Universidad de las Ciencias Informáticas

Facultad 1

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

Título:

Módulo Ventas del Sistema de Gestión para la Agencia de Renta de Vehículos REX

Autores:

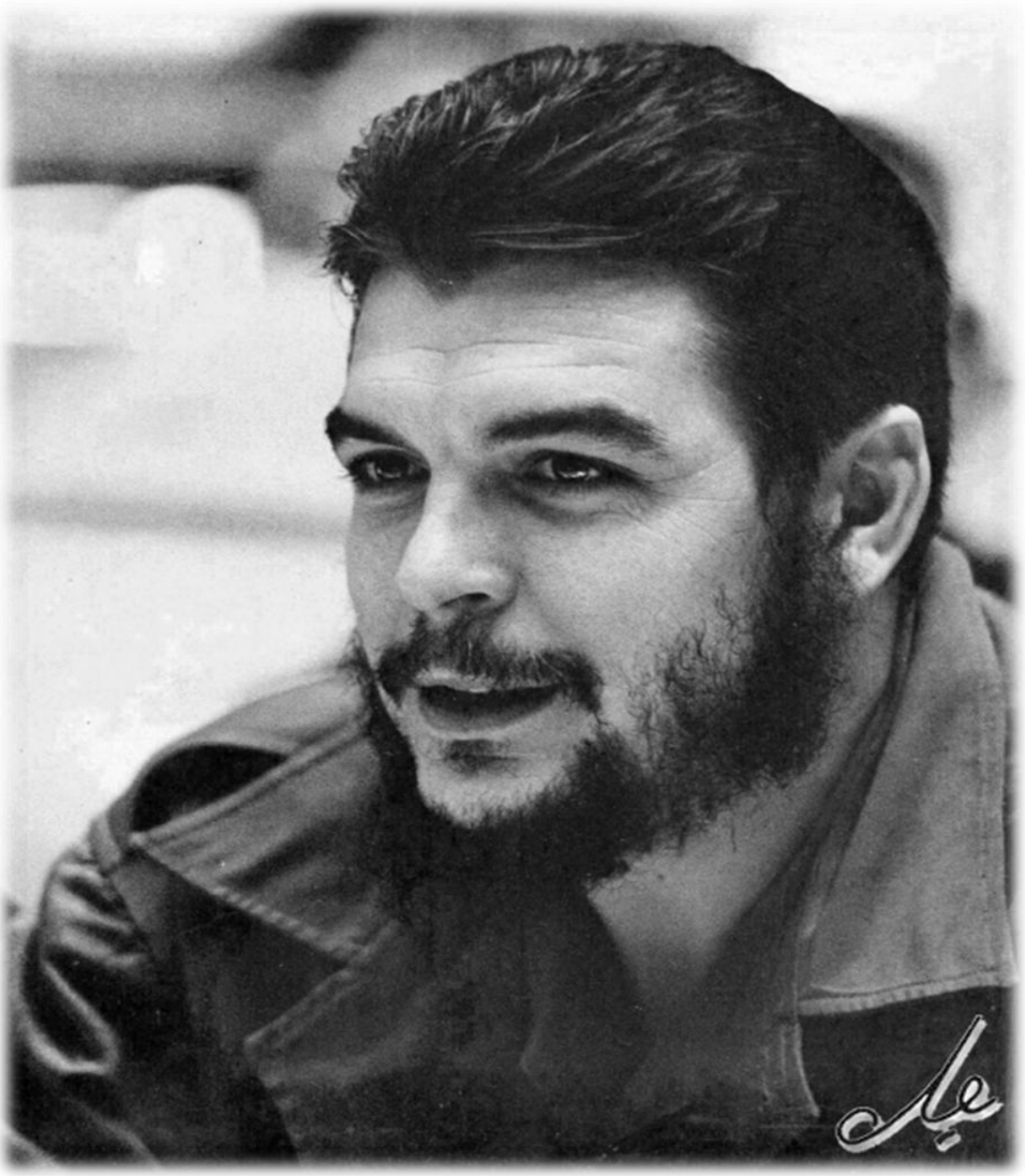
Anilia Yenisleidy Pazos Ferrera

Tutores:

MSc. Dismey Saavedra López

Ing. Osay González Fuentes

La Habana, Cuba. Junio de 2017.



"…aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de sí mismos…" *Ernesto Che Guevara*

Declaración de autoría

Declaro por este medio que yo Anilia Pazos Ferrera, con carné de identidad 93120731692 soy la autora principal del trabajo titulado “Módulo Ventas del Sistema de Gestión para la Agencia de Renta de Vehículos REX” 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 ____ del año ____.

_____

Anilia Yenisleidy Pazos Ferrera.

Firma del Autor

_____ _____

MSc. Dismey Saavedra López

Firma del Tutor

Ing. Osay González Fuentes

Firma del Tutor

Datos del contacto

Tutor: MSc. Dismey Saavedra López

Ingeniera en Ciencias Informáticas. Graduada en la Universidad de las Ciencias Informáticas en el 2008. MSc. en Informática en Salud, graduada en el Centro de Cibernética Aplicada a la Medicina (CECAM) en el 2015. Actualmente se desempeña como especialista, en el Centro de Identificación y Seguridad Digital (CISED) en el departamento de aplicaciones. Ha sido tutora de tesis de pregrado, oponente y tribunal de otras tesis, obteniendo siempre buenos resultados. Se ha desempeñado además como profesora de Matemática durante cinco años trabajando siempre con primer año.

Correo electrónico: dsaavedra@uci.cu

Tutor: Ing Osay González Fuentes

Ingeniero en Ciencias Informáticas. Graduado en la Universidad de las Ciencias Informáticas en el 2006. Actualmente se desempeña como Director del Centro de Identificación y Seguridad Digital. Ha sido tutor de tesis de pregrado en otras ocasiones teniendo buenos resultados.

Correo electrónico: ogonzalezf@uci.cu

Agradecimientos

Primero que todo le quiero agradecer a mis padres que lo han sido mi inspiración, gracias mami, gracias papi por todo el apoyo incondicional que me han brindado toda mi vida.

A mi novio que ha sido mi motor impulsor y gracias por cambiarme la vida.

En general a toda mi familia que también han aportado su granito de arena para que yo saliera adelante con mi carrera, pero en especial a mi tía Aris que es la persona que cuando los caminos se me cerraban por una parte ella con su forma de ser y carisma sin saberlo me estaba abriendo una ventana.

A todos los muchachos de mi grupo decirle que los quiero a todos por igual. Quiero hacer una mención especial a dos personitas que de una forma u otra me han soportado todos estos años ellos son Wendita y Hecti siempre los voy a tener en mi corazón.

A los profesores Yandri, Yusdel y Walber que me han ayudado incondicionalmente en mi proceso de tesis.

A mis tutores.

RESUMEN

Debido a la necesidad de fomentar la industria de *software* en Cuba, se creó la Universidad de las Ciencias Informáticas, centro de enseñanza superior que tiene como propósito convertirse en una potencia informática en el desarrollo de *software*. En la universidad se lleva a cabo el desarrollo del Sistema de Gestión para la Agencia de Renta de Vehículo REX, que tiene como misión llevar el control, manejo y distribución de su flota. Debido a que el sistema actualmente en explotación en la agencia, posee una desactualización tecnológica, lo que imposibilita la realización de los procesos que ejecuta. El presente trabajo de diploma persigue como objetivo: desarrollar un módulo Venta que facilite el proceso de gestión de renta de autos, el mismo cuenta con funcionalidades que ayudan al proceso de gestión de las rentas de autos en la agencia. Para darle solución a lo anteriormente planteado, se realizaron las investigaciones necesarias en cuanto al desarrollo de sistemas de gestión de rentas de autos en aplicaciones *web*, además se seleccionaron las diferentes tecnologías que se utilizaron para llevar a cabo la propuesta de solución y quedaron implementados todos los requisitos funcionales, lográndose un módulo integrable y escalable dentro del sistema SIGREX.

Palabras clave: gestión de la información, renta, REX, venta.

Índice

Índice de Tablas

Índice de Figuras

Introducción

Las Tecnologías de la Información y las Comunicaciones (TIC), se han trasformado en uno de los recursos más importantes de la sociedad, lo que ha traído consigo un auge en la transmisión e intercambio de datos, información y conocimientos. En los últimos años, casi todos los países del mundo han establecido e implementado proyectos, políticas y estrategias para promover el uso de las TIC y aprovechar los beneficios y los aportes que estas ofrecen.

Cuba ha impulsado la informatización de la sociedad como una de las políticas más importantes del país (1). Este proceso lo ha logrado mediante la utilización de las TIC, las cuales se han insertado en diferentes sectores de la sociedad cubana jugando un papel importante y desempeñando cada día un mayor rol. (2) Uno de los sectores en los cuales ha tenido un gran impacto es el turístico, debido a que las maneras en que los turistas realizan los viajes, las consultas de servicios, las publicidades y formas de ventas actuales están sumamente ligadas a las tecnologías. Con su uso se han logrado grandes beneficios, como el conocer mejor las necesidades de los clientes, ofrecer una mejor entrega del servicio, llegar a un mayor número de turistas y optimizar sus recursos logrando aumentar su eficiencia. (3)

El turismo está muy ligado a la actividad de viajar, esencialmente a través de la aviación y el transporte terrestre. Significativo es el número de empresas dedicadas a la transportación turística. Los servicios que ofertan la renta de autos es uno de los más importantes. En Cuba, existen empresas que se encargan de ofertar estos servicios y una de ellas es Transtur, la mayor empresa de servicios de ómnibus y renta de autos para el turismo, esta empresa tiene sucursales en todo el territorio nacional.

La Agencia de Renta de Vehículos REX, como empresa independiente con alcance nacional, que pertenece al Grupo Transtur, tiene como objetivo ofrecer posibilidades de transportación exclusivas en Cuba, ya sea por negocios o placer; esta empresa, cuenta con una flota única distribuida en todo el país a través de puntos de venta cuyo control, manejo y distribución se realiza mediante un sistema automatizado con más de diez años de explotación.

Este sistema que lleva por nombre Sistema Integral de Gestión de REX (SIGREX) consta de ocho módulos para manipular toda la información de la renta de autos y está diseñado como una aplicación de escritorio que se basa en la arquitectura cliente-servidor. Utiliza el sistema gestor de base de datos *Microsoft SQL Server 2000*, está desarrollada sobre la herramienta *Visual Studio 2003* y el lenguaje de programación *C#*. Además, usa una base de datos distribuida bajo el funcionamiento de réplicas de la información. Producto al continuo avance de las tecnologías y a la falta de soporte tecnológico, hoy el sistema cuenta con funcionalidades que no cumplen con los requisitos necesarios para su funcionamiento.

Lo anteriormente mencionado trae consigo un conjunto de limitantes, las cuales se resumen a continuación: la información que hoy maneja el sistema no produce confianza debido a la demora de las actualizaciones de los datos, las cuales se realizan cada diez minutos aproximadamente, pues puede suceder que un auto se encuentre disponible y ya se encuentre rentado o reservado por un cliente, lo que ocasiona que los servicios no se puedan brindar con la calidad esperada. Otra de las dificultades radica en que los datos son insertados en el sistema y una vez en él no pueden ser modificados, lo que hace que el proceso de actualización de los datos sea lento y engorroso, debido a que para actualizar un dato debe hacerse de manera central por los especialistas informáticos. Además, su condición de aplicación de escritorio constituye una desventaja adicional producto a que impide su despliegue en distintas plataformas.

Dentro del sistema se encuentra el módulo Ventas el cual posibilita la gestión de las rentas de autos, este módulo también se ve afectado por los problemas antes expuestos.

Hoy la agencia está interesada en realizar cambios y orientarse a la *web*. Con este objetivo se desarrolla en la Universidad de las Ciencias Informáticas, el proyecto productivo del Sistemas de Gestión de Renta de Vehículos para la agencia REX (SIGREX), perteneciente al Centro de Identificación y Seguridad Digital (CISED).

Dada la situación problemática antes descrita, se identifica como **problema de investigación:** ¿Cómo facilitar el proceso de gestión de la información de las rentas de autos en el SIGREX?

Se define como **objeto de estudio** el proceso de gestión de la información de las rentas en agencias de autos. Enmarcado el **campo de acción** en la gestión de la información de las rentas de autos en la Agencia de Renta de Vehículos REX.

Para dar respuesta al problema antes mencionado, se traza como **objetivo general** de la investigación: Desarrollar un módulo que facilite el proceso de gestión de renta de autos para SIGREX.

Las **preguntas científicas** que guían y orientan el desarrollo del proceso investigativo son las siguientes:

- ¿Cuáles son los principales referentes teóricos que sustentan el desarrollo del módulo Ventas del SIGREX?
- ¿Qué metodología, tecnologías y herramientas conforman el ambiente de desarrollo para la implementación del módulo Ventas del SIGREX?
- ¿Qué artefactos se generan a partir del análisis y diseño del módulo Ventas del SIGREX?

- ¿Cómo comprobar que la solución propuesta facilita el proceso de gestión de ventas en la agencia de renta de vehículos REX?

Para dar solución al problema antes planteado y a las preguntas científicas formuladas, se proponen los siguientes **objetivos específicos**:

- Analizar los principales referentes teóricos que sustentan el desarrollo del módulo de gestión de renta de autos del SIGREX.
- Seleccionar el ambiente de desarrollo para la implementación del módulo Ventas del SIGREX.
- Elaborar los artefactos generados a partir del análisis y diseño del módulo Ventas del SIGREX.
- Implementar el módulo Ventas del SIGREX.
- Realizar pruebas para validar la solución propuesta.

Para darle cumplimiento a los objetivos específicos se emplearán varios **métodos científicos** para la búsqueda y procesamiento de los datos tales como:

Métodos teóricos:

Analítico-Sintético: para el estudio de conceptos fundamentales que se encuentran intrínsecos en el proceso de rentas de autos, con el análisis de toda la documentación necesaria para la extracción de los elementos importantes sobre el tema de la presente investigación.

Modelación: para la confección del diseño del sistema mediante los diagramas y modelos.

Métodos Empíricos:

Análisis documental: para extraer información necesaria de la bibliografía existente y consultar información en sitios de interés nacional e internacional, con el objetivo de apoyar los objetivos específicos de la investigación.

El contenido de la investigación se encuentra estructurado en tres capítulos, los cuales se congregan de la manera siguiente:

Capítulo 1: Fundamentación Teórica.

Este capítulo se centra en los aspectos teóricos y conceptos fundamentales relacionados con el objeto de estudio de la investigación. Se analizan las soluciones existentes tanto en el ámbito internacional como en el nacional y se describen las herramientas a utilizar para el diseño e implementación de la solución.

Capítulo 2: Análisis y Diseño del sistema.

En este capítulo se realiza una propuesta del sistema, se describe cómo debe funcionar y se destacan sus características distintivas; además, se especifican sus requisitos funcionales y no funcionales y se muestran los modelos necesarios para la realización del diseño de la propuesta de solución.

Capítulo 3: Implementación y Validación del Sistema.

Se analizan los procesos referentes al desarrollo de la solución y las pruebas realizadas para validar el correcto funcionamiento del módulo.

Finalmente, se presentan las Conclusiones, Recomendaciones, Referencias Bibliográficas y Anexos derivados de la investigación.

Capítulo I: Fundamentación Teórica

En el presente capítulo se realiza una exposición de los principales conceptos que, desde el punto de vista teórico, permiten un mejor entendimiento de lo que se plantea en la situación problemática y en el marco del problema. También se realiza un estudio sobre las soluciones similares en Cuba y en el mundo, con el fin de lograr una mejor comprensión y adquirir conocimientos referentes a los sistemas de gestión de rentas de autos, su alcance y aporte científico. Además, se presenta la metodología y el ambiente de desarrollo que se emplearán en la implementación de la propuesta de solución.

1.1 Información

En todo sistema de gestión de información, el elemento principal es la misma información, por lo que es necesario definirla y caracterizarla. Según la Real Academia de la Lengua Española, *se define la información como la comunicación o adquisición de conocimientos que permiten ampliar o precisar los que se poseen sobre una materia determinada*. Esta definición hace referencia a la utilidad de la información como fuente de conocimientos.

La información tiene distintas facetas y tendrá distintos significados en dependencia del punto de vista que se analice. Según la autora Dolores Vizcaya Alonso, haciendo un análisis de la información desde el punto de vista filosófico la define como " una parte de una reflexión, diferente de los factores materiales y energéticos, que es percibida por los sistemas materiales en una etapa organizativa definida y tan voluminosa que puede almacenarse, expresa en mensajes ordenados respecto a la probabilidad de uno u otro hecho entre la multitud de acontecimientos de una naturaleza dada" (4), definición que se centra fundamentalmente en el carácter informativo y/o explicativo de la información.

Otro concepto con el que se define a la información es en el ámbito de Sistemas de Información (SI), en el cual Börje Langefors define la información como "cualquier mensaje o conocimiento que pueda usarse para posibilitar o mejorar una acción o decisión" (5), mencionando el valor de la información en la toma de decisiones en cualquier organización.

A partir de las definiciones anteriores, el autor de la investigación concluye que "la información es un conjunto de datos procesados que tiene importancia para el que la recibe, sirve de ayuda en la toma de decisiones además de influir en la obtención de nuevos conocimientos o reafirmar los ya existentes".

1.2 Gestión de la información.

Antes de hablar de la Gestión de la Información (GI), es necesario tener en cuenta el concepto de gestión que es “la actividad dirigida a obtener y asignar los recursos necesarios para el cumplimiento de los objetivos de la organización.” (6)

Varios son los autores que coinciden con que la información necesita una buena gestión, o sea, obtenerla de una fuente confiable, analizarla, procesarla y utilizarla en la toma de decisiones de una organización. Según Phil Bartle,“ la gestión de la información es el proceso de analizar y utilizar la información que se ha recabado y registrado para permitir a los administradores (de todos los niveles) tomar decisiones documentadas.” (7)

Aterrizando la definición antes mencionada desde el punto de vista de la ingeniería, se puede decir que la GI, se pudiera entender por “el proceso de extracción, manipulación, tratamiento, depuración, conservación y acceso a la información obtenida a través de una o varias fuentes, además de gestionar los derechos de los usuarios a usarla.”

1.3 Sistemas de gestión.

En el mundo el desarrollo de las tecnologías va aparejado al incremento de sistemas para la gestión de la información en las distintas esferas. Estos constituyen una estructura probada para la gestión y mejora continua de las políticas, procedimientos y procesos de toda organización. (8) La necesidad de analizar y comprender dichos sistemas, tanto en el ámbito nacional, como internacional es el factor clave para la realización de un nuevo producto informático.

Los Sistemas de Gestión (SG) deben facilitar, simplificar y realizar automáticamente procesos que tradicionalmente se realizaban de forma manual. Así pues, sustituyen ventajosamente al personal encargado, evitando errores y mejorando la velocidad de procesamiento de la información; establecen un progresivo control en las entidades financieras con ventajas incuestionables en cuanto a fiabilidad y seguridad; realizan los reiterativos procesos contables sin errores en las operaciones y con una rapidez y agilidad considerable.

1.4 Sistemas de gestión de rentas de autos a nivel internacional.

En el mundo han sido desarrollados numerosos sistemas de gestión de rentas de autos, con el objetivo de controlar de forma efectiva las reservas, contratos y otros indicadores asociados a las rentas. Ejemplos de estos sistemas son:

- **Sistema de Alquiler de Vehículo ALVESIA:**

Es un sistema que abarca e integra todas las funciones de un negocio de *Car Rental*, conformado por varios módulos: módulo Comercial que permite el manejo de clientes, servicios, precios y tarifas, por clasificación de vehículos y localidad; módulo Reservaciones en él se pueden generar cotizaciones y reservaciones de autos con y sin prepagos; el módulo Alquiler gestiona los contratos y las facturas de alquiler, alquiler y siniestro y de siniestro; el módulo Flota permite el manejo del auto desde su pre-ingreso hasta su desincorporación, en él se registran todos los movimientos del vehículo con sus características (kilómetros, combustible, chequeo de inventario) y consecuencias; el módulo Siniestro y Seguridad permite controlar y manejar los siniestros, desde los menos relevantes hasta los más impactantes; la seguridad se maneja una lista negra de la empresa, donde se registran aquellos clientes que su comportamiento no ha sido el más adecuado con respecto a los alquileres de autos.

Otros de los módulos con el que cuenta el sistema es el de Seguridad Interna, el cual permite la auditoría de contratos, donde se controla el uso correcto de las reglas del negocio en el momento de su generación. Un contrato que se haya generado con las reglas del negocio incorrectas, se penaliza, afectando esto la comisión de venta del operador responsable; y el módulo Venta, permite consignar o vender un vehículo y generarle la factura. En el módulo de Mantenimiento, se registran las órdenes de trabajos para aquellos autos que requieren de un mantenimiento preventivo o correctivo, se especifica el tipo de mantenimiento y se asocian los costos que los mismos ocasionaron. En el de Configuración se inicializan todos los parámetros bases del sistema y donde se definen algunas de las reglas del negocio. (9)

Características del Sistema de Alquiler de Vehículo ALVESIA:

- Es un sistema totalmente *web*.
- La inversión que se requiere en plataforma es mínima: un computador con conexión a Internet y una impresora. El servidor puede manejarlo con hospedaje externo.
- Está desarrollado con PHP y Mysql.
- Integra todo el negocio de un *car rental*.
- Es modular.
- Es configurable.
- Todo el negocio está sobre una misma plataforma y base de datos. No hay información dispersa.

- ***Visual Rent a Car:***

Es una completa aplicación de gestión de alquileres de vehículos. Control de vehículos, cliente y documentación de los mismos. *Software* para empresas de *Rent a Car*, para alquileres de

vehículos. Clasificación de vehículos en diferentes categorías, configurables por el usuario. *Software* sencillo e intuitivo que se adapta a las necesidades de cualquier negocio. (10)

Características del *Visual Rent a Car*:

- Control de la situación del vehículo por sucursal y localización (configurables).
- Realización automática de los contratos en *Word* del alquiler.
- Control y seguimiento de las reservas de vehículos. Generación del alquiler a partir de la misma.
- Conexión del programa a la *web* que permite realizar las reservas de vehículos a través de Internet.
- Utiliza la base de datos de *Microsoft SQL server* 2008.

1.5 Sistema de gestión de rentas de autos a nivel nacional.

El desarrollo informático en Cuba, en los últimos años ha evolucionado el pensamiento de la sociedad cubana en busca de nuevas soluciones para los problemas presentados en numerosas ramas. Las empresas turísticas no están ajenas a esta nueva revolución que ha generado el sector de la informática y es por ello que se ha extendido la informatización de los procesos que se realizan como búsqueda de mejores soluciones.

Las empresas transportistas con mayor servicio que tiene Cuba en el sector turísticos son:

- **El Grupo de Turismo Gaviota S.A.:** actualmente las gestiones de las rentas se realizan con la ayuda de la herramienta ofimática *Excel y* el correo, que es por donde realizan las actualizaciones diarias del negocio y los datos se almacenan en formato duro, además utilizan la modalidad de rentas de autos *Online* a través del sitio *web* de la entidad.
- **El Grupo Empresarial Transtur S.A.:** es el transportista oficial del Ministerio de Turismo de Cuba y la mayor empresa de su tipo en el país. La cual cuenta con una amplia red de empresas dedicadas a la renta de autos, dentro de las que se encuentran Cubacar, Habanautos y REX, apoyadas por la modalidad de ventas *Online* que se realizan por medio del sitio *web* trantur.com y otros sitios donde están asociados. También cuentan con el apoyo de:
    - **El Sistema de Explotación de la Renta (SER):** es una aplicación de escritorio, su función no es vender sino llevar el control de la explotación de la venta.
    - **Sistema de Gestión de Renta de Vehículos REX (SIGREX):** el sistema automatizado con el que REX actualmente gestiona su flota vehicular, está dividido en ocho módulos que controlan los procesos de la empresa, todos

desarrollados sobre la plataforma .NET 2.0 usando *Microsoft SQL Server* 2000 como gestor de base de datos. Sus principales características son: es una aplicación de escritorio con base de datos en todas las oficinas donde se encuentre desplegado, utiliza réplicas para la actualización de los datos y no cuenta con un servicio de soporte al cliente.

Valoración de los sistemas de gestión de rentas de autos.

A partir de estudio de algunos sistemas de gestión renta de autos, anteriormente mencionados, se tomaron en cuenta, no solo sus características como aplicaciones informáticas, sino por los aportes que estos brindan. En cuanto a los sistemas internacional sus funcionalidades adaptables a cualquier negocio, pero en su mayoría son privativos y su aplicación en Cuba resulta inapropiada a partir de los altos costos por concepto de licencia y soporte de las tecnologías que utilizan. Los sistemas nacionales no responden aceptadamente a las necesidades específicas de la agencia REX.

Por todo lo antes expuesto se concluye, que se hace necesario crear un módulo Ventas nuevo que se corresponden con los requisitos de la agencia. Igualmente, que sea consecuente con la filosofía de conseguir la soberanía tecnológica, haciéndose necesario el desarrollo del módulo sobre herramientas y tecnologías libres basadas en la *web*, con el fin de lograr una correcta gestión de la información de las rentas.

1.4 Lenguajes, Metodologías y Herramientas de desarrollo.

El desarrollo de un proyecto de *software* es un proceso que engloba procedimientos, técnicas, documentación y herramientas que se utilizan en la creación de un producto de *software.* Convirtiéndose dicha metodología en un plano que apoya a los desarrolladores, para así lograr clientes satisfechos con el resultado final y a la vez lograr quién debe hacer qué, cuándo y cómo dentro del equipo de trabajo. (11)

1.4.1 Metodología para el desarrollo del *software*.

Proceso Unificado Ágil

El Proceso Unificado Ágil (AUP por sus siglas en inglés) es una versión simplificada del Proceso Unificado de Rational (RUP). Este describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. AUP aplica técnicas ágiles incluyendo: (12)

Desarrollo Dirigido por Pruebas (*test driven development* -TDD en inglés).

- Modelado ágil.
- Gestión de cambios ágil.
- Refactorización de Base de Datos para mejorar la productividad.

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva:
Inicio: el objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
Elaboración: el objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
Construcción: durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
Transición: el sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación, aceptación y finalmente se despliega en los sistemas de producción.
Ventajas.

- Simplicidad: apuntes concisos.
- Agilidad: procesos simplificados del RUP
- Centrarse en actividades de alto valor, esenciales para el desarrollo.
- Herramientas independientes: a disposición del usuario.

Desventajas.

- Como es un proceso simplificado, muchos desarrolladores eligen trabajar con el RUP, por tener a disposición más detalles en el proceso.

Variación de la AUP para la UCI

Al no existir una metodología de *software* universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, entre otros) exigiéndose así que el proceso sea configurable, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. (12)

Tabla 1: *Comparación de las fases de AUP con las AUP-UCI.*

Fases AUP	**Fases Variación AUP-UCI**	**Objetivos de las fases (Variación AUP-UCI)**
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la

		planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, se obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
Construcción		
Transición		
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

De las cuatro fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI, mantener la fase de Inicio, pero modificando el objetivo de la misma. Se unifican las restantes tres fases en una sola, la cual tiene como nombre Ejecución y se agrega la fase de Cierre.

Con la adaptación de AUP que se propone para la actividad productiva de la UCI, se logra estandarizar el proceso de desarrollo de *software*, dando cumplimiento además a las buenas prácticas que define CMMI-DEV v1.3. Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos.

Debido a que el módulo debe ser correctamente documentado, se decide el uso de AUP-UCI como metodología de desarrollo porque posee un enfoque ágil, es ideal para pequeños equipos de trabajo, se apoya en las buenas prácticas de CMMI-DEV v1.3 y además porque se recomienda para los proyectos de la Universidad.

1.4.2 El Lenguaje Unificado de Modelado (UML).

Lenguaje Unificado de Modelado (UML[1], por sus siglas en inglés) es el lenguaje de modelado de sistemas de *software* más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables (13).

Dentro de sus características resaltan:

- Tecnología orientada a objetos.
- Viabilidad en la corrección de errores.
- Participación del cliente en todas las etapas del proyecto.
- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas).

1.4.3 Herramienta de modelado.

El propósito de una herramienta CASE (*Computer Aided Software Engineering*) es brindar soporte de forma automatizada para la aplicación de técnicas usadas por una o varias metodologías. Es beneficioso usarla cuando se utiliza una metodología porque dispone de funciones automatizadas tales como: obtención de prototipos, generación de código, generación de pantallas e informes, generación de diseños físicos de bases de datos y facilita:

- La verificación y mantenimiento de la consistencia de la información del proyecto.
- El establecimiento de estándares en los procesos de desarrollo y documentación.
- El mantenimiento del sistema y las actualizaciones de su documentación.
- La aplicación de las técnicas de una metodología.
- La planificación y gestión del proyecto informático.

Dicha tecnología supone la automatización del desarrollo del *software*, lo que contribuye a mejorar la calidad en el desarrollo de sistemas de información. (14)

[1] *Unified Modeling Language*

- ***Visual Paradigm para UML 8.0.***

Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de *software*: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. También ayuda de forma rápida a la construcción de aplicaciones de calidad, permite dibujar todo tipo de diagramas de clases y generar documentación. La herramienta proporciona abundantes tutoriales de UML, posee una interfaz sugerente, pertenece a la familia de *software* libre y soporta UML como notación principal

- ***Rational Rose.***

Rational Rose es una herramienta de diseño orientada a objetos, que da soporte al modelado visual, es decir, que permite representar gráficamente el sistema, permitiendo hacer énfasis en los detalles más importantes, centrándose en los casos de uso y enfocándose hacia un *software* de mayor calidad, empleando un lenguaje estándar común que facilita la comunicación. Proporciona mecanismos para realizar la Ingeniería Inversa, es decir, que a partir del código se pueda obtener información sobre su diseño; adicionalmente permite generar código en diferentes lenguajes a partir de un diseño en UML. (15)

Después de haber realizado un estudio entre estas dos herramientas de modelado, se llegó a la conclusión que se utilizará *Visual Paradigm*, porque es una herramienta multiplataforma y robusta, permite el control de versiones y genera la documentación en varios formatos, lo que favorece la rapidez del trabajo.

1.4.4 Lenguaje de desarrollo.

Un lenguaje de programación es una herramienta que permite la comunicación e instruir a un computador para que ejecute una tarea específica. Cada lenguaje de programación posee una forma de escribirse que es diferente en cada uno por la forma que fue creado y por la forma que trabaja su compilador para revisar, acomodar y reservar el mismo programa en memoria. (16)

- **Python 2.7.**

Python es un lenguaje de programación multiparadigma, pues soporta orientación a objetos, programación imperativa, programación orientada a aspectos y en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico, es fuertemente tipado y multiplataforma.

El entorno de ejecución detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos. Una ventaja fundamental es la gratuidad de su intérprete, el cual tiene versiones para prácticamente

cualquier plataforma en uso: sistemas PC bajo Linux, sistemas PC bajo *Microsoft Windows*, sistemas Macintosh de Apple, entre otras. (17)

Ventajas del uso de Python:

- Propósito general: se pueden crear todo tipo de programas.
- Multiplataforma: hay versiones disponibles de Python en muchos sistemas informáticos distintos. Originalmente se desarrolló para Unix, aunque cualquier sistema es compatible con el lenguaje siempre y cuando exista un intérprete programado para él.
- Facilidad de uso: lenguaje con una rápida curva de aprendizaje lo que permite empezar a hacer programas sencillos en Python en muy poco tiempo.
- Legibilidad del código. la estructura del código es bastante natural y promueve una forma de escribir que facilita su lectura. Esta es una ventaja importante frente a lenguajes dirigidos al mismo sector, como Perl.
- Interactivo: dispone de un intérprete por línea de comandos en el que se pueden introducir sentencias. Cada sentencia se ejecuta y produce un resultado visible, que puede ayudar a entender mejor el lenguaje y probar los resultados de la ejecución de porciones de código rápidamente.
- Orientado a Objetos: la programación orientada a objetos está soportada en Python y ofrece en muchos casos una manera sencilla de crear programas con componentes reutilizables. Además, Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.
- Funciones y librerías: existen una gran variedad de bibliotecas disponibles para extender la funcionalidad básica de Python a cualquier campo.
- Sintaxis clara: tiene una sintaxis muy visual, producto de que utiliza una notación con márgenes de obligado cumplimiento. Esto ayuda a que todos los programadores adopten unas mismas notaciones y que los programas de cualquier persona tengan un aspecto muy similar.
- Mixto: se puede integrar de manera "fácil" con otros lenguajes de programación.
- Gratuito: una ventaja fundamental de Python es la gratuidad de su intérprete que está liberado bajo la licencia GPL, se puede descargar desde la página *web*: http://www. Python.org. (18)

Python es el lenguaje seleccionado para la programación de la propuesta de solución, proporcionando la rapidez y robustez necesarias para un módulo como el que se desea

desarrollar. Además, el uso del lenguaje está regido principalmente por definición de la dirección del proyecto SIGREX.

1.4.5 Framework.

Un *framework* es una estructura de archivos y utilidades que aceleran la programación de una aplicación informática, que provee una metodología de trabajo que sistematiza y facilita la generación de formularios, funciones y módulos de uso común, permitiendo al desarrollador dedicar su atención hacia los aspectos específicos de cada aplicación. (19)

- **Django.**

Es un *framework* de alto nivel basado en Python que facilita el desarrollo de aplicaciones *web* dinámicas que: (20)

- Abstrae a los programadores de los problemas comunes del desarrollo *web* y acelera las tareas más frecuentes en la programación.
- Proporciona un método de mapear las *URL* ejecutando un código en especial para cada una. Permite mostrar y validar formularios de manera muy simple, manipulando el código del formulario y adaptándolo a las necesidades de la aplicación.
- Convierte los datos enviados por los usuarios, en estructuras de datos que pueden ser manipuladas fácilmente.
- A través de plantillas ayuda a separar el contenido de la presentación evitando tener que manipular la lógica de negocio cuando se necesite realizar cambios de apariencia en la página.
- Permite lidiar con trescientas peticiones *web* por segundo.
- Django incluye muchas aplicaciones comunes a todos los sitios *web*, como la autenticación de usuarios o la administración del contenido del sitio.
- Su arquitectura está inspirada en el patrón Modelo-Vista-Controlador (MVC), encargándose en gran parte de los controladores, y proveyendo herramientas para facilitar el desarrollo de las vistas y los modelos.

Basados en estas características, se escoge este marco de trabajo en su versión 1.10 para la implementación del sistema, pues se centra en el desarrollo rápido, la reutilización y la seguridad.

1.4.6 Entorno de Desarrollo Integrado (IDE).

Un entorno de desarrollo integrado o IDE (*Integrated Development Environment*) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Estas herramientas pueden estar creadas para su utilización con un único lenguaje.

- ***PyCharm.***

PyCharm Community Edition-2016.1, es un entorno de desarrollo integrado multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Provee funcionalidades que permiten una experiencia única y aumentan en gran medida la productividad: (21)

- Completamiento de código de manera inteligente y señalamiento de errores con reparación de los mismos de forma automatizada.
- Integración con marcos de trabajo de desarrollo *web* modernos como Django.
- Depurador integrado y herramientas de pruebas.
- Soporte para bases de datos e integración con sistemas de control de versiones.
- En adición a Python, PyCharm soporta *JavaScript, CoffeeScript, TypeScript*, HTML/CSS, *Cython*, lenguajes de plantilla, *AngularJS* y más.

Por todas las características anteriormente expresada, se decidió la utilización del IDE de desarrollo *PyCharm Community Edition-2016.1*.

1.4.7 Sistema gestor de base de datos.

*"*Un Sistema Gestor de Base de Datos (SGBD, en inglés DBMS: *DataBase Management System*) es un sistema de *software* que permite la definición de bases de datos; así como la elección de las estructuras de datos necesarios para el almacenamiento y búsqueda de los datos, ya sea de forma interactiva o a través de un lenguaje de programación*".* (22)

Se define también como "el conjunto de programas que administran y gestionan la información contenida en una base de datos". (22) Ayuda a realizar las siguientes acciones:

- Definición de los datos.
- Mantenimiento de la integridad de los datos dentro de la base de datos.
- Control de la seguridad y privacidad de los datos.
- Manipulación de los datos.

Este tipo de sistema facilita el trabajo de los desarrolladores con el manejo de las bases de datos, aumentando el nivel de productividad.

- ***PostgreSQL* 9.4.**

Es un gestor de bases de datos relacional orientado a objetos y libre. Es capaz de manejar una enorme cantidad de datos permitiendo el acceso simultáneo de un conjunto de usuarios. Brinda seguridad y estabilidad a los mismos y facilita el trabajo con procedimientos almacenados y consultas. (22)

Cuenta con las características tales como:

- Extensible: el código fuente está disponible para todos sin costo.
- Multiplataforma: *PostgreSQL* está disponible para plataformas Unix (34 plataformas en la última versión estable) y Windows.
- Diseñado para ambientes de alto volumen: *PostgreSQL* está considerado como la base de datos de código abierto más avanzada del mundo.

Es multiplataforma y totalmente compatible con ACID (*Atomicity, Consistency, Isolation and Durability*, en español Atomicidad, Consistencia, Aislamiento y Durabilidad). Tiene soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados (en varios idiomas). Soporta el almacenamiento de grandes objetos binarios, como imágenes, sonidos o vídeos. Dispone de interfaces de programación nativas para C/C++, Java, .Net, Perl, Python, Ruby, ODBC, entre otros. Es altamente escalable tanto en la enorme cantidad de datos que puede manejar como en el número de usuarios concurrentes que puede soportar.

Esta herramienta constituirá el soporte para la base de datos que almacenará toda la información de manera centralizada, la cual será gestionada por el módulo a desarrollar.

1.5 Conclusiones parciales.

La utilización de los métodos de investigación científica definidos en la introducción de la actual investigación, permitió estudiar y comprender a fondo el proceso de gestión de renta de autos. También permitió realizar un análisis de los sistemas de este tipo, de mayor calidad a nivel nacional e internacional. Después de realizados estos estudios, se llegó a la conclusión que es preciso desarrollar un nuevo módulo que capaz de integrarse con SIGREX y que cumpla con las necesidades de la agencia.

Luego del estudio del estado del arte se decidió emplear como metodología para guiar el proceso de desarrollo del *software*: AUP-UCI. Como lenguaje de programación se eligió Python por ser potente y posibilitar la creación de una aplicación robusta; auxiliándose de la valiosa ayuda que proporciona el marco de trabajo *Django*, que agiliza el desarrollo de aplicaciones *web* en Python. El Entorno de Desarrollo Integrado que se utilizará para el desarrollo del sistema

será *Pycharm*. Para lograr la persistencia de la información, se seleccionó el sistema de base de datos *PostgreSQL*, que brinda una alta fiabilidad e integridad de los datos.

Capítulo II: Análisis y Diseño del sistema.

En el presente capítulo se abordan temas relacionados con el funcionamiento del sistema, guiados por la etapa de ejecución, donde se realiza una descripción textual de los casos de uso del sistema, guiada por los requisitos funcionales y no funcionales definidos y se generan los artefactos (modelo de dominio, modelo de datos, diagramas de clases del diseño) propuestos por la metodología AUP-UCI, para obtener una concepción general del módulo a implementar.

2.1 Descripción de la propuesta de solución.

Para contribuir con la solución a las limitantes antes planteadas, se implementará un módulo Ventas destinado al nuevo sistema de gestión denominado SIGREX. El módulo resultante, permite a los usuarios del sistema, crear, modificar, eliminar una reserva, gestionar los contratos y listar las reservas diarias. Otras de las funcionalidades que dispone el módulo, es gestionar los accesorios que el cliente desee rentar, así como gestionar conductores, los códigos de operaciones y los depósitos a realizar, teniendo en cuenta la interacción con los demás módulos del sistema.

2.2 Modelo del dominio.

A través del modelo de dominio se realiza una representación las clases conceptuales del dominio del problema y los conceptos del mundo real. Una vez generado este modelo, se crean las bases para adentrarse en el análisis del módulo como siguiente paso dentro del desarrollo del *software*.

En la **Figura 1**, se muestra el modelo conceptual de la problemática a resolver, identificando los objetos fundamentales y sus relaciones.

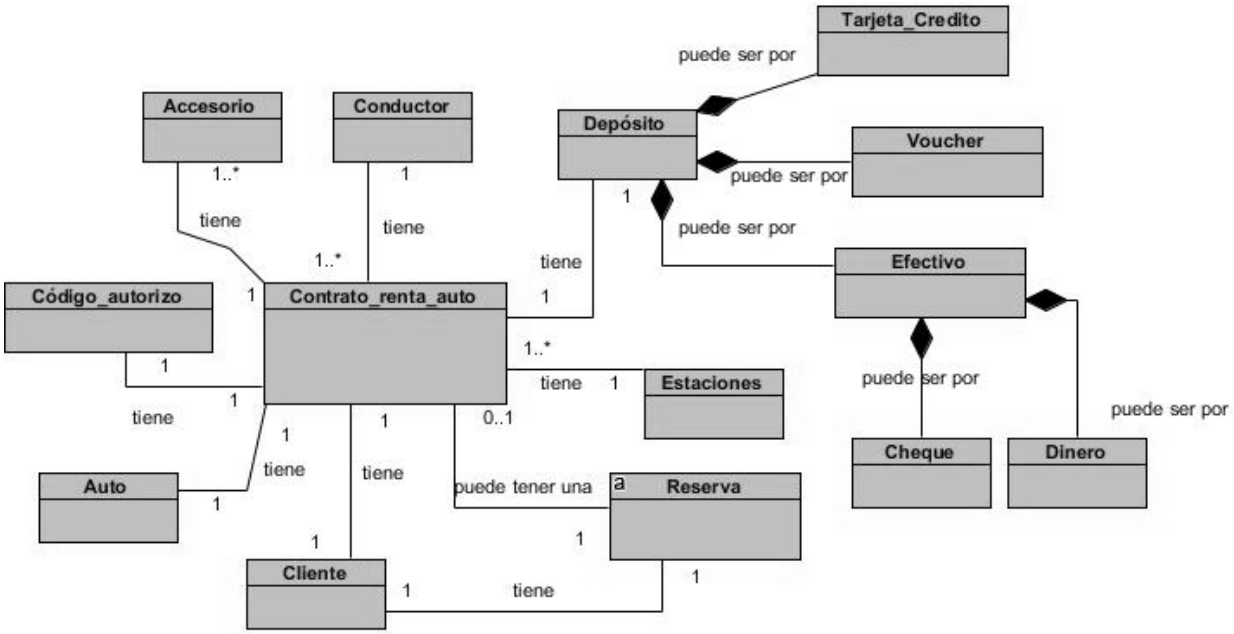


Figura 1: *Modelo de dominio*.

Descripción de los elementos del modelo de dominio:

Contrato de renta de auto: elemento donde se manejan todos los datos referentes al acuerdo que se efectúa entre el cliente y la agencia, a través del operario de la estación de ventas. Este puede ser con una reserva previa o no.

Reserva: elemento que representa la operación que realiza el operario a solicitud del cliente para reservar un auto en el tiempo que este desee.

Cliente: persona encarga de solicitar un contrato o una reserva.

Conductor: elemento disponible para que se pueda insertar el o los conductores que el cliente desee.

Accesorio: elemento disponible para que se pueda insertar el o los accesorios que el cliente desee o no rentar.

Depósito: elemento que representa la forma de pago en la que el cliente puede realizar la renta del auto.

Tarjeta de crédito: elemento que representa uno de los tipos de forma de pago.

Voucher: elemento que representa unos de los tipos de forma de pago.

Efectivo: elemento que representa unos de los tipos de forma de pago.

Cheque: elemento que representa un tipo de pago en efectivo.

Auto: elemento que representa el auto que se va ser rentado.

Estaciones: elemento que representa la estación de venta en que se realizó el contrato.

2.3 Requerimiento del sistema.

Luego de analizarse el dominio del problema, se definen los requerimientos del sistema, que son la manera de describir de forma clara y sin ambigüedades el comportamiento de la propuesta de solución. Estos se clasifican en requisitos funcionales (RF), que son las funcionalidades que deben cumplir el módulo y en requisitos no funcionales (RNF), que son las propiedades o cualidades que el producto debe tener.

2.3.1 Requisitos funcionales.

Se determinan como requisitos funcionales del módulo Ventas:

Tabla 2: *Requisitos funcionales.*

Código	Requisitos Funcionales
RF1	Adicionar reserva
RF2	Actualizar reserva
RF3	Eliminar reserva
RF4	Listar reserva
RF5	Adicionar contrato con reserva
RF6	Actualizar contrato con reserva
RF7	Eliminar contrato con reserva
RF8	Listar contrato con reserva
RF9	Adicionar contrato sin reserva
RF10	Actualizar contrato sin reserva
RF11	Eliminar contrato sin reserva
RF12	Listar contrato sin reserva
RF13	Adicionar cliente
RF14	Actualizar cliente
RF15	Eliminar cliente
RF16	Listar cliente
RF17	Adicionar conductor
RF18	Actualizar conductor
RF19	Eliminar conductor

RF20	Listar conductor
RF21	Adicionar accesorios
RF22	Actualizar accesorios
RF23	Eliminar accesorios
RF24	Listar accesorios
RF25	Adicionar depósito en efectivo
RF26	Actualizar depósito en efectivo
RF27	Eliminar depósito en efectivo
RF28	Listar depósito en efectivo
RF29	Adicionar depósito con tarjeta de crédito
RF30	Actualizar depósito con tarjeta de crédito
RF31	Eliminar depósito con tarjeta de crédito
RF32	Listar depósito con tarjeta de crédito
RF33	Adicionar código de autorizo
RF34	Actualizar código de autorizo
RF35	Eliminar código de autorizo
RF36	Listar código de autorizo
RF37	Adicionar depósito en Voucher
RF38	Actualizar depósito en Voucher
RF39	Eliminar depósito en Voucher
RF40	Listar depósito en Voucher
RF41	Listar contrato de renta de autos abiertos
RF42	Listar contrato de renta de autos cerrados
RF43	Listar reservas diarias
RF44	Listar reservas no efectuadas
RF45	Listar entradas planificadas
RF46	Listar salidas planificadas

2.3.2 Requisitos no funcionales.

Los requisitos no funcionales son características que hacen al producto atractivo, usable, rápido o confiable. Son importantes para que el cliente pueda valorar las características no funcionales del producto, pues si se conoce que el mismo cumple con toda la funcionalidad requerida, las

propiedades no funcionales, marca la diferencia entre un producto bien aceptado y uno con poca aceptación. (22)

A continuación, se muestran los requisitos no funcionales que debe cumplir el módulo. Para ello se debe tener en cuenta, que la propuesta de solución es parte de un sistema que hoy ya tiene definido entre otros aspectos, los requisitos no funcionales, por tanto, los que se exponen, constituyen definiciones de la dirección del proyecto SIGREX, por las cuales debe regirse también la propuesta de desarrollo.

- **Usabilidad.**

RNF-1. El módulo debe tener una interfaz amigable para los usuarios con conocimientos básicos en el manejo de una computadora y de un ambiente *web*.

Hardware.

RNF-2 Del lado del servidor la computadora empleada como mínimo debe tener:

- 1 Gb de RAM.
- 40 Gb de Disco Duro.
- Microprocesador a velocidad de 3.3 GHz.

RNF-3. Del lado del cliente las computadoras empleadas deben tener como mínimo:

- 1 Gb de RAM.
- Microprocesador de un núcleo a velocidad de 1.5 GHz.
- Tarjeta de red 100 Mb/s.

- ***Software.***

RNF-4. Del lado del servidor debe instalarse

- Intérprete de Python versión 2.7.11 o superior, y anterior a la versión 3.3.
- Marco de trabajo Django versión 1.10.
- PostgreSQL versión 9.4.

- **Seguridad.**

RNF-5. Los usuarios deben autenticarse antes de comenzar a interactuar con el sistema.

RNF-6. El sistema se encarga de controlar los diferentes niveles de acceso.

2.4 Historias de usuarios (HU).

Las HU son unas de las variantes que permiten encapsular los requisitos funcionales del sistema. A continuación, se muestran algunas de las más significativas (ver HU en su totalidad, en el **Anexo 1**).

Tabla 3: *HU_1 Gestionar reserva.*

Número: 1	**Nombre del requisito:** Insertar reserva, Modificar reserva, Eliminar reserva, listar reserva, crear un contrato
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá insertar, modificar, listar, eliminarlas reservas de REX y crear un contrato a partir de la reserva. Para registrar una reserva se necesitan los siguientes datos:

- **Número de reserva:** campo numérico de seis dígitos, de carácter obligatorio, que representa el identificativo de la reserva.
- **Cliente:** campo seleccionable, de carácter obligatorio, que representa el nombre del cliente.
- **Fecha/hora de inicio:** campo seleccionable, de carácter obligatorio, que representa la fecha/hora de inicio del servicio de la renta.
- **Fecha/hora de fin:** campo seleccionable, de carácter obligatorio, que representa la fecha/hora de fin del servicio de la renta.
- **Tipo de Servicio:** campo seleccionable, de valor booleano y de carácter obligatorio, que representa el tipo de servicio brindado al cliente.
    - **Servicio entrega:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le entrega el auto al cliente.
    - **Servicio de recogida:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le recoge el auto al cliente.
- **Estación de recibo:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente recibe el auto.
- **Estación de entrega:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente entrega el auto.

• **Vuelo de llegada:** campo alfanumérico, de carácter obligatorio, que representa el número de vuelo de llegada del cliente. • **Vuelo de salida:** campo alfanumérico, de carácter obligatorio, que representa el número vuelo de salida del cliente. • **Fecha de la solicitud:** campo generado automáticamente por el sistema.

Observaciones:
• El campo Tipo de servicio está definido por dos tipos:
  • Con Chofer
  • Traslado

Tabla 4: *HU_2 Gestionar contrato con reserva.*

Número: 2	**Nombre del requisito:** Insertar contrato con reserva, Modificar contrato con reserva, Eliminar contrato con reserva, listar contrato con reserva
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá insertar, modificar y listar los contratos de REX. Para registrar un contrato se necesitan los siguientes datos: • **Número de contrato (RA):** campo numérico de siete dígitos, de carácter obligatorio, que representa el identificativo del contrato. El primer dígito es una C y los restantes dígito corresponde a al número de la reserva separados por un -. • **Número de reserva:** campo numérico de seis dígitos que representa el identificador de la reserva realizada previamente (si procede). Este campo se autocompleta a partir de una entrada de dígitos. Los primeros dígitos hacen referencia a la estación de salida y los últimos tres dígitos números aleatorios. • **Estación de recibo:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente recibe el auto.	

- **Estación de entrega:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente entrega el auto.
- **Auto:** campo alfanumérico, de carácter obligatorio, que representa el identificador del auto en la flota de REX.
- **Kms. de Salida:** campo generado automáticamente por el sistema a partir del cierre del último DTI. Se registra el kilometraje que marca el auto en el momento de apertura del contrato.
- **Fecha/Hora de salida:** campo generado automáticamente por la fecha del sistema, con la hora y la fecha actual en el que se rentó el auto.
- **Fecha/Hora de entrada:** campo seleccionable que permite actualizar la fecha y hora planificada para la entrada del auto.
- **Precio Kms. Extra:** campo que mostrará automáticamente según la categoría del auto el precio a aplicar por los kilómetros extras recorridos.
- **Gasolina de salida:** campo que permite registrar la cantidad de combustible del auto en el momento de la salida. Como valor predeterminado visualiza la capacidad del tanque del modelo del auto seleccionado.
- **Forma de pago:** campo obligatorio y seleccionable que permite escoger las formas de pago, las cuales pueden ser de tipo *Voucher*, efectivo y tarjeta de crédito.

Otros datos adicionales:

- **Número de autorizo de Operaciones (Ventas):** cuadro de texto para insertar un código de autorizo para realizar *Walk in* en caso que sea preciso.
- **Conductor:** se muestran dos recuadros para visualizar los datos de choferes adicionales en caso que se requiera. El primero de ellos, accede a la interfaz donde se pueden insertar o seleccionar conductores ya registrados con anterioridad.
- **Por ciento de descuento:** en este campo se muestra el descuento que tiene el cliente en caso de que proceda, pero también el botón habilita un formulario para un eventual descuento autorizado por algún directivo.
- **Daños:** este botón da acceso a un nuevo formulario que permite insertar el estado técnico en que queda el auto. También permite insertar el valor de los daños.

Los datos que se registran en los daños son:

- **Estado del auto:** campo donde se selecciona el estado en que se encuentra el auto que pueden ser: fuera de servicio o disponible.
- **Motivo fuera de servicio:** campo donde se selecciona el motivo por el cual el auto esta fuera de servicio. Los posibles valores pueden ser: Mantenimiento, Problemas técnicos y Daños.
- **Valor del daño:** este campo donde se selecciona si el auto tiene seguro o no y además el monto del daño.
- **Precio por cantidad de conductores:** muestra estos datos en función de la cantidad de conductores que se registraron en el contrato.
- **Servicio de entrega:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le entrega el auto al cliente.
- **Servicio de recogida:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le recoge el auto al cliente.
- **Impuesto APO:** Impuesto a pagar si la estación de apertura, está ubicada en el aeropuerto.
- **Servicio 24 horas:** si este servicio se va a cobrar, debe seleccionar el *check box* que se muestra a su izquierda.
- **Multas de tránsito:** si existen multa este campo muestra el monto a pagar.
- **Penalidades:** si existen penalidades se muestra el listado de las penalidades y el monto total a pagar por las infracciones.

2.5 Análisis y Diseño.

2.5.1 Descripción del estilo arquitectónico.

Los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de *software* y proveen un conjunto de subsistemas predefinidos. Además, facilitan la adaptabilidad a requerimientos cambiantes, el rendimiento del sistema y acoplamiento, brindándole solución a las llamadas entre los objetos.

Entre los patrones arquitectónicos empleados en la construcción de aplicaciones *web* se encuentra el Modelo-Vista-Controlador (MVC), que separa el modelo de datos, la lógica de

control y las interfaces de usuario. Particularmente el *framework* Django utiliza como arquitectura base el patrón MVC.

Django es un *framework* MTV del inglés *Model-Template-View* que es una modificación de MVC (Modelo-Vista-Controlador). Para poder entender Django se debe tener en cuenta su analogía con MVC de la siguiente forma: (24)

- El modelo en Django sigue siendo Modelo (M).
- La vista en Django pasa a llamarse *Template* (T).
- El controlador en Django pasa a llamarse Vista (V).

El funcionamiento del MTV de Django representado en la **Figura 2**, se explica detalladamente:

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.

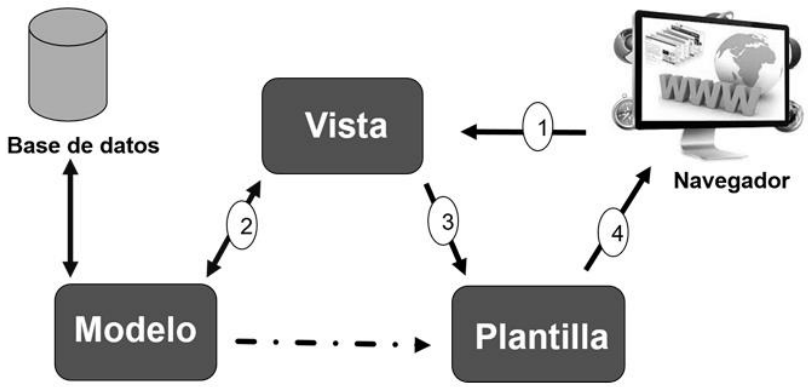


Figura 2: *Funcionamiento MTC de Django.*

De lo visto anteriormente acerca de cómo trabaja el MTV de Django, se derivan los siguientes elementos:

- **El modelo:** define lo datos almacenados, es representado en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros, también posee métodos. Todo esto permite indicar y controlar el comportamiento de los datos.
- **La vista:** su propósito es determinar qué datos serán visualizados, es representado en forma de clases. El ORM (*Object Relational Mapping*) de Django permite escribir código Python en lugar de SQL (*Structured Query Language*) para realizar las

consultas. También, se encarga de tareas como el envío de correo electrónico, autenticación con servicios externos y la validación de datos a través de formularios.

- **La plantilla:** recibe los datos de la vista y luego los organiza para la presentación al navegador *web*. Básicamente es una página HTML (*HyperText Markup Language*) con algunas etiquetas extras que son propias del Django, dichas etiquetas permiten flexibilidad para los desarrolladores de la interfaz.

De manera general cada una de las aplicaciones del sistema cuenta con un fichero **models.py** donde se definen todos los modelos, **views.py** que es donde se definen los controladores y una serie de archivos HTML que representan las vistas.

En **model.py** se definen clases, a través de las cuales se garantiza el mapeo de los datos, garantizando un trabajo más fácil para el manejo de los datos en la base de datos, por ejemplo: en el caso de la reserva existe una clase denominada “Reserva”, donde las funcionalidades que brinda el ORM de Django posibilita el trabajo con los datos de una reserva utilizando una instancia de esta clase. Las instancias de reserva siempre son generadas en el controlador, “**views.py**”, componente encargado de controlar el negocio utilizando instancias del modelo que también envía a las vistas. Las vistas son tratadas en un directorio denominado “*Templates*”, donde se almacenan los archivos HTML. Es necesario acotar que la relación entre las vistas y el controlador es tratada desde un componente denominado “**url.py**”.

Ejemplo de los componentes distribuidos en el módulo con la utilización del patrón MVC:

Modesl.py:

```
class Reserva(models.Model):
```

Views.py:

```
class ReservaCreation(CreateView):
class ReservaUpdate(UpdateView):
class ReservaDelete(DeleteView):
class ReservaList(ListView):
```

Templates:

```
reserva_form.html
reserva_delete.html
reserva_list.html
```

url.py:

```
url(r'^reserva/', ReservaList.as_view(), name='list_reserva'),
url(r'^reserva(?P<pk>\d+)$', ReservaDetail.as_view(), name='detail_reserva'),
url(r'^reserva_nuevo$', ReservaCreation.as_view(), name='new_reserva'),
```

```
url(r'^reserva_editar/(?P<pk>\d+)$', ReservaUpdate.as_view(), name='edit_reserva'),
url(r'^reserva_borrar/(?P<pk>\d+)$', ReservaDelete.as_view(), name='delete_reserva'),
```

2.5.2 Descripción de los patrones de diseño.

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. El mismo identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades. Estos modelos que se presentan como parejas de problema/solución con un nombre, codifican buenos principios y sugerencias relacionados con la asignación de responsabilidades, basados en la recopilación del conocimiento de los expertos en desarrollo de *software*. (25) Existen varios tipos de patrones entre los que se encuentran los patrones de creación y los patrones de asignación de responsabilidad. Entre los patrones de diseño utilizados se encuentran los siguientes:

Patrones de asignación de responsabilidades o Grasp (*General Responsabilty Assignment Software Patterns*): experto, bajo acoplamiento, creador y controlador.

- **Experto:**

Es un patrón que asigna responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Da origen a diseños donde el objeto de *software* realiza las operaciones que normalmente se aplican a lo que se desea representar, por lo que ofrece una analogía con el mundo real.

Con la utilización de este patrón se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando la creación de clases sencillas y más cohesivas que son fáciles de comprender y mantener.

```
class ClienteCreation(CreateView):
  model = Cliente
  form_class = ClienteForm
  success_url = reverse_lazy('ventas:list_cliente')
```

Figura 3: *Ejemplo de aplicación del patrón experto y creador.*

Un ejemplo se evidencia, en la clase *ClienteCreation* (ver **figura 3**), especializada en crear clientes con la utilización de la clase Cliente creada en el modelo.

- **Creador:**

El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar

con el objeto producido en cualquier evento. Brinda soporte a un bajo acoplamiento lo que supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

Este patrón es utilizado en el componente **views.py** donde a través de las clases creadas, se producen instancias para el manejo de los datos del módulo. Con el fin de mapear los datos almacenados o por almacenar.

- **Bajo acoplamiento:**

El acoplamiento indica el nivel de dependencia entre los diferentes módulos de un *software* de un sistema informático, se puede decir que es el grado en que un módulo puede funcionar sin recurrir a otro; dos funciones son absolutamente independientes entre sí (el nivel más bajo de acoplamiento) cuando una puede hacer su trabajo completamente sin recurrir a la otra. Una clase con bajo (o débil) acoplamiento no depende de muchas otras clases. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras.

Este patrón ya viene incluido con Django, que permite un bajo acoplamiento entre las piezas, lo que evita las dependencias, por ejemplo a la hora de realizar cambios en las configuraciones de las *URL*[2] en la Base de Datos, plantillas HTML, basta solo con realizarlo una sola vez.

- **Controlador:**

Un controlador es un objeto de interfaz no destinada al usuario. Se asocia a operaciones del sistema generadas por un actor externo o en respuesta a eventos del sistema.
El patrón controlador define quién debería encargarse de atender un evento del sistema. En el diseño de la propuesta de solución el componente encargado de desempeñar este papel es el componente ***views.py***, el cual es el encargado de controlar los diferentes eventos que rigen el comportamiento funcional del módulo.

2.7 Diagrama de clases del diseño

El diagrama de clases del diseño se realizó con el fin de representar estructura estática que describe gráficamente las clases e interfaces de la aplicación. Este contiene información como clases, asociaciones, atributos, interfaces con sus operaciones y constantes, métodos, navegabilidad y dependencias.

[2] *Uniform Resource Locator*

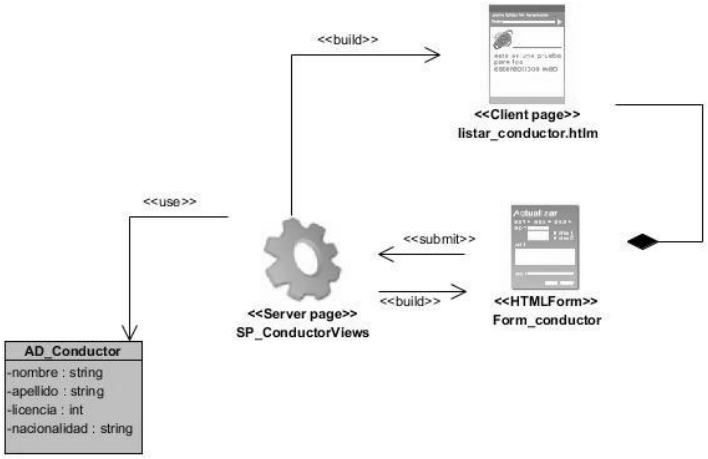


Figura 4: *Diagrama de clases de diseño.*

2.6 Modelo de datos.

Un modelo de datos consiste en un conjunto de herramientas conceptuales para describir la representación de la información en términos de datos. Los modelos de datos comprenden aspectos relacionados con: estructuras y tipos de datos, operaciones y restricciones.

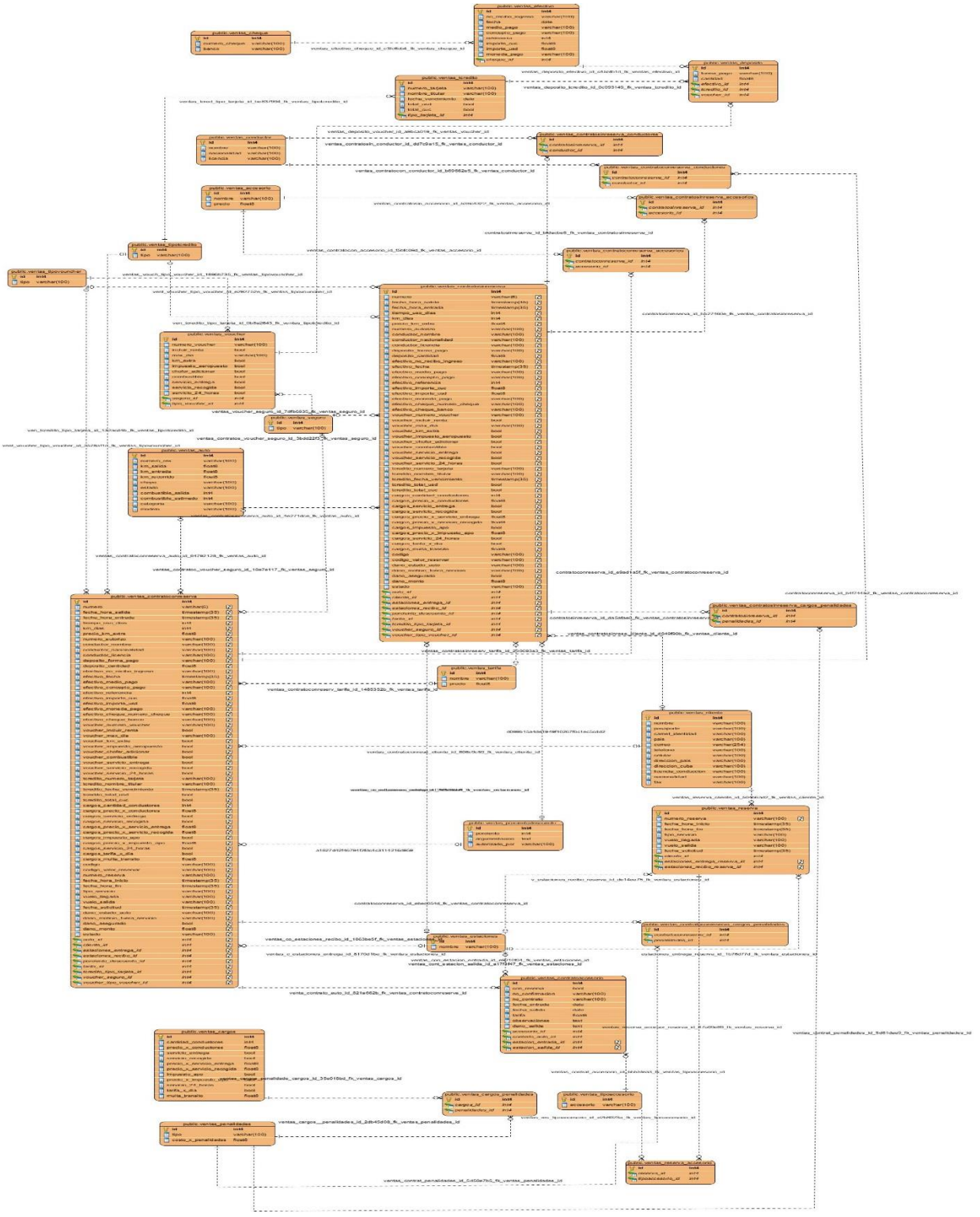


Figura 5: *Modelo de dato.*

2.7 Conclusiones parciales.

Con la especificación de los requisitos funcionales y no funcionales del sistema, se logró una mejor comprensión de los resultados que se pretenden obtener y sirvió de guía para la implementación del módulo. La representación y descripción de los artefactos generados, garantizaron un mejor entendimiento de los flujos de trabajos presentes en el proceso de la gestión de la información de las ventas en la Agencia de Renta de Vehículos REX. La definición de la arquitectura y los patrones de diseño a utilizar, permitieron establecer las bases para fomentar la reutilización y las buenas prácticas de programación, así como disminuir el impacto de los cambios futuros en el código fuente.

Capítulo III: Implementación y Validación del Sistema.

La fase de implementación constituye una parte transcendental en el proceso de desarrollo de un *software*, es en este momento donde se define y organiza el código de la propuesta de solución. Durante esta etapa se materializan, en forma de código, todos los artefactos de implementación, descripciones y arquitectura propuestos en la fase de análisis y diseño; que permite conformar el producto final requerido por el cliente.

Todo *software* debe ser puesto a prueba, para garantizar que cumpla con todos los estándares requeridos y con todas aquellas condiciones y funcionalidades que el cliente final necesita. A esta etapa se le conoce como validación del sistema y en ella, pueden realizarse diferentes tipos de pruebas en función de los objetivos de las mismas.

3.1 Estándares de codificación utilizados.

Un estándar de codificación no es más que un conjunto de pautas que tiene como objetivo uniformar la escritura del código fuente de un *software* para facilitar su legibilidad (25).

Las pautas fundamentales definidas para la implementación son las siguientes:

Indentación

- Las líneas de continuación deben alinearse verticalmente con el carácter que se ha utilizado (paréntesis, llaves, corchetes).
- Utilizar una indentación de una tabulación para cada línea con excepción de la primera.
- La indentación se realizará solamente con tabulaciones, no deben utilizarse nunca los cuatro espacios.

Máxima longitud de las líneas

- Todas las líneas deben estar limitadas a un máximo de setenta y nueve 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

- Separar las funciones de alto nivel y definiciones de clases con dos líneas en blanco.
- Las definiciones de métodos dentro de una clase deben separarse por una línea en blanco.
- Se pueden utilizar líneas en blanco escasamente para separar secciones lógicas.

Codificaciones

- Utilizar la codificación UTF-8.

Importaciones

- Las importaciones deben estar en líneas separadas.
- Siempre deben colocarse al comienzo del archivo.
- Deben quedar agrupadas de la siguiente forma:
  - Importaciones de la librería estándar.
  - Importaciones terceras relacionadas.
  - Importaciones locales de la aplicación / librerías.
- Cada grupo de importaciones debe estar separado por una línea en blanco.
- Evitar utilizar espacios en blanco en las siguientes situaciones:
  - Dentro de paréntesis, corchetes y llaves.
  - Antes de una coma, un punto y coma o dos puntos.
  - Antes del paréntesis que comienza la lista de argumentos en la llamada a una función.
  - Antes de un corchete que empieza una indexación.
  - Más de un espacio alrededor de un operador de asignación (u otro) para alinearlo con otro.

Espacios en blanco en expresiones y sentencias

- Deben rodearse con exactamente un espacio los siguientes operadores binarios:
  - Asignación (=).
  - Asignación de aumentación (+=, -=, etc.).
  - Comparación (==, <, >, >=, <=, !=, <>, in, not in, is, is not).
  - Expresiones lógicas (and, or, not).
- Si se utilizan operadores con prioridad diferente se aconseja rodear con espacios a los operadores de menor prioridad.
- 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, entre otros.
- Si un comentario es corto el punto final puede omitirse.

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.

Conversiones de nombramientos

- Los nombres de las clases, las funciones y las propiedades adoptarán la notación *UpperCamelCase* y no se utilizará el guión bajo para delimitar palabras.
- Los nombres de los atributos, las variables y los parámetros adoptarán la notación *lowerCamelCase.* Esta define que todas las palabras que conforman el nombre, excepto la primera, comenzarán con la primera letra en mayúscula y el resto de las letras en minúsculas.

3.2 Modelo de implementación.

El modelo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo éstos se organizan de acuerdo a los nodos específicos en el modelo de despliegue. La implementación comienza con el resultado del diseño y se implementa el sistema en términos de componentes.

Los artefactos principales generados en esta etapa, son el diagrama de despliegue, de paquete y el de componentes, que conforman lo que se conoce como el modelo de implementación, donde se describen los componentes, su organización y dependencias entre los nodos físicos en los que funcionará la aplicación.

3.2.1 Diagrama de paquetes.

El diagrama de paquetes en la presente investigación se realiza para mostrar cómo el módulo está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Se realiza con el objetivo de obtener una visión más clara del sistema de información orientado a

objetos, organizándolo en subsistemas, agrupando los elementos del análisis, diseño, construcción y detallando las relaciones de dependencia entre ellos.

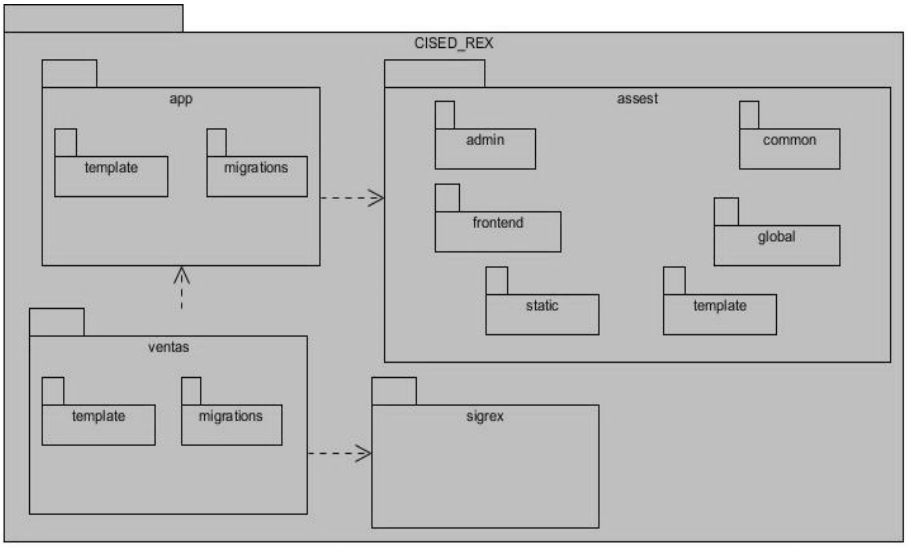


Figura 6: *Diagrama de paquete.*

3.2.2 Diagrama de componentes.

El diagrama de componentes se realizó con el fin de mostrar la estructura de alto nivel del modelo de implementación en términos de subsistemas de implementación, componentes y mostrar las relaciones entre los componentes que conforman la implementación.

Este diagrama también se utiliza para representar la vista estática de la aplicación, la estructura y las relaciones lógicas entre los componentes tales como código fuente, librerías, tablas, archivos y de forma general componentes que conforman el módulo Ventas.

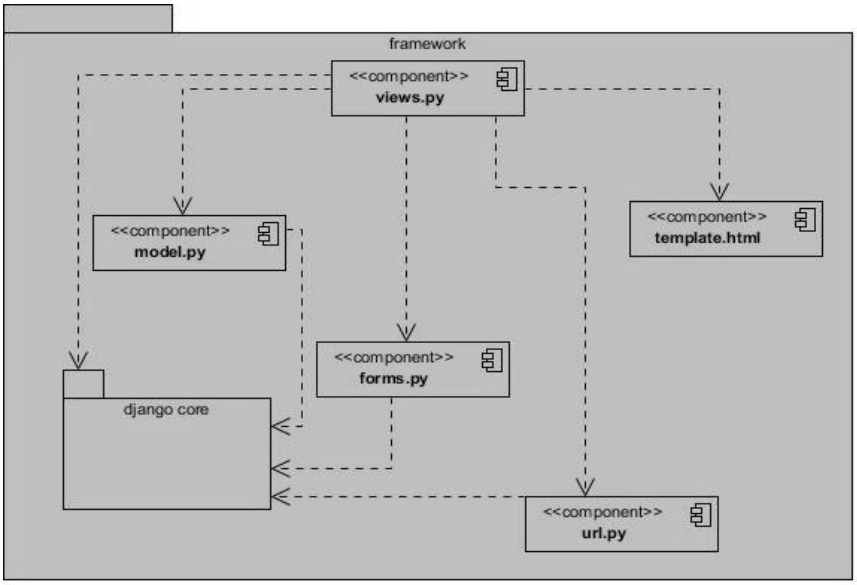


Figura 7: *Diagrama de componente.*

3.2.3 Diagrama de despliegue.

El diagrama de despliegue se utilizó para mostrar la estructura física del sistema, incluyendo las relaciones entre el *hardware* y el *software* que se despliega, estas relaciones son representadas por los protocolos de comunicación que se utilizan para acceder a cada uno. En la siguiente figura se visualiza el diagrama de despliegue definido para la propuesta de solución:

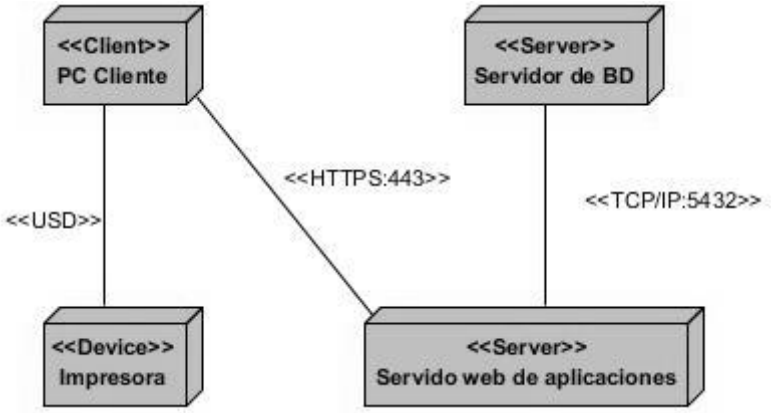


Figura 8: *Diagrama de despliegue.*

Descripción de los nodos

Nodo PC Cliente: representa una computadora desde la cual el usuario podrá acceder a la aplicación.

Nodo Servidor *web*: representa una estación donde estará instalada el servidor Apache, sobre el cual correrá la aplicación y los usuarios accederán a través de la maquina cliente.

Nodo Servidor de base de datos: representa el servidor donde estará el sistema gestor de base de datos, que dará respuesta a las peticiones hechas por la aplicación.

Nodo Impresora: representa el dispositivo donde se imprimen los contratos.

Descripción de elementos e interfaces de comunicación

- <<HTTPS>>:es el protocolo que representa la conexión segura que se establece entre una PC Cliente y un servidor de aplicaciones, esta conexión se realiza por el puerto 443.
- <<TCP/IP>>: la conexión entre el servidor *web* y el servidor de base de dato se hace a través de este protocolo y la conexión se realiza por el puerto 5432.
- <<USB>>: conexión que existe entre la impresora y la PC Cliente del usuario. Se modela con USB, pero puede existir otro tipo de conexión, todo depende del tipo de impresora que esté usando el usuario y el tipo de conexión que utilice esta.

3.2.4 Reseña de la implementación.

La implementación de la aplicación web comenzó con la creación de la estructura necesaria para el trabajo modular con el módulo Ventas dentro de SIGREX. Una vez definido el modelo en la herramienta CASE, fueron implementadas cada una de las clases del modelo con sus atributos. Posteriormente se definen los componentes visuales en la clase **forms.py**, la cual está destinada a visualizar los formularios, en el componente **forms.py** que representaran a cada uno de los atributos de las clases del modelo en las interfaces de usuarios del módulo.

```
class Reserva(models.Model):
  numero_reserva = models.CharField(max_length=100, null=True)
  fecha_hora_inicio = models.DateTimeField(default=datetime.now(), blank=True)
  fecha_hora_fin = models.DateTimeField(default=datetime.now(), blank=True)
  tipo_servicio = models.CharField(max_length=100, default='')
  vuelo_llegada=models.CharField(max_length=100, default='')
  vuelo_salida =models.CharField(max_length=100, default='')
  fecha_solicitud = models.DateTimeField(default=datetime.now(), blank=True)
  cliente = models.ForeignKey(Cliente)
  accesorio = models.ManyToManyField(TipoAccesorio)
  estaciones_recibo_reserva = models.ForeignKey(Estaciones, null=True,
lated_name='estaciones_recibo_reserva')
estaciones_entrega_reserva = models.ForeignKey(Estaciones, null=True,
related_name='estaciones_entrega_reserva')
```

```
    def __str__(self): # __unicode__ en Python 2
        return str(self.numero_reserva)
```

***Figura 9**: Fragmento de código del componente modelo.py*

```
class ReservaForm(forms.ModelForm):
    class Meta:
        model = Reserva
        fields = ['numero_reserva', 'fecha_hora_inicio', 'fecha_hora_fin', 'tipo_servicio',
'vuelo_llegada', 'vuelo_salida', 'fecha_solicitud',
                'cliente', 'accesorio', 'estaciones_recibo_reserva', 'estaciones_entrega_reserva']
        widgets = {
            'numero_reserva': TextInput(attrs={'class': 'form-control'}),
            'fecha_hora_inicio': DateTimeInput(attrs={'class': 'form-control'}),
            'fecha_hora_fin': DateTimeInput(attrs={'class': 'form-control'}),
            'tipo_servicio': TextInput(attrs={'class': 'form-control'}),
            'vuelo_llegada': TextInput(attrs={'class': 'form-control'}),
            'vuelo_salida': TextInput(attrs={'class': 'form-control'}),
            'fecha_solicitud': DateTimeInput(attrs={'class': 'form-control'}),
            'cliente': Select(attrs={'class': 'form-control'}),
            'accesorio': Select(attrs={'class': 'form-control'}),
            'estaciones_recibo_reserva': Select(attrs={'class': 'form-control'}),
            'estaciones_entrega_reserva': Select(attrs={'class': 'form-control'}),
        }
        labels = {
            'numero_reserva':'Numero de reserva',
            'fecha_hora_inicio': 'Fecha/hora de Inicio',
            'fecha_hora_fin': 'Fecha/hora fin',
            'tipo_servicio': 'Tipo de Servicio',
            'vuelo_llegada': 'Vuelo de Llegada',
            'vuelo_salida': 'Vuelo de salida',
            'fecha_solicitud': 'Fecha de Solicitud',
            'cliente': 'Cliente',
            'accesorio': 'accesorio',
            'estaciones_recibo_reserva': 'Estacion de recibo',
            'estaciones_entrega_reserva': 'Rstacion de Entrega',
        }
```

***Figura 10:** Fragmento de código del componente forms.py.*

Se definieron las clases que manejan los diferentes eventos que soportan las funcionalidades del módulo, entre las que se encuentra crear, editar, eliminar y listar los datos. Cada clase controladora de estos eventos, tiene asociado su modelo correspondiente y componentes que soportan su interfaz visual (*templates*).

```
class ReservaList(ListView):
    model = Reserva
```

```
class ReservaDetail(DetailView):
    model = Reserva

class ReservaCreation(CreateView):
    model = Reserva
    success_url = reverse_lazy('ventas:list_reserva')
    fields = ['numero_reserva', 'fecha_hora_inicio', 'fecha_hora_fin', 'tipo_servicio',
'vuelo_llegada', 'vuelo_salida', 'fecha_solicitud', 'cliente', 'accesorio',
'estaciones_recibo_reserva', 'estaciones_entrega_reserva']

class ReservaUpdate(UpdateView):
    model = Reserva
    success_url = reverse_lazy('ventas:list_reserva')
    fields = ['numero_reserva', 'fecha_hora_inicio', 'fecha_hora_fin', 'tipo_servicio',
'vuelo_llegada', 'vuelo_salida', 'fecha_solicitud', 'cliente', 'accesorio',
'estaciones_recibo_reserva', 'estaciones_entrega_reserva']

class ReservaDelete(DeleteView):
    model = Reserva
    success_url = reverse_lazy('ventas:list_reserva')
```

***Figura 11**: Fragmento de código del componente views.py.*

Mientras se crean las clases en el componente **views.py**, estas son asociadas a direcciones *web* para su renderización, a través de la *web* con la utilización del componente. **url.py**

```
url(r'^reserva/', ReservaList.as_view(), name='list_reserva'),
url(r'^reserva(?P<pk>\d+)$', ReservaDetail.as_view(), name='detail_reserva'),
url(r'^reserva_nuevo$', ReservaCreation.as_view(), name='new_reserva'),
url(r'^reserva_editar/(?P<pk>\d+)$', ReservaUpdate.as_view(), name='edit_reserva'),
url(r'^reserva_borrar/(?P<pk>\d+)$', ReservaDelete.as_view(), name='delete_reserva'),
```

***Figura 12**: Fragmento de código del componente urls.py.*

3.3 Pruebas funcionales.

Antes que el módulo Ventas se considere listo para desplegarse, debe someterse previamente a una etapa de pruebas, las cuales se realizan con el objetivo de llevar a cabo la ejecución de detectar errores, verificar que el *software* funcione como fue diseñado, validar y probar el cumplimiento de los requisitos.

3.3.1 Nivel de prueba.

Existen varios niveles de prueba, dentro de los que se pueden encontrar pruebas de unidad, de integración, de sistema y de aceptación donde cada uno contiene una técnica de prueba específica según los atributos de calidad que se deseen verificar. Las técnicas a su vez pueden

derivarse en distintos tipos de pruebas, las cuales utilizan métodos para llevar a cabo la ejecución de estas al *software.*

3.3.2 Tipos de prueba.

Los tipos de pruebas están asociados en dependencia de las pruebas de *software* que se desee realizar entre las que se encuentran las pruebas de rendimiento, pruebas de carga, pruebas funcionales, pruebas de regresión, entre otras.

Dentro de las pruebas de funcionalidad se utilizó las pruebas funcionales, que es un tipo de prueba que tiene como objetivo asegurar el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados.

3.3.3 Método de prueba.

Los métodos de pruebas se realizan en dependencia del tipo de pruebas que se estén realizando dentro de las pruebas funcionales se encuentran los métodos de caja negra y el de caja blanca. El método utilizado es el de Caja Negra que consiste en ejecutar cada requisito, usando datos válidos y no válidos, para verificar:

- Que se aplique apropiadamente cada regla de negocio.
- Que los resultados esperados ocurran cuando se usen datos válidos.
- Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.

Con las pruebas de caja negra se demostró que las funciones del *software* son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto.

3.3.4 Diseño de casos de prueba.

El método de prueba de caja negra se aplicó a la interfaz de la aplicación, este se realizó mediante los casos de prueba con el objetivo de identificar y comunicar las condiciones que se llevarán a cabo en la prueba ya que son necesarios para verificar la aplicación exitosa y aceptable de los requisitos del producto.

Dentro del método de caja negra se utiliza la técnica “Partición Equivalente” siendo considerada como una de las más efectivas en la evaluación de los valores válidos e inválidos, aunque pueden existir valores no relevantes a los que no sea necesario proporcionar un valor real de dato en las entradas existentes en la aplicación. La partición equivalente divide el dominio de entrada de un programa en clases de datos a partir de las cuales se derivan casos de prueba. En el **Anexo 2** se encuentran los casos de pruebas elaborados.

3.3.5 Resultados de las pruebas.

Para obtener los resultados se realizaron cuatro iteraciones de pruebas en los que se detectaron varias No Conformidades (NC). En el transcurso de las iteraciones la cantidad de errores tuvo una tendencia a disminuir hasta el punto de eliminar dichas NC en la cuarta iteración. A continuación, se muestra el resumen de los resultados de las pruebas:

***Tabla 5**: No conformidades.*

Iteraciones	No conformidades
1	15
2	8
3	5
4	0

Entre las No Conformidades más significativas se destacan las siguientes:

***Tabla 6:** Principales no conformidades.*

No. NC	Requisitos Funcionales	Tipo de NC	Etapa de la iteración	Impacto	Estado con respecto a la solución
1	Gestionar contrato con reserva.	Errores de validación.	1ra etapa	Alta	Resuelta y aprobada
2	Gestionar contrato sin reserva.	Errores de validación.	2da etapa	Alta	Resuelta y aprobada
3	Gestionar cliente.	Errores de validación.	3ra etapa	Alta	Resuelta y aprobada

3.4 Prueba de rendimiento.

Las pruebas de rendimiento están diseñadas para comprobar este aspecto de calidad en tiempo de ejecución dentro del contexto de un sistema integrado. Se dan durante todos los pasos del proceso de pruebas y permiten además descubrir situaciones que lleven a degradaciones y posibles fallos del software. (22)

Con la realización de las pruebas de rendimiento fue posible hallar tendencias y comportamientos de los elementos de la propuesta de solución, las cuales se generan bajo carga y estrés. Este tipo de pruebas permitió identificar cuellos de botella, capacidad de concurrencia

de usuarios, tiempos de respuesta de operaciones de negocio, establecer un marco de referencia para pruebas futuras, entre otros.

Pruebas de estrés: Mediante las pruebas de estrés es posible identificar la capacidad de respuesta de un sistema bajo condiciones de carga extrema, representadas por una alta concurrencia de usuarios o procesos (23).

Prueba de carga: Mediante la ejecución de las pruebas de carga es posible identificar la capacidad de recuperación de un sistema cuando es sometido a cargas variables tanto de usuarios como de procesos, determinando el tiempo de respuesta de todas las transacciones críticas del sistema (23).

Para la realización de las pruebas de carga y estrés, se empleó la herramienta Apache JMeter. El ambiente de prueba estuvo conformado por:

- Sistema Operativo: Windows 10
- Microprocesador: Intel(R) Core(TM) i3-2120 CPU @ 3.30 HGz 3.30 HGz
- Memoria RAM: 1GB
- Disco Duro: 4 TB
- Tipo de Conexión: Ethernet 10/100Mbps.

Además, se tuvo en cuenta que la agencia REX está conformada por veintiuna estaciones de ventas y que el módulo será utilizado por un solo usuario, en cada estación. Teniendo en cuenta esta cantidad de usuarios concurrente en el sistema, se decide realizar la prueba de rendimiento con un número de usuario cuatro veces superior a los usuarios promedio que utiliza el sistema.

Resultados y análisis de las pruebas de carga y estrés.

La prueba se definió para 84 hilos de concurrencia, los cuales simulan 84 usuarios accediendo concurrentemente al sistema. Se simularon un total de 420 peticiones a cinco direcciones del módulo en el servidor, en el tiempo mínimo de respuesta es de 49 milisegundo. En la **Figura 13** se puede observar los resultados.

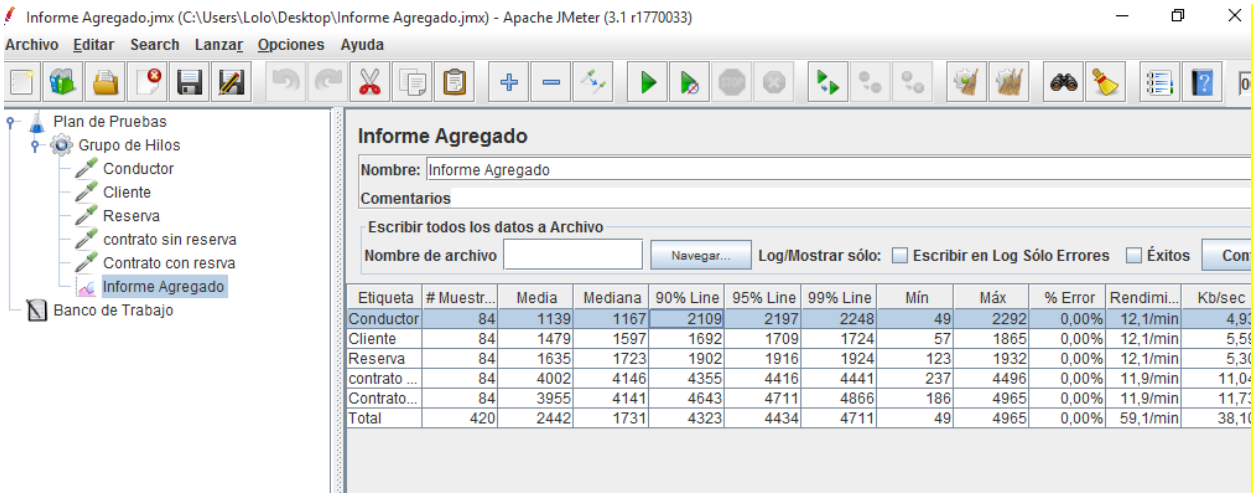


***Figura 13**: Valores obtenidos para una muestra de 84 usuarios.*

Descripción de los parámetros evaluados:

Muestras: cantidad de hilos utilizados para la URL.

Media: tiempo promedio en milisegundos para un conjunto de resultados.

Min: tiempo mínimo que demora un hilo en acceder a una página.

Max: tiempo máximo que demora un hilo en acceder a una página.

%Error: porciento de error de las respuestas de las peticiones.

Rendimiento: rendimiento medido en los requerimientos por segundo / minuto / hora.

Kb/s Recibidos: rendimiento medido en *Kbyte* por segundo.

3.5 Prueba de integración.

La prueba de integración es una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es coger los módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. (26)

Dado que es el módulo Ventas utiliza los datos de otros módulos, el enfoque seleccionado para realizar las pruebas de integración es el ascendente, que empieza con la construcción y prueba de los módulos de los niveles más bajos de la estructura del programa.

Resultados de las pruebas de integración.

Las pruebas de integración realizadas al módulo Venta demostró la integración que existe entre este módulo y los módulos restantes de SIGREX, un ejemplo de esto se puede apreciar en la

Figura 14 donde se ve la dependencia que existe entre la entidad reserva con el módulo Flota a través del atributo **auto** y con el módulo Comercial a partir del atributo **accesorio**.

```
class Reserva(models.Model):
    numero_reserva = models.CharField(max_length=100, null=True)
    fecha_hora_inicio = models.DateTimeField(default=datetime.now(), blank=True)
    fecha_hora_fin = models.DateTimeField(default=datetime.now(), blank=True)
    tipo_servicio = models.CharField(max_length=100, default='')
    vuelo_llegada=models.CharField(max_length=100, default='')
    vuelo_salida =models.CharField(max_length=100, default='')
    fecha_solicitud = models.DateTimeField(default=datetime.now(), blank=True)
    cliente = models.ForeignKey(Cliente)
    accesorio=models.ManyToManyField(TipoAccesorio)
    estaciones_recibo_reserva=models.ForeignKey(Estaciones,null=True
    related_name='estaciones_recibo_reserva')
    estaciones_entrega_reserva=models.ForeignKey(Estaciones,null=True,
    related_name='estaciones_entrega_reserva')
```

Figura 14: *Clase reserva.*

3.6 Conclusiones parciales.

Tomando como punto de partida el diseño propuesto en el capítulo anterior y unido a las herramientas, metodología, tecnologías empleadas y estándares de codificación empleados, se desarrolló un módulo Ventas del SIGREX que permite contribuir a la gestión de las rentas autos. Además, se definieron las pruebas que se realizaron al sistema para validar la calidad de las funcionalidades del mismo.

.

Conclusiones

Una vez finalizada la investigación que sirvió de base para el desarrollo del módulo Ventas para el Sistema de Gestión para la Agencia de Renta de Vehículos REX, se puede concluir que:

- A través del estudio de los principales sistemas de renta de autos existentes, se logró comprender las características del funcionamiento de este proceso.
- La selección de la metodología AUP-UCI utilizada para el desarrollo del sistema permitió la guía del proceso y posibilitó la creación de los artefactos fundamentales que permitieron la construcción del módulo Ventas, con una alta calidad.
- Se obtuvo una aplicación *web* funcional probada y validada durante su proceso de desarrollo utilizando pruebas funcionales, de rendimiento, desarrollada con tecnologías libres y provista de elementos de seguridad y un entorno amigable.
- Mediante la implementación y validación de la solución desarrollada, se constató que el sistema facilita el proceso de gestión de la información de las rentas de autos de la agencia REX.

Recomendaciones

El autor de la investigación recomienda:

- Actualizar el módulo Ventas con el desarrollo de las políticas correspondientes para el manejo tarifas y costos.

Referencias Bibliográficas

1. Isasi-Genix A, Gómez-Acosta MI, Stuart-Cárdenas ML. *Diseño del proceso de implementación de software en DESOFT.* Habana : s.n., 2012.

2. Carolyn A Lin, David J Atkin. *Communication technology and social change: Theory and implications.* 2014.

3. Hernández, Martha Beatriz. Las tecnología de la información y la idustria del turismo. [En línea] junio 9, 2014. gestiopolis.com.

4. *Lenguaje e información.* Alonso, Dolores Vizcaya. s.l. : Revista de Ciência da Informação, Agosto de 2001, Vol. Vol. 2.

5. *Teoría de los sistemas de información.* Langefors, Börje. Buenos Aire : s.n., 1976.

6. *Gestión - Información - Conocimiento.* Murray, Pablo. Lima : BIBLIOS, Noviembre de 2002,.

7. Bartle, Phil. Información para la gestión y gestión de la información. [En línea] CEC Community Empowerment Collective., 01 2011, 10. www.scn.org/mpfc/modules/mon-miss.html.

8. The British Standards Institution. [En línea] 2012. http://www.bsigroup.com.mx/es-mx/Auditoriay-Certificacion/Sistemas-de-Gestion/De-un-vistazo/Que-son-los-sistemas-de-gestion/..

9. globalwebtek. 2003. . globalwebtek. [En línea] 2003. globalwebtek.com/productos/control-de-alquiler-de-vehiculos..

10. FloreSoft. FloreSoft. [En línea] 2007. www.floresssoft.com..

11. Teleformación . Dirección de Entorno Virtual de Aprendizaje. METODOLOGIAS TRADICIONALES VS. METODOLOGIAS AGILES. [En línea] 2013. http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodologias/METODOLOGIAS_TRADICION.

12. . *Metodología de desarrollo para la UCI.* Sánchez, Tamara Rodríguez. habana : s.n., 2015.

13. Larman, C. *UML y Patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 2004.

14. Perissé, C.M. Herramientas case. [En línea] 2010. http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm...

15. Grupo de Soluciones Innova. . Rational Rose Enterprise. [En línea] 2007. http://www.rational.com.ar/herramientas/roseenterprise.html...

16. Cortez Vásquez, MsC. Augusto, Vega Huerta, MsC. Hugo and Pariona Quispe, Lic. Jaime. Revista de investigación de Sistemas e Informática. Procesamiento de lenguaje natural. [En línea] 2016. http://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/5923/5121.

17. González, Raúl Duque. Paython para todos . [En línea] 2010.

18. Darly, Brad. *Python Phrasebook.* 2007.

19. *Sistema web de control y monitoreo de flotas.* Luis López Fillad, Jonathan Vega González. 2013.

20. *El libro de Django.* holovaty, adrian y Kaplan-Moss, Jacob. 2008.

21. Developers, PyCharm. Python IDE for Professional. PyCharm. Python IDE for Professional Developers. [En línea] 2016. [Citado el: 11 10, 2016.] http://www.jetbrains.com/pycharm/.

22. *Base de Datos en postgresSQL.* [En línea] 2010. http://ocw.uoc.edu/computer-science-technology-and-multimedia/bases-de-datos/bases-de- datos/P06_M2109_02152.pdf. .

23. Torets, Jose Manuel López. V&V Quality. Pruebas de seguridad. [En línea] 2016. [Citado el: 12 20, 2016.] http://vyvquality.com/pruebas-seguridad/.

24. Real Academia Española. Real Academia Española. [En línea] 2010. [Citado el: enero 24, 2017.] http://dle.rae.es/?w=diccionario.

25. Busca Palabras. *Busca Palabras.* [En línea] mayo 15, 2016. [Citado el: 12 5, 2016.] buscapalabra.com/definiciones.html?palabra=trasiego.

26. Cuba Travel Network. *Renta de autos en Cuba.* [En línea] 2017. [Citado el: Enero 18, 2017.] http://www.cubatravelnetwork.com/es/autos/alquilar_auto_cuba.asp.

27. lenguaje de programación. [En línea] 2016. http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

28. Mastermagazine. *Definición de UML.* [En línea] 2008. http://www.mastermagazine.info/termino/7006.php. .

29. Schulz R, Beach SR, Matthews JT, Courtney K, Dabbs AD, Mecca LP. *Willingness to Pay for Quality of Life Technologies to Enhance Independent Functioning Among Baby Boomers and the Elderly Adults. The Gerontologist.* [ed.] 54(3):363-74. 2014.

30. Isasi-Genix A, Gómez-Acosta MI, Stuart-Cárdenas ML. *Diseño del proceso de implementación de software en DESOFT .* habana : s.n., 2012.

31. Saavedras López, Dismey. *Representación del conocimiento causal. Aplicaciones en la medicina.Informática Salud 2013;.* universidad de las ciencias Informática, habana : 2012.

32. Carolyn A Lin, David J Atkin. *Communication technology and social change: Theory and implications.* 2014.

33. Hernández, Martha Beatriz. Las tecnología de la información y la idustria del turismo. [En línea] julio 9, 2014. gestiopolis.com.

34. Galindos, R. *Guión Visual Paradigm for UML.* 2014.

35. *Entornos de Desarrollo Integrado (IDE´s).* [En línea] 2013. http://carlosblanco.pro/2012/04/entornos-desarrollo-integrado-introduccion/..

36. Grupo de Soluciones Innova. Rational Rose Enterprise. [En línea] 2007. http://www.rational.com.ar/herramientas/roseenterprise.html..

37. Targetware Informática S.A.C. . software. *Visual Paradigm para UML.* [En línea] 2010. http://www.software.com.ar/visual- paradigm-para-uml.html..

38. *UML en 24 Horas.*

39. Teleformación, Dirección de. Entorno Virtual de Aprendizaje. *METODOLOGIAS TRADICIONALES VS. METODOLOGIAS AGILES.* [En línea] 2013. http://eva.uci.cu/file.php/161/Documentos/Materiales_complementarios/UD_1_Procesos/Metodolo gias/METODOLOGIAS_TRADICIONALES_VS._METODOLOGIAS_AGILES.pdf..

40. Thompson, Ivan. Promonegocios. [En línea] agosto 2005. [Citado el: noviembre 25, 2016.] http://www.promonegocios.net/venta/.

Anexos

Anexo 1: Historias usuarios.

Tabla 7: *HU_3 gestionar contrato sin reserva.*

Número: 3	**Nombre del requisito:** Insertar contrato sin reserva, Modificar contrato sin reserva, Eliminar contrato sin reserva, listar contrato sin reserva
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá insertar, modificar y listar los contratos sin reserva de REX. Para registrar un contrato se necesitan los siguientes datos:

- **Número de contrato (RA):** campo numérico de 7 dígitos, de carácter obligatorio, que representa el identificativo del contrato. El primer dígito es una C y los restantes dígito corresponde a al número de la reserva separados por un -.
- **Estación de recibo:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente recibe el auto.
- **Estación de entrega:** campo seleccionable, de carácter obligatorio, que representa el lugar en donde el cliente entrega el auto.
- **Auto:** campo alfanumérico, de carácter obligatorio, que representa el identificador del auto en la flota de REX.
- **Kms. de Salida:** campo generado automáticamente por el sistema a partir del cierre del último DTI. Se registra el kilometraje que marca el auto en el momento de apertura del contrato.
- **Fecha/Hora de salida:** campo generado automáticamente por la fecha del sistema, con la hora y la fecha actual en el que se rentó el auto.
- **Fecha/Hora de entrada:** campo seleccionable que permite actualizar la fecha y hora planificada para la entrada del auto.

- **Precio Kms. Extra:** campo que mostrará automáticamente según la categoría del auto el precio a aplicar por los kilómetros extras recorridos.
- **Gasolina de salida:** campo que permite registrar la cantidad de combustible del auto en el momento de la salida. Como valor predeterminado visualiza la capacidad del tanque del modelo del auto seleccionado.
- **Forma de pago:** campo obligatorio y seleccionable que permite escoger las formas de pago, las cuales pueden ser de tipo *Voucher*, efectivo y trajeta de crédito.

Otros datos adicionales:

- **Número de autorizo de Operaciones (Ventas):** cuadro de texto para insertar un código de autorizo para realizar *Walk in* en caso que sea preciso.
- **Conductor:** se muestran dos recuadros para visualizar los datos de choferes adicionales en caso que se requiera. El primero de ellos, accede a la interfaz donde se pueden insertar o seleccionar conductores ya registrados con anterioridad.
- **Por ciento de descuento:** en este campo se muestra el descuento que tiene el cliente en caso de que proceda, pero también el botón habilita un formulario para un eventual descuento autorizado por algún directivo.
- **Daños:** este botón da acceso a un nuevo formulario que permite insertar el estado técnico en que queda el auto. También permite insertar el valor de los daños.

Los datos que se registran en los daños son:

- **Estado del auto:** campo donde se selecciona el estado en que se encuentra el auto que pueden ser: fuera de servicio o disponible.
- **Motivo fuera de servicio:** campo donde se selecciona el motivo por el cual el auto esta fuera de servicio. Los posibles valores pueden ser: Mantenimiento, Problemas técnicos y Daños.
- **Valor del daño:** este campo donde se selecciona si el auto tiene seguro o no y además el monto del daño.
- **Precio por Cantidad de conductores:** muestra estos datos en función de la cantidad de conductores que se registraron en el contrato.

- **Servicio de entrega:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le entrega el auto al cliente.
- **Servicio de recogida:** campo requerido cuando el tipo de servicio es con chofer. Campo seleccionable, de carácter obligatorio, que representa el lugar en donde se le recoge el auto al cliente.
- **Impuesto APO:** Impuesto a pagar si la estación de apertura, está ubicada en el aeropuerto.
- **Servicio 24 horas:** si este servicio se va a cobrar, debe seleccionar el *check box* que se muestra a su izquierda.
- **Multas de tránsito:** si existen multa este campo muestra el monto a pagar.
- **Penalidades:** si existen penalidades se muestra el listado de las penalidades y el monto total a pagar por las infracciones.

Tabla 8: *HU_4 Gestionar cliente.*

Número: 4	**Nombre del requisito:** Insertar cliente, Modificar cliente, Eliminar cliente, listar cliente	
Programador: No Aplica		**Iteración Asignada:** 1
Prioridad: Alta		**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica		**Tiempo Real:** No Aplica

Descripción:

Se permitirá insertar, modificar y listar los clientes de REX. Para registrar un contrato se necesitan los siguientes datos:

- **Nombre:** campo de solo letras, de carácter obligatorio, que representa el nombre del cliente.
- **Pasaporte:** campo numérico entero de 10 dígitos, de carácter obligatorio, que representa el número de pasaporte del cliente.

- **Carnet de identidad(CI):** campo numérico entero de 11 dígitos, de carácter obligatorio, que representa el número de carnet de identidad del cliente.
- **País:** campo seleccionable, de carácter obligatorio, que representa el país que pertenece el cliente.
- **Correo: c**ampo alfanumérico, que representa el correo del cliente que realiza la reserva.
- **Teléfono:** campo numérico, de carácter obligatorio, que representa el número del contacto del cliente que realiza la solicitud de reserva.
- **Nacionalidad:** campo de solo letras, de carácter obligatorio, que representa la nacionalidad del cliente.
- **Dirección(Cuba):** campo alfanumérico, que representa la dirección en la que el cliente va a establecerse en Cuba.
- **Dirección(País):** campo alfanumérico, que representa la dirección donde él reside en su país.
- **País de residencia:** campo de solo letras, de carácter obligatorio, que representa la residencia del cliente.
- **Número de licencia:** campo numérico, de carácter obligatorio, que representa el número de la licencia de conducción del cliente.

Tabla 9: *HU_5 Gestionar accesorios.*

Número: 5	**Nombre del requisito:** Insertar accesorios, Modificar accesorios, Eliminar accesorios, listar accesorios
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá insertar, modificar y listar los accesorios de REX. Para registrar un accesorio se necesitan los siguientes datos:

- **Con reservación:** campo booleano, que representa si el accesorio o no está reservado.
- **Tipo de accesorio:** campo donde se seleccionan el tipo de accesorio.

• **Número de REX:** campo numérico de 6 dígito, de carácter obligatorio, que representa el número que identifica al accesorio. • **Precio:** campo numérico que representa el precio del accesorio. • **Daño de salida:** campo de letra que representa los daños que presenta el accesorio inicialmente.

Tabla 10: *HU_6 Gestionar conductor.*

Número: 6	**Nombre del requisito:** Insertar conductor, Modificar conductor, Eliminar conductor, listar conductor
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá insertar, modificar y listar los conductores. Para registrar un conductor se necesitan los siguientes datos: • **Nombre:** campo de solo letras, de carácter obligatorio, que representa el nombre del conductor. • **Nacionalidad:** campo de solo letras, de carácter obligatorio, que representa la nacionalidad del conductor. • **Número de licencia de conducción:** campo numérico, de carácter obligatorio, que representa el número de la licencia de conducción del conductor.	

Tabla 11*: HU_7 Gestionar depósito en efectivo.*

Número: 7	**Nombre del requisito:** Insertar depósito en efectivo, Modificar depósito en efectivo, Eliminar depósito en efectivo, listar depósito en efectivo
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica

Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá insertar, modificar y listar los depósitos en efectivo. Para registrar los depósitos en efectivo se necesitan los siguientes datos: • **Número de recibo de ingreso (RI):** campo numérico de 4 dígitos, de carácter obligatorio, que representa el identificativo del recibo de ingreso. • **Fecha:** campo generado automáticamente por la fecha del sistema, con la hora y la fecha actual en que se realizan los depósitos. • **Moneda de pago:** campo seleccionable que hace referencia a la moneda con la que se va a realizar el pago que puede ser en CUC o USD. • **Importe en CUC:** campo numérico, de carácter obligatorio, que representa el importe a cobrar en CUC. • **Importe USD:** campo numérico, de carácter obligatorio, que representa el importe a cobrar en CUC. • **Medio de pago:** campo seleccionable que hace referencia a la forma de pago que puede ser en efectivo o en cheque. Si en caso de que el depósito a realizar sea a través de un cheque se registran los siguientes datos: • **Número del cheque:** campo alfanumérico, de carácter obligatorio que representa el número del cheque. • **Banco:** campo de solo letra, de carácter obligatorio, que representa el banco que proviene el cheque.	

Tabla 12: *HU_8 Gestionar depósito en tarjeta de crédito.*

Número: 8	**Nombre del requisito:** Insertar depósito en tarjeta de crédito, Modificar depósito en tarjeta de crédito, Eliminar depósito en tarjeta de crédito, listar depósito en tarjeta de crédito
Programador: No Aplica	**Iteración Asignada:** 1

Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá insertar, modificar y listar los depósitos en tarjeta de crédito. Para registrar un depósito en tarjeta de crédito se necesitan los siguientes datos: • **Nombre del titular:** campo de letras, de carácter obligatorio, que representa el nombre del dueño de la tarjeta. • **Número de la tarjeta de crédito:** campo numérico, de carácter obligatorio, que representa el número de la tarjeta. • **Fecha de vencimiento:** campo seleccionable, de carácter obligatorio, que representa la fecha en que vence la tarjeta de crédito. • **Tipo de tarjeta:** campo seleccionable, de carácter obligatorio, que representa el tipo de la tarjeta de crédito. • **Total, de CUC:** campo seleccionable, de carácter obligatorio, que representa la cantidad a cobrar en CUC de la tarjeta de crédito. • **Total, de USD:** campo seleccionable, de carácter obligatorio, que representa la cantidad a cobrar en USD de la tarjeta de crédito.	

Tabla 13: *HU_9 Gestionar códigos de autorizo.*

Número: 9	**Nombre del requisito:** Insertar código de autorizo , Modificar código de autorizo, Eliminar código de autorizo, listar código de autorizo
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción:	

Se permitirá insertar, modificar y listar los códigos de autorizo. Para registrar los códigos de autorizo se necesitan los siguientes datos: • **Código de autorizo:** campo alfanumérico, de carácter obligatorio, que representa el código de autorización. • **Valor a reservar:** campo numérico, que representa el valor a reservar por cada una de las tarjetas de crédito. • **Monto CUC:** campo numérico, que representa el monto en CUC a retirar de la tarjeta de crédito. • **Monto USD:** campo numérico, que representa el monto en USD a retirar de la tarjeta de crédito.

Tabla 14: *HU_10 Gestionar depósito en Voucher.*

Número: 10	**Nombre del requisito:** Insertar depósito en Voucher, Modificar depósito en Voucher, Eliminar depósito en Voucher, listar depósito en Voucher
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá insertar, modificar y listar depósito en *Voucher*. Para registrar el depósito en *Voucher* se necesitan los siguientes datos: • **Tipo de *Voucher*:** campo que es una lista desplegable, de carácter obligatorio, que contiene los 4 tipos: *Full Credit*, Renta, Renta y Seguro, y Reserva. • **Número de *Voucher*:** campo alfanumérico, de carácter obligatorio, que representa el número que identifica al *voucher*. • **Incluir renta:** campo booleano, de carácter obligatorio, que representa si se va incluir o no a la renta el *voucher*. Elementos incluidos en la renta**:** • **Kms extras:** campo booleano, que representa si se va incluir o no a Kms extras.	

- **Chofer incluido:** campo booleano, que representa si se va incluir o no a chofer.
- **Seguro:** campo booleano, que representa si se va incluir o no seguros al auto y si selecciona, hay que seleccionar el tipo de seguro.
- **Impuestos de aeropuertos:** campo booleano, que representa si se va incluir o no los impuestos de aeropuertos.
- **Servicio de entrega (*Pick Up*):** campo booleano, que representa si se va incluir o no el servicio de entrega del auto.
- **Servicio de recogida (*Dropn Of*):** campo booleano, que representa si se va incluir o no el servicio de recogida del auto.
- **Combustible:** campo booleano, que representa si se va incluir o no el servicio de combustible.
- **Servicio de 24 horas:** campo booleano, que representa si se va incluir o no el servicio de 24 horas.

Tabla 15: *HU_11 Listar contratos de rentas de autos abiertos.*

Número: 11	**Nombre del requisito:** Listar contratos de renta de autos abiertos.
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá listar el informe de contratos de renta de autos abiertos. Del mismo se visualiza la siguiente información:

- **Número de contrato (RA):** campo que representa el número del contrato.
- **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa.
- **Auto:** campo que representa el número del auto rentado.
- **Fecha/Hora de salida** campo que representa la fecha y hora de salida del auto de la estación.

• **Fecha/Hora de entrada planeada:** campo que representa la fecha y hora de la entrada planificada del auto. • **Cliente:** campo que representa el nombre y apellido del cliente.

Tabla 16: *HU_12 Listar contratos de renta de autos cerrado.*

Número: 12	**Nombre del requisito:** Listar contratos de renta de autos cerrados.
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá listar el informe de contratos de renta de autos cerrados. Del mismo se visualiza la siguiente información: • **Número de contrato (RA):** campo que representa el número del contrato. • **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa. • **Auto:** campo que representa el número del auto rentado. • **Fecha/Hora de salida** campo que representa la fecha y hora de salida del auto de la estación. • **Fecha/Hora de entrada planeada:** campo que representa la fecha y hora de la entrada planificada del auto.	

Tabla 17: *HU_13 Listar reservas diarias.*

Número: 13	**Nombre del requisito:** Listar reservas diarias.
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica

Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá listar el informe de reservas diarias. Del mismo se visualiza la siguiente información: • **Número de contrato (RA):** campo que representa el número del contrato. • **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa. • **Fecha/Hora de inicio:** campo que representa la fecha y hora de inicio de la reserva. • **Fecha/Hora de entrada fin:** campo que representa la fecha y hora fin de la reserva. • **Cliente:** campo que representa el nombre y apellido del cliente.	

Tabla 18: *HU_14 Listar reservas no realizadas.*

Número: 14	**Nombre del requisito:** Listar reservas no realizadas..
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica
Descripción: Se permitirá listar el informe de reservas no realizadas. Del mismo se visualiza la siguiente información: • **Número de contrato (RA):** campo que representa el número del contrato. • **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa. • **Fecha/Hora de inicio:** campo que representa la fecha y hora de inicio de la reserva. • **Fecha/Hora de entrada fin:** campo que representa la fecha y hora fin de la reserva. • **Cliente:** campo que representa el nombre y apellido del cliente.	

Tabla 19: *HU_15 Listar entradas planificadas.*

Número: 15	**Nombre del requisito:** Listar entradas planificadas.
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá listar el informe de entradas planificadas. Del mismo se visualiza la siguiente información:

- **Número de contrato (RA):** campo que representa el número del contrato.
- **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa.
- **Auto:** campo que representa el número del auto rentado.
- **Fecha/Hora de salida** campo que representa la fecha y hora de salida del auto de la estación.
- **Fecha/Hora de entrada planeada:** campo que representa la fecha y hora de la entrada planificada del auto.
- **Cliente:** campo que representa el nombre y apellido del cliente.

Tabla 20: *HU_16 Listar salidas planificadas.*

Número: 16	**Nombre del requisito:** Listar salidas planificadas.
Programador: No Aplica	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** No Aplica
Riesgo en Desarrollo: No Aplica	**Tiempo Real:** No Aplica

Descripción:

Se permitirá listar el informe de entradas planificadas. Del mismo se visualiza la siguiente información:

- **Número de contrato (RA):** campo que representa el número del contrato.

- **Número de reserva:** campo que representa el número de reserva si el contrato tiene una reserva previa.
- **Auto:** campo que representa el número del auto rentado.
- **Fecha/Hora de salida** campo que representa la fecha y hora de salida del auto de la estación.
- **Fecha/Hora de entrada planeada:** campo que representa la fecha y hora de la entrada planificada del auto.
- **Cliente:** campo que representa el nombre y apellido del cliente.

Anexo 2: Casos de pruebas.

Tabla 21: *Caso de prueba de la HU gestionar reserva.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E1	Crear reserva	Crea la reservas y va al listado de las reserva incluyendo la creada.	Ventas/reserva
E2	Editar reserva	Edita la reserva y va para el listado de la reserva.	Ventas/reserva
E3	Eliminar reserva	Elimina la reserva y va para el listado de las reservas.	Ventas/reserva
E4	Listar reserva	Muestra el listado de todas las reservas realizadas donde se listan con los siguientes campos: Número de reserva. Fecha/hora de salida. Fecha/hora de entrada.	Ventas/reserva

***Tabla 22**: Caso de prueba de la HU gestionar contrato con reserva.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E4	Crear contrato con reserva	Crea el contrato con reserva y va al listado de los contratos con reserva incluyendo el creado.	Ventas/Contrato con reserva
E5	Editar contrato con reserva	Edita el contrato con reserva y va para el listado de los contratos con reserva	Ventas/Contrato con reserva
E6	Eliminar contrato con reserva	Elimina el contrato con reserva y va para el litado de los contratos con reserva.	Ventas/Contrato con reserva
E7	Listar contrato con reserva	Muestra el listado de todos los contratos con reserva realizados que se listan con los siguientes campos: Número de reserva. Número del contrato. Fecha/hora de Entrada. Fecha/hora de salida	Ventas/Contrato con reserva

***Tabla 23:** Caso de prueba de la HU gestionar contrato sin reserva.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central

E8	Crear contrato sin reserva.	Crear el contrato sin reserva y va para el listado de los contrato sin reserva incluyendo el creado.	Ventas/Contrato sin reserva
E9	Editar contrato sin reserva	Edita el contrato sin reserva y va para el listado de los contratos sin reserva.	Ventas/Contrato sin reserva
E10	Eliminar contrato sin reserva	Elimina el contrato sin reserva y va para el listado de los contratos sin reserva.	Ventas/Contrato sin reserva
E11	Listar contrato	Muestra el listado de todos los contratos sin reserva realizadas donde se listan con los siguientes campos: Número del contrato. Fecha/hora de Entrada. Fecha/hora de salida	Ventas/Contrato sin reserva

Tabla 24: *Caso de prueba de la HU gestionar cliente.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E12	Crear cliente	Crea el cliente y va al listado de los clientes incluyendo el creado.	Ventas/cliente
E13	Editar cliente	Edita el cliente va para el listado de los clientes.	Ventas/cliente

E14	Eliminar cliente	Elimina el cliente y va para listado de clientes.	Ventas/cliente
E15	Listar cliente	Muestra el listado de todos los clientes donde se listan con los siguientes campos: Nombre. Pasaporte.	Ventas/cliente

Tabla 25: *Caso de prueba de la HU gestionar conductor.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E16	Crear conductor	Crea el conductor y va al listado de los conductores incluyendo el creado.	Ventas /Conductor
E17	Editar conductor	Edita el conductor y va para el listado de los conductores.	Ventas /Conductor
E18	Eliminar conductor	Elimina el conductor y va para listado de conductores.	Ventas /Conductor
E19	Listar conductor	Muestra el listado de todos los conductores donde se listan con los siguientes campos: Nombre. Nacionalidad. Licencia.	Ventas /Conductor

Tabla 2726: *Caso de prueba de la HU listar reservas diarias.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E20	Listar reservas diarias	Muestra el listado de todas las reservas diarias donde se listan con los siguientes campos: Número de reserva. Fecha/hora de inicio. Fecha/hora fin.	Ventas/ Lista de reservas diarias

Tabla 28: *Caso de prueba de la HU listar reservas no efectuadas.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E21	Listar reservas no efectuadas	Muestra el listado de todas las reservas no efectuadas donde se listan con los siguientes campos: Número de reserva. Fecha/hora de inicio. Fecha/hora fin.	Ventas/ Lista de reservas no efectuadas

Tabla 29: *Caso de prueba de la HU listar entradas planificadas.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E30	Listar entradas planificadas.	Muestra el listado de todas las entradas planificadas que se listan	Ventas/ Lista de entradas planificadas.

		con los siguientes campos	

***Tabla 27**: Caso de prueba de la HU listar salidas planificadas.*

Id del escenario	Escenario	Respuesta del sistema	Flujo central
E31	Listar salidas planificadas.	Muestra el listado de todas las salidas planificadas que se listan con los siguientes campos	Ventas/ Lista de salidas planificadas.