Universidad de las Ciencias Informáticas

Facultad 1

CubanCloud: Catálogo de servicios para una nube privada

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

Autor:
Roberto Cárdenas Alemán

Tutores:
Ing. Cecilia Esther Hernández Espinosa
1er Tte. Ing. Dany Esquijarosa Bonilla

La Habana, junio de 2017
“Año 59 de la Revolución”

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo Roberto Cárdenas Alemán, con carnet de identidad 93092211264 soy el autor principal del trabajo de diploma titulado: *"CubanCloud: Catálogo de Servicios para una nube privada"* y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

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

Autor:

_____

Roberto Cárdenas Alemán

Tutores:

_____ _____

Ing. Cecilia Esther Hernández Espinosa 1er Tte. Ing. Dany Esquijarosa Bonilla

DEDICATORIA

A quien pagó en una cruz el precio de la salvación de mi alma,
al Rey y gran Sumo Sacerdote, Jesucristo.

A la memoria de mis abuelos Bienvenido, Zenaida y Benito.

A mi abuela Ángela.

A mi madre y a mi esposa.

A mis familiares y amigos.

AGRADECIMIENTOS

A Dios mi Señor Jesucristo por confortar mi alma, por guiarme por sendas de justicia. Porque me ha acompañado atravesando los peores momentos. Su corrección y su amor nunca han dejado de infundirme aliento.

A mi madre amada por toda la confianza que depositó en mi desde pequeño. Por todos los sacrificios que ha tenido que hacer para que pudiera cumplir mi deseo de graduarme en la UCI. Por el buen ejemplo que es para mí.

A mi esposa por su amor y comprensión quien ha cuidado de mí en todo momento. Por darme buenísimas ideas y consejos para la realización de este trabajo. También agradezco a mis suegros por acogerme con tanto amor como si fuese un hijo más.

A mi tutora Cecilia que a pesar de su maestría y de los deberes que tiene como madre, estando licencia de maternidad me ha dedicado de su tiempo para que este trabajo salga lo mejor posible.

A mis hermanos de la iglesia porque sus oraciones me han sustentado.

No olvidaré a mis compañeros de brigada porque sin dudas han sido parte importante de mi vida, a Edel, tu amistad es una de las mayores bendiciones que he recibido en la UCI.

Sobrino, Lexys y Haileen ustedes son esos amigos y hermanos que nunca olvidaré, gracias por ser fieles e incondicionales.

A todos los que de una forma u otra se preocuparon por mi tesis.

RESUMEN

En la actualidad el término computación en la nube ha dejado de ser algo del futuro para convertirse en el presente de las redes de comunicaciones. La computación en la nube consta de cuatro modelos de despliegues: nube privada, pública, comunitaria e híbrida que utilizan tres modelos de servicio: Infraestructura como Servicio, Plataforma como Servicio y Software como Servicio. Cualquier proveedor de servicios nube requiere de una interfaz que le permita la gestión, promoción y contratación de los servicios que ofrece. La Empresa de Tecnologías de la Información para la Defensa trabaja en la creación de un ecosistema nube, por esta razón la presente investigación tuvo como objetivo el desarrollo de un catálogo de servicios que permita la gestión, promoción y contratación de los servicios de una nube privada. Para la implementación de este catálogo se utilizó *Extreme Programming* como metodología de desarrollo, como lenguaje de programación *Python* 2.7.11, como marco de trabajo web *Django* 1.8.7 y como sistema de gestión de base de datos a *PostgreSQL* 9.4. La realización de esta investigación arrojó una solución que permite el proceso de gestión, promoción y contratación de servicios nube a terceros.

Palabras claves: catálogo de servicios, computación en la nube, contrato, servicio.

ÍNDICE

INTRODUCCIÓN

El término negocio se define como una actividad que se lleva a cabo con un fin lucrativo. Por lo general, dicha actividad está asociada a la producción y a la compra-venta de productos y servicios (EAE Business School, 2017). Los empresarios han visto en el desarrollo de las Tecnologías de la Información (TI) un nuevo modelo de negocio que traslada a la empresa del plano físico al virtual (Mendelson, 2015). Actualmente se pueden encontrar empresas que desarrollan negocios de todo tipo. Entre estas figuran las dedicadas a ofrecer hardware y software como servicio que con el desarrollo de la computación en la nube han alcanzado gran presencia en la web porque sus ofertas son cada vez más variadas y la demanda de sus servicios es creciente (El País, 2014).

La adopción de las tecnologías de la computación en la nube permite comercializar porciones de hardware virtualizado abstrayendo al cliente de la infraestructura subyacente. Es así como el cliente solo contrata los recursos necesarios en el momento, sin preocuparse de una planificación detallada, sobre el crecimiento futuro de la empresa o la redundancia necesaria para proteger el negocio contra catástrofes informáticas (García Parellada, 2014). La Empresa de Tecnologías de la Información para la Defensa (XETID) es una de las empresas cubanas que está trabajando en la implementación de una solución nube para ofrecer servicios a un grupo de clientes específicos.

Este proveedor desea darle a conocer a sus clientes los servicios que ofrecerán. Esta es la razón por la que están interesados en eliminar las barreras comunicativas y las explicaciones con exceso de vocabulario técnico. De igual manera se requiere elaborar un material que sea lo suficientemente atractivo como para captar la atención del cliente. Por lo que es necesario explicar de forma sencilla en qué segmento de la economía se mueve la empresa, qué es lo que proveen, el valor de sus servicios ya sea en cuanto a calidad, garantía, experiencia o cualquier otro aspecto que marque la diferencia (Gálvez, 2015).

El objetivo de la XETID es proveerle al cliente el acceso bajo demanda a un conjunto compartido de recursos tales como: redes, almacenamiento, aplicaciones y servicios, a los que puede acceder a través de internet. Si un cliente deseara contratar uno de los servicios ofrecidos debería conciliar una cita con el encargado del negocio para conocer las ofertas, precios, planes y plazos. Cuando el cliente conozca las ofertas deberá elegir teniendo en cuenta su necesidad. Una vez cumplidos los dos pasos anteriores se procedería a firmar el contrato y se realizaría el pago por los servicios seleccionados. La forma en que la XETID ofrecería estos servicios de computación en la nube no es la idónea ya que incurriría en las siguientes deficiencias:

- El cliente debería trasladarse hasta el lugar donde radica la empresa cada vez que necesite conocer algún detalle de los servicios ofrecidos.
- Las ofertas carecerían de soportes gráficos.
- Las explicaciones de los servicios no serían atractivas porque podrían ser extensas y el cliente perdería el interés.
- La contratación sería lenta debido a la necesidad de conciliar reuniones.

Ante estas dificultades la XETID desea hacerle llegar a sus clientes los servicios que ofrecerá de forma amena, intuitiva, explicativa, que no genere distorsiones y que pueda hacer su elección con un mínimo de esfuerzo.

Ante la problemática descrita anteriormente se define como **problema de investigación:** ¿Cómo contribuir a la gestión, promoción y contratación de los servicios de una nube privada para un negocio de computación en la nube?

Se establece como **objeto de estudio**: los catálogos de servicios de TI para la computación en la nube. Se declara como **campo de acción**: los catálogos de servicios de TI para una nube privada.

Lo que permite que se plantee como **objetivo general**: desarrollar un catálogo de servicios que permita la gestión, promoción y contratación de los servicios para una nube privada.

Para dar cumplimiento al objetivo general se definieron las siguientes **tareas de investigación**:

1. Sistematización de los referentes teóricos que sustentan la investigación, relacionados con el uso de los catálogos de servicios de TI para la gestión, promoción y contratación de servicios de una nube privada.
2. Caracterización de los catálogos de servicios de TI utilizados en la gestión, promoción y contratación de servicios de una nube privada.
3. Análisis del estado actual de los catálogos de servicio de TI en proveedores de servicios de computación en la nube.
4. Análisis y diseño de las funcionalidades del catálogo de servicios de TI para gestionar, promocionar y contratar los servicios de una nube privada.
5. Implementación de las funcionalidades del catálogo de servicios de TI para gestionar, promocionar y

contratar los servicios de una nube privada.

6. Realización de las pruebas funcionales al catálogo de servicios implementado.

El cumplimiento satisfactorio de las tareas de investigación contribuirá al desarrollo de la **idea a defender:** La implementación del catálogo de servicios de TI permitirá gestionar, promocionar y contratar los servicios que se ofrecen en una nube privada.

Con el propósito de dar solución a las tareas antes mencionadas se utilizaron los siguientes métodos:

Métodos Teóricos

- **Histórico-Lógico:** Se utilizó con el objetivo de realizar un estudio sobre los antecedentes, evolución y tendencias actuales de los catálogos de servicios para la computación en la nube, lo que posibilitó la determinación de los aspectos más relevantes.
- **Analítico-Sintético:** Se utilizó con el objetivo de realizar un análisis bibliográfico de documentos, trabajos de diploma, artículos y la consulta bibliográfica de diferentes autores en la temática, para así extraer los elementos más significativos en el campo de los catálogos de servicio.

Métodos Empíricos

- **Observación:** Se utilizó para hacer un análisis y evaluación del comportamiento de los catálogos de servicios para soluciones nube, con el objetivo de identificar elementos que hagan de este una solución competente.

El presente trabajo está estructurado en introducción, tres capítulos de contenido, conclusiones, recomendaciones y referencias bibliográficas. El orden y contenido de los capítulos es el siguiente:

Capítulo 1. "Fundamentación teórica del catálogo de servicios para una nube privada**"**, donde se definen los principales conceptos a tener en cuenta, resultando la base que sustenta el desarrollo de las tareas de investigación. Se realiza un análisis de los catálogos de servicio de TI de diferentes proveedores de nubes privadas, con el objetivo de determinar elementos similares a tener en cuenta para las fases de análisis, diseño e implementación. Además, se determinan las herramientas y tecnologías que se emplearon en la construcción de la solución.

Capítulo 2. “Análisis, diseño e implementación del catálogo de servicios para una nube privada”, se muestra el análisis y el diseño realizado previo a la implementación. Para lograrlo se presenta la descripción de la propuesta de solución y los requisitos obtenidos durante el análisis. Además de un grupo de artefactos que ayudan a comprender el diseño del sistema.

Capítulo 3. “Pruebas al catálogo de servicios para una nube privada”, En este capítulo se hace alusión a la fase de prueba propuesta por la metodología de desarrollo la cual indica realizar pruebas unitarias y pruebas de aceptación para comprobar el cumplimiento de los requerimientos establecidos en las historias de usuarios.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA DEL CATÁLOGO DE SERVICIOS PARA UNA NUBE PRIVADA

En este capítulo se abordan los conceptos asociados al dominio del problema. Además, se realiza un estudio de los sistemas homólogos existentes a partir de las tendencias a nivel internacional, se enfatiza en el catálogo de servicios que ofrecen. Por otra parte, se especificarán las herramientas y tecnologías que sirven de base a la propuesta de solución.

1.1. Conceptos asociados al dominio del problema

A continuación, se relacionan los principales conceptos que ayudan a entender el desarrollo de la investigación.

1.1.1. Definición de catálogo de servicios de Tecnología de la Información (TI)

Un catálogo de servicios es la manifestación formal de la provisión de servicios entre el proveedor TI y los clientes. Provee una descripción general, en términos del negocio y de infraestructura tecnológica, de los servicios ofrecidos, su costo o precio, y el esquema de facturación (Murcia, 2010). También como el menú de un restaurante: presenta a los clientes los servicios que se pueden prestar. Típicamente un catálogo debe incluir los sistemas que ya están en funcionamiento (Arcilla, y otros, 2012). Además, se añade que este debe:

- Definir los servicios y los acuerdos de nivel de servicio que se alinean con las necesidades del negocio.
- Gestionar los procesos de pedido de servicios y adaptarlos a la demanda del cliente.
- Asegurar una entrega de servicios eficientes al automatizar el cumplimiento y monitorear la calidad del servicio.
- Comunicar el valor que esos servicios representan para el negocio (NewScale, 2005).

Por otra parte, se dice que este es el registro de productos y servicios que se usa para describirlos, publicarlos y promoverlos. Define los servicios estándares y sus niveles de servicio. Se recomienda que incluya precios y métodos de pago. Es el punto de partida para que el cliente solicite nuevos servicios y que

se puedan negociar Acuerdos de Nivel de Servicio (SLA[1]) apropiados. Permite observar y calcular las utilidades que proporcionan los diferentes servicios, y en su caso hacer ajustes sobre la oferta. Además, debe asignársele un responsable para mantenerlo actualizado (Sánchez, 2012).

Se concluye que un catálogo de servicios es una herramienta que recopila toda la información relacionada con los servicios que se ofertan para lograr un entendimiento entre la empresa y los clientes. Entre los requisitos que debe cumplir dicha herramienta se destacan: la correcta redacción, evitando a toda costa el empleo de lenguaje técnico, la disponibilidad para todos aquellos que intervienen de una forma u otra en la impartición del servicio, servir de guía a los clientes, mostrar el precio correspondiente de cada servicio, y registrar sus clientes actuales.

1.1.2. Definición de computación en la nube

La computación en la nube es un modelo que provee el acceso bajo demanda a un conjunto compartido de recursos tales como redes, almacenamiento, aplicaciones y servicios, a través de una red de datos. Este modelo promueve la disponibilidad de recursos de cómputo y se encuentra compuesto por cinco características esenciales: auto servicio bajo demanda, acceso de red con banda ancha, concentración de recursos, elasticidad rápida y servicio contabilizado, tres modelos de servicios: Infraestructura como Servicio (*IaaS, Infrastructure as a Service*), Plataforma como Servicio *(PaaS, Platform as a Service*) y Software como Servicio (*SaaS*, *Software as a Service*) y cuatro modelos de despliegue (Nube Pública, Nube Privada, Nube Comunitaria y Nube Híbrida) (Mell, y otros, 2009).

1.1.3. Definición de nube privada

El NIST[2], añade en su definición de nube privada que su infraestructura es operada únicamente para una organización y esta puede ser administrada por la organización o un tercero y residir dentro de sus instalaciones o fuera de ellas (Mell, y otros, 2009) . Según el sitio oficial de *Microsoft*, (…) Nube privada es la implementación de servicios de la nube sobre los recursos que se dedican a su organización (…). Además, añade que (…) Con una nube privada, obtendrá muchos de los beneficios de los servicios de una nube pública, lo que incluye autoservicio, escalabilidad y elasticidad, con el control adicional y

[1] Siglas correspondientes al término en inglés: *Service Level Agreement.*
[2] Siglas correspondientes al término en inglés: *National Institute of Standards and Technology.*

personalización para disponer de recursos dedicados(...). Siguiendo esta última definición se concluye que la nube privada además de brindar los servicios descritos en la definición de computación en la nube posee la particularidad de ser operada o utilizada por una sola organización con lo que se logran la seguridad de sus datos y el ahorro de recursos tecnológicos al disponer de los principales servicios de forma virtualizada.

1.1.4. Infraestructura como servicio (IaaS)

Contempla la entrega de servicios de infraestructura, también denominados servicios computacionales fundamentales, entre los cuales se encuentran: almacenamiento, procesamiento y memoria. Dicha infraestructura es entregada bajo demanda, permitiendo a los usuarios el despliegue de aplicaciones sobre un sistema operativo principal. El usuario final no podrá manejar la plataforma de computación en la nube según su conveniencia, aunque si podrá controlar dispositivos de almacenamiento, de red y sistemas operativos (Mendoza Ariza, y otros, 2012).

1.1.5. Plataforma como servicio (PaaS)

Permite ofrecer como servicio plataformas para desplegar aplicaciones suministradas por los usuarios, teniendo como condición que este debe ajustarse a los parámetros de configuración exigidos por el proveedor, como son el lenguaje de programación y el servidor de aplicaciones. En general, puede decirse que *PaaS* es un servicio de plataforma con el cual es posible el desarrollo de software a través de la red (Mendoza Ariza, y otros, 2012).

1.1.6. Software como servicio (SaaS)

Consiste en la entrega de software alojado en un servidor externo y perteneciente a un proveedor de servicios. Este modelo oculta totalmente aspectos de administración y control de la infraestructura base de computación en la nube y sólo permite una configuración limitada al software por parte del usuario final. El concepto de software como servicio se basa en que la empresa cliente se registra para usar la aplicación que está alojada en un servidor externo, en lugar de comprar licencias para instalarlo en computadoras individuales. Esta característica le da al comprador más flexibilidad para cambiar de proveedor y menos problemas de mantenimiento de software (Mendoza Ariza, y otros, 2012).

1.2. Elementos presentes en un catálogo de servicios

Según el marco de referencia de buenas prácticas para la administración de servicios de TI (ITIL) en su manual versión 3.0 refiere que: un catálogo de servicios contiene información de los servicios específicos por los que el cliente está dispuesto a pagar. Se trata de una herramienta de gestión del conocimiento, que permite a los empleados y consultores dirigir su solicitud sobre los servicios. Cada servicio en el catálogo debe contener los siguientes elementos:

- Una etiqueta de identificación para el servicio.
- Descripción de los servicios.
- Tipos de solicitud de servicios relacionados.
- Categorización del servicio o tipo que le permite ser agrupados con otros servicios similares.
- Interfaces y dependencias entre todos los servicios y componentes de apoyo y elementos de configuración en el catálogo de servicios y el CMS[3].
- Claramente la propiedad y responsabilidad sobre el servicio.
- Costos asociados.
- Puntos de escalamiento y contratos clave.
- Acuerdos de nivel de servicio (SLA) de datos.

Los catálogos de servicios de TI son necesarios para cualquier empresa que brinda servicios, debe estar enfocado a los clientes, e incluir tan sólo aquellos servicios que la organización está prestando actualmente. El enfoque ha de ser comercial, y debe cuidarse mucho el lenguaje para no emplear tecnicismos.
La elaboración de este catálogo de servicios puede resultar una tarea compleja, pues es necesario alinear aspectos técnicos con políticas de negocio. Además, debe garantizar:

- Servir de guía a los clientes a la hora de seleccionar un servicio que se adapte a sus necesidades.
- Delimitar las funciones y compromisos de la organización TI.
- Ser utilizado como herramienta de venta.

[3]Siglas correspondientes al término en inglés: *Content Management System*, es un programa desarrollado para que cualquier usuario pueda administrar y gestionar contenidos de una web sin poseer conocimientos de programación web.

- Evitar malentendidos entre los diferentes actores implicados en la prestación de servicios.

Para lograr este fin, se pueden emplear herramientas informáticas que permitan la creación de catálogos de servicios. A continuación, se realiza un estudio de las soluciones que presentan otros proveedores para mostrar los servicios de computación en la nube que ofrecen.

1.3. Estudio de sistemas homólogos

En la actualidad existen varios proveedores que utilizan un catálogo para mostrar sus servicios de computación en la nube. Con el objetivo de identificar las buenas prácticas y casos de error, se realiza el siguiente análisis haciendo énfasis en las características y funcionalidades comunes en este tipo de soluciones.

1.3.1. Catálogos de servicios de nubes privadas

OneDrive: en su catálogo de servicios están las opciones de almacenamiento de *Microsoft*. Puede almacenar cualquier tipo de archivo en el servicio, incluyendo fotos, video y documentos, y luego acceder a ellos desde cualquier dispositivo móvil o computadora con Windows. Sus planes se limitan a 1 TB. Además, cuenta con un plan de 50 GB por $ 1.99 por mes. Por último, cualquier persona con una cuenta de *Microsoft* sólo obtendrá 5 GB de almacenamiento gratuito (Mitroff, 2016).

Dropbox: es otro de los favoritos en el mundo del almacenamiento en la nube, recibe elogios por su diseño limpio y adaptable al tamaño de la pantalla del dispositivo donde se acceda a su catálogo de servicios. Su diseño es muy básico y no da muchas opciones para ver y organizar los archivos, sus aplicaciones móviles y aplicaciones de escritorio permiten una buena navegación. Ofrece a sus usuarios oportunidades para obtener más espacio de almacenamiento para reforzar los 2 GB que se obtienen cuando el cliente se registra (Mitroff, 2016). Actualmente cuenta con ofertas para personas y para negocios lo que hace que el servicio esté disponible para distintos tipos de clientes (Dropbox, 2016).

Google Drive: combina un conjunto completo de herramientas de oficina con almacenamiento en la nube. Su catálogo posee un diseño limpio, plano con pocos elementos y adaptable a las dimensiones de la pantalla de cualquier dispositivo. Además, resaltan sus opciones de almacenamiento que invitan a los clientes a usar de forma gratis 15 GB, compartidos con la cuenta de *Gmail*, las fotos que el cliente desee subir a *Google+*

y cualquier documento que realice en *Google Drive*. En su guía de precio aparecen solo tres opciones: 100 GB al mes por $ 1.99, 1 TB al mes por $ 9.99 y 10 TB al mes por 99.99. Para mostrar estos planes no utiliza ningún efecto que llame la atención, su estrategia es impactar al cliente con los precios más que con la interfaz de usuario que ofrece su catálogo. (Google, 2016).

Amazon Cloud Drive: su catálogo conserva el concepto de la tienda online que representa Amazon. Su diseño es plano y carece de animaciones. Para comenzar a usar cualquiera de sus servicios se debe especificar en qué tipo de dispositivo se desea trabajar. Posee diferentes planes para almacenamiento entre ellos uno solo para fotos y otro para todos los otros tipos de archivos. Ninguno de los planes es gratuito, pero ambos tienen la opción de pruebas gratis por tres meses (Amazon Drive, 2016).

Para obtener un resumen de las principales características de los planes que ofertan estas plataformas para el almacenamiento en una nube privada ver tabla 29 del Anexo 2. Por otra parte, los servicios de *hosting* también están asociados al tema de la computación en la nube. Al analizar algunos de los sitios que brindan este servicio se pueden obtener algunas funcionalidades que no deben faltar en la aplicación. A continuación, se resume cada uno de los sistemas analizados:

Dinahosting: la cantidad de servicios que ofrece no sobrepasa de 3 por cada vista o categoría de servicio. Su catálogo posee un carrito de pedidos que es el encargado de ir almacenando los servicios que el cliente desee contratar (Dinahosting, 2016).

Hostsonny: inicialmente muestra una tabla con todos sus planes, pero no es en la portada del sitio donde se contrata el servicio. Primero el cliente tiene que solicitar el servicio, luego introducir el dominio donde se publicará su sitio web y por ultimo pasará por el carrito de compra y verá todas sus solicitudes y efectuará el pago en línea (HostSonny, 2016).

Hostinger: posee una portada donde se promociona el servicio y cuenta con un menú que entre sus opciones resalta el carrito de compras donde se realiza el pago. Dicho carrito muestra a los clientes la cantidad de servicios que tiene seleccionados. Una vez que el cliente desea pagar tiene que dirigirse al carrito y escoger con qué tipo de tarjeta desea realizar el pago. Este sitio muestra los planes en una tabla donde contempla los nombres de los servicios, sus características, y el precio. En un extremo inferior de la

tabla aparece para cada servicio el porcentaje de descuento y la opción de seleccionar el plan (Hostinger, 2017).

Interoute: al igual que los anteriormente mencionados se dedica al hosting y ofrece SaaS y PaaS, pero su catálogo no se limita a vender sus servicios. También se puede encontrar abundante información sobre la computación en la nube y cuenta con un blog. Por otra parte, el diseño de su catálogo es plano sin muchos gráficos y emplea colores serios (Interoute, 2017).

Una vez culminado el estudio de los sistemas homólogos se puede concluir que ninguna de las soluciones cumple con el objetivo de la investigación porque resultan ser muy específicas para el negocio al que representan lo que impide su uso en otro negocio. Además, las soluciones nubes promocionan sus servicios por planes de pago, pero solo se enfocan en brindar servicio de almacenamiento haciendo distinción en algunos casos del tipo de archivo que permiten almacenar, carecen de la posibilidad de brindar SaaS y PaaS. Por otra parte, los sitios de hosting si ofrecen SaaS y PaaS, pero también se dedican a representar su propia empresa. Sin embargo, permitió la identificación de funcionalidades y tecnologías que pueden contribuir a un mejor desarrollo de la propuesta de solución que se propone en la investigación.

1.4. Herramientas y tecnologías

1.4.1. Metodología de desarrollo de software

Según (Sommerville, 2011) una metodología de desarrollo de software es un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas, sugerencias de diseño y guías de procesos. Las metodologías se pueden clasificar como ágiles y tradicionales. Las tradicionales son aquellas que fundamentalmente están centradas en el control del proceso, además son las más efectivas para proyectos de gran envergadura. Las metodologías ágiles representan una opción razonable a las metodologías de software tradicionales para ciertas clases de software y ciertos tipos de proyectos. Ha demostrado su utilidad al entregar sistemas exitosos con rapidez (Vinent Puig, y otros, 2015).

Proceso Unificado Ágil: En inglés *Agile Unified Process (AUP)* es una versión simplificada del *Rational Unified Process (RUP)*. Este describe simple la forma de desarrollar aplicaciones de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en *RUP*. La metdología *AUP* aplica técnicas ágiles incluyendo desarrollo dirigido por pruebas, modelado ágil, gestión de cambios ágil, y

refactorización de base de datos para mejorar la productividad. De igual manera ocurre gestión de riesgos y se propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Por lo que se dice que *AUP* establece un modelo más simple que el que aparece en *RUP* porque reúne en una única disciplina las disciplinas de modelado de negocio, requisitos, análisis y diseño. El resto de las disciplinas (implementación, pruebas, despliegue, gestión de configuración, gestión y entorno) coinciden con las restantes de *RUP* (Cordero, 2014).

***AUP*-Variante para la UCI:** Surge debido a la necesidad de contar con una metodología que se adapte al ciclo de vida definido para la actividad productiva de la UCI y se apoya en las buenas prácticas que plantea el modelo CMMI-DEV v1.3. La variante de *AUP* para la UCI está formada por tres fases, (inicio, ejecución y cierre) para el ciclo de vida de los proyectos de la universidad, las cuales contienen las características de las cuatro fases (inicio, elaboración, construcción y transición) propuestas en *AUP* (Rodríguez Sánchez, 2015) .

Scrum: es un marco de trabajo para el desarrollo y el mantenimiento de productos complejos. Es considerado como metodología ágil. Se basa en el control de procesos de forma empírica. Emplea un enfoque iterativo e incremental para aumentar la predictibilidad y el control de riesgo. Está especialmente indicada para proyectos con un rápido cambio de requisitos. El trabajo a realizar durante cada una de las iteraciones es seleccionado por el equipo de desarrollo de una lista de requerimientos priorizados conocida como el *Product Backlog*. De esta lista, el equipo selecciona los requerimientos con mayor prioridad que puede desarrollar completamente. Luego se define y estima las tareas necesarias para cumplir con esos requerimientos; ese listado de tareas es conocido como *Sprint Backlog*. Los roles, artefactos, eventos y reglas de Scrum son inmutables y aunque es posible implementar solo partes de Scrum, el resultado no es Scrum. Este solo existe como un todo y funciona bien como contenedor para otras técnicas, metodologías y prácticas (Schwaber, y otros, 2013).

Extreme Programming (XP) propone la retroalimentación continua entre el cliente y el equipo de desarrollo, la comunicación fluida entre todos los participantes y la simplicidad en las soluciones implementadas. Los valores y principios de las metodologías ágiles expuestos en el Manifiesto Ágil son la inspiración de la misma (Letelier, et al., 2006). Su ciclo de vida consiste en seis fases: exploración, planificación de la entrega (*Release*), iteraciones, producción, mantenimiento y muerte del proyecto. Esta metodología se define como adecuada para proyectos con requisitos imprecisos y cambiantes, y donde existe un alto riesgo técnico. Es

una metodología encaminada para el desarrollo; consiste en una programación rápida o extrema, cuya particularidad consiste en tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto (Brito, 2009). Teniendo en cuenta que el equipo es pequeño, integrado en su totalidad por dos personas, el desarrollador y el usuario final, quien participa constantemente con el desarrollador. De igual manera los requerimientos están sujetos a cambios y que el proyecto es considerado pequeño y de corto plazo. Resulta decisiva la utilización de XP para guiar el proceso de desarrollo de la solución.

1.4.2. Herramienta CASE

En el presente proyecto se decidió utilizar como herramienta CASE[4] a *Visual Paradigm 8.0* pues esta ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo de software a través de la representación de todo tipo de diagramas. Presenta licencia gratuita y comercial. Es fácil de instalar y actualizar y compatible entre versiones. Esta herramienta fue diseñada para una gran variedad de usuarios interesados en la construcción de sistemas de software de forma confiable mediante la utilización de un enfoque orientado a objetos. Entre sus características resaltan, la disponibilidad en múltiples plataformas (*Windows, Linux*), diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad. Así mismo usa un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación, capacidades de ingeniería directa e inversa, modelo y código que permanece sincronizado en todo el ciclo de desarrollo. También soporta aplicaciones web, varios idiomas, compatibilidad entre ediciones, soporte de UML versión 2.1, generación de base de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, generador de informes y editor de figuras entre otras. Por otra parte las imágenes y reportes generados no poseen buena calidad (Visual Paradigm International, 2016)

Esta herramienta permite aumentar la calidad del software, a través de la mejora de la productividad en el desarrollo y mantenimiento del software. Aumenta el conocimiento informático de una empresa ayudando así a la búsqueda de soluciones para los requisitos. También permite la reutilización del software, portabilidad y estandarización de la documentación, además del uso de las distintas metodologías propias

[4] (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de software.

de la ingeniería de software. En la UCI se ha generalizado su uso y es la herramienta que más se usa para realizar todo el modelado del proceso de desarrollo de software.

1.4.3. Lenguaje de modelado

UML (*Unified Modeling Language*): es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML es un lenguaje expresivo, claro y uniforme, que no garantiza el éxito de los proyectos, pero si mejora sustancialmente el desarrollo de los mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios. Es independiente del proceso, aunque para utilizarlo óptimamente se debe usar en un proceso que sea dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental. Es importante resaltar que UML es un lenguaje para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar (Alonso Martínez, y otros, 2010). UML en su versión 2.1 es el lenguaje de modelado empleado ya que es empleado en la versión seleccionada de la herramienta CASE.

1.4.4. Tecnologías del lado del servidor

Lenguaje de programación

Python fue creado por *Guido van Rossum*, es potente y posee una elegante sintaxis. Es interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos (Python, 2016). Un lenguaje interpretado es aquel que se ejecuta con un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora. *Python* tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi-interpretado. En *Python*, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo código de máquina intermedio llamado *bytecode* la primera vez que se ejecuta, genera archivos .pyc o .pyo (*bytecode* optimizado), que son los que se ejecutarán en sucesivas ocasiones (Duque, 2008).

Se dice que un lenguaje es dinámicamente tipado si una misma variable puede tomar valores de distinto tipo en distintos momentos. Su tipo se determinará en tiempo de ejecución según el tipo de valor al que se asigne y el tipo de esta variable puede cambiar si se le asigna un valor de otro tipo. Por otra parte, es

fuertemente tipado cuando no se permite tratar a una variable como si fuera de un tipo distinto al que tiene, es necesario convertir de forma explícita dicha variable al nuevo tipo previamente. Para los desarrolladores el hecho que sea multiplataforma es muy ventajoso pues permite utilizarlo sobre plataformas open source, añadiendo solo las librerías específicas de la plataforma en cuestión permitiendo que código se ejecute sin grandes cambios (Duque, 2008).

Para el desarrollo de la propuesta de solución se empleó *Python* en su versión 2.7.11. debido a que es un lenguaje potente cuya sintaxis es considerada elegante debido a que posee un sistema de indentación que ahorra las llaves, los *begin* y *end*, los paréntesis y otros elementos. Este cuenta con un amplio repositorio de aplicaciones reusables, librerías y documentación que facilitan el trabajo a los desarrolladores. Además, es de código abierto, su licencia permite su uso y distribución de forma gratuita siempre que no viole los derechos de terceros. En la actualidad cuenta con varios marcos de trabajo para el desarrollo web, entre ellos *Django* es preferido por una amplia comunidad de desarrolladores.

Marco de trabajo de desarrollo web

Existe una gran variedad de marcos de trabajo de *Python* para el desarrollo web, a continuación, se resume un estudio de estos destacando sus características esenciales:

Flask: es un *microframework* depende de dos bibliotecas externas: el motor de plantilla Jinja2 y el kit de herramientas Werkzeug WSGI. Es recomendado para aprender a programar, crear prototipos o aplicaciones independientes. No cuenta con una estructura de la cual partir, el trabajo comienza con una página en blanco. Este no proporciona grandes funcionalidades pero existen extensiones disponibles para agregar ORM, validación de formularios, manejo de carga, entre otros (Flask, 2016).

Web2py: es gratuito y de código abierto escrito en *Python*, se enfoca al rápido desarrollo de aplicaciones web, con la aplicación de un controlador de base de datos escalable, seguro y portátil, siguiendo LGPLv3 como licencia. Proporciona soluciones integrales para que todo el proceso de desarrollo pueda llevarse a cabo en el navegador. Además, posee funciones como la versión web para el desarrollo en línea, la escritura de plantillas HTML, la carga de archivos estáticos, la escritura de bases de datos y otras funciones como la función de diario y una interfaz de administración automatizada (LinkedIn, 2015). Sin embargo, por ser más reciente que *Django* posee menos documentación y una menor comunidad de desarrollo. Además, los desarrolladores lo prefieren para aquellos sistemas que requieran la especialización de roles según la labor de cada usuario (Python, 2012).

***Django*:** es un marco de trabajo web de código abierto y libre que permite construir aplicaciones web de forma rápida y con menos código porque se centra en automatizar todo lo posible. Como se plantea en su sitio oficial www.djangoproject.com "Con *Django*, se puede concebir las aplicaciones web desde el concepto hasta el lanzamiento en cuestión de horas. Este se encarga de gran parte de las molestias del desarrollo web, para que el desarrollador pueda centrarse en la escritura de la aplicación sin necesidad de reinventar la rueda". Su interfaz de administración es una de las características más atractivas que posee.

En el presente trabajo se utilizará *Django 1.8.7* porque es una versión estable y compatible con *Python 2.7.11.* También posee una interfaz administrativa que supera a la de otros *marcos de trabajo.* Posee diseño de URL[5] sin limitaciones, un potente sistema de plantillas independiente al código *Python*, mapeador objeto-relacional que define los datos en su totalidad en *Python*, soporte para aplicaciones multilenguaje y permite especificar cadenas de traducción. Otros elementos a tener en cuenta figuran la seguridad que ofrece, ya que elimina de forma interna las vulnerabilidades que comúnmente suelen dejar los desarrolladores. También pone énfasis en la reutilización de código, la conectividad y extensibilidad de componentes y el principio: no te repitas (DRY, del inglés *Don't Repeat Yourself*). (Django Software Fundation, 2016).

Sistema Gestor de Base de Datos (SGBD)

***PostgreSQL 9.4*:** este es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD[6] y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarles a otras bases de datos comerciales (PostgreSQL Global Development Group, 2013). Entre sus muchas características se sabe que utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. Para el almacenamiento de los datos persistentes se selecciona *de este* porque el cliente solicitó que los datos fueran almacenados sobre un gestor de bases de datos *PostgreSQL*, pues su filosofía de trabajo está orientada al software libre y posee más experiencia de trabajo con esta tecnología.

[5] Siglas correspondientes al término en inglés:*Uniform Resource Locator.*
[6] Es una licencia de software libre permisiva como la licencia de *OpenSSL* o la *MIT License*.

Herramienta para la administración de la base de datos

***PgAdmin III*:** se usa para manejar bases de datos hechas en *PostgreSQL 9.4* y superiores, funciona sobre casi todas las plataformas. Este software fue diseñado para responder a las necesidades de todos los usuarios, desde la escritura de simples consultas *SQL* a la elaboración de bases de datos complejas. La interfaz gráfica es compatible con todas las características de *PostgreSQL* y facilita su administración. La aplicación también incluye un editor de la sintaxis *SQL*, un editor de código del lado del servidor y un agente para la programación de tareas (PgAdmin, 2016). Para instalarlo en Linux solo hay que instalar el paquete pgadmin3 pues ya se encuentra en los repositorios.

1.4.5. Entorno Integrado de Desarrollo (IDE)

El IDE seleccionado es PyCharm 2016.3.2 Professional, es desarrollado por la empresa JetBrains, este es multiplataforma y es utilizado para desarrollar usando el lenguaje de programación *Python*. Proporciona análisis de código, depuración gráfica, posee integración con VCS[7] / DVCS[8] y soporte para el desarrollo web con *Django* (Django Software Fundation, 2016).

1.4.6. Servidores web

Para el despliegue de la aplicación en un entorno de producción se valoró las siguientes alternativas se servidores web:

Apache: es un servidor web flexible, rápido y eficiente, continuamente actualizado y adaptado a los nuevos protocolos HTTP[9]. También es multiplataforma y modular por lo que puede ser adaptado a diferentes entornos y necesidades. Esto es posible debido a los diferentes módulos de apoyo que proporciona, y con la API[10] de programación de módulos, para el desarrollo de módulos específicos. De igual forma gracias a que es modular se han desarrollado diversas extensiones (Apache, 2016). Al habilitar el módulo mod_wsgi se pueden servir archivos *Python* (Libros Web, 2016).

[7] Siglas correspondientes al término en inglés: *Version Control System*.
[8] Siglas correspondientes al término en inglés: *Distributed Version Control System*.
[9] Siglas correspondientes al término en inglés: *Hypertext Transfer Protocol*.
[10] Siglas correspondientes al término en inglés: *Application Programming Interface.*

Nginx: es uno de los servidores web más populares del mundo y es usado por la mayoría de los sitios web con más tráfico de internet. Usa menos recursos que Apache en la mayoría de los casos, y puede ser usado como servidor web o proxy inverso (Lucas, 2014). Según el sitio oficial de Nginx, este es un servidor HTTP libre, de código abierto, de alto rendimiento y *proxy* inverso, así como un servidor proxy IMAP/POP3. Es conocido por su alto rendimiento, estabilidad, conjunto de características ricas, configuración sencilla y bajo consumo de recursos. Para la selección de este servidor web, se tuvo en cuenta sus principales características. Entre estas se destacan: su capacidad para manejar más de 10 000 conexiones simultáneas con bajo uso de memoria, el bajo consumo de recursos, el balanceo de la carga, el soporte para TLS[11]/SSL[12] y el manejo de ancho de banda.

Gunicorn: es un servidor HTTP de *Python* WSGI para UNIX[13]. es ampliamente compatible con varios marcos de trabajo web y bastante rápido (Gunicorn, 2017). Este fue el empleado para servir la aplicación en *Python*.

1.4.7. Tecnologías del lado del cliente

AJAX: es una técnica de desarrollo web para crear aplicaciones interactivas o *RIA (Rich Internet Applications)*. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones (Gómez Pérez, y otros, 2011).

Lenguaje de Marcado de Hipertexto versión 5 (HTML5): es un lenguaje de etiquetas y las páginas web habituales están formadas por cientos o miles de pares de etiquetas. La principal ventaja de los lenguajes de etiquetas es que son muy sencillos de leer y escribir por parte de las personas y de los sistemas electrónicos. La principal desventaja es que pueden aumentar el tamaño del documento, por lo que en general se utilizan etiquetas con nombres muy cortos. En HTML5 se añaden nuevas etiquetas como *canvas*,

[11] Siglas correspondientes al término en inglés: *Transport Layer Securit*.
[12] Siglas correspondientes al término en inglés: *Secure Sockets Layer*.
[13] Es el nombre de un sistema operativo creado por Ken Thompson y Dennis Ritchie en 1969, este es la base para el sistema operativo Linux.

audio, *video*, *header*, *footer*, *article*, *nav*, *time* entre otras. Permite generar tablas dinámicas que pueden filtrar, ordenar y ocultar contenido en el cliente (Libros Web, 2016).

Bootstrap 3.0: Es un marco de trabajo que permite la creación de diseños web combinando HTML, CSS y JavaScript. Este permite crear diseños adaptables a las dimensiones de la pantalla de cualquier dispositivo (laptop, Tablet, teléfono inteligente)

CSS 3 (*Cascading Style Sheets*): es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. Es utilizado para dar estilo a documentos HTML y XML[14], separando el contenido de la presentación. Los estilos definen la forma de mostrar los elementos. CSS permite a los desarrolladores web controlar el estilo y el formato de múltiples páginas web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en el CSS afectará a todas las páginas vinculadas a ese CSS en las que aparezca dicho elemento (W3C, 2016).

JavaScript: es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas, donde no es necesario compilar los programas para ejecutarlos. Por tal motivo los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios (Eguiluz, 2016).

1.5. Consideraciones finales

En este capítulo se han abordado los elementos teóricos que dan sustento a la propuesta de solución del problema planteado, en tal sentido se puede arribar a las siguientes consideraciones:

- ✓ Con la definición de los principales conceptos se logra una mejor comprensión de los elementos teóricos que se abordan en la investigación.
- ✓ El análisis de la propuesta del ITIL para la implementación de catálogos de servicios de TI da la posibilidad de conocer los elementos que deben estar presentes en la solución.

[14] Es un meta-lenguaje que nos permite definir lenguajes de marcado.

- ✓ De las soluciones estudiadas todas cuentan un con catálogo de servicios que comprende los planes de pago, los límites de almacenamiento gratuito y las formas de contratación del servicio, lo que permitió identificar funcionalidades y características que deberían de estar presentes en la propuesta de solución.

CAPÍTULO 2. ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DEL CATÁLOGO DE SERVICIOS PARA UNA NUBE PRIVADA

El presente capítulo muestra el análisis, diseño e implementación realizado. Para lograrlo se presenta la descripción de la propuesta de solución y los requisitos obtenidos durante el análisis. Además de un grupo de artefactos que ayudan a comprender el diseño y la implementación del sistema.

2.1. Modelo de dominio

Se utiliza para expresar el entendimiento ganado en el análisis y es el paso previo al diseño del sistema (Larman, 1999). Este es un medio para comprender como implementar el catálogo de servicios. La siguiente imagen muestra el modelo de dominio que expresa como se comportaba el negocio en cuestión antes de desarrollar la propuesta de solución. El cliente sería atendido por el empleado, una vez que el cliente solicite las ofertas este se las mostraría. Si el cliente se decide por algún servicio solicitaría un contrato al empleado.

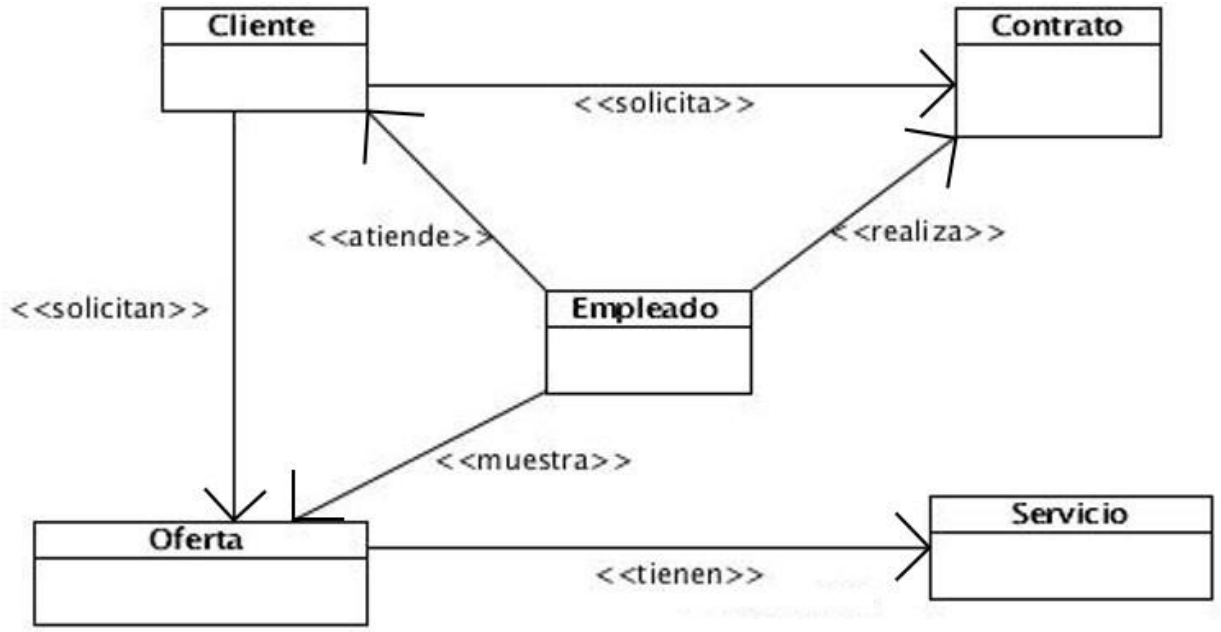


Figura 1. Diagrama del modelo de dominio de la propuesta de solución (Elaboración propia).

2.1.1. Descripción de clases del modelo del dominio

Cliente: Es la persona o entidad que solicita un servicio.

Contrato: Es la forma en la que queda registrado los acuerdos alcanzados entre las partes, es decir entre el cliente y el proveedor.

Empleado: Es quien atiende las solicitudes y representan al proveedor de servicios ante los clientes.

Oferta: Son todos aquellos servicios que se encuentran disponibles en un momento dado.

Servicios: Son los que componen las ofertas.

2.2. Descripción de la propuesta de solución

Se requiere implementar un catálogo de servicios donde serán ofertados servicios o paquetes de servicios de computación en la nube que pueden ser de cualquiera de los tres modelos de servicios: *IaaS*, *SaaS* o *PaaS*. Por otra parte, cada servicio contará con su debida descripción, tipo de modelo de servicio al que pertenece precio y por el plazo de tiempo que puede ser contratado. En la solución deben de estar presentes los precios y planes de los servicios. Entre otras opciones deben aparecer los servicios de forma ordenada y clasificados por categorías (*IaaS*, *SaaS* y *PaaS*). El cliente debe tener la opción de ver los acuerdos de nivel del servicio por cada servicio que esté disponible en el catálogo. También el cliente debe tener la posibilidad de seleccionar más de un servicio y llevar el control de la selección realizada a través del carrito de compras. El cliente, en el carrito de compras, deberá tener la opción de sacar del carrito cualquier servicio que previamente haya seleccionado. Una vez que se decida a pagar por los servicios deberá seleccionar la opción de contratar en el carrito de compra e inmediatamente se le mostrará una interfaz para que ingrese los datos de su cuenta bancaria. Es importante señalar que este interfaz solamente es una conexión entre el sitio y la pasarela de pago que no viene con dicho catálogo. Una vez rubricado el contrato el cliente deberá tener acceso a este en una lista de contratos y si desea podrá exportarlo a PDF. Por otra parte, el catálogo deberá contar con una iconografía que les permita a los clientes orientarse en él. Además, será tarea del empleado del catálogo mantener la oferta disponible en la web. El sistema deberá contar con una correcta administración para ser capaz de asignar usuarios, roles y permisos.

2.2.1. Descripción de roles del sistema

El **cliente anónimo** es aquel usuario que desea realizar consultas sobre el catálogo o los precios y no se haya registrado todavía como cliente. Sin estar registrado, a este usuario no se le permite añadir servicios al carrito. Solo podrá observar el contenido del catálogo, pero no podrá realizar solicitudes.

El **cliente registrado** es aquel usuario que ha introducido sus datos como cliente del catálogo y ya cuenta con un nombre de usuario y una contraseña. Este cliente verá su nombre en un cuadro de la parte superior derecha y la información actualizada de su carrito. Además, tendrá la opción de visualizar sus contratos. También, si así lo desea, podrá finalizar la etapa de la realización del pedido obteniendo un acuerdo sobre el o los servicios solicitados el cual deberá aceptar para finalizar su trámite y automáticamente se realizará el pago a través de la pasarela.

El **empleado** tendrá acceso a la intranet del catálogo de servicios para realizar las operaciones y gestiones que esta facilita.

El **administrador** es el encargado de la administración del sitio, así como listar, bloquear, desbloquear y modificar usuarios, también podrá tener acceso a la intranet del catálogo y podrá realizar las operaciones de gestión que esta facilita.

2.3. Técnica de obtención de requisitos

Para la implementación de la propuesta de solución se llevó a cabo la siguiente técnica de obtención de requisitos:

Observación de sistemas semejantes: se realizó un estudio de soluciones semejantes, lo que permitió obtener una base de propuestas de requisitos que sirvió para el desarrollo de la solución en la presente investigación.

2.4. Levantamiento de requisitos funcionales y no funcionales de software

2.4.1. Definición de los requisitos funcionales de software

Tabla 1. Listado de requisitos funcionales de software (Elaboración propia).

Nº	Nombre
RF1	Registrar usuario
RF2	Autenticar usuario
RF3	Modificar usuario
RF4	Listar usuarios
RF5	Buscar usuario por el nombre de usuario
RF6	Bloquear usuario

RF7	Eliminar usuario
RF8	Consultar servicios según categoría
RF9	Consultar los detalles de un servicio
RF10	Consultar servicios recientes
RF11	Consultar carrito
RF12	Añadir servicio al carrito
RF13	Eliminar servicio del carrito
RF14	Guardar contrato en la base de datos
RF15	Mostrar contratos
RF16	Buscar contratos por cliente
RF17	Cancelar contrato
RF18	Mostrar información de los usuarios
RF19	Asignar roles y permisos a los usuarios
RF20	Mostrar estados de los contratos
RF21	Modificar servicio
RF22	Mostrar información de los empleados
RF23	Guardar servicio en la base de datos
RF24	Exportar contrato como PDF

2.4.2. Historias de usuarios

Según (Letelier, y otros, 2006) las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software. El tratamiento de las historias de usuario es muy dinámico y flexible, en cualquier momento estas pueden romperse, reemplazarse por otras más específicas o generales, añadirse nuevas o

ser modificadas. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas.

Además, se dice que son escritas por el cliente, en su propio lenguaje, como descripciones cortas de lo que el sistema debe realizar. La diferencia más importante entre las historias y los tradicionales documentos de especificación funcional se encuentra en el nivel de detalle requerido. Las historias de usuarios deben tener el detalle mínimo como para que los programadores puedan realizar una estimación poco riesgosa del tiempo que llevará su desarrollo. Cuando llegue el momento de la implementación, los desarrolladores dialogarán directamente con el cliente para obtener todos los detalles necesarios. Las historias de usuarios deben poder ser programadas en un tiempo entre una y tres semanas. Si la estimación es superior a tres semanas, debe ser dividida en dos o más historias. Si es menos de una semana, se debe combinar con otra historia (Joskowicz, 2008). A continuación, se presentan algunas de las historias de usuarios registradas en el proceso de desarrollo.

Tabla 2. Historia de usuario # 1 (Elaboración propia).

Historia de Usuario	
Código: RF1	**Nombre Historia de Usuario:** Registrar usuario
Referencia: RF1	
Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 2 días**
Riesgo en Desarrollo: Alta	**Tiempo Real: 2 ½ días**
Descripción: El sistema registra al usuario la primera vez que el mismo se autentique, asignándole el rol cliente registrado.	
Observaciones: El usuario tendrá acceso a todas las opciones disponibles para clientes registrados. Se creará el perfil, luego tendrá la opción de cambiar el avatar, pues por defecto el avatar será uno vacío.	
Interfaz:	

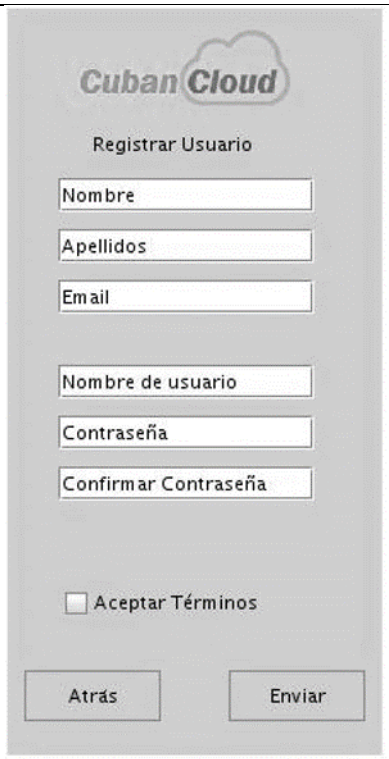


Tabla 3. Historia de usuario # 23 (Elaboración propia).

Historia de Usuario	
Código: RF23	**Nombre Historia de Usuario:** Insertar servicio en la base de datos
Referencia: RF23	
Usuario: empleado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Media	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Medio	**Tiempo Real: 1 día**
Descripción: Una vez que el empleado se ha autenticado podrá insertar servicios a la base de datos para ser publicados en el catálogo.	
Observaciones:	
Interfaz:	

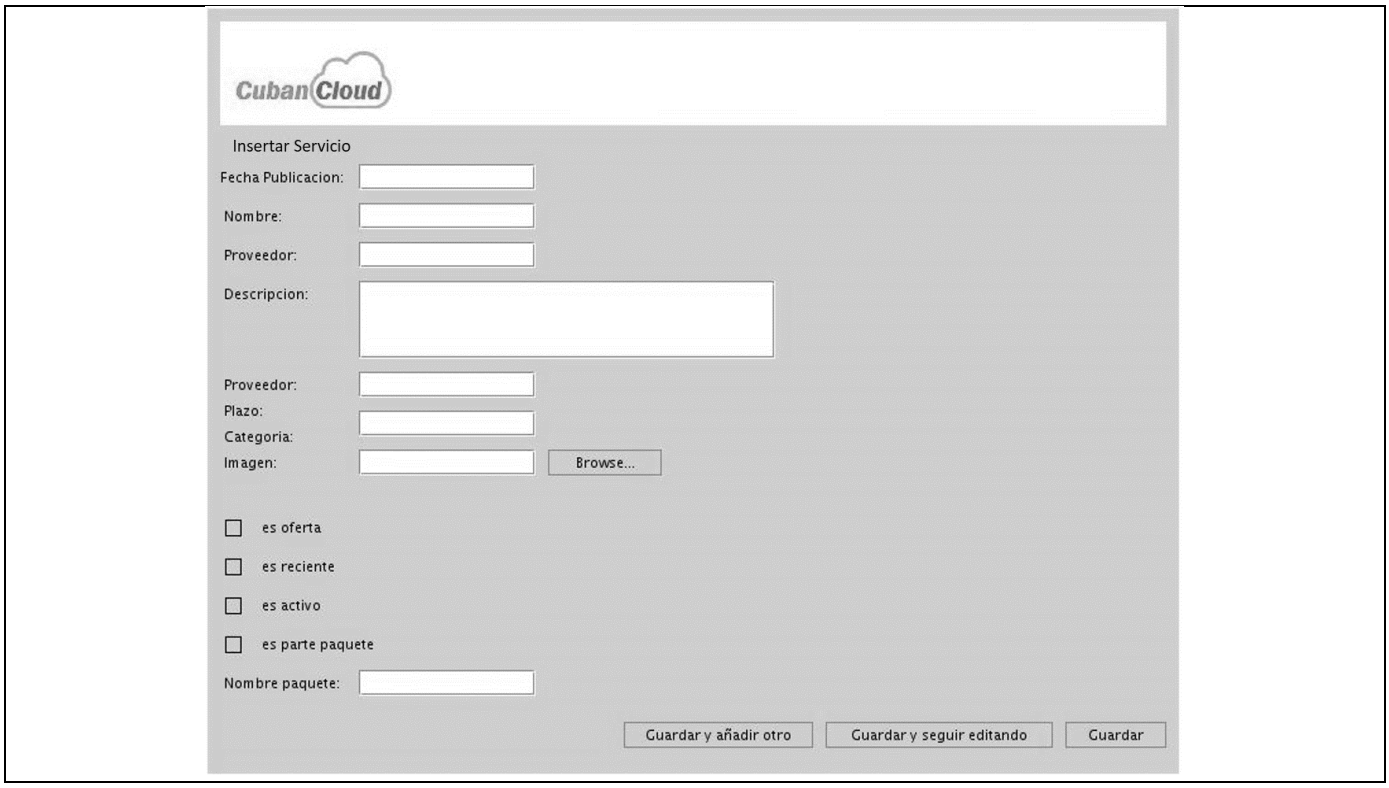


Tabla 4. Historia de usuario # 8 (Elaboración propia).

Historia de Usuario	
Código: RF8	**Nombre Historia de Usuario:** Consultar servicios según categoría
Referencia: RF8	
Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1día**
Descripción: El cliente tendrá la posibilidad de consultar los servicios según la categoría a la que pertenezcan, las que pueden ser *IaaS, SaaS y PaaS.*	
Observaciones:	
Interfaz:	

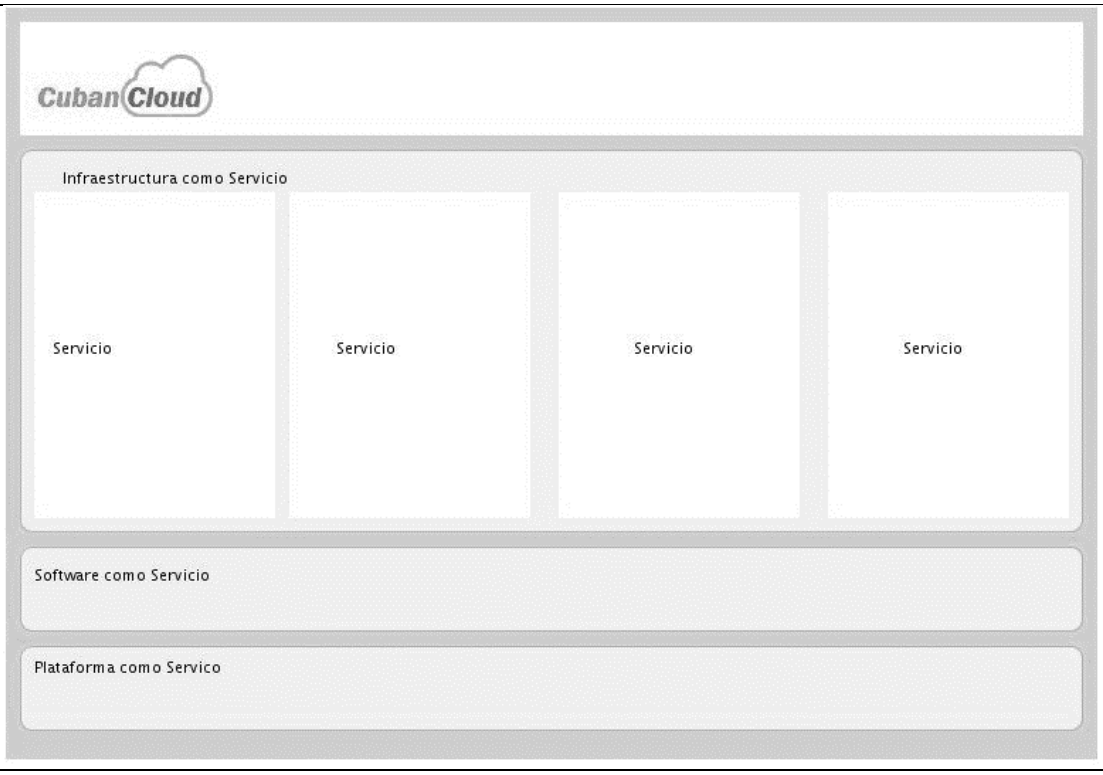


Tabla 5. Historia de usuario # 6 (Elaboración propia).

Historia de Usuario	
Código: RF6	**Nombre Historia de Usuario:** Bloquear usuario
Referencia: RF6	
Usuario: administrador	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1día**
Descripción: El usuario no podrá acceder al sistema. Este requisito incluye el desbloquearlo, para que pueda acceder.	
Observaciones: Esta acción se hará cambiando el valor de la *checkbox* Activo(*True* / *False*) siendo las opciones (*False* para bloquear y *True* para desbloquear).	
Interfaz: ☐ Activo	

2.4.3. Descripción de los requisitos no funcionales de software

Los requisitos no funcionales son propiedades que hacen al producto atractivo, usable, rápido o confiable. Se conocen como un conjunto de características de calidad, que es necesario tener en cuenta al diseñar e implementar el software (Sommerville, 2011).

Usabilidad:

- La aplicación debe estar dirigida a registrar la información referente al proceso de gestión del catálogo de servicios.
- Solo se mostrarán a los usuarios aquellas opciones o informaciones a las que su responsabilidad o rol dentro del negocio necesitan acceder mostrando en la vista los íconos y menús correspondientes.
- Los íconos deben estar representados por una imagen acorde a la acción que se está realizando mediante el mismo.

Confiabilidad:

- El sistema de autenticación recogerá datos de los usuarios y tendrá control de identidades de los mismos.
- Debe proveer algún mecanismo seguro de encriptación y transferencia de datos.
- Se debe mantener una seguridad a nivel de usuarios y contraseñas codificadas para el acceso a los servicios del catálogo.

Soporte:

- La base de datos que utilizará el sistema como medio de almacenamiento de la información estará soportada sobre un gestor de base de datos PostgreSQL.

2.5. Plan de iteraciones

Cada iteración corresponde a un periodo de tiempo de desarrollo del proyecto de entre una y tres semanas (Rodríguez Corbea, y otros, 2007). Este proyecto constará de tres iteraciones, a realizarse en un total de 7 semanas. A cada iteración se le asignó una cantidad de historias de usuarios que deberán ser implementadas.

Iteración 1: Serán implementadas historias de usuarios con prioridad alta, obteniéndose una primera versión de prueba y dotando al catálogo de sus principales funcionalidades y se realiza la primera entrega al cliente donde se decidirá cuales no conformidades pasarán a la siguiente iteración para darle solución. Debido a que hay muchas historias de usuarios con prioridad alta el cliente decidió pasar algunas historias de usuarios con prioridad alta para la segunda iteración.

Iteración 2: Se implementarán algunas historias de usuarios con prioridad alta y las de prioridad mediana, obteniéndose una segunda versión de prueba y corrigiéndose los errores obtenidos durante la primera iteración.

Iteración 3: Se implementarán las historias de usuarios con prioridad baja, obteniéndose una tercera versión que haya mitigado totalmente las no conformidades obtenidas en la segunda iteración.

Tabla 6. Plan de Iteraciones (Elaboración propia).

Iteración No.	Historias de Usuarios que se implementarán	Tiempo Asignado
1	1, 2, 3, 9, 11, 12, 13, 14, 15, 17, 18	3 semanas
2:	16, 19, 20, 21, 22, 23	2 semana
3:	4, 5, 6, 7, 8, 10, 24	2 semana

2.6. Plan de entregas

El plan de iteración consiste en seleccionar las historias de usuario que, según el plan de entregas, corresponderían a esta iteración. También se eligen qué pruebas de aceptación fallidas se corregirán (Rodríguez Corbea, y otros, 2007). En el presente plan de entregas se definen las fechas de cierre de cada iteración, donde se le entrega una versión del producto al cliente, con el propósito de entregar en tiempo el producto final. La implementación comenzó el 13 de marzo de 2017.

Tabla 7. Plan de entregas (Elaboración propia).

Iteración	Fecha de entrega
1	3/04/2017
2	17/04/2017
3	01/05/2017

2.7. Arquitectura de la propuesta de solución

2.7.1. Arquitectura Cliente-Servidor

Es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, que le da respuesta (Paszniuk, 2013).

2.7.2. Patrón arquitectónico

Los patrones arquitectónicos son los que definen la estructura de un sistema, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo, con el objetivo de facilitar la tarea del diseño. El patrón que implementa *Django* es el Modelo Vista Plantilla que es semejante al Modelo Vista Controlador solo que al controlador se le llama vista y a la vista plantilla (Montero, 2012).

El **modelo** define los datos almacenados, se encuentra en forma de clases de *Python*, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos.

La **vista** se presenta en forma de funciones en *Python* y su propósito es determinar los datos que serán visualizados. El ORM[15] de *Django* permite escribir código *Python* en lugar de *SQL* para hacer las consultas que necesita la vista. La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. La vista sólo se encarga de los datos, la presentación es tarea de la plantilla.

La **plantilla** es la encargada de realizar la presentación de los datos en el navegador web. En ella no solo se puede crear contenido HTML además de algunas etiquetas extras que implementa *Django*, también se crea contenido XML, CSS, JavaScript, entre otros.

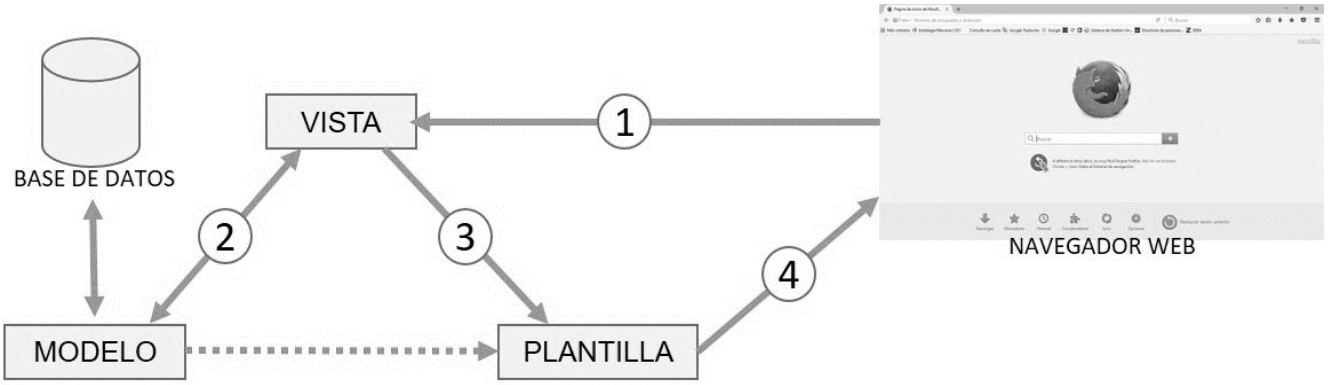


Figura 2. Funcionamiento de MVT de Django (Elaboración propia).

1. El navegador envía una solicitud.

2. La vista interactúa con el modelo para obtener datos.

3. La vista llama a la plantilla.

4. La plantilla renderiza la respuesta a la solicitud del navegador (Ayala, 2012).

[15] Mapeo Objeto Relacional, en inglés *Object Relational Mapping:* Es una capa intermedia que hace la traducción entre el lenguaje que entiende la base de datos SQL, es decir, relacional, y el lenguaje de programación, orientado a objetos.

2.8. Patrones de diseño

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos. El concepto de patrones de diseño fue el resultado de un trabajo realizado por un grupo de cuatro personas (*Erich Gamma, Richard Helm, Ralph Johnson* y *John Vlissides*, conocidos como "la pandilla de los cuatro") que se publicó en 1995 en un libro titulado "Patrones de diseño: Elementos de software orientado a objetos reutilizables" en el que se esbozaban veintitrés patrones de diseño. Un patrón de diseño puede considerarse como un documento que define una estructura de clases que aborda una situación particular (CCM, 2017).

2.8.1. Patrones GRASP (Responsability Assignment Software Patterns)

Los patrones GRASP se usan para la asignación de responsabilidades, también son buenas prácticas que se enfocan en lograr un diseño refinado de software (Kord, 2011). En la propuesta de solución se implementaron los siguientes patrones.

Experto en información: define la asignación de responsabilidades. Se manifiesta en aquella clase que tiene la información necesaria para realizar una responsabilidad obteniéndose un diseño con mayor cohesión y menor acoplamiento (Larman, 1999).

```
@login_required(login_url=LOGIN_URL)
def tienda(request):
    precompras = Precompra.objects.filter(id_user=request.user.pk)
    serviciosPaaS = Servicio.objects.filter(categoria='PaaS')
    serviciosIaaS = Servicio.objects.filter(categoria='IaaS')
    serviciosSaaS = Servicio.objects.filter(categoria='SaaS')
    return render(request, 'tienda.html', locals())
```

Figura 3. Segmento de código de fichero view.py donde se hace uso del patrón Experto en información (Elaboración propia).

Controlador: es un objeto que no pertenece a la interfaz de usuario, es responsable de recibir o manejar un evento del sistema. Un controlador define el método para la operación del sistema (Larman, 1999). En el patrón arquitectónico Modelo-Vista-Plantilla la Vista hace función de controlador. Dado que las vistas están programadas en forma de funciones, se aprecia este patrón cuando una función retorna un objeto de tipo *render.*

```
@login_required(login_url=LOGIN_URL)
def tienda(request):
    precompras = Precompra.objects.filter(id_user=request.user.pk)
    serviciosPaaS = Servicio.objects.filter(categoria='PaaS')
    serviciosIaaS = Servicio.objects.filter(categoria='IaaS')
    serviciosSaaS = Servicio.objects.filter(categoria='SaaS')
    return render(request, 'tienda.html', locals())
```

Figura 4. Segmento de código de fichero view.py donde se hace uso del patrón Controlador (Elaboración propia).

Creador: este patrón está presente siempre que una clase tenga la responsabilidad de crear una instancia de otra (Larman, 1999). En la solución está implícito en varias clases que pertenecen al marco de trabajo, por ejemplo, la clase *SimpleDocTemplate.*

```
    def build(self,flowables,onFirstPage=_doNothing, onLaterPages=_doNothing, canvasmaker=canvas.Canvas):
        """build the document using the flowables.  Annotate the first page using the onFirstPage
               function and later pages using the onLaterPages function.  The onXXX pages should follow
               the signature

                  def myOnFirstPage(canvas, document):
                      # do annotations and modify the document
                      ...

               The functions can do things like draw logos, page numbers,
               footers, etcetera. They can use external variables to vary
               the look (for example providing page numbering or section names).
        """
        self._calc()    #in case we changed margins sizes etc
        frameT = Frame(self.leftMargin, self.bottomMargin, self.width, self.height, id='normal')
        self.addPageTemplates([PageTemplate(id='First',frames=frameT, onPage=onFirstPage,pagesize=self.pagesize),
                               PageTemplate(id='Later',frames=frameT, onPage=onLaterPages,pagesize=self.pagesize)])
        if onFirstPage is _doNothing and hasattr(self,'onFirstPage'):
            self.pageTemplates[0].beforeDrawPage = self.onFirstPage
        if onLaterPages is _doNothing and hasattr(self,'onLaterPages'):
            self.pageTemplates[1].beforeDrawPage = self.onLaterPages
        BaseDocTemplate.build(self,flowables, canvasmaker=canvasmaker)
```

Figura 5. La clase SimpleDocTemplate implementa el patrón Creador (Elaboración propia).

Bajo acoplamiento: este patrón tiene como objetivo tener las clases lo menos relacionadas posibles logrando tener la menor repercusión posible en el resto de las clases en caso de que ocurra una modificación en una de ellas, de esta manera se disminuye la dependencia entre las clases y se logra una mayor reutilización de código (Johnson, y otros, 1995). Este patrón ya viene incluido con *Django*, lo que evita las

dependencias, por ejemplo, cuando se realizan cambios en las configuraciones de las URL[16], la base de datos, plantillas HTML y las vistas, basta solo con realizar dichos cambios una sola vez (García, 2015).

2.8.2. Patrones GoF

Los patrones GoF[17] proporcionan una solución programable con su propio diagrama de clases que muestra la forma en que se debe usar (Kord, 2011). Además, están agrupados en tres categorías: Creación, Estructural y Comportamiento (Johnson, y otros, 1995).

Decorador*:* Extiende la funcionalidad de un objeto dinámicamente de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Provee una alternativa muy flexible para agregar funcionalidad a una clase (Larman, 1999). En la propuesta de solución este es empleado en los documentos HTML para reutilizar código, y además está presente en aquellos métodos del fichero views.py donde se requiere estar autenticado por lo cual se usó *@login_required* que es uno de los decoradores de *Django*.

```
@login_required(login_url=LOGIN_URL)
def ayuda(request):
    return render(request, 'ayuda.html')
```

Figura 6. Segmento de código del fichero views.py donde se hace uso del patrón decorador (Elaboración propia).

2.9. Modelo de datos físicos

Un modelo de datos es una representación abstracta de los datos de una organización y las relaciones entre ellos. Más aún, podemos decir que, en cierta medida, un modelo de datos describe una organización. El propósito de un modelo de datos es, por una parte, representar los datos y, por otra, ser comprensible (Salazar, 2012).

[16] Siglas en inglés para el término: *Uniform Resource Locator.*
[17] Siglas en inglés para el término: *Gang of Four*

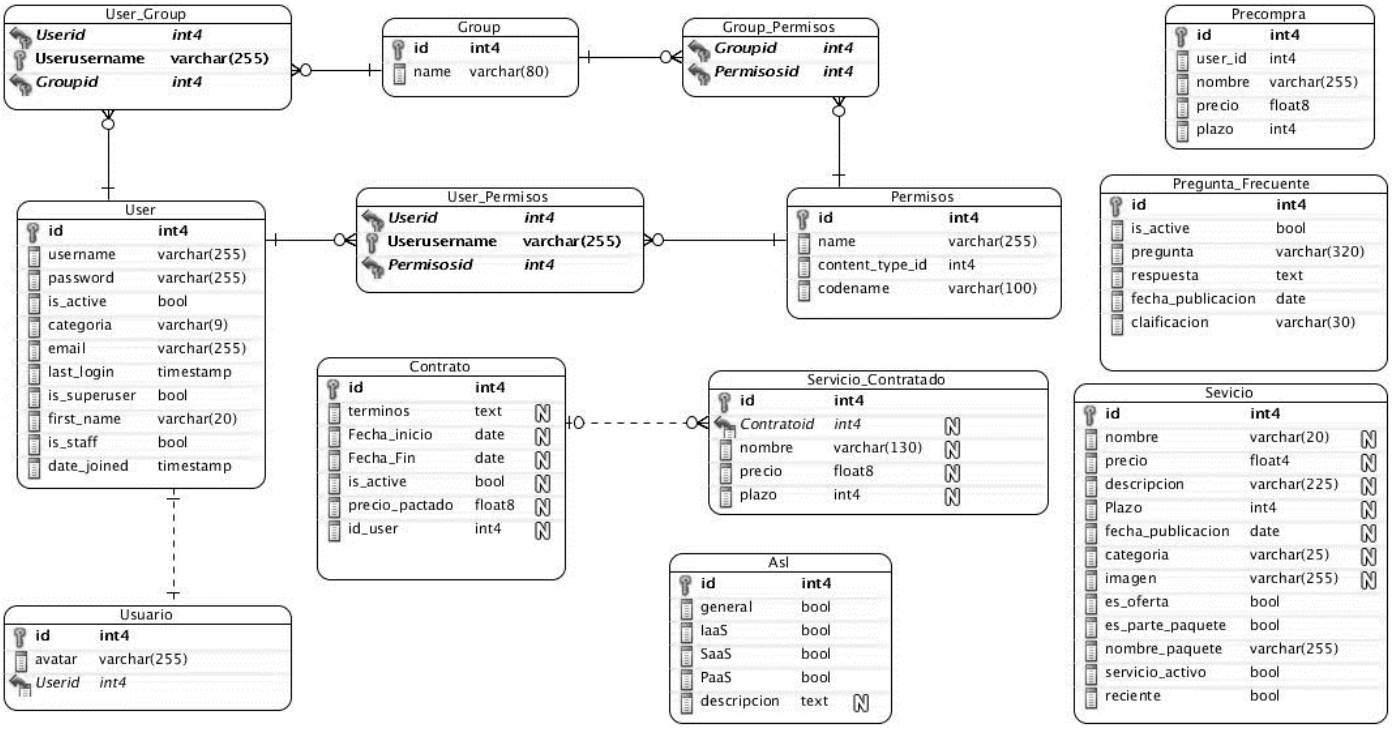


Figura 7 . Modelo de datos físicos (Elaboración propia).

2.10. Modelo de despliegue

Un diagrama de despliegue consiste en un diagrama estructurado que muestra la arquitectura del sistema desde el punto de vista del despliegue o distribución de los artefactos del software en los destinos de despliegue; además define a los artefactos como representaciones de elementos concretos en el mundo físico que son el resultado de un proceso de desarrollo. Ejemplos de artefactos son archivos ejecutables, bibliotecas, archivos, esquemas de base de datos, archivos de configuración, etc. (Sarmiento, 2013). Cuando se refiere a destino de despliegue se hace referencia a un nodo que es o bien un dispositivo de hardware o bien un entorno de ejecución de software.

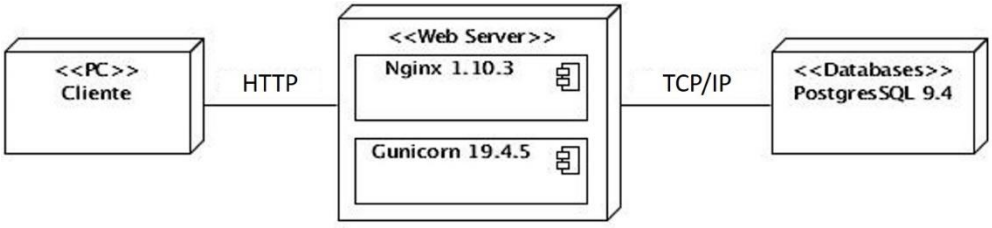


Figura 8. Modelo de Despliegue (Elaboración propia).

2.10.1. Descripción de los elementos de interface y comunicación

HTTP: Protocolo para establecer la solicitud y transmisión de archivos a través de una red de datos, especialmente páginas web y sus componentes.

TCP/IP: Protocolos para establecer la conexión entre el servidor de aplicaciones y el servidor de base de datos a través del puerto definido para el gestor de base de datos *PostgreSQL*. La conexión entre estos servidores permitirá ejecutar un conjunto de órdenes y obtener rápidamente respuesta a las mismas.

2.11. Estándares de codificación

La metodología XP enfatiza en la comunicación de los programadores a través del código, por lo que es indispensable que se sigan estándares de programación para mantener el código legible entre los miembros del equipo de desarrollo para facilitar los cambios que se puedan presentar (Rodríguez Corbea, y otros, 2007).

Tabla 8. Estándares de codificación (Elaboración propia).

Estándar	Descripción
Indentación	- Usa 4 espacios por cada nivel de indentación. - Para nuevos proyectos, se recomienda firmemente el uso de espacios en lugar de tabuladores. La mayoría de los editores tienen características que hacen esto bastante sencillo. - Las líneas de continuación deben alinearse verticalmente con el carácter que se ha utilizado (paréntesis, llaves, corchetes).
Tamaño máximo de línea	- Todas las líneas deben estar limitadas a un máximo de 79 caracteres. - Dentro de paréntesis, corchetes o llaves se puede utilizar la continuación implícita para cortar las líneas largas. - En cualquier circunstancia se puede utilizar el carácter "\" para cortar las líneas largas.
Líneas en blanco	- Separa las funciones no anidadas y las definiciones de clases con dos líneas en blanco.

	- Las definiciones de métodos dentro de una misma clase se separan con una línea en blanco. - Se puede utilizar líneas en blanco escasamente para separar secciones lógicas.
Codificaciones	- Utilizar la codificación UTF-8. - Se pueden incluir cadenas que no correspondan a esta codificación utilizando “\x”, “\u” o “\U”.
Importaciones	- Las importaciones deben estar en líneas separadas. - Siempre deben colocarse al comienzo del archivo. - Deben quedar agrupadas de la siguiente forma: 1. Importaciones de la librería estándar. 2. Importaciones terceras relacionadas. 3. Importaciones locales de la aplicación / librerías. - Cada grupo de importaciones debe estar separado por una línea en blanco.
Espacios en blanco en expresiones y sentencias	- Evitar utilizar espacios en blanco en las siguientes situaciones: ▪ Inmediatamente dentro de paréntesis, corchetes y llaves. ▪ Inmediatamente antes de una coma, un punto y coma o dos puntos. ▪ Inmediatamente antes del paréntesis que comienza la lista de argumentos en la llamada a una función. ▪ Inmediatamente antes de un corchete que empieza una indexación. ▪ Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro. - Deben rodearse con exactamente un espacio los siguientes operadores binarios: asignación (=), asignación aumentada (+=, -= etc.), comparación (==, <, >, !=, <>, <=, >=, in, not in, is, is not), booleanos (and, or, not). - No utilizar espacios alrededor del igual (=) cuando es utilizado para indicar un argumento de una función o un parámetro con un valor por defecto.

Comentarios	- Los comentarios deben ser oraciones completas. - Si un comentario es una frase u oración su primera palabra debe comenzar con mayúscula a menos que sea un identificador que comience con minúscula. - Nunca cambiar las minúsculas y mayúsculas en los identificadores de clases, objetos, funciones, etc. - Si un comentario es corto el punto final puede omitirse. Los comentarios de bloque generalmente consisten en uno o más párrafos construidos con frases completas, y cada frase debería terminar con un punto
Comentarios en bloque	- Deben estar indentados al mismo nivel que el código a comentar. - Cada línea de un comentario en bloque comienza con un numeral (#) y un espacio en blanco.
Comentarios en línea	- Se recomienda utilizarlos escasamente. - Se debe definir comenzando por un numeral (#) seguido de un espacio en blanco. - Deben ubicarse en la misma línea que se desea comentar.
Cadenas de documentación	- Deben quedar documentados todos los módulos, funciones, clases y métodos públicos. - Para definir una cadena de documentación debe quedar encerrada dentro de (“””). - Los (“””) que finalizan una cadena de documentación deben quedar en una línea a no ser que la cadena sea de una sola línea.
Convenciones de nombramiento	- Nunca se deben utilizar como simple caracteres para nombres de variables los caracteres ele minúscula “l”, o mayúscula “O”, ele mayúscula “L” ya que en algunas fuentes son indistinguibles de los números uno (1) y cero (0). - Los módulos deben tener un nombre corto y en minúscula.

	- Los nombres de clases al igual que los nombres de excepciones deben utilizar la convención "*CapWords*" (palabras que comienzan con mayúsculas). - Los nombres de las funciones deben estar escrito en minúscula separando las palabras con un guión bajo "_". - Las constantes deben quedar escritas con letras mayúsculas separando las palabras por un guión bajo (_).

2.12. Consideraciones finales

- ✓ El establecimiento de los estándares de codificación permitió un mejor entendimiento de las funcionalidades implementadas.
- ✓ Se definieron las funcionalidades que debe tener el catálogo de servicios.
- ✓ El patrón arquitectónico empleado fue el Modelo-Vista-Plantilla que implementa el marco de trabajo web.
- ✓ Los datos que son obtenidos de los procesos de gestión, promoción y contratación de los servicios deben estar debidamente almacenados por lo que se definió el modelo de datos a implementarse en la solución.
- ✓ La distribución física del sistema se estableció con el modelo de despliegue.

CAPÍTULO 3. PRUEBAS AL CATÁLOGO DE SERVICIOS PARA UNA NUBE PRIVADA

En este capítulo se hace alusión a la fase de prueba propuesta por la metodología de desarrollo la cual indica realizar pruebas unitarias y las de aceptación para comprobar el cumplimiento de los requerimientos establecidos en las historias de usuarios.

3.1. Pruebas de aceptación.

El objetivo de estas pruebas es verificar los requisitos del sistema. Estos son la principal fuente de información a la hora de construir las pruebas de aceptación, las cuales son creadas a partir de las historias de usuario, durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación.

Tiene como propósito demostrar al cliente el cumplimiento de un requisito del software. Describe un escenario (secuencia de pasos) de ejecución o uso del sistema desde la perspectiva del cliente. Puede estar asociada a requisitos funcionales o no funcionales de software. Un requisito tiene una o más pruebas de aceptación asociadas. Estas pruebas cubren desde escenarios típicos hasta los más excepcionales (Letelier, 2007).

El cliente o usuario especifica los aspectos a testear. Una historia de usuario puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar su correcto funcionamiento y no se considera completa hasta que no supera sus pruebas de aceptación. La garantía de calidad es una parte esencial en el proceso de XP. La realización de este tipo de pruebas y la publicación de los resultados debe ser los más rápido posibles, para que los desarrolladores puedan realizar con la mayor rapidez los cambios que sean necesarios.

Tabla 9. Prueba de Aceptación #1 Registrar usuario (Elaboración propia).

Código:	RF1	**H.U:**	Registrar usuario
Persona que realiza la prueba:	Roberto Cárdenas Alemán		

Descripción de la prueba:	Probar que se registra el usuario en la base de datos.
Condición de ejecución:	No estar registrado en el catálogo de servicios.
Entrada / Pasos de ejecución:	- Acceder a la opción Crear nueva cuenta - Llenar los campos requeridos para registrar un nuevo usuario

Escenarios	**Resultado esperado**	**Evaluación de la prueba**
SC1: Acceder a la opción Crear nueva cuenta.	Se muestra la vista de LOGIN del sistema, en ella debe aparecer la opción Crear nueva cuenta, al hacer clic en ella debe aparecer un nuevo formulario que es el de Registrar usuario.	Satisfactoria
SC2: Introducir la información correcta.	Se introduce la información correcta al aceptar los términos y condiciones de uso se hace clic en el botón Aceptar y se debe pasar a la vista inicial del LOGIN.	Satisfactoria
SC3: Introducir la información incorrecta o dejar campos obligatorios vacíos.	Al igual que en el **SC2** lo que introduciendo la información de forma incorrecta y dejando campos vacíos al dar clic en el botón Aceptar se debe mostrar un mensaje diciendo que existen errores en los datos y campos vacíos y debe mantenerse en la misma vista para volver a intentarlo.	Satisfactoria

Tabla 10. Prueba de Aceptación #2 Autenticar usuario (Elaboración propia).

Código:	RF2	**H.U:**	Autenticar usuario
Persona que realiza la prueba:	Roberto Cárdenas Alemán		
Descripción de la prueba:	Probar que el usuario se autentica correctamente		
Condición de ejecución:	Estar registrado en el catálogo de servicios.		
Entrada / Pasos de ejecución:	- Acceder a la vista del LOGIN - Llenar los campos requeridos - Hacer clic en el botón LOGIN		

Escenarios	**Resultado esperado**	**Evaluación de la prueba**
SC1: Acceder a la vista del Login	Se muestra la vista de LOGIN del sistema, en ella debe aparecer los cuadros de textos necesarios para autenticarse.	Satisfactoria
SC2: Introducir la información correcta.	Se introduce la información correcta, al hacer clic en el botón LOGIN se pasa a la portada del sitio.	Satisfactoria
SC3: Introducir la información incorrecta o dejar campos obligatorios vacíos.	Al igual que en el **SC2** lo que introduciendo la información de forma incorrecta o dejando campos vacíos al dar clic en el botón LOGIN se debe mostrar un mensaje diciendo que existen errores	Satisfactoria

	en los datos y campos vacíos y debe mantenerse en la misma vista para volver a intentarlo.	

Tabla 11. Prueba de Aceptación #12 Añadir servicio al carrito (Elaboración propia).

Código:	RF12	**H.U:**	Añadir servicio al carrito
Persona que realiza la prueba:	Roberto Cárdenas Alemán		
Descripción de la prueba:	Probar que los servicios se añadan correctamente al carrito de compras.		
Condición de ejecución:	Estar registrado en el catálogo de servicios y tener servicios activos en la base de datos.		
Entrada / Pasos de ejecución:	- Acceder a la Tienda de servicios. - Hacer clic en el botón comprar del servicio en cuestión - Confirmar que se desea añadir dicho servicio al carrito haciendo clic en el botón añadir.		

Escenarios	**Resultado esperado**	**Evaluación de la prueba**
SC1: Acceder a la vista Tienda de servicios.	Se muestra la vista de la Tienda de Servicios con los servicios y sus atributos publicados correctamente según su categoría que pueden ser *IaaS*, *SaaS* o *PaaS*.	Satisfactoria

SC2: Seleccionar el servicio deseado.	Se muestra un *Modal-Dialog* que indica que el cliente tiene que confirmar su selección haciendo clic en el botón Añadir.	Satisfactoria
SC3: Cancelar la selección.	El cuadro de dialogo debe cerrarse al hacer clic en el botón Cancelar o fuera *Modal-Dialog*.	Satisfactoria

Tabla 12. Prueba de Aceptación #8 Consultar según categoría (Elaboración propia).

Código:	RF8	**H.U:**	Consultar servicios según categoría
Persona que realiza la prueba:	Roberto Cárdenas Alemán		
Descripción de la prueba:	Probar que los servicios se muestren correctamente.		
Condición de ejecución:	Estar registrado en el catálogo de servicios y tener servicios activos en la base de datos.		
Entrada / Pasos de ejecución:	- Acceder a la Tienda de servicios.		
Escenarios	**Resultado esperado**		**Evaluación de la prueba**
SC1: Acceder a la vista Tienda de servicios.	Se muestra la vista de la Tienda de Servicios con los servicios y sus atributos publicados correctamente según su categoría que pueden ser *IaaS, SaaS* o *PaaS*.		Satisfactoria

3.2. Pruebas unitarias

Las pruebas unitarias se emplean para probar el buen funcionamiento de un módulo o parte del sistema, para asegurar su correcto funcionamiento de todos los módulos o partes por separado y evitar errores futuros en el momento de la integración de todas sus partes.

Una de las actividades que conforma la fase de pruebas en XP es la refactorización, constante actividad de reestructuración del código. Dicha actividad tiene como objetivo remover la duplicidad, mejorar la legibilidad, simplificar y hacer más flexible la codificación, para facilitar los posteriores cambios (Rodríguez Corbea, y otros, 2007).

Cada tarea que se identificó con las historias de usuario, representa una característica distinta del sistema y se realiza una prueba de unidad por cada una de ellas, existen pruebas unitarias las cuales son diseñadas para probar cada uno de los métodos y clases, dichas pruebas son realizadas por los programadores (Rosado Gómez, y otros, 2012). Con el empleo del módulo TestCase del IDE de programación *Pycharm* se realizaron las pruebas unitarias, lográndose la automatización del proceso y una retroalimentación visual del estado del mismo.

3.2.1. Entorno de prueba

Tabla 13. Propiedades del entorno de prueba (Elaboración propia).

Marca de la PC :	Asus
Tipo de CPU :	DualCore Intel Celeron N2830, 2417 MHz (29 x 83)
Nombre de la placa base:	Asus X551MA Series Notebook
Chipset de la placa base:	Intel Bay Trail-M
Memoria del sistema:	3984 MB
Tipo de BIOS :	AMI (04/10/2014)

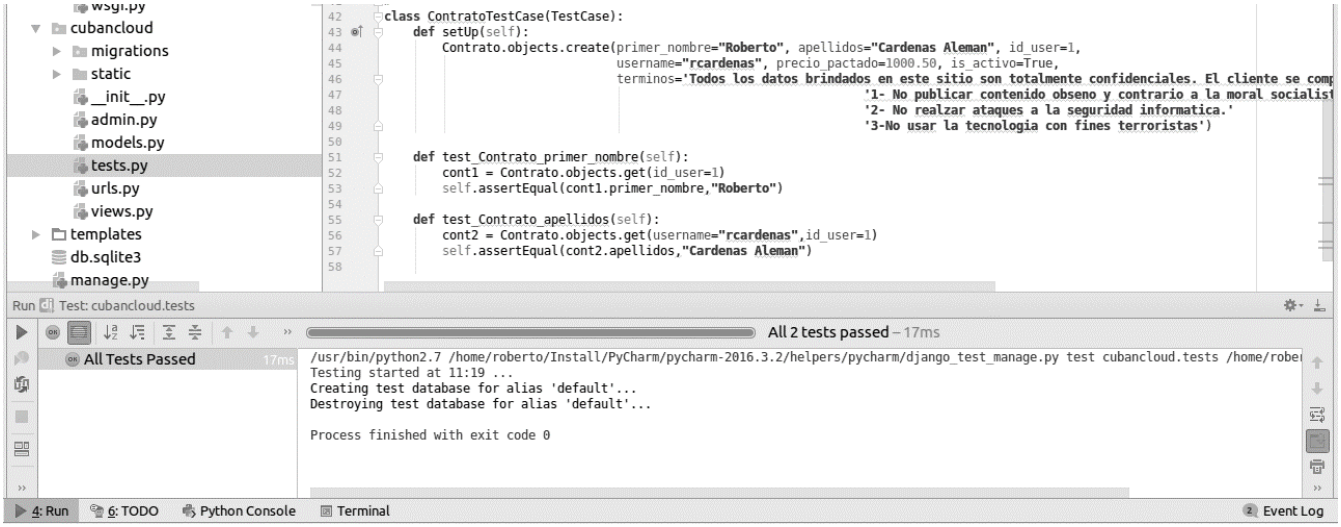


Figura 9. Prueba realizada al modelo Contrato (Elaboración propia).

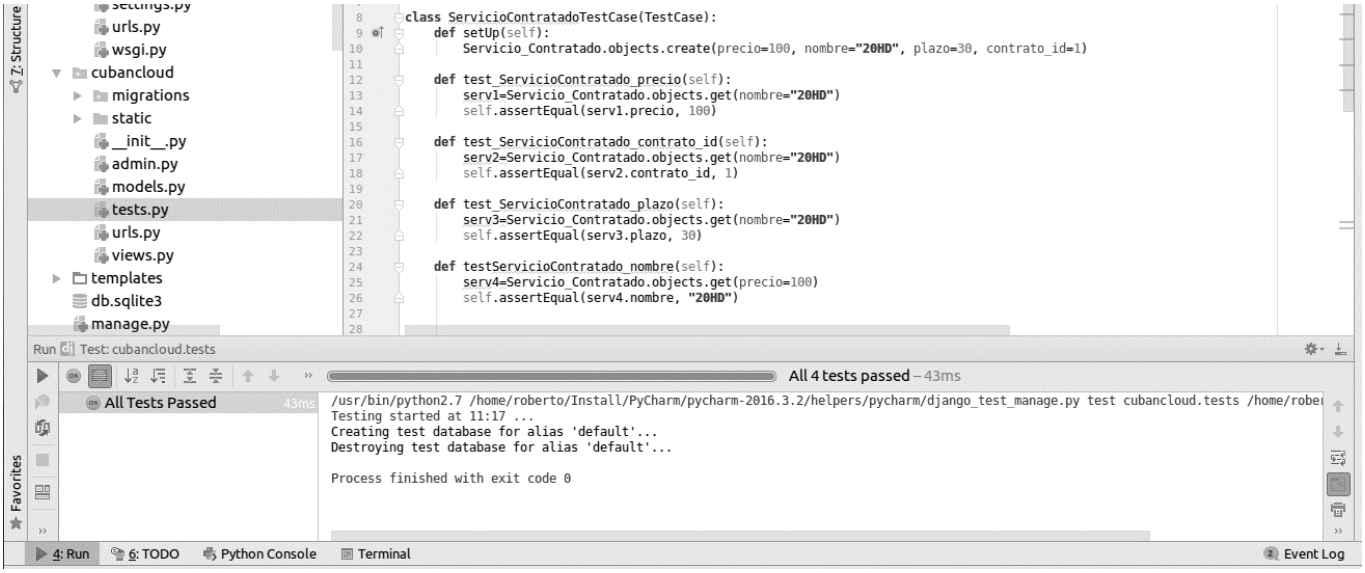


Figura 10. Prueba realizada al modelo Servicio_Contratado (Elaboración propia).

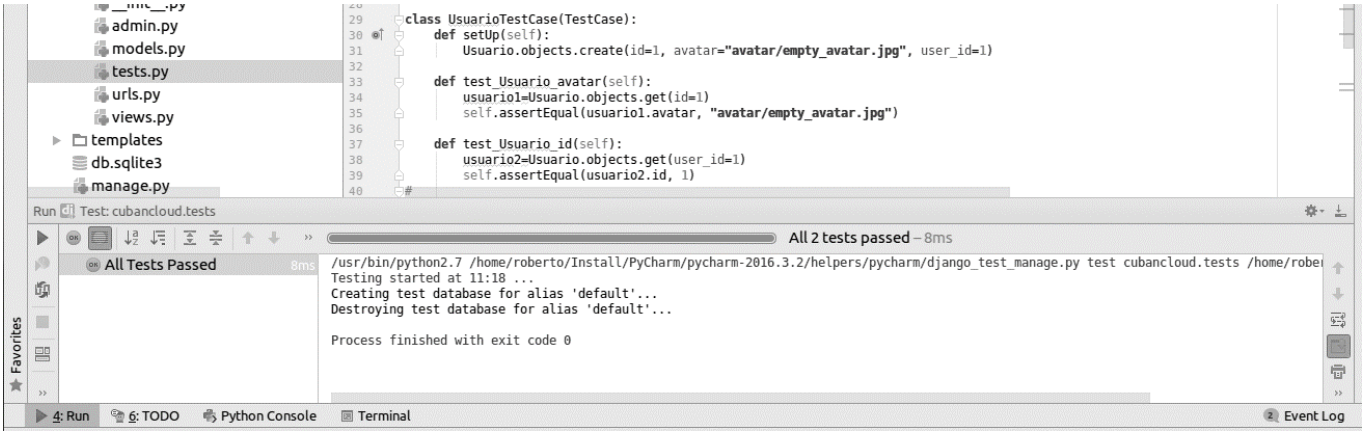


Figura 11. Prueba realizada al modelo Usuario (Elaboración propia).

3.3. No conformidades detectadas.

Entre las no conformidades detectadas se encontraban funcionalidades no deseadas por el cliente y los errores encontrados. Al final de cada iteración se le muestra al cliente una versión funcional del software de forma que pueda detectar aquellas no conformidades que serán corregidas al inicio de la subsiguiente iteración. La presente investigación está dividida en 3 iteraciones, a continuación, se listan las no conformidades encontradas en cada una de ellas.

Tabla 14. Resumen de las no conformidades por cada iteración (Elaboración propia).

Iteraciones	No conformidades
1	1- En el RF1 el campo nombre de usuario permite mayúsculas y espacios. 2- En el RF2 no aparece mensaje de error cuando se introducen datos erróneos. 3- En el RF9 no se muestra correctamente los detalles de los servicios, es necesario añadirle un *scroll*. 4- En el RF11 no aparecen los servicios añadidos al carrito. 5- En el RF11 la tabla donde se debe mostrar los servicios añadidos al carrito los colores no combinan con el resto del sistema.

2	1- En el RF20 los contratos no son mostrados correctamente. 2- En el RF20 se requiere que la opción de exportar a PDF sea mostrada luego que el usuario previamente visualiza el contrato seleccionado. 3- En el RF22 al ser modificados los requisitos se afectan los servicios contratados. 4- En el RF22 los identificadores de los botones que realizan la operación guardar deben ser cambiados de idioma.
3	1- En el RF5 en la interfaz no aparece la opción buscar. 2- En el RF8 los servicios tienen que estar separados según su categoría. 3- En el RF10 las imágenes de los servicios nuevos tienen que ser mostradas. 4- En el RF24 el documento en formato PDF es generado pero los datos aparecen desorganizados. 5- En el RF24 los datos de la tabla Servicio Contratado que deben ser mostrados en PDF aparecen de forma descendente, deben aparecer de forma ascendente. 6- En el RF24 el logo del catálogo de servicios tiene que aparecer en el PDF.

Como parte de la metodología seleccionada, las no conformidades encontradas en cada iteración son las primeras tareas a resolver de la iteración siguiente, siendo el cliente el encargado de ordenarlas por prioridad. Algunas de ellas al no ser críticas, son arrastradas a la siguiente iteración. Llevando a cabo este proceso, se logran minimizar los niveles de aceptación de errores. De esta manera quedaron resueltas las no conformidades detectadas en la aplicación desarrollada.

3.4. Consideraciones finales

- ✓ Las pruebas de aceptación permitieron comprobar la satisfacción del cliente verificando cada requisito desde la perspectiva de uso del sistema por parte del cliente.
- ✓ La realización de las pruebas unitarias permitió comprobar el correcto funcionamiento de las funcionalidades implementadas.
- ✓ La resolución de las no conformidades encontradas contribuyo a la entrega de un producto de acorde con las exigencias del cliente.

CONCLUSIONES

De modo general se concluyó con el catálogo de servicios de TI para una nube privada el cual sirve para gestionar, promocionar y contratar los servicios que se ofrecen sean de tipo *IaaS, SaaS* o *PaaS*.

Otros elementos significativos que se pueden señalar son:

- ✓ Los fundamentos teóricos permitieron adoptar una postura referente a los catálogos de servicios de TI, lo que permitió lograr una mayor comprensión del alcance de la investigación y establecer su objeto de estudio.
- ✓ La sistematización de los elementos que forman parte del marco teórico de la investigación científica y el estado actual de los catálogos de servicios, posibilitó la selección de la metodología, herramientas y tecnologías a utilizar para la implementación de la solución propuesta.
- ✓ La utilización de XP como metodología de desarrollo de software permitió guiar el proceso de desarrollo y estableció los artefactos que forman parte de la documentación del proceso ingenieril.
- ✓ La solución fue probada lo que permitió verificar el correcto funcionamiento de cada una de las historias de usuario definidas para CubanCloud.

RECOMENDACIONES

- ✓ Integrar el catálogo de servicios a la pasarela de pago.
- ✓ Añadirle un método de paginado.
- ✓ Realizar el despliegue de la aplicación empleando el protocolo HTTPS para la conexión entre la PC cliente y el servidor web.
- ✓ Integrar el catálogo de servicios a un CRM[18] con el propósito de conocer datos que permitan estudiar cómo se comporta el cliente ante las ofertas.

[18] Siglas para el término en inglés: *Customer Relationship Management.* Es un software que se encarga de estudiar el comportamiento de los clientes en sitios que se dedican a comercializar productos o servicios.

BIBLIOGRAFÍA

Alonso Martínez, Heydi, Soler Arcia, Daneidys y Reyes Hernández, Kenia. 2010. Desarrollo del Modulo de Reportes del Sistema Integrado de Gestion Bibliotecaria Koha para la Biblioteca Nacional de Cuba Jose Marti. [En línea] Junio de 2010. [Citado el: 10 de Noviembre de 2016.] https://repositorio.uci.cu/jspui/handle/ident/TD_03534_10.

Amazon Drive. 2016. [En línea] 2016. https://www.amazon.es/clouddrive/home.

Apache. 2016. [En línea] 2016. http://httpd.apache.org/docs/2.0/es/.

Arcilla, Magdalena, Calvo-Manzano, José A y Cerrada, Jose Antonio. 2012. El Catalogo de Servicios como base para la gestión Financiera en las Medianas, Pequeñas y Micro Empresas. [En línea] Julio de 2012. [Citado el: 20 de Octubre de 2016.] https://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj98_Hfqb3QAhVC4YMKHbkbAd4QFggZMAA&url=http%3A%2F%2Fwww.sapub.org%2Fglobal%2Fshowpaperpdf.aspx%3Fdoi%3D10.5923%2Fj.computer.20120001.05&usg=AFQjCNE-ttBcTxTikB4a3aI.

Ayala, José Luis Condori. 2012. Revista de Información, Tecnología y Sociedad. [En línea] 2012. [Citado el: 14 de Febrero de 2017.] http://www.revistasbolivianas.org.bo/scielo.php?pid=S1997-40442012000200016&script=sci_arttext. 1997-4044.

Brito, Karenny. 2009. *"SELECCIÓN DE METODOLOGÍAS DE DESARROLLO PARA APLICACIONES WEB EN LA FACULTAD DE INFORMÁTICA DE LA UNIVERSIDAD DE CIENFUEGOS.* Cienfuegos : s.n., 2009.

Cárdenas, Erick Rincón. 2006. [En línea] 1 de Enero de 2006. [Citado el: 5 de Diciembre de 2016.] https://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwifpMzrIlvTAhUI1oMKHV4GCB0QFggYMAA&url=http%3A%2F%2Fbooks.google.com.cu%2Fbooks%2Fabout%2FManual_de_derecho_de_comercio_electr%25C3%25B3ni.html%3Fid%3DUBTmcWxWsDsC.

CCM. 2017. CCM. *Comunidad Informática.* [En línea] Abril de 2017. [Citado el: 20 de Mayo de 2017.] https://www.genbetadev.com/metodologias-de-programacion/patrones-de-diseno-que-son-y-por-que-debes-usarlos.

Cordero, Jorge Luis. 2014. METODOLOGIAS AGILES, PROCESO UNIFICADO ÁGIL. [En línea] 18 de Noviembre de 2014. [Citado el: 22 de Febrero de 2017.] http://ingenieriadesoftware.mex.tl/images/18149/METODOLOGIAS%20AGILES.pdf.

DB-Engines. 2016. DB-ENGINES. [En línea] 2016. [Citado el: 24 de Septiembre de 2016.] www.db-engines.com.

Dinahosting. 2016. [En línea] 2016. https://dinahosting.com/.

Django Software Foundation. 2014. Django Project. [En línea] 19 de Octubre de 2014. http://djangoproject.org.

Django Software Fundation. 2016. Django. [En línea] 2016. [Citado el: 25 de Septiembre de 2016.] www.djangoproject.com.

Dropbox. 2016. [En línea] 2016. https://www.dropbox.com.

Duque, Raúl González. 2008. Python para todos. [En línea] 2008. [Citado el: 25 de Septiembre de 2016.] https://launchpadlibrarian.net/18980633/Python%20para%20todos.pdf.

EAE Business School. 2017. Artículo: Tipos de negocios y sus principales características. [En línea] 2017. http://www.eaeprogramas.es/empresa-familiar/tipos-de-negocios-y-sus-principales-caracteristicas.

Eguiluz, Javier. 2016. LibrosWeb:Introducción a JavaScript . [En línea] 2016. [Citado el: 20 de Enero de 2017.] http://librosweb.es/libro/javascript/capitulo_1.html.

El País. 2014. Artículo: Distintas opciones para almacenar en la nube. [En línea] 3 de Junio de 2014. http://cincodias.elpais.com/cincodias/2014/06/03/lifestyle/1401793260_123689.html.

Fernández, E. 2002. Comercio electrónico. [En línea] 2002. http://www.unipamplona.edu.co/unipamplona/portalIG/home_109/recursos/octubre2014/administracionde empresas/semestre9/11092015/comercioelectronico.pdf.

Flask. 2016. Documentación oficial:. [En línea] 2016. http://flask.pocoo.org/docs/0.12/.

Gálvez, Mario. 2015. La importancia de un buen catálogo de productos o servicios. [En línea] 2015. [Citado el: 3 de Marzo de 2017.] http://www.marquid.com/catalogo-productos-servicios/.

García Parellada, Lilia Rosa. 2014. *Propuesta de arquitectura para una Nube Privada con soporte para Infraestructura como Servicio.* La Habana : s.n., 2014.

García, Saul. 2015. Libro: Django, la guía definitiva. Desarrolla aplicaciones web de forma rápida y sencilla. [En línea] 2015. https://github.com/saul-g/El-libro-de-Django/blob/master/Libros/libro_django1.8.pdf.

Gómez Pérez, Miruleidis y Borges Crosa, Daniela . 2011. Metodología para el análisis forense de los principales incidentes de seguridad informática en la UCI. [En línea] 2011. https://repositorio.uci.cu/jspui/handle/ident/TD_04066_11.

González Duque, Raúl. 2016. *Python para todos.* 2016.

González, Edgar Jahaziel Carbajal. 2014. Negocio a Negocio. [En línea] 24 de Septiembre de 2014. [Citado el: 9 de Noviembre de 2016.] http://www.academia.edu/8693144/NEGOCIO_A_NEGOCIO1.

Google. 2016. Google Drive. [En línea] 2016. https://www.google.com/intl/es-419_ALL/drive/.

Gunicorn. 2017. [En línea] 2017. http://gunicorn.org/.

Hostinger. 2017. [En línea] 2017. https://www.hostinger.es.

HostSonny. 2016. [En línea] 2016. https://hostsonny.com/.

Hwaci. 2016. SQLite. [En línea] 2016. [Citado el: 20 de Septiembre de 2016.] www.sqlite.org/copyright.html.

Interoute. 2017. [En línea] 2017. http://www.interoute.es/.

ITIL. [En línea] http://www.w3ii.com/es/itil/service_catalogue_management.html.

Johnson, Ralph, y otros. 1995. Design Patterns. Elements of reusable object-oriented software. [En línea] 1995. https://pdfs.semanticscholar.org/254e/ada1041dd64c9635234b69481890d8558b98.pdf.

Joskowicz, José. 2008. [En línea] 2008. https://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

Kord, Maia. 2011. Patrones GRASP. Patrones GoF. Diferencia entre GRASP y GoF. [En línea] 3 de Diciembre de 2011. [Citado el: 10 de Diciembre de 2016.] https://sites.google.com/site/tuxnots/materias-de-la-facu/metodologia-de-sistemas/patronesgrasppatronesgofdiferenciaentregraspygof.

Larman, Craig. 1999. UML y Patrones. 2da Edición. [En línea] 1999. [Citado el: 20 de Enero de 2017.] http://www.academia.edu/download/32421917/PREVIEW-LIBRO-9788483229279.pdf.

Letelier, Patricio. 2007. *Pruebas de Aceptación como conductor del Proceso Software.* Valencia : Universidad Politénica de Valencia, 2007.

Letelier, Patricio y Penadés, María Carmen. 2006. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP). [En línea] 2006. http://www.cyta.com.ar/ta0502/v5n2a1.htm. 1666-1680.

Libros Web. 2016. Libros Web. [En línea] 2016. [Citado el: 10 de Diciembre de 2016.] http://librosweb.es/libro/xhtml/capitulo_2.html.

LinkedIn. 2015. The Best 10 Python Frameworks for Web Development. [En línea] 13 de Octubre de 2015. https://www.linkedin.com/pulse/best-10-python-frameworks-web-development-elyn-z-6059523132912394240.

Lucas, Jesús. 2014. Tutorial: Como instalar Nginx en Ubuntu 14.04 LTS. [En línea] 6 de Mayo de 2014. [Citado el: 10 de Abril de 2017.] https://openwebinars.net/blog/tutorial-como-instalar-nginx-en-ubuntu-14-04-lts/.

M., Saul Garcia. 2015. [En línea] 15 de Enero de 2015. http://github.com/jacobian/djangobook.com.

Mell, Peter y Grance, Tim. 2009. National Institute of Standads and Technology. [En línea] 10 de 7 de 2009. [Citado el: 10 de Noviembre de 2016.] https://www.nist.gov/sites/default/files/documents/itl/cloud/cloud-def-v15.pdf.

Mendelson, Haim. 2015. Artículo: Modelos de negocio, tecnologías de la información y la empresa del futuro. [En línea] 2015. https://www.bbvaopenmind.com/articulo/modelos-de-negocio-tecnologias-de-la-informacion-y-la-empresa-del-futuro/?fullscreen=true.

Mendoza Ariza, Jose Ricardo, y otros. 2012. Artículo: Prototipo e-Commerce B2C soportado en Cloud Computing. [En línea] 2012. https://www.educacioneningenieria.org/index.php/edi/article/viewFile/226/155. 1900-8260.

Microsoft. 2012. www.microsoft.com. [En línea] 2012. [Citado el: 10 de Noviembre de 2016.] https://www.microsoft.com/spain/virtualizacion/private/overview/default.mspx.

Mitroff, Sarah. 2016. CNET. [En línea] 1 de Febrero de 2016. [Citado el: 7 de Noviembre de 2016.] https://www.cnet.com/how-to/onedrive-dropbox-google-drive-and-box-which-cloud-storage-service-is-right-for-you/.

Montero, Sergio Infante. 2012. Maestros del Web. [En línea] Abril de 2012. [Citado el: 10 de Mayo de 2016.] http://www.maestrosdelweb.com/guias/#guias-django.

Murcia, Angélica Guzman. 2010. ArandaSoft. [En línea] Septiembre de 2010. [Citado el: 24 de febrero de 2017.] http://arandasoft.com/webcast-anteriores/catalogo_%20servicioV3-2.pdf.

NewScale. 2005. www.newScale.com. [En línea] Febrero de 2005. [Citado el: 9 de Noviembre de 2016.] http://hosteddocs.ittoolbox.com/RFF121205.pdf.

Nginx. 2017. Nginx.org. [En línea] 2017. [Citado el: 20 de Abril de 2017.] https://www.nginx.com/resources/wiki/.

Oracle Corporation. 2016. Oracle MySQL. [En línea] 2016. [Citado el: 20 de Septiembre de 2016.] www.oracle.com/mysql/index.html.

Paszniuk, Rodrigo. 2013. Programación Paraguay. [En línea] 19 de Julio de 2013. https://www.programacion.com.py/varios/arquitectura-cliente-servidor.

PgAdmin. 2016. [En línea] 2016. https://www.pgadmin.org/docs/pgadmin3/1.22/.

PostgreSQL Global Development Group. 2013. PostgreSQL. [En línea] 2013. [Citado el: 22 de Septiembre de 2016.] www.postgresql.org.es/sobre_postgresql.

Python. 2016. [En línea] 2016. https://docs.python.org/3/license.html.

Python. 2012. Archivos de Python-es:. [En línea] 2012. https://mail.python.org/pipermail//python-es/2012-June/032085.html.

Rodríguez Corbea, Maite y Ordóñez Pérez, Mayelin. 2007. *La metodología XP aplicable al software educativo en Cuba.* La Habana. UCI : s.n., 2007.

Rodríguez Sánchez, Tamara. 2015. *Metodología de desarrollo para la Actividad productiva en la UCI.* La Habana. Cuba : Universidad de las Ciencias Informáticas, 2015.

Roldán, José Ignacio Herranz. 2011. Blog Tecnología para desarrollo. [En línea] 17 de Enero de 2011. [Citado el: 13 de Abril de 2017.] https://www.paradigmadigital.com/dev/tdd-como-metodologia-de-diseno-de-software/.

Rosado Gómez, Alveiro, Quintero Duarte, Alexander y Guevara Meneses, Cesar Daniel. 2012. Revista Ingenio. [En línea] 12 de Agosto de 2012. [Citado el: 10 de Febrero de 2017.] http://revistas.ufpso.edu.co/index.php/ringenio/article/download/23/10. 2011-642X.

Salazar, Cristian. 2012. SlideShare. [En línea] 28 de Septiembre de 2012. [Citado el: 10 de Abril de 2017.] https://es.slideshare.net/csalazarc/modelo-de-datos-14506949.

Sánchez, Roberto. 2012. SlideShare. [En línea] 28 de Diciembre de 2012. [Citado el: 4 de Marzo de 2017.] https://es.slideshare.net/Inteli_SC/el-catlogo-de-servicio-de-ti-con-vista-al-negocio.

Sarmiento, Johana. 2013. [En línea] 12 de Abril de 2013. [Citado el: 21 de Febrero de 2017.] http://umldiagramadespliegue.blogspot.com/.

Schwaber, K y Sutherland, J. 2013. La guía de scrum: La guía definitiva de scrum, las reglas del juego. [En línea] 2013. http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-es.pdf.

Software como servicio: necesidades y retos en los sistemas de servicio de la Industria Cubana del Software. **Suárez Batista, Anisbert, Febles Estrada, Ailyn y Trujillo Casañola, Yaimí. 2016.** Especial, Informática 2016, La Habana : Revista Cubana de Ciencias Informáticas, 2016, Vol. 10. 2227-1899.

Sommerville, I. 2011. 2011.

SXP, metodología de desarrollo de software. **Gladys Marsi Peñalver Romero, Sergio Jesús García de la Puente, Abel Meneses Abad. 2011.** La Habana : Serie Científica de la Universidad de las Ciencias Informáticas, 2011.

Torres Castañeda, David Hazael y Guerra Zavala, Javier. 2012. Revista:Contribuciones a la Economía. [En línea] 2012. [Citado el: 6 de Noviembre de 2016.] http://www.eumed.net/ce/2012/tcgz.html. 1696-8360.

Vinent Puig, Caridad y Cornell Milián, Yoel. 2015. Módulo para monitorear el uso de software de las computadoras en la red con el sistema Gestión de Recursos de Hardware y Software. [En línea] Junio de 2015. [Citado el: 20 de Febrero de 2017.] https://repositorio.uci.cu/jspui/handle/123456789/7096.

Visual Paradigm International. 2016. Visual-Paradigm. [En línea] 2016. [Citado el: 6 de Octubre de 2016.] https://www.visual-paradigm.com/support/documents/vpuserguide/12/13/5963_visualparadi.html.

W3C. 2016. W3C. [En línea] 2016. [Citado el: 20 de Enero de 2017.] http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo.

ANEXOS

Anexo 1 Historias de Usuarios

Tabla 15. Historia de Usuario # 2 (Elaboración propia).

Historia de Usuario	
Código: RF2	**Nombre Historia de Usuario:** Autenticar usuario
Referencia: RF2	
Usuario: todos los usuarios	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 4 días**
Riesgo en Desarrollo: Alta	**Tiempo Real: 3 días**
Descripción: Permitirá a todos los usuarios entrar al sistemas una vez que se comprueban sus datos.	
Observaciones: En caso de surgir algún error el sistema deberá mostrar un mensaje indicando lo que ha sucedido.	
Interfaz: 	

Tabla 16. Historia de Usuario # 3 (Elaboración propia).

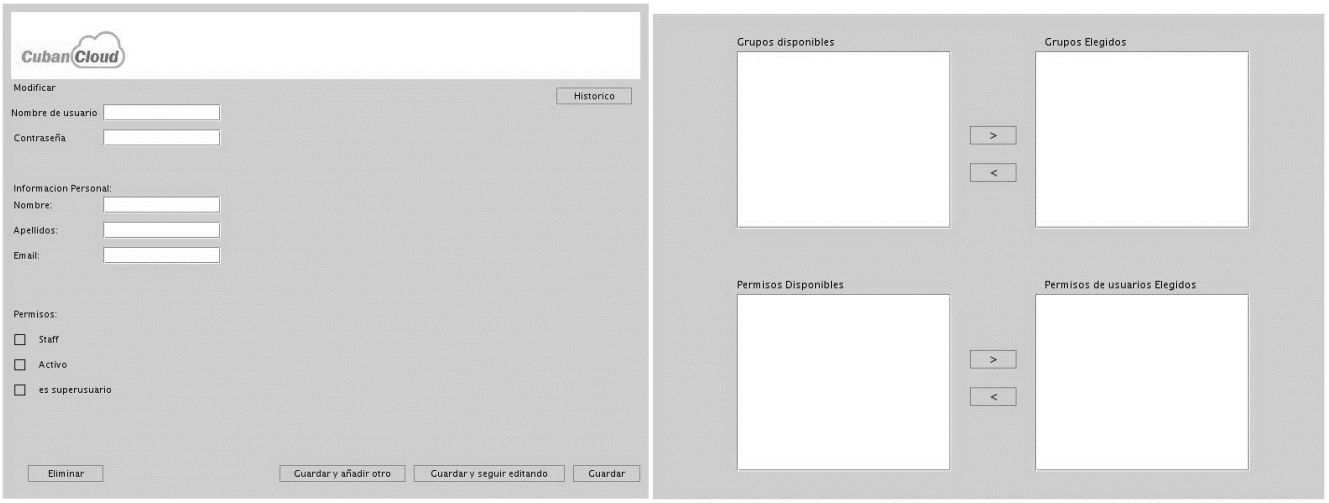
Historia de Usuario	
Código: RF3	**Nombre Historia de Usuario:** Modificar usuario
Referencia: RF3	
Usuario: administrador	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 2 días**
Riesgo en Desarrollo: Alto	**Tiempo Real: 2 días**
Descripción: Modifica los datos del usuario, lo que incluye sus roles y permisos en el sistema.	
Observaciones: Deberá ofrecer las opciones de eliminarlo, guardarlo, o guardarlo y seguir modificando otro usuario.	
Interfaz: 	

Tabla 17. Historia de Usuario # 4 (Elaboración propia).

Historia de Usuario	
Código: RF4	**Nombre Historia de Usuario:** Listar usuarios
Referencia: RF4	

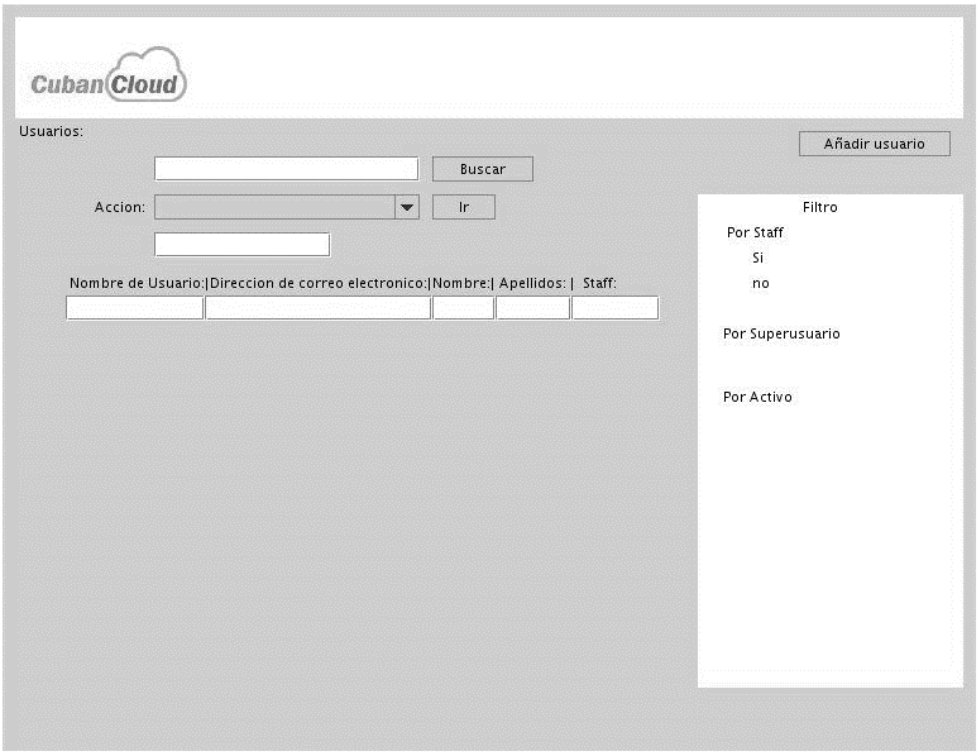
Usuario: administrador, empleado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real:1 día**
Descripción: Muestra los usuarios registrados.	
Observaciones: Aparecerá un filtro que está ubicado en el lado derecho de la lista.	
Interfaz: 	

Tabla 18. Historia de Usuario # 5 (Elaboración propia).

Historia de Usuario	
Código: RF5	**Nombre Historia de Usuario:** Buscar usuario por el nombre de usuario
Referencia: RF5	

Usuario: administrador	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1 día**
Descripción: Se realiza una búsqueda por el nombre en la lista de usuarios registrados en el sistema.	
Observaciones: Se escribe el nombre en el cuadro de texto superior y se hace clic en el botón buscar.	
Interfaz: 	

Tabla 19. Historia de Usuario # 6 (Elaboración propia).

Historia de Usuario	
Código: RF6	**Nombre Historia de Usuario:** Bloquear usuario
Referencia: RF6	
Usuario: administrador	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1día**
Descripción: El usuario no podrá acceder al sistema. Este requisito incluye el desbloquearlo, para que pueda acceder.	
Observaciones: Esta acción se hará cambiando el valor de la *checkbox* Activo(*True*/*False*) siendo las opciones (*False* para Bloquear y *True* para Desbloquear).	
Interfaz: 	

Tabla 20. Historia de Usuario # 7 (Elaboración propia).

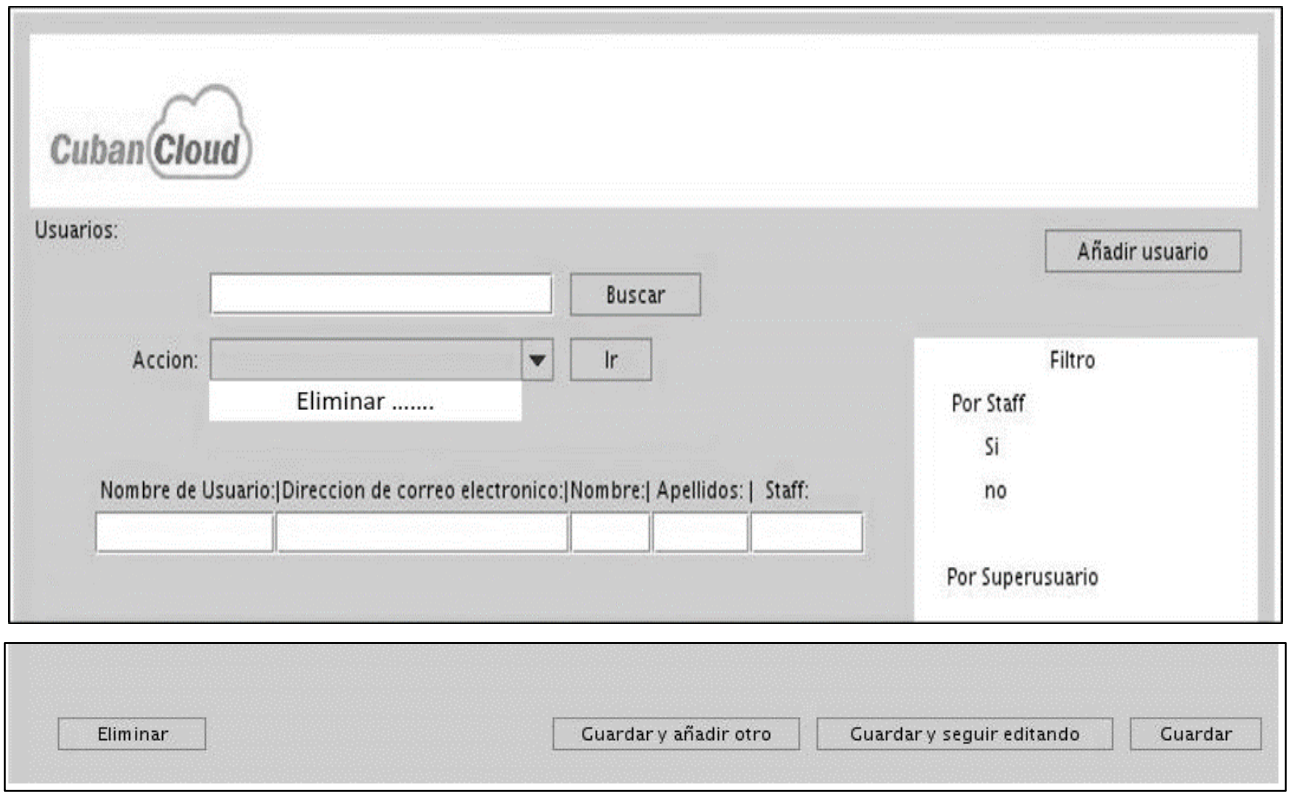
Historia de Usuario	
Código: RF7	**Nombre Historia de Usuario:** Eliminar usuario
Referencia: RF7	
Usuario: administrador	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1 día**
Descripción: Elimina a un usuario de la base de datos	
Observaciones: Existen dos formas de hacerlo, la primera es seleccionando un usuario de la lista de usuarios y en el *listbox* superior seleccionar la opción “Eliminar”, y la segunda forma es abriendo el servicio para modificarlo, al final de formulario aparece un botón “Eliminar”.	
Interfaz: 	

Tabla 21. Historia de Usuario # 8 (Elaboración propia).

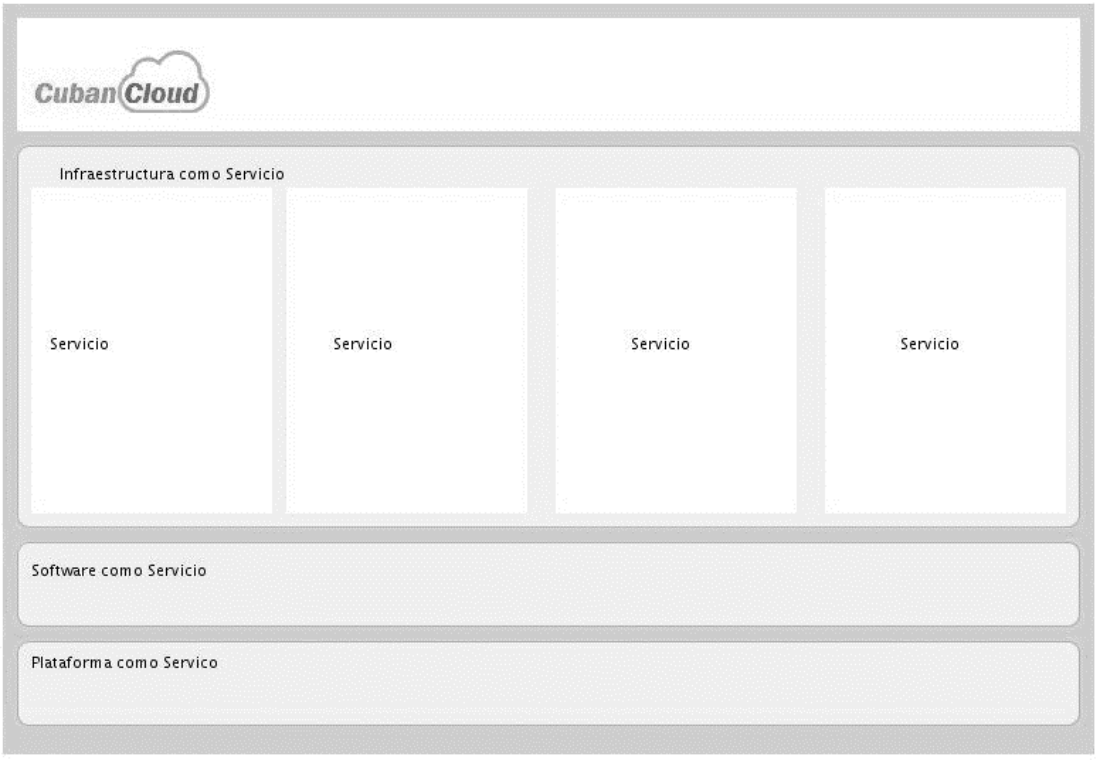
Historia de Usuario	
Código: RF8	**Nombre Historia de Usuario:** Consultar servicios según categoría
Referencia: RF8	
Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Alto	**Tiempo Real: 1día**
Descripción: El cliente tendrá la posibilidad de consultar los servicios según la categoría a la que pertenezcan, las que pueden ser *IaaS, SaaS* y *PaaS.*	
Observaciones: Estas tres categorías son las únicas que se verán en el sitio.	
Interfaz: 	

Tabla 22. Historia de Usuario # 9 (Elaboración propia).

Historia de Usuario	
Código: RF9	**Nombre Historia de Usuario:** Consultar los detalles de un servicio
Referencia: RF9	
Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Alto	**Tiempo Real: 1 ½ día**
Descripción: El cliente o usuario anónimo podrá ver una información más detallada acerca del servicio que desee. Uno de los campos que se mostraran de cada servicio será los detalles.	
Observaciones: En caso de ser extensos los detalles de un servicio debe aparecer un *scroll* para que el tamaño del cuadro del servicio no se modifique.	
Interfaz: 	

Tabla 23. Historia de Usuario # 10 (Elaboración propia).

Historia de Usuario	
Código: RF10	**Nombre Historia de Usuario:** Consultar servicios recientes
Referencia: RF10	

Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 3**
Prioridad en Negocio: Baja	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Bajo	**Tiempo Real: 1 día**
Descripción: Desde un carrusel en la página principal se irán mostrando las imágenes de todos aquellos servicios que están referenciados en la base de datos como novedad.	
Observaciones: Se podrá contratar desde la página principal haciendo clic encima del *link* que aparece con el nombre del servicio sobre la imagen.	

Tabla 24. Historia de Usuario # 11 (Elaboración propia).

Historia de Usuario	
Código: RF11	**Nombre Historia de Usuario:** Consultar carrito
Referencia: RF11	
Usuario: cliente registrado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 3 días**
Riesgo en Desarrollo: Alto	**Tiempo Real: 3 días**
Descripción: El cliente podrá consultar en todo momento y se encuentre donde se encuentre los servicios que ha seleccionado para ser contratados.	
Observaciones: Debe poder guardar un historial con los artículos que estaban en el carrito para que en caso de que ocurra algún error de conexión estos aun permanezcan en él.	
Interfaz:	

Tabla 25. Historia de Usuario # 12 (Elaboración propia).

Historia de Usuario	
Código: RF12	**Nombre Historia de Usuario:** Añadir servicio al carrito
Referencia: RF12	
Usuario: cliente registrado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 2 días**
Riesgo en Desarrollo: Alto	**Tiempo Real: 2 día**
Descripción: El usuario que entre a la página y se registre podrá añadir servicios en el carrito.	
Observación: Aparecerá un *Modal-Dialog* que indicará que se debe confirmar la opción de añadir al carrito. A medida que se van añadiendo el total a pagar va aumentando. Y la cantidad que se ve reflejada en un icono en el menú superior se va actualizando.	
Interfaz:	

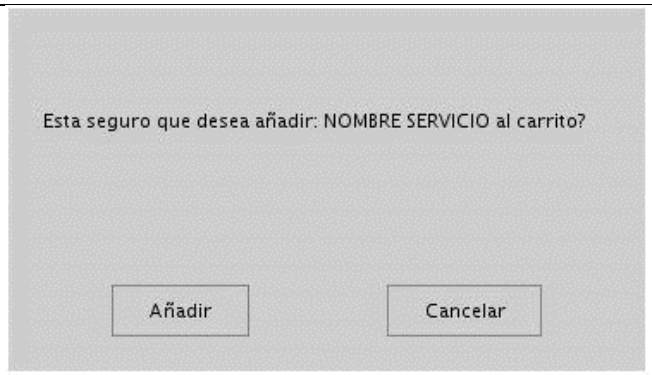


Tabla 26. Historia de Usuario # 13 (Elaboración propia).

Historia de Usuario	
Código: RF13	**Nombre Historia de Usuario:** Eliminar servicio del carrito
Referencia: RF13	
Usuario: cliente registrado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Alto	**Tiempo Real: 1 día**
Descripción: Desde la página de detalles del carrito, el cliente puede eliminar todos los servicios que desee. La tabla que muestra los servicios que hay en el carrito, dispone de la columna “Eliminar” con un botón para cada artículo.	
Observaciones: Haciendo clic sobre el botón “Eliminar”, se eliminarán todos los artículos marcados borrándolos de la variable sesión. A medida que se eliminan servicios, el total a pagar va disminuyendo.	
Interfaz: 	

Tabla 27. Historia de Usuario # 14 (Elaboración propia).

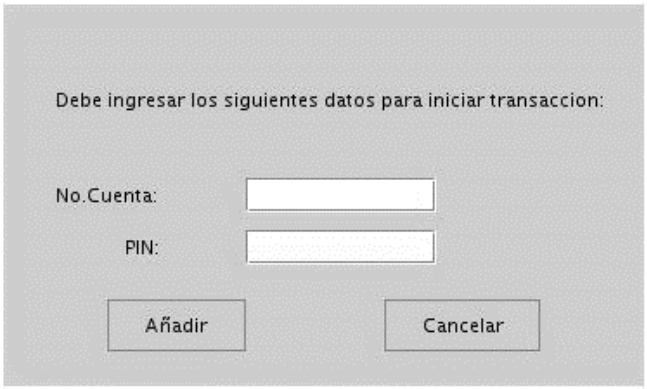
Historia de Usuario	
Código: RF14	**Nombre Historia de Usuario:** Guardar contrato en la base de datos
Referencia: RF14	
Usuario: cliente anónimo	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 2 días**
Riesgo en Desarrollo: Medio	**Tiempo Real: 1 día**
Descripción: Desde la página de detalles del carrito, el cliente puede pagar por todos los servicios seleccionados haciendo clic en el botón "pagar". Luego aparecerá un *Modal-Dialog* en el que se listan los servicios por los que se paga y se muestra el precio pactado y los acuerdos de nivel de servicio asociados a cada categoría de servicio.	
Observaciones: Para que se guarde el contrato en la Base de datos es necesario realizar el pago a través de la interfaz de la pasarela de pago.	
Interfaz: 	

Tabla 28. Historia de Usuario # 15 (Elaboración propia).

Historia de Usuario	
Código: RF15	**Nombre Historia de Usuario:** Mostrar contratos

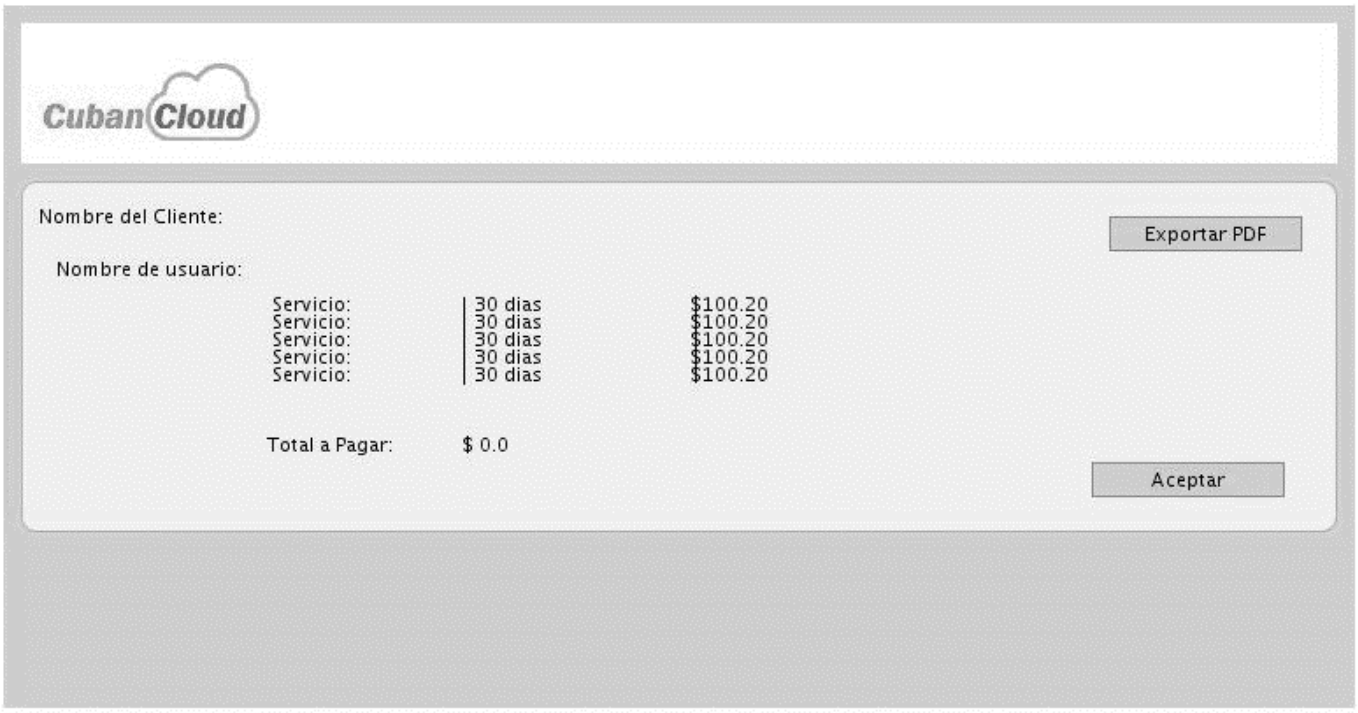
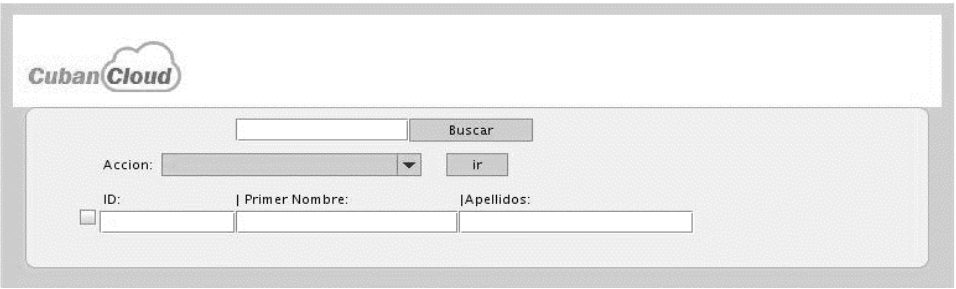
Referencia: RF15	
Usuario: cliente registrado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 1**
Prioridad en Negocio: Alta	**Tiempo Estimado: 1 día**
Riesgo en Desarrollo: Alto	**Tiempo Real: 1 ½ día**
Descripción: El cliente podrá ver en una tabla todos los contratos rubricados a través del catálogo.	
Observaciones: Al hacer clic sobre una fila de la tabla contrato se podrá visualizar el contrato en un *Modal-Dialog*.	
Interfaz: 	

Tabla 29. Historia de Usuario # 16 (Elaboración propia).

Historia de Usuario	
Código: RF16	**Nombre Historia de Usuario:** Buscar contratos por cliente

Referencia: RF16	
Usuario: empleado	
Programador: Roberto Cárdenas Alemán	**Iteración Asignada: 2**
Prioridad en Negocio: Alta	**Tiempo Estimado: 2 días**
Riesgo en Desarrollo: Medio	**Tiempo Real: 2 días**
Descripción: El empleado podrá visualizar en una tabla todos los contratos que tiene un cliente.	
Interfaz: 	

Anexo 2 Resultados en eventos científicos


UCI
Universidad de las Ciencias Informáticas

Universidad de las Ciencias Informáticas
Federación Estudiantil Universitaria Facultad1

Otorga el presente:

Reconocimiento

A: Roberto Cárdenas

Por haber participado en:
Jornada Científica Estudiantil
y obtener la categoría:
Destacado

"¿Para qué, sino para poner paz entre los hombres han de ser los adelantos de la ciencia?"
José Martí

Dado en La Habana a los 5 días del mes de mayo de 2017.

Oscar Henry Torres Caballero
Presidente FEU Facultad 1

MSc. Delly Lien González Hernández
Decana Facultad 1
FACULTAD 1

JICI

vive FEU

Jornada científica estudiantil

Figura 12. CubanCloud obtuvo destacado en la Jornada Científica (Elaboración propia)

Figura 13. CubanCloud obtuvo relevante en la 12 Peña Tecnológica. (Elaboración propia)