory Access Protocol*) para el control de los usuarios que acceden al sistema que se comunica con la capa de servicios mediante el protocolo LDAP. A continuación, se muestra la imagen del diagrama de despliegue donde se evidencia lo antes expuesto.

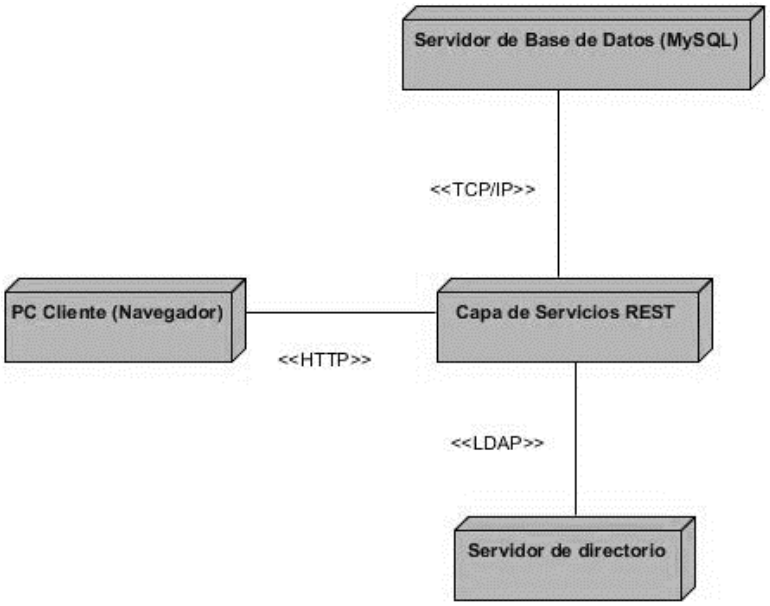


Figura 15: Diagrama de despliegue de la tienda virtual.

3.4. Interfaces de la tienda virtual

Finalmente se obtuvo una tienda que permite publicar y descargar aplicaciones de sistema operativo Android desarrolladas en la universidad. La tienda contribuye además a fomentar el interés de desarrollo de las mismas por parte de la comunidad universitaria ya que otorga la posibilidad a los desarrolladores de mostrar las aplicaciones que desarrollan. Además, se garantiza la disponibilidad de un sitio propio para la UCI. Teniendo en cuenta que internet es una herramienta potente sin importar en que ámbito se maneje, esta tienda virtual es otro recurso la cual facilita a la comunidad no tener que buscar en otras tiendas internacionales.

3.4.1. Interfaz “Listar aplicaciones de la tienda”

Cuando el usuario se autentica en la tienda se le muestra un listado de todas las aplicaciones de la misma. En las aplicaciones se muestra: el nombre, cantidad de puntos, el promedio de votación reflejado en estrellas, el botón de descarga y el icono que la representa, como se muestra a continuación en la Figura 16.

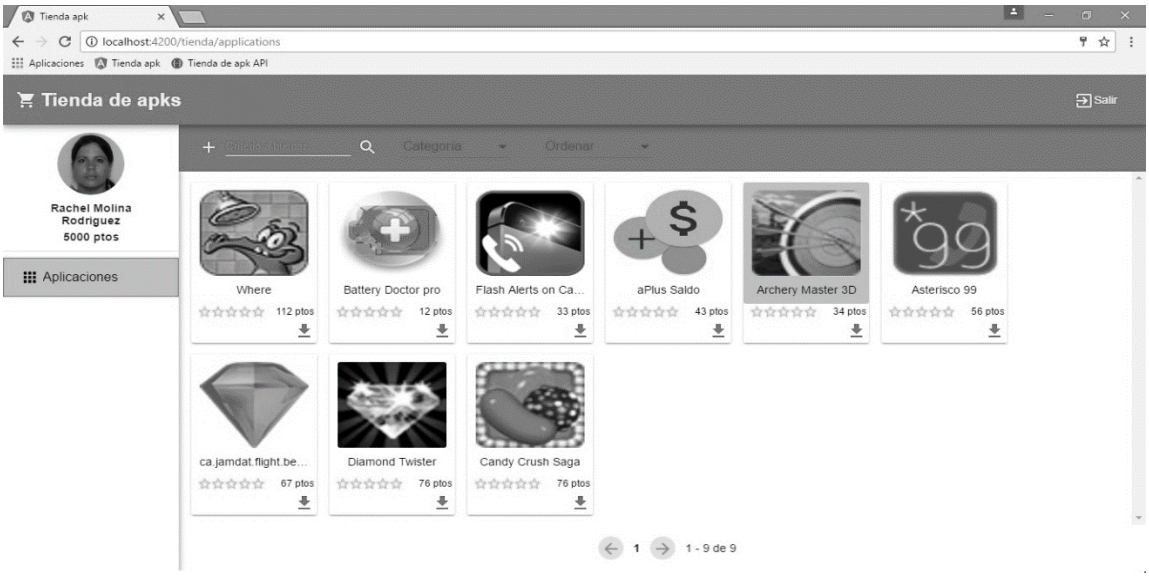


Figura 16: Interfaz "Listar aplicaciones de la tienda".

3.4.2. Interfaz "Subir aplicación"

Cuando un usuario se autentica en la tienda puede subir aplicaciones a la misma, en el símbolo de más que aparece al lado de la barra de búsqueda es donde puede ejecutar esta acción. Se muestra una ventana donde se da la opción de buscar la aplicación que se desea subir, de la cual se muestra el nombre en el campo Archivo, como se muestra a continuación en la Figura 17.

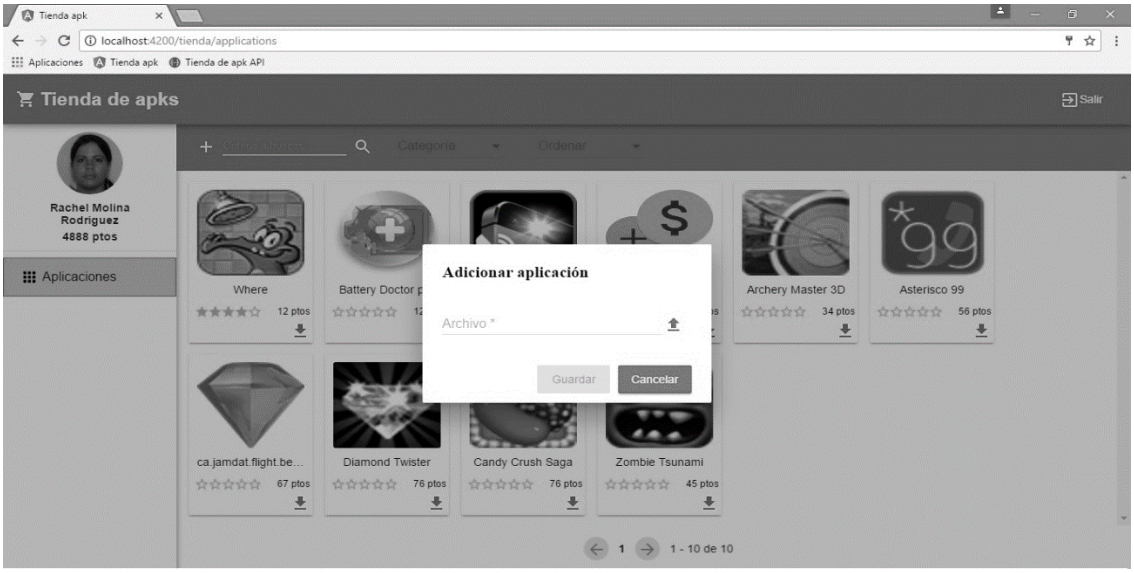


Figura 17: Interfaz “Subir aplicación”.

3.4.3. Interfaz “Analizar aplicación”

Cuando el administrador se autentica en la tienda revisa que la nueva aplicación subida por otro usuario funcione correctamente para otorgarle puntos y una categoría, para posteriormente publicarla en la tienda. Se muestra una ventana en la cual se realiza la acción descrita además de la opción de publicación representada por un checkbox, como se muestra a continuación en la Figura 18.

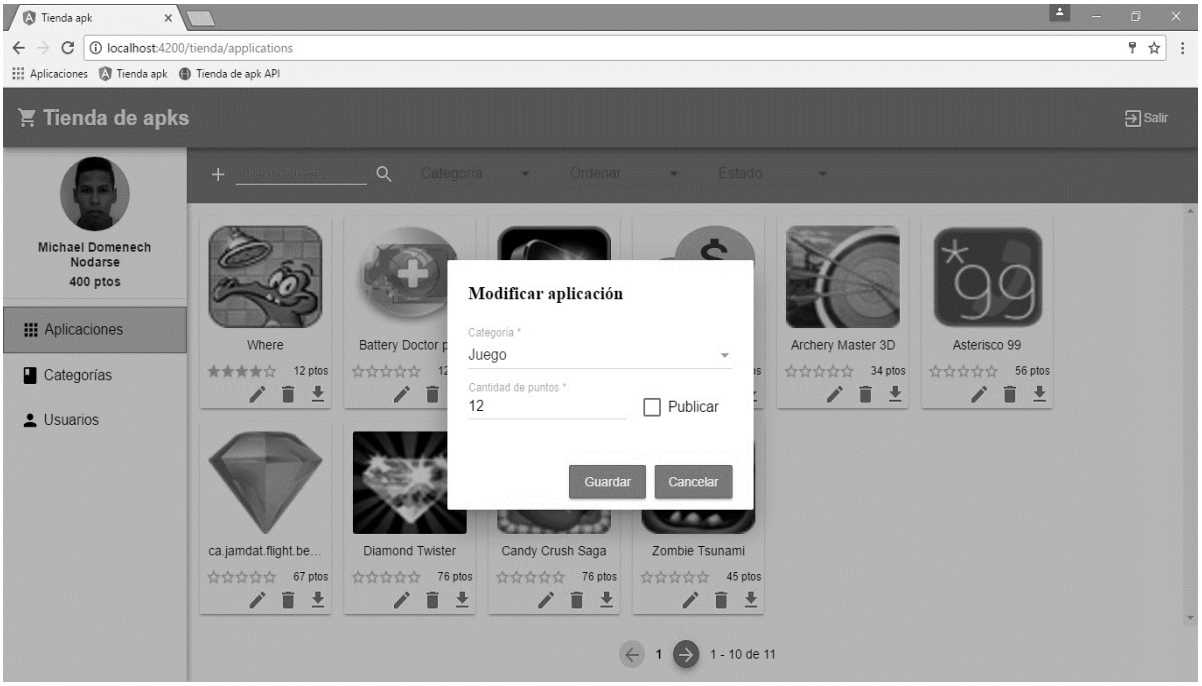


Figura 18: Interfaz “Analizar aplicación”.

3.4.4. Interfaz “Descargar aplicación”

Cuando un usuario se autentica en la tienda se le muestran las aplicaciones y en cada una de estas se muestra el botón de descarga, el cual al ser presionado, se muestra una ventana que permite al usuario buscar el lugar del ordenador donde desea guardar la aplicación, como se muestra a continuación en la Figura 19.

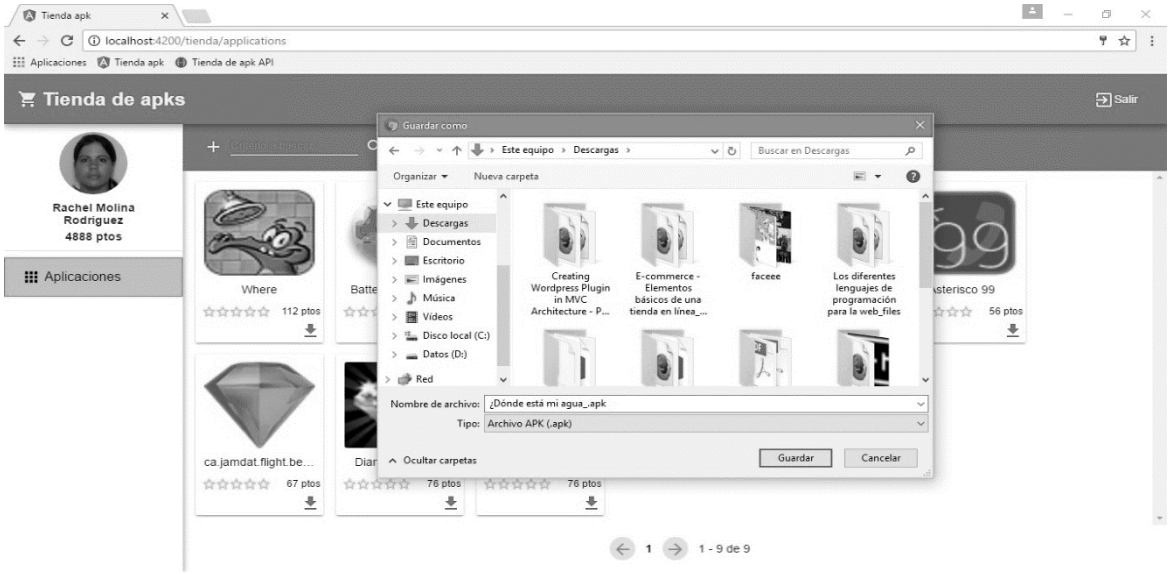


Figura 19: Interfaz “Descargar aplicación”.

3.5. Pruebas de software

La prueba del software es un elemento crítico para la garantía de calidad del software y representa una revisión de las especificaciones, del diseño y de la codificación. Una vez generado el código fuente es necesario probar el software para descubrir y corregir la mayor cantidad de errores posibles antes de ser entregado. Su objetivo es diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores (Pressman, 2009).

La metodología de software AUP en su variante para la UCI establece 3 disciplinas para la ejecución de pruebas: internas, liberación y aceptación. Como parte de las pruebas se decide la realización de las pruebas internas y de aceptación, se descartan las pruebas de liberación ya que estas son diseñadas y ejecutadas por una entidad certificadora de la calidad externa. En las pruebas internas se propone verificar el resultado de la aplicación. En el caso de las pruebas de aceptación, se llevan a cabo para verificar que el software está listo y que puede ser utilizado por usuarios finales, para ejecutar las tareas y funciones para las que fue construido.

3.5.1. Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas

(Rodríguez, 2015). Debido a esto se realizan dentro de las pruebas internas las pruebas de unidad para probar el correcto funcionamiento del código mediante el método de caja blanca y las pruebas funcionales para probar el correcto funcionamiento de las interfaces usando el método de caja negra.

3.5.1.1. Pruebas de unidad

Las pruebas de unidad o unitarias centran el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo. Se prueba el módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada (Pressman, 2009).

El método aplicado para esta prueba fue la **prueba de caja blanca**, donde las pruebas se centran en los detalles procedimentales del software, por lo que su diseño está fuertemente ligado al código fuente. Se escogen distintos valores de entrada para examinar cada uno de los posibles flujos de ejecución del programa y cerciorarse de que se devuelven valores de salida adecuados (Pressman, 2009).

Con el fin de automatizar este tipo de pruebas sobre la solución se decidió emplear el framework de pruebas QUnit, el cual se integra a la herramienta Node.js que se utilizó para desarrollar el *back-end* del sitio. Se implementaron los casos de prueba a través de la clase ApplicationHandler, los cuales comprueban el correcto funcionamiento de las principales funcionalidades definidas en esta clase, responsables de la lógica principal del sitio. A partir de la interpretación de los resultados se puede afirmar que se ejercitan al menos una vez todos los caminos independientes de cada método, se ejercitan las decisiones lógicas en sus vertientes verdadera y falsa, además se ejercitan las estructuras de datos para asegurar su validez; garantizando el correcto funcionamiento de los métodos de la clase ApplicationHandler.

3.5.1.2. Pruebas funcionales

Aseguran el apropiado trabajo de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Las metas de estas pruebas son verificar la apropiada aceptación de datos y verificar el procesamiento, recuperación e implementación adecuada de las reglas del negocio (Pressman, 2009).

Para llevarlas a cabo, el método empleado fue el de la **prueba de caja negra**, el cual se centra en los requisitos funcionales del software. Es decir, la prueba de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales

de un programa (Pressman, 2009). Para efectuar las pruebas de caja negra se usó la técnica de partición equivalente.

Partición equivalente

Es una técnica de prueba de caja negra que divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El diseño de casos de prueba para partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada (Pressman, 2009).

Los casos de prueba se diseñaron según las funcionalidades descritas en las Historias de Usuario. La intención que se persigue es lograr una comprensión específica de las condiciones que la solución debe cumplir. Cada planilla de caso de prueba recoge la especificación de una Historia de Usuario, dividida en secciones y escenarios, detallando las funcionalidades descritas en ella y describiendo cada variable.

A continuación, se muestran los casos de prueba correspondientes a las Historias de Usuario “Subir aplicación” y “Descargar aplicación” ya que representan el proceso fundamental del negocio.

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 8: Caso de Prueba Funcional para la Historia de Usuario “Subir aplicación”.

Caso de Prueba Funcional	
CP1_HU2	**HU_2:** Subir aplicación.
Responsable: Michael Domenech Nodarse	
Descripción: El caso de prueba se inicia luego de que el usuario entra al sitio. En la barra que se muestra encima de las aplicaciones aparece el símbolo de adición, en el cual el usuario escoge la acción y le permite subir una aplicación a la tienda, pero no se mostrará automáticamente, solo al administrador pues	

el mismo debe revisar que esta funcione correctamente para posteriormente ser publicada. El caso de prueba termina cuando la aplicación se sube satisfactoriamente.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC_1.1 Subir una aplicación a la tienda.	El usuario selecciona la acción de subir una aplicación a la tienda y selecciona la aplicación deseada.	El sitio muestra al usuario una ventana en la cual tiene que seleccionar la opción de buscar la aplicación que desea subir a la tienda. Cuando la encuentra y se muestra el nombre en la ventana se habilita la opción de guardar. Al ejecutar este botón se mostrará un mensaje indicando que se está subiendo la aplicación.	El usuario selecciona la acción de subir una aplicación a la tienda.
EC_1.2 Error cuando se sube un archivo con la extensión diferente a (.apk)	El usuario selecciona la acción de subir una aplicación a la tienda. Cuando intenta subir un archivo cualquiera que no es una aplicación con la extensión (.apk) o que la haya renombrado poniéndole ese formato, se muestra un error.	El sitio muestra al usuario un error indicando que el archivo no es una aplicación.	El usuario selecciona la acción de subir una aplicación a la tienda.

Tabla 9: Caso de Prueba Funcional para la Historia de Usuario "Descargar aplicación".

Caso de Prueba Funcional			
CP2_HU4		**HU_4:** Descargar aplicación.	
Responsable: Michael Domenech Nodarse			
Descripción: El caso de prueba se inicia luego de que el usuario entra al sitio. Se le muestra el listado de aplicaciones de las cuales cada una tiene la opción de descargar y ejecuta la acción en cualquier aplicación. El caso de prueba termina cuando la descarga ha finalizado.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC_2.1 Descargar una aplicación de la tienda.	El usuario selecciona la acción de descargar en una aplicación de la tienda.	El sitio muestra al usuario una ventana en la cual tiene que seleccionar el lugar donde quiere guardar la aplicación que va a descargar. Luego se muestra un mensaje indicando que se completó la descarga satisfactoriamente.	El usuario selecciona la acción de descargar en una aplicación de la tienda.
EC_2.2 Error en la descarga de una aplicación de la tienda.	El usuario selecciona la acción de descargar en una aplicación de la tienda pero los puntos del usuario no son suficientes.	El sitio muestra al usuario un mensaje de error.	El usuario selecciona la acción de descargar en una aplicación de la tienda.

Con la realización de estas pruebas se validaron los requisitos funcionales, demostrando que los resultados que se obtuvieron fueron satisfactorios, descartando funciones incorrectas o ausentes, errores de interfaz, errores en estructuras de datos o en acceso a datos externos.

3.5.2. Pruebas de aceptación

Esta es la etapa final en el proceso de pruebas, antes de que el sistema sea aceptado para uso operacional. El sistema se pone a prueba con datos suministrados por el cliente del sistema, en vez de datos de prueba simulados. Las pruebas de aceptación revelan los errores y las omisiones en la definición de requerimientos del sistema, ya que los datos reales ejercitan el sistema en diferentes formas a partir de los datos de prueba. Asimismo, las pruebas de aceptación revelan problemas de requerimientos, donde las instalaciones del sistema en realidad no cumplan las necesidades del usuario o cuando sea inaceptable el rendimiento del sistema (Sommerville, 2011). A continuación, se muestran los casos de prueba realizados a algunas de las Historias de Usuario.

Tabla 10: Caso de Prueba de Aceptación para la Historia de Usuario "Editar aplicación".

Caso de Prueba de Aceptación
Historia de usuario: Editar aplicación.
Persona que realiza la prueba: Yoandy Pérez Villazón
Descripción de la prueba: Probar que se pueden realizar cambios en los datos de las aplicaciones de la tienda.
Condiciones de ejecución: • Estar autenticado en la tienda con el rol de administrador. • Que exista al menos una aplicación en la tienda.
Entrada/Pasos de ejecución: 1. Acceder a la lista de aplicaciones. 2. Dar clic en una de las aplicaciones.

3. Modificar los campos: cantidad de puntos, categoría, publicación.
Resultado Esperado: Se efectúan los cambios esperados en la aplicación.
Evaluación de la prueba: Satisfactoria

Tabla 11: Caso de Prueba de Aceptación para la Historia de Usuario "Calificar aplicación".

Caso de Prueba de Aceptación
Historia de usuario: Calificar aplicación.
Persona que realiza la prueba: Yoandy Pérez Villazón
Descripción de la prueba: Probar que se puede votar por una aplicación de la tienda.
Condiciones de ejecución: • Estar autenticado en la tienda con el rol de cliente. • Que exista al menos una aplicación en la tienda.
Entrada/Pasos de ejecución: 4. Acceder a la lista de aplicaciones. 5. Dar clic en una de las aplicaciones. 6. Dar clic en la estrella por la que se quiere votar.
Resultado Esperado: Se efectúa la votación en la aplicación.
Evaluación de la prueba: Satisfactoria

Al finalizar estas pruebas el cliente fue capaz de verificar el cumplimiento del objetivo planteado. Los resultados obtenidos para cada uno de los casos de prueba efectuados fueron satisfactorios, por lo que el cliente firmó el acta de aceptación del producto, como se aprecia en el Anexo 2.

3.6. Conclusiones parciales

A través del diagrama de despliegue se obtuvo la especificación de las características que deben cumplir el sitio y la capa de servicios. Las pruebas de unidad y funcionales permitieron comprobar el correcto funcionamiento del código de la aplicación y validar la completitud de los requisitos. Con la realización de las pruebas de aceptación se demostró el cumplimiento del objetivo planteado en la presente investigación.

Conclusiones generales

Con la realización de este trabajo se dio cumplimiento a cada uno de los objetivos trazados, a partir de los resultados obtenidos se puede arribar a las siguientes conclusiones:

1. El estudio de los referentes teóricos relacionados con las tiendas virtuales de aplicaciones móviles permitió un mejor entendimiento de los conceptos asociados.
2. El análisis de tiendas similares permitió la comprensión del funcionamiento de las mismas, identificándose las funcionalidades que poseen, algunas de las cuales se ponen de manifiesto en la propuesta de solución realizada.
3. La identificación de los requisitos funcionales y no funcionales, descritos a través de Historias de Usuario, facilitaron la descripción, comprensión y posterior codificación por parte del equipo de desarrollo.
4. Con la realización de una capa de servicios web REST se garantizó un mejor funcionamiento del sitio.
5. Los resultados obtenidos a partir de la ejecución de las pruebas permitieron la comprobación de los requisitos identificados y la aceptación por parte del cliente.

Recomendaciones

Tomando como base la investigación realizada y el análisis de los resultados obtenidos se recomienda añadir una funcionalidad que permita a los usuarios personalizar su perfil y puedan ver las aplicaciones que subieron.

Referencias Bibliográficas

Ambler, S.W. The Agile Unified Process (AUP) Home Page. [En línea]. 2014. [Consultado: 20 de noviembre de 2016]. Disponible en: http://www.ambysoft.com/unifiedprocess/agileUP.html.

Angular. AngularJS: Developer Guide: Introduction. [En línea]. 2017. [Consultado: 7 de abril de 2017]. Disponible en: https://docs.angularjs.org/guide/introduction.

Aptoide. Acerca de | Aptoide - Android Apps Store. [En línea]. 2016. [Consultado: 6 de diciembre de 2016]. Disponible en: http://m.aptoide.com/about.

Atraura.com. Ventajas y desventajas de programar en AngularJS | Atraura. [En línea]. 2016. [Consultado: 7 de abril de 2017]. Disponible en: https://www.atraura.com/angularjs/.

Bonata, **Maximiliano**. Programación y Algoritmos. Mp Ediciones Corporation. Octubre, 2003. 296 páginas.

Booch, **Grady**, **Rumbaugh**, **James**, **Jacobson y Ivar**. El Manual de Referencia del Lenguaje de Modelado Unificado. ADDISON-WESLEY. 2009. 688 páginas.

Bruegge, Bernd, DUTOIT, Allen. Ingeniería de software orientado a objetos. Prentice Hall Mexico. 2008. 576 páginas.

Buzz, S. Arquitectura de Software. [En línea]. 2015. [Consultado: 15 de febrero de 2017]. Disponible en: http://sg.com.mx/content/view/922.

Cuello, **J.** y **Vittone**, **J.** Capítulo 1: Las aplicaciones – Diseñando apps para móviles. [En línea]. 2015. [Consultado: 9 de noviembre de 2016]. Disponible en: http://appdesignbook.com/es/contenidos/las-aplicaciones/.

De la Torre, Cesar. Guía de Arquitectura de N-Capas orientada al Dominio con .NET 4.0 - Guia_Arquitectura_N-Capas_DDD_NET_4_Borrador_Marzo_2010.pdf [En línea]. 2010. [Consultado: 13 de abril de 2017]. Disponible en: http://wpf.com.es/wp-content/uploads/2014/02/Guia_Arquitectura_N-Capas_DDD_NET_4_Borrador_Marzo_2010.pdf.

DesarrolloWeb.com. Qué es AngularJS. [En línea]. 2014. [Consultado: 7 de abril de 2017]. Disponible en: http://www.desarrolloweb.com/articulos/que-es-angularjs-descripcion-framework-javascript-conceptos.html.

Referencias bibliográficas

F-Droid. F-Droid. [En línea]. 2016. [Consulta: 6 de diciembre 2016]. Disponible en: https://f-droid.org/wiki/page/Main_Page.

Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. [En línea]. 2000. [Consultado: 9 de junio de 2017]. Disponible en: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

Fowler, M. y Scott, K. UML distilled: a brief guide to the standard object modeling language. 2. ed., 10. Print. Boston: Addison-Wesley. Object technology series. ISBN 978-0-201-65783-8. 2002.

Garlan, **David y Shaw**, **Mary**. An Introduction to Software Architecture. Financial Times Prentice Hal. 1994. 42 páginas.

Gartner. Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016. [En línea]. 2017. [Consultado: 3 de mayo de 2017]. Disponible en: http://www.gartner.com/newsroom/id/3609817.

Genbetadev. Lenguaje Modelado. [En línea]. 2013. [Consultado: 30 de noviembre de 2016]. Disponible en: https://www.genbetadev.com/herramientas/visual-studio-2013.

Git. About – Git. [En línea]. 2017. [Consultado: 11 de noviembre de 2016]. Disponible en https://git-scm.com/about.

Google. Aplicaciones de Android en Google Play. [En línea]. 2016. [Consultado: 7 de octubre de 2016]. Disponible en: https://play.google.com/store?hl=es.

Headways Media. Tienda virtual - Glosario Mercadotecnia. [En línea]. 2015. [Consultado: 9 de noviembre de 2016]. Disponible en: http://www.headways.com.mx/glosario-mercadotecnia/definicion/tienda-virtual/.

Hipertextual. Qué es un sistema de control de versiones. [En línea]. 2014. [Consultado: 11 de noviembre de 2016]. Disponible en: https://hipertextual.com/archivo/2014/04/sistema-control-versiones/.

IBM. ¿Simplemente qué es Nodejs?. [En línea]. 2011. [Consultado: 15 de marzo de 2017]. Disponible en: https://www.ibm.com/developerworks/ssa/opensource/library/os-nodejs/.

Itunews. Auge de los dispositivos móviles - Tendencias de las telecomunicaciones. [En línea]. 2015. [Consultado: 6 de febrero de 2017]. Disponible en: https://itunews.itu.int/es/1090-auge-de-los-dispositivos-moviles.note.aspx.

Referencias bibliográficas

Jacobson, Ivar y Booch, Grady, Rumbaugh, James. El Proceso Unificado. Addison Wesley. 2004. 464 páginas.

Kaisler, **S.H.** Software Paradigms. New Jersey: John Wiley & Sons. ISBN 0-471-48347-8. Wiley-Interscience. 2005. 440 páginas.

Kincaid, **Jason**. Amazon's Android App Store Launches: Test Drive Apps Directly From Your Browser | TechCrunch. [En línea]. 2011. [Consultado: 7 de octubre de 2016]. Disponible en: https://techcrunch.com/2011/03/22/amazon-android-app-store-3/.

Larman, **C.** Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition. [En línea]. 2004. [Consultado: 23 febrero 2017]. Disponible en: https://aanimesh.files.wordpress.com/2013/09/applying-uml-and-patterns-3rd.pdf.

Librosweb. Capítulo 1. Introducción (Introducción a CSS). [En línea]. 2016. [Consultado: 7 de diciembre de 2016]. Disponible en: https://librosweb.es/libro/css/capitulo_1.html.

Microsoft. MSDN. Técnicas de codificación. [En línea]. 2014**.** [Consultado: 23 de marzo de 2017]. Disponible en: http://msdn.microsoft.com/es-es/library/aa291593(v=vs.71).aspx.

Mozilla Developer. HTML5 - HTML | MDN. [En línea]. 2016. [Consultado: 10 de noviembre de 2016]. Disponible en: https://developer.mozilla.org/es/docs/HTML/HTML5.

MySql. MySQL: Why MySQL? [En línea]. 2017. [Consultado: 25 de febrero de 2017]. https://www.mysql.com/why-mysql/.

Object Management Group. OMG UML. [En línea]. 2015. [Consultado: 27 de octubre de 2016]. Disponible en: http://www.omg.org/gettingstarted/what_is_uml.htm.

Padilla, Manuel. E-commerce - Elementos básicos de una tienda en línea. [En línea]. 2013. [Consultado: 30 de noviembre de 2016]. Disponible en: http://es.slideshare.net/manuelpadillac/ecommerce-23586566.

Pérez Valdés, **Damián**. ¿Qué es Javascript? [En línea]. 2007. [Consultado: 8 de noviembre de 2016]. Disponible en: http://www.maestrosdelweb.com/que-es-javascript/.

Pressman, Roger. Ingeniería del Software. Un enfoque práctico. s.l.: McGraw-Hill/Interamericana. S.A. MCGRAW-HILL. 2009. 640 páginas.

Referencias bibliográficas

Quesada Cruz, Leonardo. Cliente para dispositivos móviles de la Tienda de Aplicaciones Android del Caribe. Tesis para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, junio 2016.

Rodríguez Sánchez, Tamara. Metodología de desarrollo para la actividad productiva de la UCI. [En línea]. 2015. [Consultado: 10 de noviembre de 2016]. Disponible en: http://excriba.prod.uci.cu/proxy/alfresco/api/node/content/workspace/SpacesStore/a622adab-eac5-4fb3-ba08-a266767fff5f/Metodologia%20UCI.pdf?a=true.

Smith, Josh. Patterns - WPF Apps With The Model-View-ViewModel Design Pattern. [En línea]. 2009. [Consultado: 7 de abril de 2017]. Disponible en: https://msdn.microsoft.com/en-us/magazine/dd419663.aspx.

Sommerville, I. Ingeniería de Software. 9na. México: Addison Wesley. Pearson Education, Inc. ISBN 978-607-32-0603-7. 2011. 792 páginas.

Stack Overflow. Stack Overflow Developer Survey 2017. [En línea]. 2017. [Consultado: 29 de marzo de 2017]. Disponible en: https://stackoverflow.com/insights/survey/2017/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2017&utm_content=social-share.

Tecnología Qode. ¿Qué es Google Play? | Blog de Tecnología Qode Apps. [En línea]. 2013. [Consultado: 7 de octubre de 2016]. Disponible en: http://qode.pro/blog/que-es-google-play/.

Tilves, Mónica. La demanda de apps móviles crecerá mucho más rápido que su desarrollo. [En línea]. 2009. [Consultado: 17 de marzo de 2017]. Disponible en: http://www.silicon.es/la-demanda-de-apps-moviles-crecera-mucho-mas-rapido-que-su-desarrollo-83082.

Unified Modeling Language. Welcome To UML Web Site! [En línea]. 2015. [Consultado: 8 de noviembre de 2016]. Disponible en: http://www.uml.org/.

Visual Paradigm. What is Visual Paradigm? [En línea]. 2016. [Consultado: 8 de noviembre de 2016]. Disponible en: https://www.visual-paradigm.com/features/.

Xatakamovil. La saturación en las tiendas de apps: ¿es este mercado un "negocio unicornio" en la actualidad? [En línea]. 2015. [Consultado: 7 de octubre de 2016]. Disponible en:

Referencias bibliográficas

http://www.xatakamovil.com/aplicaciones/la-saturacion-en-las-tiendas-de-apps-es-este-mercado-un-negocio-unicornio-en-la-actualidad.

Zapata, Carlos Mario. Lenguaje Modelado. [En línea]. 2009. [Consultado: 30 de noviembre de 2016]. Disponible en: http://www.scielo.org.co/pdf/ring/n29/n29a3.pdf.

Anexos

Anexo 1: Historias de Usuario

Tabla 12: Historia de Usuario "Calificar aplicación".

Número: 6	**Nombre de la Historia de Usuario:** Calificar aplicación.
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 15 horas
Descripción: Cuando un usuario con el rol de cliente se autentica y entra a la tienda puede dar clic sobre una aplicación. Se le muestra la información de la misma y las votaciones por estrellas, de las cuales el usuario puede seleccionar una de las cinco estrellas para emitir su voto para esa aplicación.	
Observaciones: 1. Si el usuario vota por una aplicación, automáticamente se actualiza el total de votos por dicha estrella y además el promedio de votos para esa aplicación basado en la fórmula (cantidad_total_estrellas/cantidad_personas_votaron). Para calcular el total de estrellas se suma la cantidad que posee cada una de las 5 calificaciones. 2. Si el usuario ya votó por esa aplicación e intenta volver a votar, el sistema emite un mensaje notificando que ya votó.	
Prototipo de interfaz:	

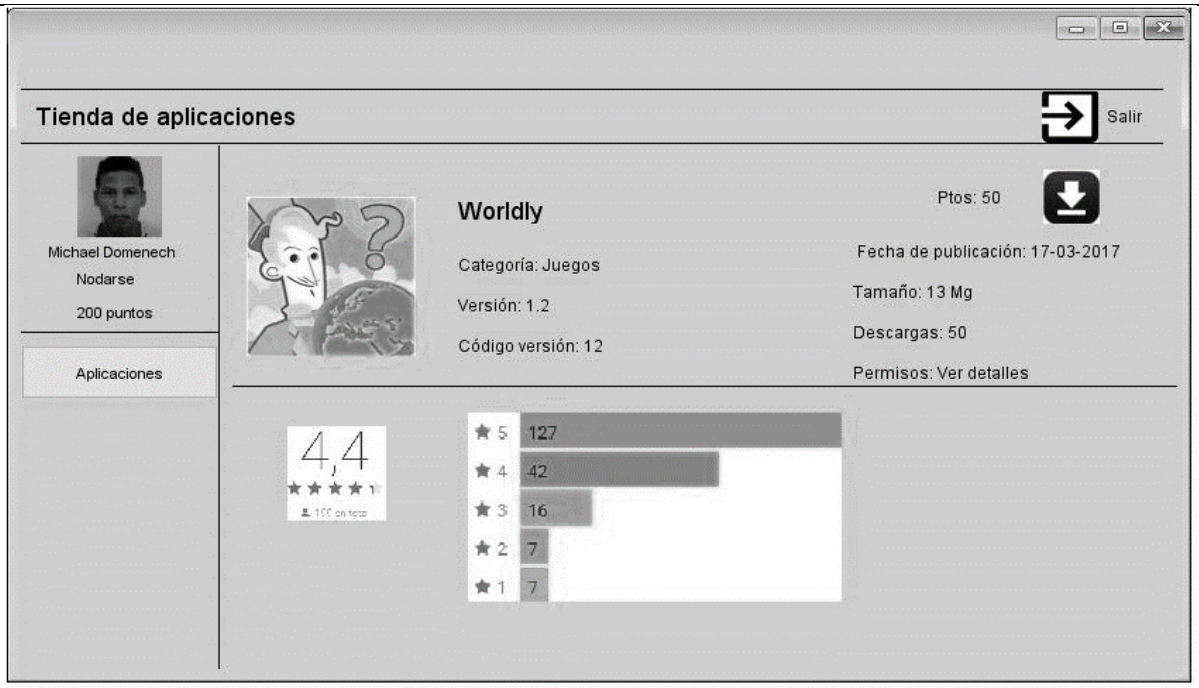


Figura 20: Interfaz "Calificar aplicación".

Tabla 13: Historia de Usuario "Listar categorías de la tienda".

Número: 7	**Nombre de la Historia de Usuario:** Listar categorías de la tienda.
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 20 horas
Descripción: Cuando un usuario con el rol de cliente se autentica y entra a la tienda, puede ver en la parte superior de la misma un combobox con el nombre Categoría. Si el usuario da clic en Categoría se despliegan todas las categorías de las aplicaciones de la tienda.	
Observaciones: No aplica.	

Prototipo de interfaz:

Figura 21: Interfaz "Listar categorías de la tienda".

Tabla 14: Historia de Usuario "Listar aplicaciones de la tienda".

Número: 8	**Nombre de la Historia de Usuario:** Listar aplicaciones de la tienda.
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Alta	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 20 horas
Descripción: Cuando un usuario con el rol de cliente se autentica y entra a la tienda, se listan todas las aplicaciones de la misma. El listado de aplicaciones muestra de estas: nombre, votación, cantidad de puntos asignados y la opción de descargar.	

Observaciones: No aplica.
Prototipo de interfaz:

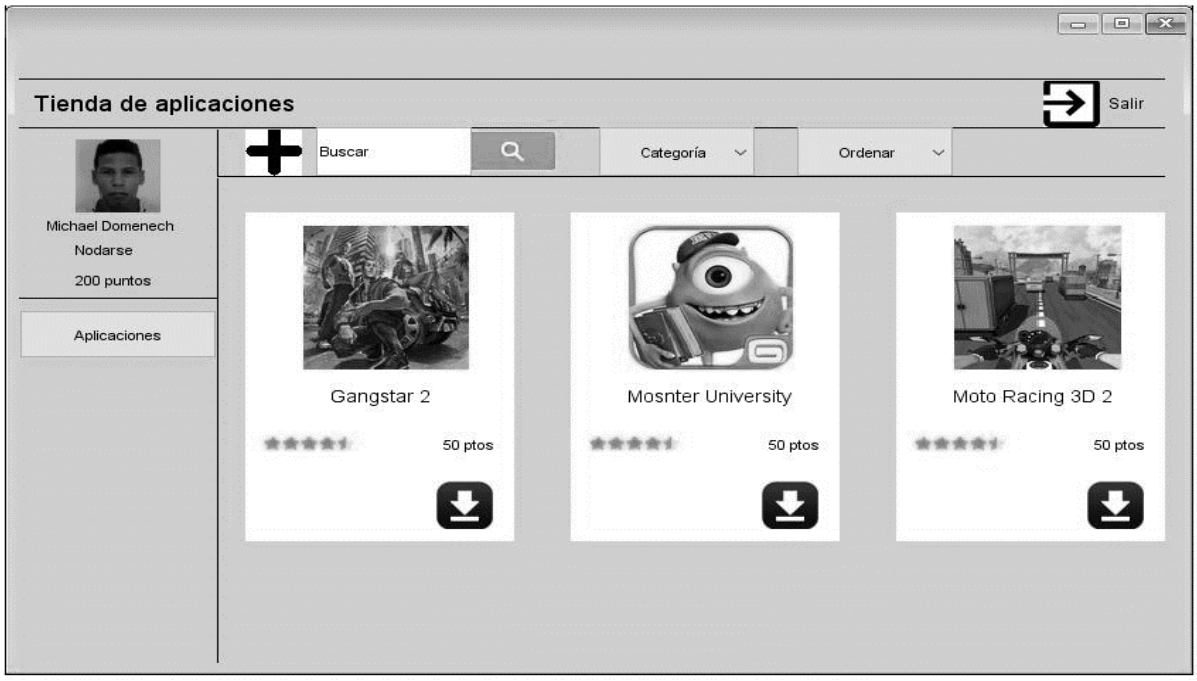


Figura 22: Interfaz "Listar aplicaciones de la tienda".

Tabla 15: Historia de Usuario "Eliminar aplicación".

Número: 9	**Nombre de la Historia de Usuario:** Eliminar aplicación.
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Media	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 15 horas
Descripción: Cuando un usuario con el rol de administrador se autentica puede eliminar una aplicación de la tienda. Para ello puede dar clic en el botón de eliminar: ➢ Eliminar: (Opcional. Se muestra en todas las aplicaciones.)	

Observaciones: 1. Si el usuario desea eliminar una aplicación, se muestra un mensaje de confirmación. 2. Si el usuario presiona Aceptar, la aplicación se elimina totalmente de la tienda.
Prototipo de interfaz:

Figura 23: Interfaz "Eliminar aplicación".

Tabla 16: Historia de Usuario "Mostrar información de una aplicación de la tienda".

Número: 10	**Nombre de la Historia de Usuario:** Mostrar información de una aplicación de la tienda.
Programador: Michael Domenech Nodarse	**Iteración Asignada:** Primera iteración
Prioridad: Media	**Tiempo Estimado:** 20 horas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 20 horas

Descripción: Cuando un usuario con el rol de cliente se autentica y entra a la tienda, de las aplicaciones listadas puede dar clic en cualquiera de ellas para ver la información que posee.
Observaciones: No aplica.
Prototipo de interfaz:

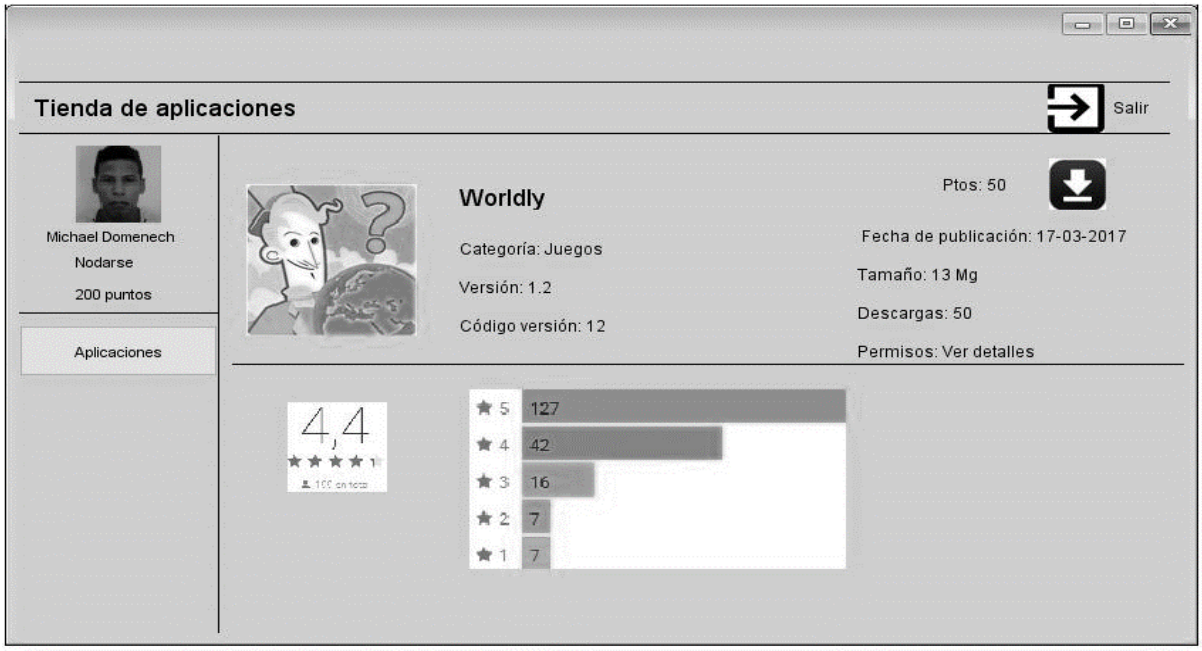
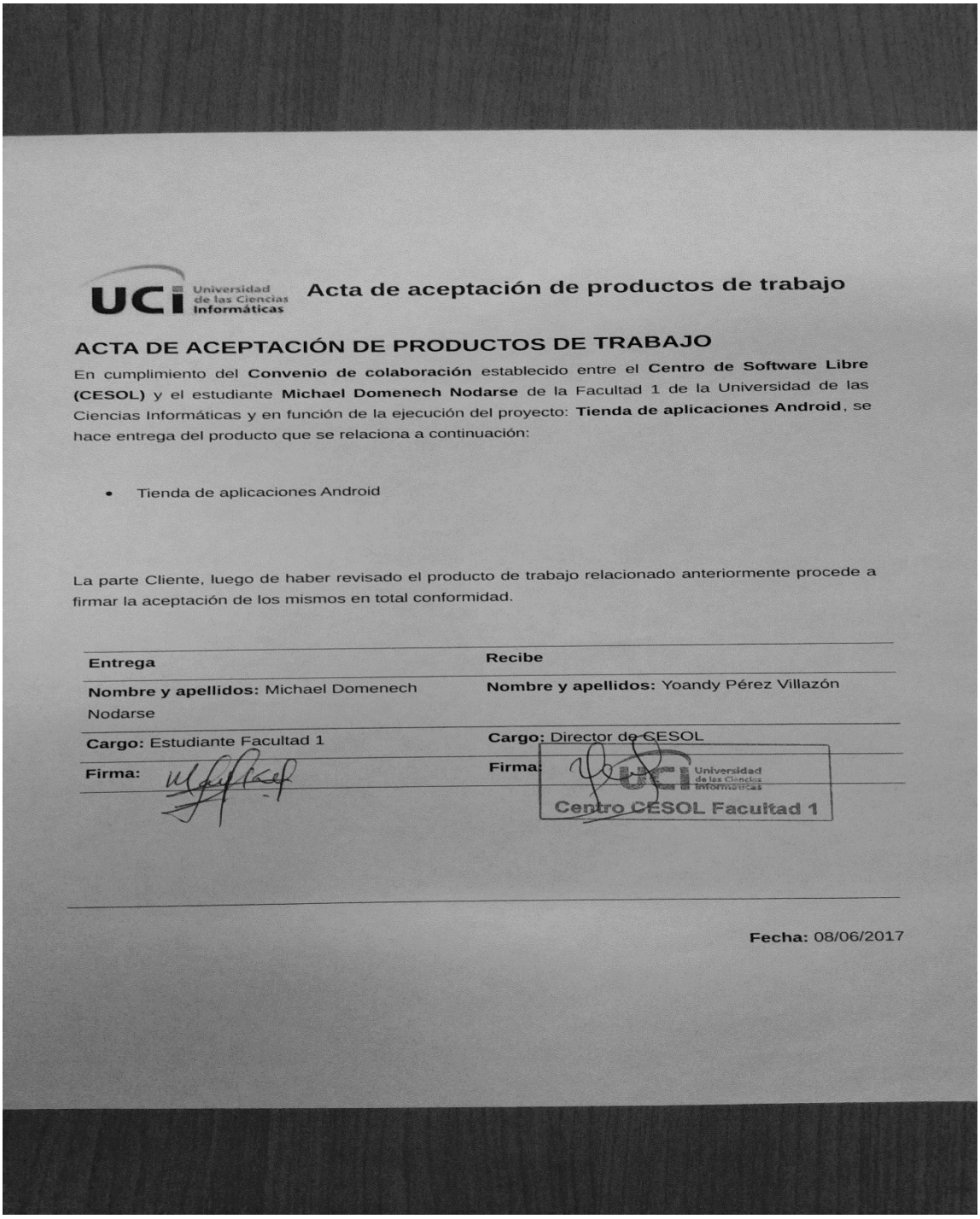


Figura 24: Interfaz "Mostrar información de una aplicación de la tienda".

Anexo 2: Acta de aceptación



UCI Universidad de las Ciencias Informáticas

Acta de aceptación de productos de trabajo

ACTA DE ACEPTACIÓN DE PRODUCTOS DE TRABAJO

En cumplimiento del **Convenio de colaboración** establecido entre el **Centro de Software Libre (CESOL)** y el estudiante **Michael Domenech Nodarse** de la Facultad 1 de la Universidad de las Ciencias Informáticas y en función de la ejecución del proyecto: **Tienda de aplicaciones Android**, se hace entrega del producto que se relaciona a continuación:

- Tienda de aplicaciones Android

La parte Cliente, luego de haber revisado el producto de trabajo relacionado anteriormente procede a firmar la aceptación de los mismos en total conformidad.

Entrega	**Recibe**
Nombre y apellidos: Michael Domenech Nodarse	**Nombre y apellidos:** Yoandy Pérez Villazón
Cargo: Estudiante Facultad 1	**Cargo:** Director de CESOL
Firma:	**Firma:**

UCI Universidad de las Ciencias Informáticas
Centro CESOL Facultad 1

Fecha: 08/06/2017