



UNIVERSIDAD AUTÓNOMA DE ENCARNACIÓN

DESARROLLO DE APLICACIÓN WEB PARA LA ADMINISTRACIÓN DE INMUEBLES

Orlando Javier Cardozo Benítez

Tesis presentada a la Facultad de Ciencia, Arte y Tecnología como requisito para la obtención del título de Licenciado/a en Análisis de Sistemas Informáticos.

Encarnación, Paraguay

Diciembre, 2018

Orlando Javier Cardozo Benítez

**DESARROLLO DE APLICACIÓN WEB PARA LA ADMINISTRACION DE
INMUEBLES**

Tesis presentada a la Facultad de Ciencia, Arte y
Tecnología, Universidad Autónoma de
Encarnación (UNAE) como requisito para la
obtención del título de Licenciado/a en Análisis
de Sistemas Informáticos.

Línea de investigación: Administración de
Negocios

Orientador/a:

Encarnación, Paraguay

Diciembre, 2018

Autorizo la reproducción total o parcial de este trabajo, por cualquier medio convencional o electrónico, para fines de estudio e investigación, siempre que sea citada la fuente.

Los hechos e ideas expresados en este trabajo de investigación son de responsabilidad exclusiva del/la autor/a.

FICHA CATALOGRÁFICA

Cardozo Benítez, Orlando Javier (2018). Desarrollo De Aplicación Web para la administración de inmuebles. Encarnación, Universidad Autónoma de Encarnación, 63p.

Orientadoras: Ing. Gabriela Cuba

Tesis presentada a la Facultad de Ciencia, Arte y Tecnología, Universidad Autónoma de Encarnación (UNAE) como requisito para la obtención del título de Licenciado/a en Análisis de Sistemas Informáticos.

Línea de investigación: Ingeniería de Software

Palabras claves: Aplicación Web Interno, Administración.

ORLANDO JAVIER CARDOZO BENITEZ

**DESARROLLO DE APLICACIÓN WEB PARA LA ADMINISTRACIÓN DE
INMUEBLES**

Tesis presentada a la Facultad de Ciencia, Arte y Tecnología, Universidad Autónoma de Encarnación (UNAE) como requisito para la obtención del título de Licenciado/a en Análisis de Sistemas Informáticos.

Línea de Investigación: Administración Inmobiliaria

Orientador/a: Ing. Gabriela Cuba

Aprobado en (lugar) _____, el/...../.....

Calificación: (en números) y letras.....

Sínodo Examinador

Prof. _____

Firma: _____

Prof. _____

Firma: _____

Prof. _____

Firma: _____

Dedico esta tesis a:

A Dios por el don de la vida,

A mis padres por el soporte incondicional y la
confianza puesta en mí.

A mis hermanos y amistades por sus alientos para
seguir adelante en momentos difíciles.

A los profesores por su vocación en la enseñanza
para con nosotros.

A los compañeros por agregarse a escalar juntos
esta montaña llamada carrera universitaria.

“Tanto si piensas que puedes, como si piensas que no puedes, estás en lo cierto”. Ford H.

Tabla de contenido

| | |
|---|-----------|
| Planteamiento del Problema | 5 |
| Preguntas Específicas de Investigación..... | 6 |
| Objetivos..... | 6 |
| General..... | 6 |
| Específicos | 6 |
| Justificación | 7 |
| Revisión de Literatura | 8 |
| Antecedentes | 8 |
| Tucondominioaldia..... | 8 |
| Residentia..... | 8 |
| AdmiCond Ultimate 8.5 | 9 |
| Software para administración de inmuebles en Encarnación. | 9 |
| Bases teóricas..... | 10 |
| Gestión de Edificios | 10 |
| Inmuebles..... | 10 |
| Gestión Administrativa..... | 11 |
| Software..... | 11 |
| Gestores de Base de Datos..... | 13 |
| Sistemas de Gestión..... | 14 |
| Software Open Source | 14 |
| Software libre | 14 |
| Estado del arte de Lenguajes de programación..... | 15 |
| Frameworks de Programación | 16 |
| Gestores de Base de Datos..... | 18 |
| Metodología o Materiales y Métodos..... | 20 |
| Definición del tipo y diseño de investigación..... | 20 |
| Instrumentos y Técnicas de Recolección de Datos | 20 |
| Procedimientos de Aplicación de Instrumento | 20 |
| Metodología de Desarrollo de Software..... | 21 |
| Delimitación del Alcance del Software | 21 |
| Limitaciones..... | 23 |
| El sistema no contempla pagos parciales..... | 23 |
| No contempla roles..... | 23 |
| El modulo pagos no cuenta con impresión de factura pre-impresión legal. | 23 |
| No se cuenta con tipos de pago. | 23 |
| No se incluirán más funcionalidades de las ya especificadas en el alcance. | 23 |
| Resultados y Discusión | 24 |
| Herramientas utilizadas | 24 |

| | |
|--|-----------|
| Análisis y planificación | 25 |
| Figuras de las interfaces de la aplicación desarrollada..... | 26 |
| Módulo login..... | 27 |
| Modulo Caja | 28 |
| Modulo Edificios..... | 29 |
| Modulo Inquilinos | 31 |
| Modulo Contratos..... | 32 |
| Modulo Recibos | 34 |
| Modulo Gastos | 35 |
| Conclusión | 38 |
| Recomendaciones | 40 |
| Lista de Referencias | 41 |
| Anexos | 44 |
| Anexo 1: Enlace al proyecto..... | 44 |
| Anexo 2: Modelado del Sistema..... | 44 |
| Anexo 3: Entrevista utilizada..... | 44 |
| Anexo 4: Estructura del proyecto..... | 45 |
| Anexo 5: Repositorio local SmartGit..... | 46 |
| Anexo 6: Código del Proyecto- Controlador inquilino | 47 |
| Anexo 7: Código del Proyecto – Controlador Edificios-Departamentos..... | 48 |
| Anexo 8: Código del Proyecto – Controlador Contratos | 50 |
| Anexo 9: Código del Proyecto –Gastos..... | 51 |
| Anexo 10: Código del Proyecto –Controlador Cajas | 53 |
| Anexo 11: Código del Proyecto –Controlador Movimiento Caja..... | 54 |
| Anexo 12: Código del Proyecto –Controlador Pagos..... | 56 |
| Anexo 13: Código del Proyecto –Controlador Recibos..... | 57 |
| Anexo 14: Código del Proyecto –Controlador Detalle_recibo | 59 |
| Anexo 15: Código del Proyecto –Reporte pdf del recibo | 60 |
| Anexo 16: Código del Proyecto –Routes..... | 62 |

LISTA DE FIGURAS

| | |
|--|----|
| Figura 1 – Logo de RoR..... | 24 |
| Figura 2 – Logo de herramientas utilizadas | 25 |
| Figura 3 – Logo del versionador de códigos..... | 25 |
| Figura 4 – Logo de Trello | 26 |
| Figura 5 – Módulo de autenticación | 26 |
| Figura 7 – Interfaz Apertura de caja | 27 |
| Figura 8 – Interfaz Informe de Caja | 28 |
| Figura 9 – Interfaz Registro de Edificio con sus departamentos..... | 29 |
| Figura 10 – Interfaz Listado Edificios | 29 |
| Figura 11 – Interfaz Registro de Inquilino | 30 |
| Figura 12 – Interfaz Listado de Inquilinos..... | 31 |
| Figura 13 – Interfaz Registro de Contrato | 32 |
| Figura 14 – Interfaz Lista de Contratos | 32 |
| Figura 15 – Interfaz de Vista preliminar de Recibos..... | 33 |
| Figura 16 – Interfaz de Vista preliminar de Recibo en formato PDF..... | 34 |
| Figura 17 – Interfaz de Registro de Gastos..... | 35 |
| Figura 18 – Interfaz de Lista de Gastos | 35 |
| Figura 19 – Logo de Heroku | 36 |
| Figura 20 – Logo de Interserver | 36 |
| Figura 21 – Logo de MilesWeb..... | 37 |

LISTA DE ABREVIATURAS

| | |
|------|---------------------------|
| RoR | Ruby on Rails |
| MVC | Modelo, Vista Controlador |
| POJO | Plain Old Java Object |
| DSL | Domain Specific Language |

Desarrollo de una Aplicacion Web Para la Administracion de Inmnuebles

Cardozo Benítez, Orlando Javier

Ing. Gabriela Cuba

Eje temático: TIC e Innovación

RESUMEN

La presente investigación presenta el desarrollo de una Aplicación Web para la administración de Inmuebles. Dicho abordaje es el resultado de una investigación previa que resalto la carencia de herramientas administrativas en dicho rubro. El objetivo de este trabajo fue la de Desarrollar una aplicación web para las distintas gestiones administrativas necesarias en la administración de inmuebles. El tipo de investigación utilizada fue la investigación aplicada cuasi-experimental utilizando la metodología de desarrollo ágil Kanban. El ágil progreso del proyecto fue mediante el uso del lenguaje Ruby junto con la implantación del framework "Ruby on Rails". La persistencia de datos finalmente fue desarrollada en el motor MySQL por la envergadura de la aplicación. Como resultante se ha logrado agilizar el proceso de gestión de datos del inquilino. La incorporación del módulo de contratos y cajas permiten una correcta gestión de plazos acorde a los documentos, junto a la administración de precios, disponibilidad y pagos de arrendadores. El sinfín de posibilidades de ampliación para adaptarlo a prácticamente cualquier emprendimiento con características similares, volvió a esta aplicación en una herramienta con características indispensable en la administración de inmuebles para emprendedores de la región.

Palabras clave: Administración, Web, Kanban .

Jeypyso Jererekokuaa Ogajehepyme'ê Web Jeipypuru Rupive

Apohára: Cardozo Benítez, Orlando Javier

Sambyhyhára: Ing Gabriela Cuba

Tembikuaareka rape: TIC ha mba'epyahu

ÑEMOMBYKY

Ko tembiapo jeporeka ome'ê tenda pe "Jeipyso jererekokuaa ogajehepyme'ê web jeipuru rupive"sa'igui oî ohecháva lo mba'e.

Mba'apo rendápe jererekokuaa rupi ha umi tenda ndeikatúigui ojaopópa umi jerekohárákuéra omba'apo jave.

Ko tembiapo jehupytyrã ha'e pe Jeipyso web jeipyru opa omba'apóva ko'ã jererekokuaahárape. Upe jehekapy ojeipurúva ha'e jehekapy jeipuru ojejeiporekáre opa kuatiañe'êita digitalkuéra rupive. Ojejapyhy pe mba'eichaite (metodología) jeipyso kanban.Ñe'ê Ruby iframework ndive "Ruby on Rails"ojeiporavo chugui marandu MYSQL gui.

Ko mba'e oheja oñembopya'eve umi ohepyme'ehàra rembiapo ojeinformatisagui marandukuéra ko'ã vore rupive ojegueraha ojehechaukáma umi tembiaporajohararã mboy jasýpa oîta,arangue ha araka'épa ohepyme'eva'erã ha mba'eichaite upe óga mboy departamento ,mboýpa ohupyty ha mboýpa oguereko viru, ha avei oñeñangareko upe oikéva jeipurutaháre.

Ñe'ê okëndavoka: Jererekokuaa ,Web, Kanban

Development of a Web Application for the Property Administration

Author: Cardozo Benítez, Orlando Javier

Advisor: Ing. Gabriela Cuba

Research Line: TIC and Innovation

SUMMARY

The present investigation presents the development of a Web Application for the administration of Real Estate. This approach is the result of a previous investigation that highlighted the lack of administrative tools in this area. The objective of this work was to develop a web application for the different administrative procedures required in the administration of real estate. The type of research used was quasi-experimental applied research using the Kanban agile development methodology. The agile progress of the project was through the use of the Ruby language along with the implementation of the "Ruby on Rails" framework. The persistence of data was finally developed in the MySQL engine by the scope of the application. As a result, the tenant's data management process has been streamlined. The incorporation of the module of contracts and boxes allows a correct management of deadlines according to the documents, together with the administration of prices, availability and payments of landlords. The endless possibilities of expansion to adapt it to practically any enterprise with similar characteristics, returned to this application in a tool with indispensable characteristics in the administration of real estate for entrepreneurs of the region.

Keywords: Administration, Web, Kanban.

Desarrollo de una Aplicacion Web Para la Administracion de Inmnuebles

El tema de investigación trata sobre el desarrollo de una aplicación web para la administración de inmuebles específicamente para edificios de departamentos direccionado a aquellos encargados de gestionar los procesos manuales que implican tales tareas como tener datos del inquilino, copia de los contratos, fechas de pago, gastos de mantenimiento (agua y luz), automatizar la generación de recibo por sus pago y tener los registros de los mismos etc.

El resultado del trabajo fue la automatización de todas las diversas tareas que requieren realizar manualmente por un administrador. A través del sistema culminado considero un aporte a la comunidad de consultores local y regional por la manera en que la misma se adapta a la gran mayoría de las reglas y procedimientos necesarios para llevar a cabo una correcta administración de los inmuebles ya obtenidos al momento de recabar datos, analizar y diseñar el software.

Planteamiento del Problema

A través del relevamiento de datos de una organización encargada de la gestión administrativa de edificios ubicada en la ciudad de Encarnación Barrio Ciudad Nueva domiciliado en la calle Ana Cheraniuk casi Avenida San Blas denominada Arrúa Emprendimientos, se ha hallado en ella la gran necesidad de proporcionar una solución automatizable que ayude al personal existente a facilitar los procedimientos.

Actualmente la organización viene llevando tales trabajos con planillas electrónicas Excel y Word en algunos casos que hoy día ya no condicen adecuadamente con la necesidad de rapidez y de tener información precisa sobre el inquilino y el inmueble en cuestión

Las series de gestiones administrativas que se llevan a cabo manualmente tales como la información de los inquilinos, sus estados de cuenta, la generación de recibos de pago, los cálculos por mora, informe de pagos hechos, que la gran mayoría de los administradores de edificios no cuentan con una herramienta que le pueda facilitar automatizando los procesos con lo que, hace que una tarea conlleve mayor dedicación de tiempo y más dificultad de registro mediando libros y/o bibliorátos, al momento de requerir alguna información de algún inquilino en especial tendríamos que recurrir físicamente a dichos documentos requiriendo mucho tiempo y desorden en la búsqueda del mismo.

Cabe la necesidad infalible de contar con informes claros y concisos que se requerirán cada mes al propietario del inmueble y también al inquilino si así lo necesite, con la misma se desea resguardar todos los datos existentes referentes a los mismos dando tranquilidad al administrador

Siendo así porque no buscar la manera en que podríamos tenerlo en alguna plataforma para la agilización de las gestiones necesarias sobre el inmueble.

Preguntas Específicas de Investigación

1. ¿Qué información existe sobre gestión administrativa de la organización que administra el inmueble?
2. ¿Cómo se podría agilizar las gestiones de un administrador de inmuebles?
3. ¿Qué elementos tiene un administrador de inmueble para digitalizar la información sobre los inquilinos?
4. ¿Puede llevarse con alguna herramienta web gastos extras de un edificio de departamentos?

Objetivos

General. Desarrollar una aplicación web que permita automatizar los procesos de una empresa dedicada a la administración de inmueble.

Específicos.

- Identificar y describir las herramientas de desarrollo del software para la implementación del proyecto.
- Identificar las posibles herramientas existentes para la gestión administrativa de inmuebles en la organización.
- Desarrollar el sistema web para la administración de inmuebles.
- Analizar los servidores web existentes donde alojar la aplicación.

Justificación

El resultado de la presente investigación tendrá una gran relevancia para los administradores de los departamentos, e inmobiliarias a través de la automatización de los procesos como cobros de mensualidad, manutención, registros sobre el estado de cuenta de cada inquilino, informes sobre los contratos, cálculo por intereses de mora, que llevan a cabo manualmente, tomándoles una gran carga de tiempo y trabajo desorganizado por tenerlo necesariamente realizar en planillas electrónicas y otros tipos gestiones.

En la propuesta de solución se gestionará la administración de los departamentos en forma automatizada, la cual contará con un sistema web para que los administradores puedan realizar en forma virtual las transacciones como los pagos de mantenimiento, consultas de pagos.

Otro punto destacable es que el software desarrollado es de código abierto (open source) es la posibilidad de compartir, modificar y estudiar el código fuente de un sistema informático. Por otro lado, el código abierto promueve la colaboración entre usuarios.

Revisión de Literatura

Antecedentes

Se identificaron diversas herramientas que sirven para la administración de inmuebles que de alguna manera ya existen en el mercado internacional que son descritas a continuación:

Tucondominioaldia

Ayuda incorporada, respaldo completo y asesoría permanente.

Envía por correo electrónico los recibos, pagos realizados, con solo hacer un clic.

Flexible, sin límite de inmuebles. Pruébalo antes de comprar (sin renovaciones de licencia).

Desarrollado para llevar de una manera fácil la Gestión de gastos y cobranzas de condominios. Ambiente Windows, gráfico, amigable e intuitivo, que le permite al usuario integrarse a las herramientas, métodos disponibles en el sistema y obtener todo el potencial desde un primer contacto.

Desarrollada para la administración de condominios. Un pago único, sin renovaciones de licencia, ni límite de inmuebles. (Pc, 2018)

Residentia

Es un software de Administración de Condominios que funciona vía Internet, diseñado para el beneficio tanto de los Administradores como de los habitantes de fraccionamientos, privadas, edificios, residenciales o conjuntos cerrados que pagan mensualmente una cuota de mantenimiento para cubrir necesidades comunes.

Esta herramienta ayuda a los vecinos a mantenerse informados de todo lo que acontece en su conjunto, mediante la publicación de comunicados, logros, avisos por parte del Administrador, así como solicitudes por parte de los condóminos.

Integra un mecanismo eficaz para que el administrador o tesorero tengan el control tanto de los

pagos de mantenimiento realizados por los residentes, como los gastos efectuados mes con mes. (Residentia, Residentia, 2018).

AdmiCond Ultimate 8.5

Con funciones mejoradas, tanto para Profesionales, Administradores Independientes de Condominios, como también para las Comunidades o Condominios administrados bajo la modalidad de Auto Gestión.

Esta versión es conocida como “Software Ecológico”, en virtud a que puede enviar a través de Internet todos los documentos que genera, incluyendo de los Estados de Cuenta o Recibos de Condominios, lo que representa un gran avance en el abaratamiento de costos, que, al no requerir el uso de papel, tampoco es necesario el uso de las impresoras y sus accesorios.

Quizá el logro más importante es que el envío de los documentos se hace de manera automática y directamente al correo electrónico a cada propietario, sin importar su ubicación geográfica, ahorrándole la necesidad de consultar saldo en páginas web o de hacer costosas llamadas.

Es la solución ideal para los Condominios Auto Gestionados porque ha sido diseñada especialmente para ello. (Servicios, 2018)

Software para administración de inmuebles en Encarnación.

A través de la información captada en la ciudad de Encarnación utilizando entrevistas y encuestas citadas en la sección tipos de procedimientos de aplicación de instrumentos a algunas consultoras e inmobiliarias encargadas al rubro de administración no se registran sistemas con características similares, de hecho no existen historial de que se llegase a utilizar herramientas tecnológicas para el efecto dando lugar a cubrir una necesidad en ese sector económico.

Bases teóricas**Gestión de Edificios**

La gestión de edificios es una disciplina que se encuentra bajo la clasificación del facility management. Un gestor de edificios supervisa los servicios "hard" y "soft" de una infraestructura, asegurando que la seguridad, salubridad y mantenimiento de un edificio sean satisfactorios para las personas que habitan o trabajan en él. Hay dos tipos de gestión de edificios: comercial y residencial.

Los "hard services" son aquellos que tienen relación con el estado físico de los edificios (los sistemas de alarmas contra incendios o los ascensores), mientras que los "soft services" hacen referencia a las tareas de limpieza, seguridad u otras actividades similares, donde el elemento principal son los recursos humanos. (Association, 2008)

Inmuebles

Los inmuebles, específica y propiamente dichos, son las tierras y edificios. Por concepto similar y ampliativo lo son también todas las cosas que con los inmuebles se relacionan o que dependen de los mismos, tales como los caminos y construcciones adheridos al suelo, así como el subsuelo y el espacio ambiental superior al suelo mismo; los árboles, las plantas, los frutos y cosechas pendientes mientras estuvieren unidos a la tierra o a los árboles formando parte integrante de una finca; todo lo que esté unido a un inmueble de una manera fija, de suerte que no pueda separarse de él sin quebrantamiento de la materia o deterioro del objeto; las estatuas, relieves, pinturas u otros objetos de uso u ornamentación, colocados en tal forma que revele el propósito de unidos de un modo permanente al inmueble; las conducciones de agua, de fuerza, de luz, los canales, oleoductos, máquinas, recipientes, instrumentos o utensilios destinados a la

industria o explotación que se realice en el inmueble, así como los aparatos necesarios para utilizados, etc. (Empresas, s.f.)

Gestión Administrativa

Es la forma en la que se utilizan los recursos escasos para conseguir los objetivos deseados. Se realiza a través de 4 funciones específicas planeación, organización, dirección y control.

En los últimos años se han creados nuevos sistemas de gestión que automatizan los procesos con el fin de mejorar la calidad y la eficacia en las tareas que realiza, estos sistemas son los informáticos que permite la integración de los distintos procesos de manera a agilizar la circulación de la información y los documentos. (Principios de la Gestion Administrativa, 2009)

Software

Según Pressman, Roger S. en su libro de Ingeniería del Software- Un enfoque práctico define software como “Un producto y al mismo tiempo es el vehículo para entregar un producto. En su forma de producto, brinda el potencial de cómputo incorporado en el hardware de cómputo o, con más amplitud, en una red de computadoras a las que se accede por medio de un hardware local”

Se lo puede definir como el producto de los desarrolladores que después le dan mantenimiento a largo plazo con actualizaciones (Pressman, El Software y la Ingenieria del Software, 2010).

Tipos de Software

Software de sistema.

Está formado por todos aquellos programas cuya finalidad es servir al desarrollo o al funcionamiento de otros programas, se caracterizan por estar próximos al hardware, procesan por

sobre todo datos indeterminados así también estructura de información compleja (Aguilera, 2015).

Se caracteriza por la integración con el hardware de la computadora, por el uso intenso de usuarios múltiples, recursos compartidos, la administración de un proceso sofisticado.

Software de tiempo real.

Está formado por todos aquellos programas que miden, analizan y controlan los sucesos del mundo real a medida que ocurren, debiendo de reaccionar de forma correcta a los estímulos de entrada en un tiempo máximo prefijado.

Hace énfasis en el tiempo que realiza o en que se produce las acciones y lleva más en cuenta la rapidez o que las acciones se produzcan en el momento adecuado. No solo deberán de funcionar correctamente la parte lógica.

Dependiendo del nivel de exigencia temporal se pueden clasificar en: Críticos, esenciales, incrementables y no esenciales. Así también dependiendo de la arquitectura: propietarios y abiertos. Dependiendo de la arquitectura del sistema centralizados y distribuidos (Aguilera, 2015).

Software de Gestión.

Estos programas utilizan grandes cantidades de información almacenadas en base de datos con objeto de facilitar las transacciones comerciales o la toma de decisiones. Además de las tareas convencionales de procesamiento de datos, en las que el tiempo de procesamiento no es crítico y los errores pueden ser corregidos.

Son herramientas que sirven de manera a gestionar las tareas y agilizarlas. La mayoría lo utiliza como un medio para compartir informaciones. Como ser a través de los Blogs, fueron creadas para coordinación de información de modo a poder crearlas, modificarlas, organizarlas y potenciar sus esfuerzos (Aguilera, 2015).

Software científico y de ingeniería.

Se encarga de realizar cálculos complejos sobre datos numéricos de todo tipo. En este caso la corrección y exactitud de las operaciones que realizan es uno de los requisitos básicos que deben de cumplir.

Sin embargo estas están abandonando los algoritmos convencionales y el diseño asistido por computadora, la simulación de sistemas y otras aplicaciones interactivas han comenzado a hacerse en forma real (Aguilera, 2015).

Frameworks de Desarrollo

Se refiere a una estructura de software integrado por componentes personalizables e intercambiables para el desarrollo de una aplicación. Se puede considerar como una aplicación genérica incompleta y configurable a la que se le puede añadir elementos para desarrollar una aplicación concreta (Medición de atributos POO en frameworks de desarrollo PHP., 2012)

Gestores de Base de Datos

El Sistema de Gestión de base de datos es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, además de proporcionar un acceso controlado a la misma (Marqués, 2011).

Tiene como característica la definición de la base de datos mediante el lenguaje de definición de datos, permite la inserción, actualización, eliminación y consulta de datos, el acceso controlado a la base de datos mediante un sistema de seguridad, de integridad, un sistema de control de concurrencia, un sistema de control de recuperación y un diccionario de datos o catalogo que con la descripción de los datos de la base de datos. (Marqués, 2011)

Sistemas de Gestión

Un sistema de gestión es una herramienta o aplicación informática que permite controlar todos y cada uno de los aspectos de una empresa (pedidos, producción, control de presencia, facturación, ventas, administración, etc.) (GRUPO I.A.G, 2004).

De esta forma la empresa podrá tener una visión de los movimientos diarios con que cuenta y fijarse en los puntos negativos o positivos con lo que cuenta durante el proceso.

Software Open Source

El software Open Source es aquel que incluye el código fuente y está disponible, más bien es una metodología ya que se suele confundir con el software libre. Para ser considerado open Source debe de contar con algunos requisitos:

El software debe ser libre de distribuir, se debe permitir modificaciones derivadas del mismo, la licencia no debe discriminar a ninguna persona y la licencia no debe limitar ningún campo de aplicación o emprendimiento. El software que cuenta con una licencia Open Source permite que una comunidad de desarrolladores lo mejore, lo corrija, lo pruebe y lo mantenga (Spano, 2010).

Software libre

La Fundación de Software Libre lo define como *“el software que se puede utilizar, copiar, estudiar, modificar y redistribuir sin ninguna restricción”* Para ser libre debe de considerarse cuatro reglas la primera es la de ejecutar un programa con cualquier propósito, la segunda es la de poder modificarla como uno quiera, la tercera y cuarta la libertad de redistribuirla con las modificaciones hechas en las últimas versiones. Para ello se necesita de utilizar el código fuente (Free Software Foundation, 1985).

Estado del arte de Lenguajes de programación

Según la estadística realizada en el año 2016 por la empresa Holandesa TIOBE siguen siendo los más utilizados:

Java: el más utilizado por su legibilidad y simplicidad, además por su estabilidad que asegura el funcionamiento de las aplicaciones que la utilizan (TIOBE, 2015).

El lenguaje que permanece a través del tiempo y es el más utilizado; compilado e interpretado en tiempo real. La rapidez al momento de ejecutar los programas es otra de las características, así también se destaca por hecho de que cualquier programa creado en JAVA se puede ejecutar desde cualquier ordenador con distintos tipos de sistemas operativos.

C: es el segundo más utilizado generalmente para sistemas de escritorio. Es de propósito general que lo hace muy flexible (TIOBE, 2015).

Este lenguaje de programación se usa mayormente para desarrollar sistemas operativos, no es orientado a objeto pero en C++ se realizaron modificaciones en la cual permiten las clases, métodos y atributos, encapsulación y polimorfismo en si orientada a objeto. Tal vez no sea un lenguaje para escribir directamente aplicaciones de gestión grafica en forma nativa

C++: es orientado a objetos surge como una continuación y ampliación de C.

C++ es una extensión de C que fue diseñado en base a la POO, no hay mucha diferencia con C es por eso que los programadores de C no tienen mucha dificultad en aprender el mismo. Este lenguaje también es uno de los más utilizado por su potencia. La mayor parte de los programadores lo utiliza para desarrollar video juegos.

C#: este lenguaje de programación se destaca por su sencillez y modernidad.

El lenguaje de programación C# trabaja en con el framework .NET que es una plataforma de Windows de código abierto. Microsoft trabaja con esta tecnología y el código fuente se

pueden ver en el repositorio de GitHub. Muchos desarrolladores optan por este lenguaje de programación. Según el índice de TIOBE se encuentra en el cuarto lugar desde el año 2015

PHP: se utiliza para desarrollar plataformas web junto con la base de datos MySQL.

Una de las características que tiene este lenguaje de programación es que cuenta con una gran documentación. Es utilizado por varias empresas ya que posee la habilidad de manejar una gran cantidad de datos, además de ser multiplataforma y libre.

Cabe destacar que muchos frameworks están basados en PHP (TIOBE, 2015).

Ruby: es un lenguaje de programación dinámico y de código abierto que está enfocado en la simplicidad y productividad (TIOBE, 2015).

Es un lenguaje de programación interpretado, de alto nivel y orientado a objetos. Es denominado un lenguaje multiparadigmas. Fue creado para mejorar la productividad se hizo conocido gracias al framework RoR hoy en día no es de preferencia para muchos desarrolladores por la desventaja que tiene con la velocidad.

Frameworks de Programación

Struts 2: se utiliza para crear aplicaciones web orientadas a empresas optimizando el proceso de desarrollo. Se utiliza cuando la carga de datos es grande (OpenWebinars, 2016).

Se caracteriza por su diseño simplificado, proporciona un Framework de validación que nos permitirá desacoplar las reglas de validación del código de las acciones. Permite también usar cualquier clase Java normal (POJO) como una acción, en struts 2 desaparecen los ActionForm se usan los JavaBean que leen directamente las propiedades, arranque rápido, esto quiere decir que no hace falta volver a reiniciar el servidor.

Struts 2 es un framework basado en el patrón MVC no es el más utilizado, sin embargo hay empresas que cuentan con sistemas programados con dicho lenguaje de programación que no hacen la migración por el hecho que les conllevaría un inmenso trabajo.

CodeIgniter: es uno de los framework utilizado por los desarrolladores de PHP por su velocidad y potencia, además que resulta fácil de aprender por su sencillez.

Así como los demás framework ya mencionados este trabaja con el patrón MVC. Se destaca por poder trabajar con la mayoría de los entornos o servidores. El núcleo del sistema solo necesita algunas librerías a diferencias de los otros que requiere de más recursos. Posee una característica de auto-carga que proporciona la inicialización de librerías, asistente y complementos de forma automática durante el proceso de arranque del sistema.

Symfony2: es un framework donde su código, componentes y librerías están publicados bajo la licencia MIT de software libre. Es uno de los framework recomendado y requerido por las empresas. Se utiliza generalmente para desarrollar proyectos grandes.

Symfony se construyó en base a otros frameworks, tomo lo mejores conceptos. En cuanto a la arquitectura cuenta con un micro-kernel altamente optimizado. Cada característica de Symfony está desarrollada con un bundle, que es un conjunto estructurado de archivos.

Laravel: la característica de este framework es que es fácil de aprender y usar en el momento de desarrollar una aplicación web. Cuenta con su propio motor de plantillas en la cual se puede escribir directamente el código sobre ellas.

Cuenta con dos repositorios en uno de ellos está el núcleo y el segundo sería el proyecto base, para que este funcione de debe descargar todas sus dependencias a través de composer. En laravel con un simple comando se podrá actualizar automáticamente todas las dependencias hasta los paquetes de terceros.

Sinatra: se define como un Domain Specific Language o DSL que deja al desarrollador elegir las herramientas adicionales dependiendo al tipo de desarrollo que este esté haciendo.

Sinatra no sigue el típico patrón MVC como lo hacen los otros framework. Se enfoca en la rápida creación de aplicaciones web en Ruby. Como es un framework minimalista, no fue pensado para resolver grandes problemas. Esto hace que no sea rápido y menos popular.

Ruby on Rails: es de código abierto. Su objetivo es favorecer la convención antes de la configuración, disminuir la repetición de código. Como es de conocimiento RoR es un framework de programación desarrollado en el lenguaje Ruby, de código abierto que utiliza el paradigma MVC el cual es utilizado para el desarrollo de aplicaciones web. Es utilizado por varios desarrolladores por la rapidez que lo caracteriza y por el hecho de que simplifica las tareas repetitivas. Otro de los puntos positivos con que cuenta es que rails posee una serie de plugins que están a disposición del público. Además se pueden hacer modificaciones y aplicarlas fácilmente.

Gestores de Base de Datos

MySQL: Es uno de lo más utilizado debido a que es open source y de fácil instalación. Es un gestor relacional de base de datos, que organiza información en distintos archivos dependiendo el motor que se utilice y en los cuales podemos guardar un simple registro hasta un complejo sistema relacional orientado a objetos (Mussa, 2008).

Básicamente ejecuta todas las plataformas incluyendo Linux. Una de sus características es que se asocia más a las aplicaciones basadas en la web.

Microsoft SQL server: es un sistema gestor de base de datos relacionales producido por Microsoft. Es un sistema cliente/servidor que funciona como una extensión del sistema operativo

Windows. Entre otras características proporciona integridad de datos, optimización de consultas, control de concurrencia y backup y recuperación.

Cuenta con varias características que lo hacen especial y que se quiera usar. Una de ellas es que trae soporte aproximado como para diez motores de almacenamientos con sus características y por sobre todo bajo costo. (SQL Server, 2016).

PostgreSQL: es un Sistema Gestor de Bases de Datos Relacionales Orientadas a Objetos. Es un gestor de bases de datos de código abierto, utiliza el modelo cliente/ servidor y multiproceso, permite trabajar con grandes volúmenes de datos; soporta gran parte de la sintaxis SQL y cuenta con un extenso grupo de enlaces con lenguajes de programación (Martinez, 2010).

Metodología o Materiales y Métodos

Definición del tipo y diseño de investigación

El tipo de investigación elegida es la investigación aplicada, dando un gran enfoque, al proceso de enlace entre toda la teoría aprendida en los años anteriores cursados en la carrera de análisis de sistemas informáticos y la práctica que conllevara realizar dicha investigación. (Sampieri, 2010).

En cuanto al diseño de investigación fue la investigación cualitativa ya que la misma consiste en el relevamiento de datos antes, durante y después de esa manera poder determinar y elaborar los requerimientos para su posterior análisis.

Instrumentos y Técnicas de Recolección de Datos

En la investigación aplicada realizada en el proceso del proyecto partió desde las técnicas primarias como ser la entrevista, el cuestionario tratando de registrar, elaborar, analizar e interpretar los datos útiles que se necesitaran para aplicarlos en la investigación.

También las secundarias que serían los libros de programación, boletines informativos, foros referidos al tema en cuestión, blog, ensayos, entre otros medios disponibles.

Procedimientos de Aplicación de Instrumento

Se utilizaron en el proceso de recolección de datos todas las fuentes bibliográficas que se pudieran conseguir especialmente libros digitales, también en tutoriales de páginas web dedicada exclusivamente a la enseñanza de lenguajes de programación, entre otros.

Se elaboró una entrevista personal con el administrador de la organización con preguntas específicas que hacían referencia a los procedimientos requeridos para la correcta gestión administrativa, a modo de recabar datos precisos, que ayudaron a los requerimientos del sistema desarrollado. A parte también para el mismo se realizó sondeos a tres consultoras situadas en

nuestra ciudad a lo que respondieron que no contaban con un herramienta que ayudase a las tareas ya mencionadas

Metodología de Desarrollo de Software

La metodología utilizada fue KANBAN ya que se basa en disminuir retrasos y crear un sistema de producción eficiente.

Los sistemas KANBAN consisten en un conjunto de formas de comunicarse e intercambiar información entre los diferentes operarios de una línea de producción, de una empresa, o entre proveedor y cliente. Su propósito es simplificar la comunicación, agilizándola y evitando errores producidos por falta de información (PDCA Home, 2012).

Las principales reglas de KANBAN son tres: Visualizar el trabajo y las fases del ciclo de producción o flujo de trabajo, determinar el límite de “trabajo en curso” (o Work In Progress) y medir el tiempo en completar una tarea (lo que se conoce como “lead time”) (Garzas, 2011).

Otra de las metodologías que se utilizó para el desarrollo del sistema fue la metodología de código abierto. El software código abierto es aquel que incluye el código fuente y está disponible y permite que una comunidad de desarrolladores lo mejore, lo corrija, lo pruebe y lo mantenga. Siempre cumpliendo los requisitos que se deben tener en cuenta para que se lo pueda considerar como software de código abierto.

Delimitación del Alcance del Software

La aplicación a desarrollarse correrá en dos navegadores que son Google Chrome, Firefox dividiéndose en diferentes módulos citados seguidamente

Inquilinos.

En el módulo inquilinos se podrá visualizar una lista de inquilinos registrados.

Si se desea registrar un inquilino nuevo permitirá al administrador guardar editar u eliminar sus datos como ser cedula de identidad, nombre, apellido, teléfono, email y toda la información referente al mismo.

Contratos

Esta interfaz realizara un registro del contrato fisico hecho al momento de la firma del inquilino con el administrador teniendo en cuenta los siguientes aspectos: cedula de identidad, fecha de contrato, monto establecido, nombre de la habitación, meses de garantía, estadia mensual fijado en el documento del contrato.

Pagos y cuotas

Aquí se generan las cuotas y adeudos del inquilino, generan recargos, etc. Muestra el detalle individual de los adeudos de cada departamento y el detalle general de los adeudos de todo el edificio.

Impresión de recibos de pago

Permite generar recibos de pago duplicados para la posterior entrega y abono por parte del inquilino.

Estado de cuenta de cada inquilino

Aquí el administrador puede consultar el detalle de todos los movimientos de cada inquilino, su cuota pendiente y pago realizado.

Informes

El modulo informes dará la posibilidad de visualizar e imprimir si se desea la información precisa como la lista de inquilinos que se tiene en el inmueble, el estado de su mensualidad, los contratos entre otros.

Gastos

En este sector se registran los gastos extras con su descripción y total del importe como la de ser gastos de impuestos, trabajos de refacción, de limpieza, etc. en fin cualquier tipo de egreso que incumba a la administración del edificio.

Limitaciones

El sistema no contempla pagos parciales

No contempla roles

El modulo pagos no cuenta con impresión de factura pre-impresa legal.

No se cuenta con tipos de pago.

No se incluirán más funcionalidades de las ya especificadas en el alcance.

Resultados y Discusión

Herramientas utilizadas

Finalizado el análisis de las posibles herramientas de desarrollo que pudieran concordar con la necesidad del presente proyecto se definió por el framework RoR por su agilidad al proceso de desarrollo, además de poseer una enorme cantidad de plugins y gemas que ayudan a la rapidez de una funcionalidad en específica, la metodología utilizada fue la de KANBAN que da la posibilidad de visualizar en tableros el proceso y avance del proyecto determinando así las tareas prioritarias. El IDE fue RubyMine con licencia gratuita para estudiante que ofrece un entorno de desarrollo con un editor de autocompletado, consola, visor de esquema de modelos, depurador, soporte para diversos sistemas de control de versiones, disponible para todos los sistemas operativos



Figura 1 – Logo de RoR. (Seeklogo, 2018)

Para el diseño del modelado de datos se utilizó la herramienta MySQL WorkBench que la misma brinda diseñar de forma visual la base de datos. Como Gestor de base de datos se utilizó MySQL por sus virtudes como ser la integridad de datos que ofrece.



Figura 2 – Logo de herramientas utilizadas. (Cambiadeeso, 2016)

Se optó Git como herramienta colaborativa y de control de versiones donde se alojó la documentación y el código fuente del software, utiliza varias interfaces donde se visualiza el proceso en el que están cada funcionalidad realizada. El link donde se tiene el repositorio del proyecto es <https://github.com/orlandoca/Deparsystem2017>.



Figura 3 – Logo del versionador de códigos. (Informático, 2017)

Análisis y planificación

El software Trello fue utilizado como herramienta colaborativa del proyecto, que básicamente es una aplicación que ayuda a la gestión de proyectos mejorando así la organización en las tareas asignadas donde se crean tableros sencillos a los cuales se les agregan “cards” o etiquetas. Se elaboraron cuatro tableros, el primero define la lista de todas las tareas a realizar para el desarrollo del sistema generalmente llamado Backlog; en el segundo “To-do” en el que se

colocan las tareas que se encuentran en proceso; el tercero es el llamado “Testing”, donde se van probando las funcionalidades de las tareas desarrolladas; y el cuarto y último es el de “Production” donde se registran las tareas culminadas que fueron pasadas a producción y que están siendo usados por el usuario.



Figura 4 – Logo de Trello. (Álvarez, 2016)

Figuras de las interfaces de la aplicación desarrollada

Iniciar sesión

Email
Ingrese su email

Contraseña
Ingrese su contraseña

☐

INGRESAR

[REGISTRATE](#)

[OLVIDASTE TU CONTRASEÑA?](#)

Figura 5 – Módulo de autenticación

Módulo login

En la figura 6 del módulo login fue desarrollado a través de unas de las gemas que posee el framework RoR llamado devise , permite que los usuarios creen cuentas en la página, modifiquen sus perfiles, iniciar o cerrar sesión, también se puede usar para recuperar contraseñas y demás.

En todas las capturas de pantalla se visualizan como todos los módulos fueron hechos con un formato en común con lo cual se logra los informes, permite guardarlos e imprimir en el caso que el usuario así lo desee en formato pdf y Excel. También cuenta con un buscador, la misma ayuda a tener las vistas homogéneas respetando una regla de diseño.

DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

Nueva Caja

Apertura

CREAR CAJA

← ATRAS

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 7 – Interfaz Apertura de caja

DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

CAJA

USUARIO:
orlandoca647@gmail.com

FECHA DE APERTURA:
23/07/2018

MONTO DE APERTURA:
500.000

NUMERO DE CAJA: 001

ESTADO:
CERRADO

FECHA DE CIERRE:
2018-07-23 19:05:00 -0400

MONTO DE CIERRE:
750.000

MONTO DE ENTRADA:
750.000

MONTO DE SALIDA:

DETALLES DE LA CAJA

| Caja | Concepto | Ingreso | Egreso | Saldo | |
|------|-------------------------|---------|--------|---------|-----|
| 1 | Apertura de caja | 500.000 | 0 | 500.000 | VER |
| 1 | PAGO DE CUOTA NUMERO: 1 | 250.000 | 0 | 750.000 | VER |

EDITAR ATRAS

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 8 – Interfaz Informe de Caja

Modulo Caja

Se realizó el modulo caja creando los datos correspondientes para su correcto funcionamiento , el usuario al ingresar al sistema debe especificar un monto inicial para su caja, una vez hecho este paso se crea con la fecha y el monto de apertura.

De tal manera se obtiene un control de los movimientos de la caja, finalizando el día el usuario deberá cerrar y tener el resumen de los ingresos del día, también cabe destacar que se cuenta en este módulo la opción de informes en pdf y Excel de manera a poder mostrarlas cuando se requiera.

Figura 9 – Interfaz Registro de Edificio con sus departamentos

| Nombre del Edificio | Nombre Propietario | Dirección | Ver | Editar | Eliminar |
|---------------------|--------------------|---------------------------|-----|--------|----------|
| EDIFICIO LAS PALMAS | ANDRES LOPEZ | TOMAS ROMERO PEREIRA 1023 | VER | EDITAR | ELIMINAR |

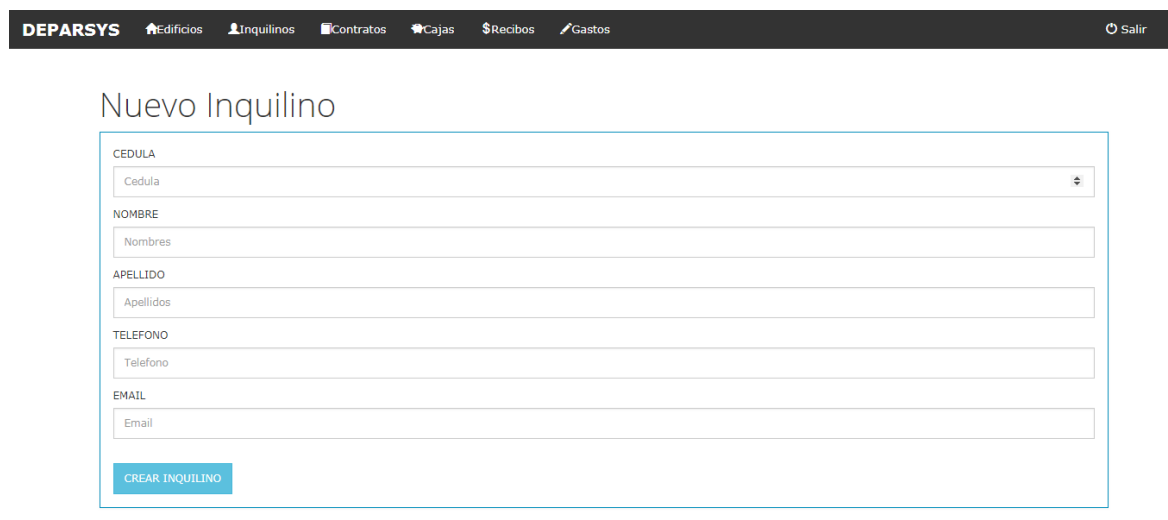
Figura 10 – Interfaz Listado Edificios

Modulo Edificios

En la figura 9 hace referencia al registro del edificio, se deberá primeramente ingresar a través del formulario que proporciona el sistema accediendo desde el botón que posee en la parte superior derecha completando el nombre del Edificio, el propietario de la misma, su dirección. También ya dando la posibilidad de ingresar la cantidad de departamentos que posee el mismo

con sus datos como ser el nombre del departamento, su precio, su estado y una pequeña descripción para mostrar las características. Si el edificio ya se encuentra registrado lo podrá buscar por su nombre con el buscador ya incorporado que trae el plugins llamado datatables , que entre sus características da la posibilidad de paginar , ordenar, filtrar por los atributos del módulo en cuestión ya registrado en la base de datos.

Una vez registrado se obtiene una lista con todos los datos del edificio con sus departamentos respectivos, en esa vista se podrá editar la información del edificio y departamento en caso de que sea necesario.



The screenshot shows the 'Nuevo Inquilino' (New Tenant) form within the DEPARSYS web application. The interface features a dark navigation bar at the top with the following menu items: Edificios, Inquilinos, Contratos, Cajas, Recibos, and Gastos. A 'Salir' (Logout) button is located on the right side of the navigation bar. The main content area is titled 'Nuevo Inquilino' and contains a form with the following fields: CEDULA (with a dropdown arrow), NOMBRE (with a placeholder 'Nombres'), APELLIDO (with a placeholder 'Apellidos'), TELEFONO (with a placeholder 'Telefono'), and EMAIL (with a placeholder 'Email'). A blue button labeled 'CREAR INQUILINO' is positioned at the bottom left of the form.

Figura 11 – Interfaz Registro de Inquilino

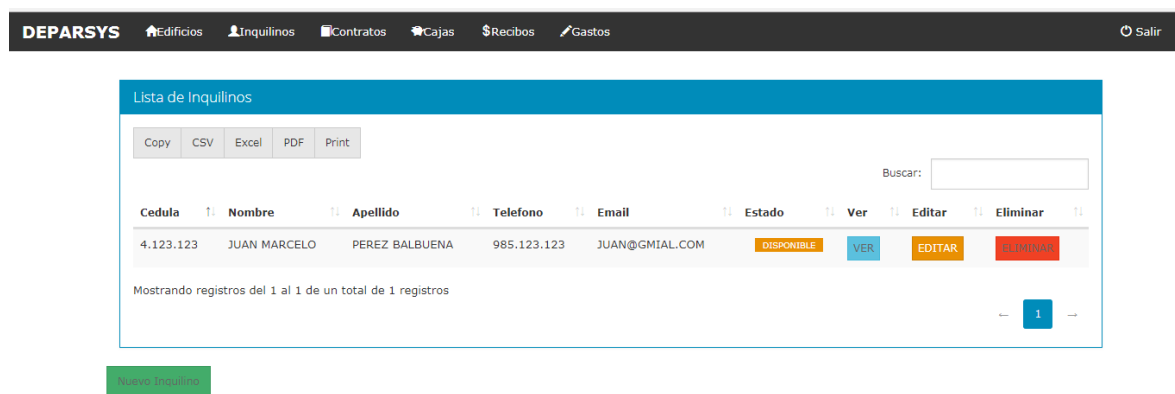


Figura 12 – Interfaz Listado de Inquilinos

Modulo Inquilinos

En la figura 11 denotando al módulo inquilinos se registra el inquilino que ya firmó el contrato para arrendar algún departamento detallando sus datos como ser sus nombres completos, sus apellidos, su número de cedula y su correo electrónico. Luego de tal registro se cuenta con una vista de los todos los inquilinos con todos sus datos donde se podrá buscar por cualquiera de sus atributos debido a la posibilidad que brinda el plugins citado en el párrafo anterior como se observa en la figura 12.

DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

Nuevo Contrato

Elige Inquilino
Por favor seleccione

Elige un Edificio con su departamento y su precio
Por favor seleccione

Inicio contrato
23 julio 2018

Fin contrato
23 julio 2018

Imagen
Seleccionar archivo Ningún archivo seleccionado

CREAR CONTRATO

← ATRAS

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 13 – Interfaz Registro de Contrato

DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

Lista de Contratos

Copy CSV Excel PDF Print

Buscar:

| Inquilino | Departamento | Precio | Fecha contrato | Inicio contrato | Fin contrato | Contrato | Ver | Editar | Eliminar |
|-----------------------------|--------------|---------|----------------|-----------------|--------------|-----------|-----|--------|----------|
| JUAN MARCELO PEREZ BALBUENA | A1 | 250.000 | 23/07/2018 | 23/07/2018 | 23/07/2019 | Descargar | VER | EDITAR | ELIMINAR |

Mostrando registros del 1 al 1 de un total de 1 registros

1

Nuevo Contrato

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 14 – Interfaz Lista de Contratos

Modulo Contratos

En el módulo contratos se realizara el contrato de prestación de servicios por el alquiler del departamento, el usuario deberá elegir el inquilino ya previamente cargado en dicho modulo que se desplegara a través de un campo , otro campo deberá seleccionar el edificio con su departamentos disponibles y precio correspondiente , otro campo con el inicio del contrato y su

finalización que dependerá exclusivamente de las políticas del titular del inmueble , ya que hay casos donde se firman contratos por seis o doce meses, previendo eso este módulo permite calcular automáticamente la cantidad de meses que se adapte al contrato del alquiler y de acuerdo a eso contar cuantas cuotas tendrá que pagar el inquilino en el lapso del contrato. También por ultimo permite cargar específicamente para el contrato firmado una vez hecho para salvaguardar la integridad del documento en caso de extravío o necesidad de volver a imprimir si así se requiera como se observa en la figura 14.

DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

← ATRAS

NOMBRE INQUILINO: JUAN MARCELO PEREZ BALBUENA
CEDULA: 4.123.123
EMAIL: JUAN@GMIAL.COM
TELEFONO: 985123123

FECHA DE PAGO: 23/07/2018
RECIBO NUMERO: 001-001- 001

Detalle del Recibo

| NUMERO DEL CONTRATO | DESCRIPCION | PRECIO |
|---------------------|-------------------------|-------------------|
| 1 | PAGO DE CUOTA NUMERO: 1 | 250.000 |
| TOTAL: | | GS 250.000 |

IMPRIMIR RECIBO

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 15 – Interfaz de Vista preliminar de Recibos

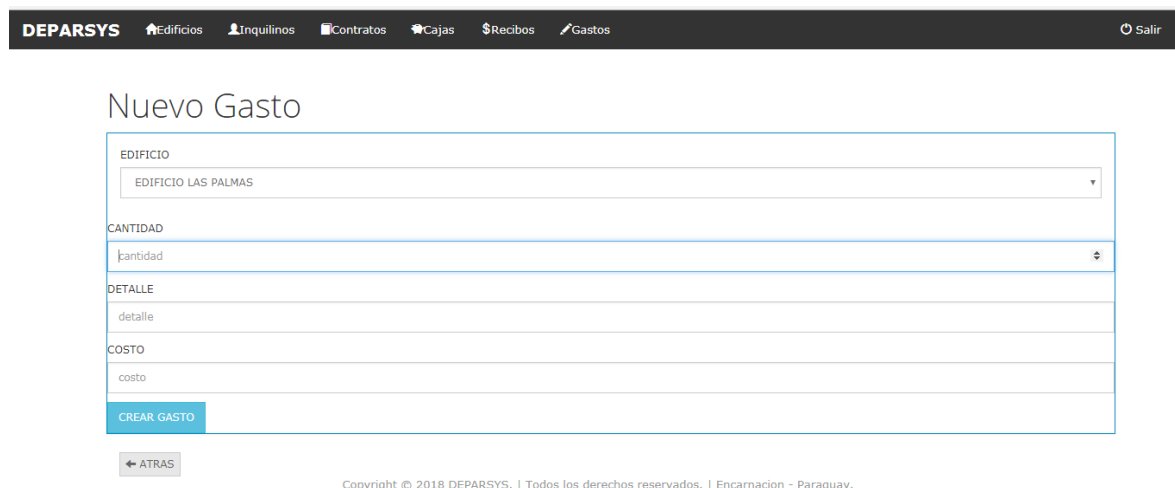
| EDIFICIO LAS PALMAS TOMAS ROMERO PEREIRA 1023 Telef.: (071) 203778 Encarnación - Paraguay | | RECIBO N° 001-001-001 | |
|--|---|---------------------------------|--|
| FECHA: 23/07/2018 | | | |
| NOMBRE DEL INQUILINO: JUAN MARCELO PEREZ BALBUENA | | | |
| NUMERO DE CUOTA* | DESCRIPCION | PRECIO UNITARIO | |
| 1 | PAGO DE CUOTA NUMERO: 1 DEL DEPARTAMENTO A1 | 250000 | |
| TOTAL | | 250000 | |
| TOTAL A PAGAR EN GUARANIES: | | DOSCIENTOS CINCUENTA MIL | |

Figura 16 – Interfaz de Vista preliminar de Recibo en formato PDF

Modulo Recibos

Otro de los ítems que se obtuvo en el requerimiento fue la de tener un módulo donde el administrador pueda tener el control y acceso del comprobante de pago cuando así lo requiera, en este caso de los recibos emitidos al momento previo de hacer el pago, donde se puede apreciar todos los datos necesarios generales que ayudan a verificar la información del edificio en cuestión, el departamento alquilado, la fecha, pequeña descripción y el total a pagar

Se utilizó la gema PRAW para imprimir la factura en formato PDF. Praw es una gema con una cantidad inmensa de funcionalidades, con soporte de dibujo vectorial, posee una variedad de herramientas de bajo nivel para las necesidades básicas del diseño.



DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

Nuevo Gasto

EDIFICIO
EDIFICIO LAS PALMAS

CANTIDAD
1

DETALLE
detalle

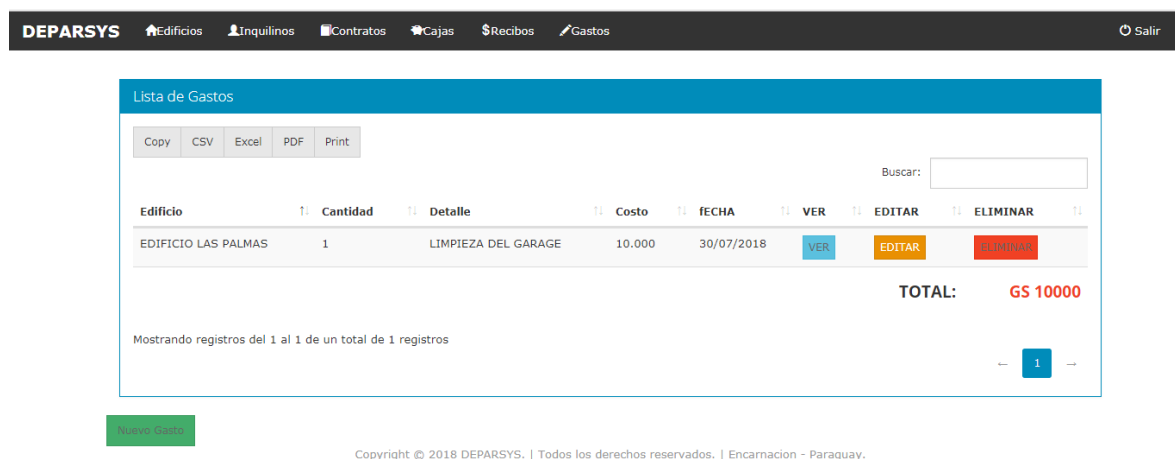
COSTO
costo

CREAR GASTO

← ATRAS

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 17 – Interfaz de Registro de Gastos



DEPARSYS Edificios Inquilinos Contratos Cajas Recibos Gastos Salir

Lista de Gastos

Copy CSV Excel PDF Print

Buscar:

| Edificio | Cantidad | Detalle | Costo | FECHA | VER | EDITAR | ELIMINAR |
|---------------------|----------|---------------------|--------|------------|-----|--------|----------|
| EDIFICIO LAS PALMAS | 1 | LIMPIEZA DEL GARAGE | 10.000 | 30/07/2018 | VER | EDITAR | ELIMINAR |

TOTAL: **GS 10000**

Mostrando registros del 1 al 1 de un total de 1 registros

← 1 →

Nuevo Gasto

Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.

Figura 18 – Interfaz de Lista de Gastos

Modulo Gastos

Para la creación del módulo gastos se tuvo en consideración aquellos costos que de alguna manera supone el mantenimiento y/o otras necesidades mensuales del edificio como la de ser el sueldo de la limpiadora, la pintura, el costo de la electricidad, el agua, las refacciones entre otros.

Por tal efecto surge esta funcionalidad donde el usuario elige el nombre del edificio , completa la cantidad , el detalle donde se describen que gasto fue, y el costo en si del mismo, también cabe resaltar que está ligada al módulo Caja, por tanto se descuenta los gastos de los ingresos de inmueble por los arrendamientos.

Teniendo como resultado de todos los gastos que se hayan hecho en el mes del edificio seleccionado posibilitando exportar en planilla electrónica o un documento pdf para tener el informe resuelto.



Figura 19 – Logo del servidor web. (LogoJoy, 2017)

Como servidor web se analizó el almacenamiento que provee Heroku tanto de forma gratuita, como con planes de pago. Es un servicio en la nube que nos permite desarrollar, alojar y correr nuestras aplicaciones en muchos lenguajes de programación como Ruby, Java, Python y PHP.



Figura 20 – Logo de interserver.net. (Vaswani, 2018)

InterServer hace que sea rápido y fácil para los clientes agregar soporte para frameworks específicos basados en sus necesidades. Para hacerlo aún más fácil, tienen scripts de instalación

de un solo clic para Ruby on Rails y otros frameworks populares. Los desarrolladores y otros usuarios están de acuerdo en que InterServer está entre los mejores cuando se trata de soporte y servicio RoR hoy. (HostAdvice.com, 2018).



Figura 21 – Logo de MilesWeb. (Sandhu, 2017)

MilesWeb apoya el desarrollo de Ruby on Rails en su hosting web compartido, hosting de revendedores y planes de hosting VPS. Las extensiones de servidor para RoR en Apache están preinstaladas en una pila LAMP con un enfoque gestionado para la seguridad de la plataforma. MilesWeb se ha ganado una buena reputación en el mercado de la India por ser una fuente confiable para que los desarrolladores de Ruby on Rails obtengan servicios de hosting de bajo coste. (HostAdvice.com, 2018)

Conclusión

En el contexto de la recolección de datos se describió esta problemática con la que contaba la organización, se ha visto de alguna manera el impacto que tendrá el sistema web para inmuebles a través de las tareas automatizadas que ayudara a realizar denotadas en la descripción de los módulos desarrollados , la empresa cuenta con unas innumerables tareas que se deben realizar manualmente dando margen a errores que afectan el rendimiento en eficacia y tiempo cabiendo lugar a la necesidad de acelerar los procesos para ser más productivos.

Se realizó un relevamiento de información tanto en la web como en entrevistas a empresas consultoras de la ciudad dedicadas a la administración de inmuebles acerca de sistemas con las características que se describen en el presente trabajo y no se han encontrado herramientas similares.

En el desarrollo del software en ese sentido nos permite encontrar y elaborar nuevas herramientas para utilizar con este tipo de procesos y hacer a medida de la realidad de la organización.

El producto del desarrollo fue un software que facilita organizar el control de una serie de tareas administrativas mediante el procesamiento de datos teniendo un acceso rápido a las informaciones que requiera el usuario, o sea el administrador. Cabe resaltar que estas informaciones son íntegras al ser resguardadas en una base de datos

Cabe mencionar como ha sido la primera vez utilizando estas herramientas colaborativas se tuvieron dificultades a la hora de culminar cada “Cards” especialmente en la tablero de testing ya que al ser solamente uno el desarrollador se debía autoevaluar lo hecho filtrándose así algunas situaciones no contempladas desde el inicio del proyecto debiendo aplicar ajustes sobre lo ya realizado.

También cabe puntualizar que los servidores que se analizaron dónde alojar la aplicación contiene restricciones en ciertos aspectos como la de ser, la cantidad de espacio dado, la base de datos obligatoriamente debe ser cambiada a otros (ya que se usó MySQL para el desarrollo), pagados la mayoría de ellos requiriendo así algunos costos extras para la implementación de la misma.

Recomendaciones

- Se recomiendan como líneas futuras agregar las siguientes funcionalidades:
 - a. La creación de roles para que el inquilino pueda contar con un usuario y contraseña para ver sus estados de cuentas.
 - b. El pago del inquilino a través de tarjetas de crédito.
 - c. La generación de pagarés a la par que se calcula la cantidad de meses de alquiler.
 - d. Diseñar e implementar un módulo de facturación para imprimir con boletas pre-impresas autorizadas por la SET.
- La implementación del sistema web.

Lista de Referencias

- GRUPO I.A.G.* (2004). Obtenido de <http://www.grupoia.com/faq/que-es-un-sistema-de-gestion-o-mis>
- Acosta, J. C. (2012). *Medición de atributos POO en frameworks de desarrollo PHP*. Argentina: In XVIII Congreso Argentino de Ciencias de la Computación.
- Aguilera, S. (2015). *Tipos de Software*. Obtenido de <http://repositorio.ub.edu.ar/bitstream/handle/123456789/5213/FInform-502-U4-7-TiposdeSw-2015.pdf?sequence=1&isAllowed=y>
- Color, A. (13 de mayo de 2009). Principios de la Gestion Administrativa. *ABC Color*.
- CV., A. M. (2018). *Comprasoft*. Obtenido de Comprasoft: <https://comprasoft.com/jetbrains/rubymine>
- Free Software Fundation. (1985). Obtenido de <https://www.fsf.org/es>
- israel965. (11 de Octubre de 2014). *Unodepiera*. Obtenido de Unodepiera: <https://www.uno-de-piera.com/heroku-servicio-de-computacion-en-la-nube/>
- Marqués, M. (2011). Sistema de Gestión de base de datos. En M. Marqués, *Base de Datos* (pág. 3).
- Muzyka, L. (20 de Agosto de 2014). *Peoplecancode* . Obtenido de Peoplecancode : <http://www.peoplecancode.com/es/tutorials/how-to-manage-users-with-devise-ruby-on-rails>
- Paramio, C. (24 de Mayo de 2011). *Genbeta*. Obtenido de Genbeta: <https://www.genbeta.com/desarrollo/herramientas-imprescindibles-para-un-desarrollador-de-ruby-on-rails>

Pc, D. (2018). *Consultatucondominio.com*. Obtenido de Consultatucondominio.com:

<https://www.consultatucondominio.com>

Pressman, R. S. (2010). El Software y la Ingeniería del Software. En R. S. Pressman, *Ingeniería del Software- Un enfoque práctico* (pág. 3). Mexico: MC Graw Hill.

Residentia, S. (2018). *Residentia*. Obtenido de Residentia: <https://www.residentia.net/>

Servicios, A. (24 de Febrero de 2018). *AdminCond*. Obtenido de AdminCond:

<http://www.admicond.com.ve/2018/02/software-gratis-para-condominios.html>

Spano, D. d. (2010). *El open source como facilitador del open access. Impacto y visibilidad de las revistas científicas*. Obtenido de

[https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia_spano_elis.pdf?](https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia_spano_elis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-disposition=inline%3B%20filename%3DEl_Open_Source_como_)

[AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=](https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia_spano_elis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-disposition=inline%3B%20filename%3DEl_Open_Source_como_)

[%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-](https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia_spano_elis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-disposition=inline%3B%20filename%3DEl_Open_Source_como_)

[disposition=inline%3B%20filename%3DEl_Open_Source_como_](https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia_spano_elis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-disposition=inline%3B%20filename%3DEl_Open_Source_como_)

HostAdvice.com. (2018). *HostAdvice.com*. Obtenido de HostAdvice.com:

<https://es.hostadvice.com/hosting-services/ruby-on-rails/>

Empresas, O. d. (s.f.). *Organizacion de Empresas*. Obtenido de Organizacion de Empresas:

<http://www.organizacionempresas.com/negocios/bienes.html>

Association, F. M. (12 de Abril de 2008). *Fmassociation*. Obtenido de Fmassociation:

<https://web.archive.org/web/20080412142951/http://www.fmassociation.org.uk/>

Álvarez, D. V. (28 de Enero de 2016). *Web Programacion*. Obtenido de Web Programacion:

[https://webprogramacion.com/508/blog-informatica-tecnologia/planificando-proyectos-](https://webprogramacion.com/508/blog-informatica-tecnologia/planificando-proyectos-con-trello.aspx)

[con-trello.aspx](https://webprogramacion.com/508/blog-informatica-tecnologia/planificando-proyectos-con-trello.aspx)

Cambiadeeso. (1 de Febrero de 2016). *Cambiadeeso*. Obtenido de Cambiadeeso:

<https://www.cambiadeso.es/entradas/mysql-anadir-usuarios-a-una-base-de-datos-con-sql/>

Empresas, O. d. (s.f.). *Organizacion de Empresas*. Obtenido de Organizacion de Empresas:

<http://www.organizacionempresas.com/negocios/bienes.html>

HostAdvice.com. (2018). *HostAdvice.com*. Obtenido de HostAdvice.com:

<https://es.hostadvice.com/hosting-services/ruby-on-rails/>

Informático, E. F. (9 de Noviembre de 2017). *El Frenético Informático* . Obtenido de El

Frenético Informático : <http://elfreneticoinformatico.com/guia-rapida-de-uso-de-github-ubuntu/>

LogoJoy. (7 de Julio de 2017). *LogoJoy*. Obtenido de LogoJoy:

https://logojoy.com/heroku_logo/

Sandhu, P. (20 de Octubre de 2017). *Bloghaul*. Obtenido de Bloghaul:

<https://bloghaul.com/milesweb-hosting-review-best-for-speed-and-reliability.html>

Seeklogo. (2018). *Seeklogo*. Obtenido de Seeklogo: <https://seeklogo.com/vector->

[logo/184637/ruby-on-rails](https://seeklogo.com/vector-logo/184637/ruby-on-rails)

Vaswani, J. (5 de Junio de 2018). *Bloggerideas*. Obtenido de Bloggerideas:

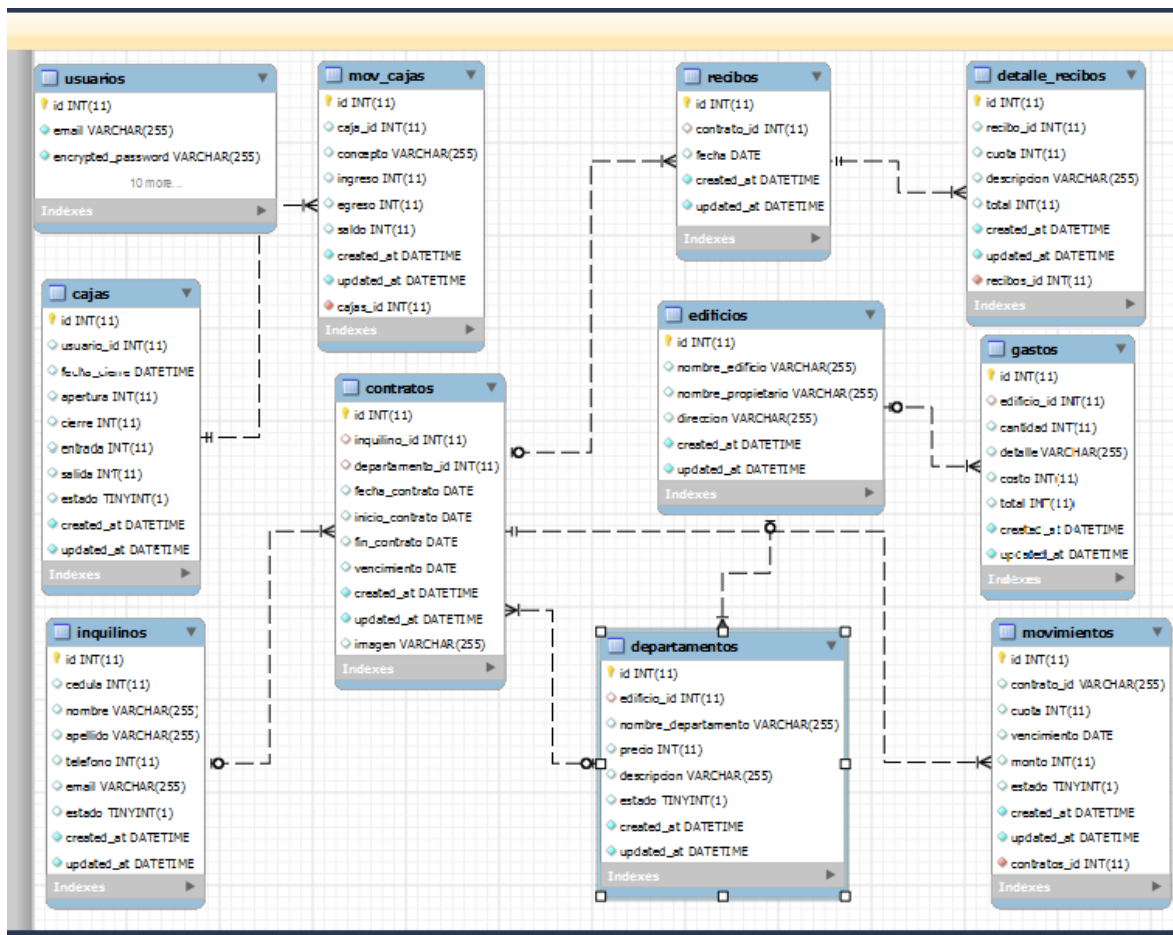
<https://www.bloggersideas.com/interserver-coupons/>

Anexos

Anexo 1: Enlace al proyecto

<https://github.com/orlandoca/Deparsystem2017>

Anexo 2: Modelado del Sistema



Anexo 3: Entrevista utilizada

1. ¿Cómo realiza el cobro de los alquileres?

Manualmente

2. ¿De qué manera registra el cobro de los alquileres?

En planilla Excel.

3. ¿Utiliza algún sistema o forma de resguardar los datos?

No utilizo ninguna

4. ¿Cómo controla el vencimiento de los contratos?

Según planilla

5. ¿Le resulta fácil utilizar planillas para sus controles de cobros?

Sí.

6. ¿Cómo Registra los gastos extras que tiene el edificio mensualmente?

Con boletas de contabilidad

7. ¿De qué manera hace las rendiciones de cuenta al propietario del inmueble?

En planillas Excel.

8. ¿Le gustaría informatizar las gestiones administrativas para agilizar su trabajo?

Porque?

Sí. Para sistematizar mis informaciones.

9. ¿Se realizan pagos parciales?

Sí.

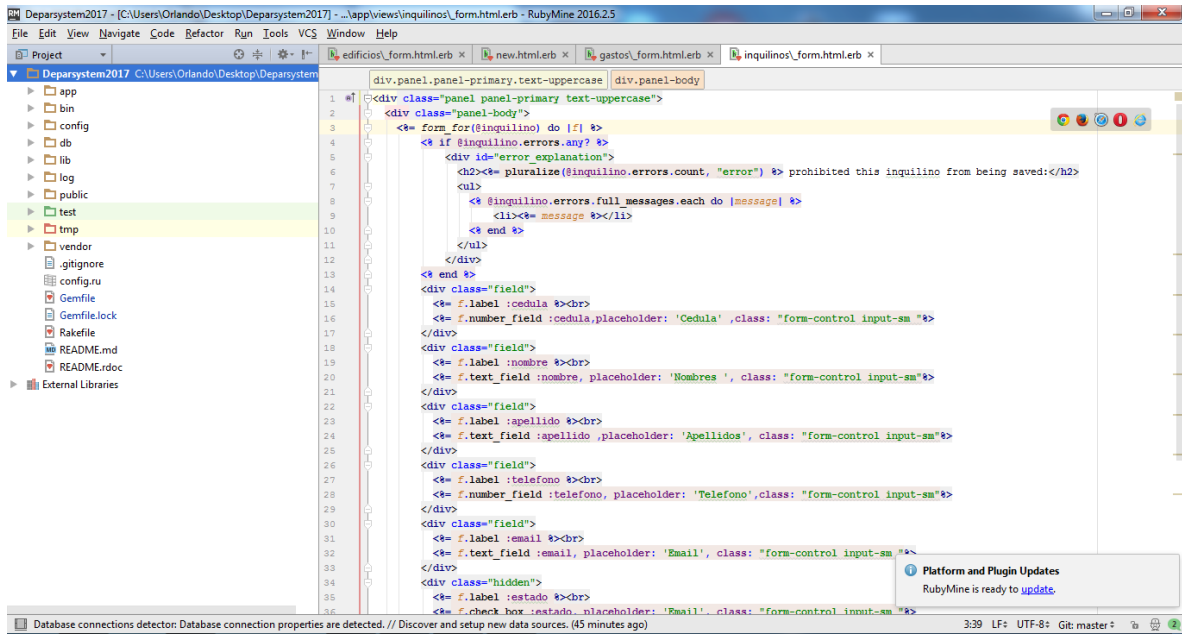
10. ¿Cuáles son los medios de pagos del inquilino?

Depósito Bancario y efectivo.

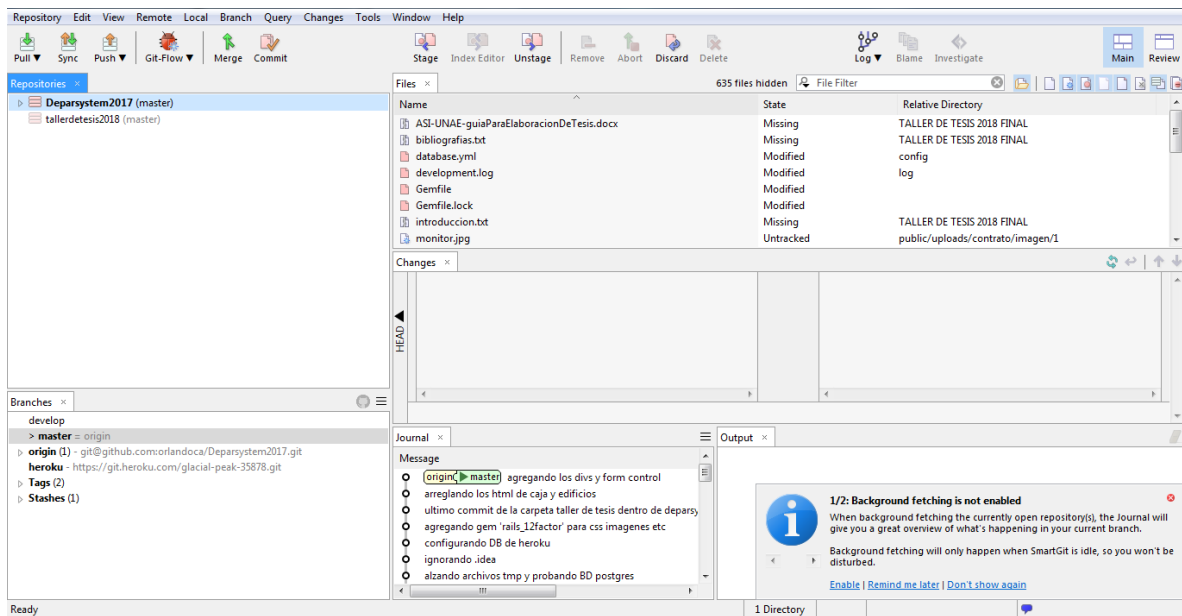
11. ¿Se realizan pagos por garantía de depósito?

Anexo 4: Estructura del proyecto

Como entorno de desarrollo se eligió el IDE RubyMine que cuenta exclusivamente con todas las especificaciones del idioma Ruby, su naturaleza dinámica y las convenciones de código



Anexo 5: Repositorio local SmartGit



Anexo 6: Código del Proyecto- Controlador inquilino

```

class InquilinosController < ApplicationController
  before_action :set_inquilino, only: [:show, :edit, :update, :destroy]

  # GET /inquilinos
  # GET /inquilinos.json
  def index
    cajaabierto = Caja.where(estado:0)
    if (cajaabierto.count > 0)
      @inquilinos = Inquilino.all
    else
      redirect_to new_caja_path
    end
  end

  # GET /inquilinos/1
  # GET /inquilinos/1.json
  def show
  end

  # GET /inquilinos/new
  def new
    @inquilino = Inquilino.new
  end

  # GET /inquilinos/1/edit
  def edit
  end

  # POST /inquilinos
  # POST /inquilinos.json
  def create
    @inquilino = Inquilino.new(inquilino_params)

    respond_to do |format|
      if @inquilino.save
        format.html { redirect_to @inquilino, notice: 'Inquilino creado exitosamente.' }
        format.json { render :show, status: :created, location: @inquilino }
      else
        format.html { render :new }
        format.json { render json: @inquilino.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /inquilinos/1
  # PATCH/PUT /inquilinos/1.json
  def update
    respond_to do |format|
      if @inquilino.update(inquilino_params)
        format.html { redirect_to @inquilino, notice: 'Inquilino actualizado exitosamente.' }
        format.json { render :show, status: :ok, location: @inquilino }
      else
        format.html { render :edit }
        format.json { render json: @inquilino.errors, status: :unprocessable_entity }
      end
    end
  end
end

```

```

# DELETE /inquilinos/1
# DELETE /inquilinos/1.json
def destroy
  @inquilino.destroy
  respond_to do |format|
    format.html { redirect_to inquilinos_url, notice: 'Inquilino eliminado exitosamente.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_inquilino
  @inquilino = Inquilino.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def inquilino_params
  params.require(:inquilino).permit(:cedula, :nombre, :apellido, :telefono, :email, :estado)
end
end

```

Anexo 7: Código del Proyecto – Controlador Edificios-Departamentos

Código utilizado para el controlador Edificios donde en el método donde especificamos los atributos del mismo también especificamos los del módulo departamentos ya que se utilizan en una misma vista.

```

class EdificiosController < ApplicationController
  before_action :set_edificio, only: [:show, :edit, :update, :destroy]

  # GET /edificios
  # GET /edificios.json

  def index
    cajaabierto = Caja.where(estado:0)
    if (cajaabierto.count > 0)
      @edificios = Edificio.all
    else
      redirect_to new_caja_path
    end
  end

  # GET /edificios/1
  # GET /edificios/1.json
  def show
  end

  # GET /edificios/new
  def new
    @edificio = Edificio.new
  end
end

```

```

# GET /edificios/1/edit
def edit
end

# POST /edificios
# POST /edificios.json
def create
  @edificio = Edificio.new(edificio_params)

  respond_to do |format|
    if @edificio.save
      format.html { redirect_to @edificio, notice: 'Edificio creado exitosamente.' }
      format.json { render :show, status: :created, location: @edificio }
    else
      format.html { render :new }
      format.json { render json: @edificio.errors, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /edificios/1
# PATCH/PUT /edificios/1.json
def update
  respond_to do |format|
    if @edificio.update(edificio_params)
      format.html { redirect_to @edificio, notice: 'Edificio actualizado exitosamente.' }
      format.json { render :show, status: :ok, location: @edificio }
    else
      format.html { render :edit }
      format.json { render json: @edificio.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /edificios/1
# DELETE /edificios/1.json
def destroy
  @edificio.destroy
  respond_to do |format|
    format.html { redirect_to edificios_url, notice: 'Edificio was successfully destroyed.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_edificio
  @edificio = Edificio.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def edificio_params
  params.require(:edificio).permit(:nombre_edificio, :nombre_propietario, :direccion, departamentos_attributes: [ :id, :nombre_departamento, :precio, :descripcion, :estado, :_destroy ])
end
end

```

Anexo 8: Código del Proyecto – Controlador Contratos

Código utilizado en el controlador Contratos donde al momento de crear el contrato calculamos la fecha de inicio y la fecha fin del mismo para saber la cantidad de meses que el inquilino va arrendar el departamento seleccionado con todos sus atributos.

```
class ContratosController < ApplicationController
  before_action :set_contrato, only: [:show, :edit, :update, :destroy]

  # GET /contratos
  # GET /contratos.json
  def index
    cajaabierto = Caja.where(estado:0)
    if (cajaabierto.count > 0)
      @contratos = Contrato.all
    else
      redirect_to new_caja_path
    end
  end

  # GET /contratos/1
  # GET /contratos/1.json
  def show
  end

  # GET /contratos/new
  def new
    @contrato = Contrato.new
  end

  # GET /contratos/1/edit
  def edit
  end

  def dynamic_edificios
    @edificios = Edificio.find(:all)
  end

  # POST /contratos
  # POST /contratos.json
  def create
    @contrato = Contrato.new(contrato_params)

    respond_to do |format|
      if @contrato.save
        @departamento = Departamento.find_by(id: @contrato.departamento_id)
        @departamento.update(estado: 1)
        @fecha = @contrato.inicio_contrato
        cantidadmeses= ((@contrato.fin_contrato.year * 12 +
@contrato.fin_contrato.month) - (@contrato.inicio_contrato.year * 12 +
@contrato.inicio_contrato.month))
        for i in 1..cantidadmeses do
          @movimiento = Movimiento.create(contrato_id: @contrato.id, cuota:i , monto:
@contrato.departamento.precio , vencimiento: @fecha + 1.months , estado: 0)
          @fecha = @fecha + 1.months
        end
        @movimiento.save
      end
    end
  end
end
```

```

    @contrato.update(fin_contrato: @fecha)
    format.html { redirect_to @contrato, notice: 'Contrato creado exitosamente.' }
    format.json { render :show, status: :created, location: @contrato }
  else
    format.html { render :new }
    format.json { render json: @contrato.errors, status: :unprocessable_entity }
  end
end
end

# PATCH/PUT /contratos/1
# PATCH/PUT /contratos/1.json
def update
  respond_to do |format|
    if @contrato.update(contrato_params)
      format.html { redirect_to @contrato, notice: 'Contrato actualizado exitosamente.' }
      format.json { render :show, status: :ok, location: @contrato }
    else
      format.html { render :edit }
      format.json { render json: @contrato.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /contratos/1
# DELETE /contratos/1.json
def destroy
  @contrato.destroy
  respond_to do |format|
    format.html { redirect_to contratos_url, notice: 'Contrato eliminado exitosamente.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_contrato
  @contrato = Contrato.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list through.
def contrato_params
  params.require(:contrato).permit(:inquilino_id, :edificio_id, :departamento_id, :fecha_contrato, :inicio_contrato, :fin_contrato, :vencimiento, :imagen)
end
end

```

Anexo 9: Código del Proyecto –Gastos

Código utilizado en el controlador gastos donde a través de métodos de un gema listamos todos los gastos del edificio al mes donde nos encontramos.


```

class GastosController < ApplicationController
  before_action :set_gasto, only: [:show, :edit, :update, :destroy]

  # GET /gastos
  # GET /gastos.json
  def index

    @gastos = Gasto.by_month(Date.today)

  end

  # GET /gastos/1
  # GET /gastos/1.json
  def show
  end

  # GET /gastos/new
  def new
    @gasto = Gasto.new
  end

  # GET /gastos/1/edit
  def edit
  end

  # POST /gastos
  # POST /gastos.json
  def create
    @gasto = Gasto.new(gasto_params)

    respond_to do |format|
      if @gasto.save
        format.html { redirect_to @gasto, notice: 'Gasto was successfully created.' }
        format.json { render :show, status: :created, location: @gasto }
      else
        format.html { render :new }
        format.json { render json: @gasto.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /gastos/1
  # PATCH/PUT /gastos/1.json
  def update
    respond_to do |format|
      if @gasto.update(gasto_params)
        format.html { redirect_to @gasto, notice: 'Gasto was successfully updated.' }
        format.json { render :show, status: :ok, location: @gasto }
      else
        format.html { render :edit }
        format.json { render json: @gasto.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /gastos/1
  # DELETE /gastos/1.json
  def destroy
    @gasto.destroy
    respond_to do |format|
      format.html { redirect_to gastos_url, notice: 'Gasto was successfully
destroyed.' }
      format.json { head :no_content }
    end
  end
end

```

```

    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_gasto
      @gasto = Gasto.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list
    through.
    def gasto_params
      params.require(:gasto).permit(:edificio_id, :cantidad, :detalle, :costo, :total)
    end
  end
end

```

Anexo 10: Código del Proyecto –Controlador Cajas

Código del controlador Cajas donde se crea siempre y cuando no haya otra caja abierta primeramente verificando a través del método “nil”, con todos los atributos que corresponden.

```

class CajasController < ApplicationController
  before_action :set_caja, only: [:show, :edit, :update, :destroy]

  # GET /cajas
  # GET /cajas.json
  def index
    @cajas = Caja.all
  end

  # GET /cajas/1
  # GET /cajas/1.json
  def show
  end

  # GET /cajas/new
  def new
    @caja = Caja.new
    @caja.usuario_id = current_usuario.id # SE OBTIENE EL USUARIO AUTOMATICAMENTE
  end

  # GET /cajas/1/edit
  def edit
    @caja.usuario_id = current_usuario.id # SE OBTIENE EL USUARIO AUTOMATICAMENTE
    @caja.estado = 1
  end

  # POST /cajas
  # POST /cajas.json
  def create
    @caja = Caja.new(caja_params)
    @caja.fecha_cierre = nil
    @caja.entrada = @caja.apertura
    @caja.cierre = @caja.apertura
    respond_to do |format|
      if @caja.save
        @mov_caja = MovCaja.create!(caja_id: @caja.id, concepto: 'Apertura de caja',

```

```

ingreso: @caja.apertura, egreso: 0, saldo: @caja.apertura)
  format.html { redirect_to @caja, notice: 'Caja abierto exitosamente.' }
  format.json { render :show, status: :created, location: @caja }
else
  format.html { render :new }
  format.json { render json: @caja.errors, status: :unprocessable_entity }
end
end
end

# PATCH/PUT /cajas/1
# PATCH/PUT /cajas/1.json
def update
  respond_to do |format|

    @caja.fecha_cierre = Time.now
    if @caja.update(caja_params)
      format.html { redirect_to @caja, notice: 'Caja cerrada exitosamente.' }
      format.json { render :show, status: :ok, location: @caja }
    else
      format.html { render :edit }
      format.json { render json: @caja.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /cajas/1
# DELETE /cajas/1.json
def destroy
  @caja.destroy
  respond_to do |format|
    format.html { redirect_to cajas_url, notice: 'Caja eliminada exitosamente.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_caja
  @caja = Caja.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list
through.
def caja_params
  params.require(:caja).permit(:usuario_id, :fecha_cierre, :apertura, :cierre,
:entrada, :salida, :estado)
end
end

```

Anexo 11: Código del Proyecto –Controlador Movimiento Caja

Código del controlador Movimiento de Caja donde se registran todos los ingresos y egresos de la aplicación en el metodo Create, tanto por los cobros de los alquileres como los gastos del edificio

```

class MovCajasController < ApplicationController
  before_action :set_mov_caja, only: [:show, :edit, :update, :destroy]

  # GET /mov_cajas

```

```

# GET /mov_cajas.json
def index
  @mov_cajas = MovCaja.all
end

# GET /mov_cajas/1
# GET /mov_cajas/1.json
def show
end

# GET /mov_cajas/new
def new
  @mov_caja = MovCaja.new
end

# GET /mov_cajas/1/edit
def edit
end

# POST /mov_cajas
# POST /mov_cajas.json
def create
  @mov_caja = MovCaja.new(mov_caja_params)

  respond_to do |format|
    if @mov_caja.save
      format.html { redirect_to @mov_caja, notice: 'Movimiento de caja creado exitosamente.' }
      format.json { render :show, status: :created, location: @mov_caja }
    else
      format.html { render :new }
      format.json { render json: @mov_caja.errors, status: :unprocessable_entity }
    end
  end
end

# PATCH/PUT /mov_cajas/1
# PATCH/PUT /mov_cajas/1.json
def update
  respond_to do |format|
    if @mov_caja.update(mov_caja_params)
      format.html { redirect_to @mov_caja, notice: 'Movimiento de caja actualizada.' }
    else
      format.json { render :show, status: :ok, location: @mov_caja }
    end
  end
end

# DELETE /mov_cajas/1
# DELETE /mov_cajas/1.json
def destroy
  @mov_caja.destroy
  respond_to do |format|
    format.html { redirect_to mov_cajas_url, notice: 'Movimiento de Caja eliminada.' }
    format.json { head :no_content }
  end
end

private

```

```

# Use callbacks to share common setup or constraints between actions.
def set_mov_caja
  @mov_caja = MovCaja.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list
through.
def mov_caja_params
  params.require(:mov_caja).permit(:caja_id, :concepto, :ingreso, :egreso, :saldo)
end
end

```

Anexo 12: Código del Proyecto –Controlador Pagos

Código del controlador Pagos, donde se crea en el método Create al momento de que el usuario del sistema selecciona una cuota o varias llevando así posteriormente al módulo recibos.

```

class PagosController < ApplicationController
  before_action :set_pago, only: [:show, :edit, :update, :destroy]

  # GET /pagos
  # GET /pagos.json
  def index
    @pagos = Pago.all
  end

  # GET /pagos/1
  # GET /pagos/1.json
  def show
  end

  # GET /pagos/new
  def new
    @pago = Pago.new
  end

  # GET /pagos/1/edit
  def edit
  end

  # POST /pagos
  # POST /pagos.json
  def create
    @pago = Pago.new(pago_params)

    respond_to do |format|
      if @pago.save
        format.html { redirect_to @pago, notice: 'Pago creado exitosamente.' }
        format.json { render :show, status: :created, location: @pago }
      else
        format.html { render :new }
        format.json { render json: @pago.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /pagos/1

```

```

# PATCH/PUT /pagos/1.json
def update
  respond_to do |format|
    if @pago.update(pago_params)
      format.html { redirect_to @pago, notice: 'Pago actualizado.' }
      format.json { render :show, status: :ok, location: @pago }
    else
      format.html { render :edit }
      format.json { render json: @pago.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /pagos/1
# DELETE /pagos/1.json
def destroy
  @pago.destroy
  respond_to do |format|
    format.html { redirect_to pagos_url, notice: 'Pago eliminado.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_pago
  @pago = Pago.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list
through.
def pago_params
  params.require(:pago).permit(:contrato_id, :inquilino_id, :fecha_pago, :monto,
:estado)
end
end

```

Anexo 13: Código del Proyecto –Controlador Recibos

```

class RecibosController < ApplicationController
  before_action :set_recibo, only: [:show, :edit, :update, :destroy]

  # GET /recibos
  # GET /recibos.json
  def index
    cajaabierto = Caja.where(estado:0)
    if (cajaabierto.count > 0)
      @recibos = Recibo.all
    else
      redirect_to new_caja_path
    end
  end

  # GET /recibos/1
  # GET /recibos/1.json
  def show
    respond_to do |format|
      format.html
      format.pdf do
        pdf = ReportePdf.new(@recibo)

```

```

        send_data pdf.render, filename: 'show.pdf', type: 'application/pdf'
      end
    end
  end

  # GET /recibos/new
  def new
    @recibo = Recibo.new
  end

  # GET /recibos/1/edit
  def edit
  end

  # POST /recibos
  # POST /recibos.json
  def create
    @recibo = Recibo.new(recibo_params)

    respond_to do |format|
      if @recibo.save
        format.html { redirect_to @recibo, notice: 'Recibo creado exitosamente.' }
        format.json { render :show, status: :created, location: @recibo }
      else
        format.html { render :new }
        format.json { render json: @recibo.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /recibos/1
  # PATCH/PUT /recibos/1.json
  def update
    respond_to do |format|
      if @recibo.update(recibo_params)
        format.html { redirect_to @recibo, notice: 'Recibo actualizado exitosamente .' }
        format.json { render :show, status: :ok, location: @recibo }
      else
        format.html { render :edit }
        format.json { render json: @recibo.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /recibos/1
  # DELETE /recibos/1.json
  def destroy
    @recibo.destroy
    respond_to do |format|
      format.html { redirect_to recibos_url, notice: 'Recibo eliminado exitosamente.' }
      format.json { head :no_content }
    end
  end

  private
  # Use callbacks to share common setup or constraints between actions.
  def set_recibo
    @recibo = Recibo.find(params[:id])
  end

  # Never trust parameters from the scary internet, only allow the white list

```

```
through.
  def recibo_params
    params.require(:recibo).permit(:contrato_id, :fecha)
  end
end
```

Anexo 14: Código del Proyecto –Controlador Detalle_recibo

```
class DetalleRecibosController < ApplicationController
  before_action :set_detalle_recibo, only: [:show, :edit, :update, :destroy]

  # GET /detalle_recibos
  # GET /detalle_recibos.json
  def index
    @detalle_recibos = DetalleRecibo.all
  end

  # GET /detalle_recibos/1
  # GET /detalle_recibos/1.json
  def show
  end

  # GET /detalle_recibos/new
  def new
    @detalle_recibo = DetalleRecibo.new
  end

  # GET /detalle_recibos/1/edit
  def edit
  end

  # POST /detalle_recibos
  # POST /detalle_recibos.json
  def create
    @detalle_recibo = DetalleRecibo.new(detalle_recibo_params)

    respond_to do |format|
      if @detalle_recibo.save
        format.html { redirect_to @detalle_recibo, notice: 'Detalle recibo was successfully created.' }
        format.json { render :show, status: :created, location: @detalle_recibo }
      else
        format.html { render :new }
        format.json { render json: @detalle_recibo.errors, status: :unprocessable_entity }
      end
    end
  end

  # PATCH/PUT /detalle_recibos/1
  # PATCH/PUT /detalle_recibos/1.json
  def update
    respond_to do |format|
      if @detalle_recibo.update(detalle_recibo_params)
        format.html { redirect_to @detalle_recibo, notice: 'Detalle recibo was successfully updated.' }
        format.json { render :show, status: :ok, location: @detalle_recibo }
      else
        format.html { render :edit }
        format.json { render json: @detalle_recibo.errors, status:

```



```

:unprocessable_entity }
  end
end
end

# DELETE /detalle_recibos/1
# DELETE /detalle_recibos/1.json
def destroy
  @detalle_recibo.destroy
  respond_to do |format|
    format.html { redirect_to detalle_recibos_url, notice: 'Detalle recibo was
successfully destroyed.' }
    format.json { head :no_content }
  end
end

private
# Use callbacks to share common setup or constraints between actions.
def set_detalle_recibo
  @detalle_recibo = DetalleRecibo.find(params[:id])
end

# Never trust parameters from the scary internet, only allow the white list
through.
def detalle_recibo_params
  params.require(:detalle_recibo).permit(:recibo_id, :cuota, :descripcion, :total)
end
end

```

Anexo 15: Código del Proyecto –Reporte pdf del recibo

Aquí es donde se diseña el reporte del pdf, las medida de los campos del módulo recibos y detalle_recibo, los atributos que se mostraran, en fin todo diseño a gusto del desarrollador.

```

class ReportePdf < Prawn::Document
  def initialize(recibo)
    super()
    @recibo = recibo
    header
    text_content
    table_content
    table_content2
    table_context3
  end

  def header
    #This inserts an image in the pdf file and sets the size of the image
    #image "#{Rails.root}/app/assets/images/header.png", width: 530, height: 150
  end

  def text_content
    # The cursor for inserting content starts on the top left of the page. Here we
    move it down a little to create more space between the text and the image inserted
    above
    y_position = cursor - 50

    # The bounding_box takes the x and y coordinates for positioning its content and

```

```

some options to style it
# bounding_box([0, cursor - 10], :width => 1200, :height => 100) do
#   text "FACTURA", size: 20, style: :bold
# end
# bounding_box([0, y_position], :width => 1200, :height => 100) do
#   text "N° FACTURA", size: 10, style: :bold
#   text "#{@factura.nro_fac}"
# end
bounding_box([0, 600], :width => 1200, :height => 100) do
  text "FECHA: #{@recibo.created_at.strftime('%d/%m/%Y')}", size: 15, style: :bold
end

bounding_box([0, 580], :width => 500, :height => 300) do
  text "NOMBRE DEL INQUILINO: #{@recibo.contrato.inquilino.nombre}
#{@recibo.contrato.inquilino.apellido}", size: 15, style: :bold
end

move_down -420
bounding_box([0, cursor], :width => 290, :height => 90) do
  transparent(0.5) { stroke_bounds }
  move_down 10
  indent(20) do
    text " #{@recibo.contrato.departamento.edificio.nombre_edificio }", size: 20,
style: :bold
  end
  indent(30) do
    text " #{@recibo.contrato.departamento.edificio.direccion }", size: 15
    text "Telef.: (071)203778 Encarnacion - Paraguay", size: 10
    # stroke_horizontal_rule
  end
end

move_down -300
bounding_box([300, 700], :width => 240, :height => 90) do
  transparent(0.5) { stroke_bounds }
  move_down 10
  indent(20) do
    text "RECIBO ", size: 20, style: :bold
    text "N° 001-001-00#{@recibo.id}", size: 15, style: :bold
  end
end

end

def table_content
  # This makes a call to product_rows and gets back an array of data that will
  # populate the columns and rows of a table
  # I then included some styling to include a header and make its text bold. I made
  # the row background colors alternate between grey and white
  # Then I set the table column widths
  table detalles_rows do
    row(0).font_style = :bold
    self.header = true
    self.row_colors = ['DDDDDD', 'FFFFFF']
    self.column_widths = [100, 240, 200]
  end
end

def detalles_rows
  move_down 45

```

```

[[['NUMERO DE CUOTA°', 'DESCRIPCION', 'PRECIO UNITARIO']] +
  @recibo.detalle_recibos.map do |detalle|
    [detalle.cuota, detalle.descripcion + " DEL DEPARTAMENTO",
     #{@recibo.contrato.departamento.nombre_departamento} , detalle.total]
  end
end

def table_content2
  table detalles_rows2 do
    row(0).font_style = :bold
    self.header = true
    self.row_colors = ['DDDDDD', 'FFFFFF']
    self.column_widths = [120, 420]
  end
end

def detalles_rows2
  move_down 40
  @total=0

  @recibo.detalle_recibos.each {|detalle| @total += detalle.total}

  [['TOTAL ', @total ]]
end

def table_context3
  table detalles_rows3 do
    row(0).font_style = :bold
    self.header = true
    self.row_colors = ['DDDDDD', 'FFFFFF']
    self.column_widths = [200, 340]
  end
end

def detalles_rows3
  move_down 40
  @total=0

  @recibo.detalle_recibos.each {|detalle| @total += detalle.total}

  [['TOTAL A PAGAR EN GUARANIES: ', @total.to_words.upcase ]]
end
end

```

Anexo 16: Código del Proyecto –Routes

Código de Routes, es el archivo de pequeños bloques de código donde indicamos a la aplicación donde están las páginas.

```

Rails.application.routes.draw do

  resources :gastos
  resources :gastos
  resources :inquilinos
  resources :mov_cajas
  resources :cajas

```

```

resources :detalle_recibos
resources :recibos
resources :movimientos do
  collection do
    put :pagar
  end
end
resources :departamentos
resources :contratos

resources :inquilinos do
  collection do
    get :autocomplete_inquilino_nombre
  end
end

resources :edificios

#devise_for :usuarios

devise_for :usuario, controllers: { sessions: "usuario/sessions", registrations:
"usuario/registrations", passwords: "usuario/passwords" }, :path_names => {:sign_in
=> 'login', :sign_up => 'registro', :sign_out => 'logout'}
as :usuarios do
  get 'sign_in' => 'usuario/sessions#new', :as => :new_usuario_session_path
  get 'sign_up' => 'usuario/registrations#create', :as => :usuario_registration_path
  delete 'sign_out' => 'usuario/sessions#destroy', :as =>
:destroy_usuario_session_path
  get 'new' => 'usuario/sessions#destroy', :as => :new_usuario_password_path
end

# The priority is based upon order of creation: first created -> highest priority.
# See how all your routes lay out with "rake routes".

# You can have the root of your site routed with "root"

root 'contratos#index'

# Example of regular route:
# get 'products/:id' => 'catalog#view'

# Example of named route that can be invoked with purchase_url(id: product.id)
# get 'products/:id/purchase' => 'catalog#purchase', as: :purchase

# Example resource route (maps HTTP verbs to controller actions automatically):
# resources :products

# Example resource route with options:
# resources :products do
#   member do
#     get 'short'
#     post 'toggle'
#   end
#
#   collection do
#     get 'sold'
#   end
# end

# Example resource route with sub-resources:
# resources :products do
#   resources :comments, :sales
#   resource :seller

```

```
# end

# Example resource route with more complex sub-resources:
# resources :products do
#   resources :comments
#   resources :sales do
#     get 'recent', on: :collection
#   end
# end

# Example resource route with concerns:
# concern :toggleable do
#   post 'toggle'
# end
# resources :posts, concerns: :toggleable
# resources :photos, concerns: :toggleable

# Example resource route within a namespace:
# namespace :admin do
#   # Directs /admin/products/* to Admin::ProductsController
#   # (app/controllers/admin/products_controller.rb)
#   resources :products
# end
end
```