



**UNIVERSIDAD AUTÓNOMA DE ENCARNACIÓN**

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

Orlando Javier Cardozo Benitez

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**

**Ficha Catalográfica (Al dorso de la Portada de la Tesis)**

**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 Benitez, Orlando Javier (2012). Desarrollo De Aplicación Web para la administración de inmuebles. Encarnación, Universidad Autónoma de Encarnación, 122 p.

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 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 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<sup>1</sup>

<b>Desarrollo de Aplicación Web para la Administracion de Inmuebles .....</b>	<b>4</b>
<b>Planteamiento del Problema .....</b>	<b>5</b>
Preguntas Específicas de Investigación .....	5
Objetivos.....	6
General.....	6
Específicos .....	6
Justificación .....	7
<b>Este es un Título Uno .....</b>	<b>¡Error! Marcador no definido.</b>
Este es un Título Dos.....	<b>¡Error! Marcador no definido.</b>
Este es un título tres.....	<b>¡Error! Marcador no definido.</b>
Este es un título cuatro.....	<b>¡Error! Marcador no definido.</b>
Este es un título cinco. ....	<b>¡Error! Marcador no definido.</b>
<b>Revisión de Literatura .....</b>	<b>8</b>
Antecedentes .....	16
Este es un título tres.....	16
Este es un título cuatro.....	17
Este es un título cinco. ....	<b>¡Error! Marcador no definido.</b>
Bases teóricas.....	12
Este es un título tres.....	<b>¡Error! Marcador no definido.</b>
<b>Metodología o Materiales y Métodos.....</b>	<b>19</b>
Definicion del Tipo y Diseño de Investigacion .....	19
Definicion de la poblacion.....	22
Descripcion de tecnicas e instrumentos de recoleccion , medicion , procesamiento y analisis de los datos. ....	<b>¡Error! Marcador no definido.</b>
Delimitacion .....	<b>¡Error! Marcador no definido.</b>
Alcances .....	22
Limitaciones .....	<b>¡Error! Marcador no definido.</b>
<b>Resultados y Discusión.....</b>	<b>23</b>
Titulo .....	23
Este es un título tres.....	<b>¡Error! Marcador no definido.</b>
Este es un título cuatro.....	<b>¡Error! Marcador no definido.</b>
Este es un título cinco. ....	<b>¡Error! Marcador no definido.</b>
Titulo .....	<b>¡Error! Marcador no definido.</b>
Este es un título tres.....	<b>¡Error! Marcador no definido.</b>

---

<b>Conclusión .....</b>	<b>30</b>
<b>Recomendaciones .....</b>	<b>36</b>
<b>Lista de Referencias .....</b>	<b>37</b>
<b>Anexos .....</b>	<b>39</b>

## **LISTA DE TABLAS**

TABLA 1 –Nombre de la primera tabla	45
TABLA 2 – Nombre de la segunda tabla	52
TABLA 3 – Etc.	61

## LISTA DE GRÁFICOS

FIGURA 1 – Logo de RoR.....	23
FIGURA 2 – Logo de herramientas utilizadas.....	24
FIGURA 3 – Logo del versionador de códigos.....	24
FIGURA 4 – Logo del servidor web.....	25
FIGURA 5 – Logo del Trello.....	25
FIGURA 6 –Modulo de autenticación.....	26
FIGURA 7 –Modulo Apertura de Caja.....	27
FIGURA 8 –Interfaz Informe de caja.....	27
FIGURA 9 – Interfaz Registro de Edificios con sus departamentos.....	28
FIGURA 10 –Lista de Edificios.....	29
FIGURA 11 –	
FIGURA 12 –	
FIGURA 13 –	
FIGURA 14 –	
FIGURA 15 –	

## **LISTA DE FIGURAS**

## **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 Inmuebles**

Cardozo Benítez, Orlando Javier

Ing. Gabriela Cuba

Eje temático: TIC e Innovación

### **RESUMEN**

La presente investigación dará lugar al “Desarrollo de una Aplicación Web Para la administración de Inmuebles” debido a la carencia que visualice .En el campo laboral administrativo propio y en la región en procesar de alguna manera digitalmente las diferentes tareas que tienen el personal administrativo al momento de gestionar un arrendamiento. El objetivo de este trabajo fue la de Desarrollar una aplicación web

Para las series de gestiones administrativas. El tipo de investigación utilizada fue la investigación aplicada con el proceso de recolección de datos de todas las fuentes bibliográficas digitales. Se tomó la metodología de desarrollo Kanban. El lenguaje Ruby con su framework "Ruby on Rails" en la versión 4.2. Se eligió la base de datos MySQL en la versión 5.5.

Los resultados fueron que agilizamos de alguna manera las tareas al informatizar los datos del inquilino, con el módulo de contratos vemos el comienzo con sus contratos, la duración de meses, sus fechas de vencimiento de la renta, en el módulo edificios tenemos la cantidad de departamentos, precios, con los detalles de la misma, en el módulo pagos sus recibos de pago, sus estados de cuenta, en caja gestionamos las entradas del arrendamiento.

Palabras clave: Administración, Web.

Cornisa: **Desarrollo de Aplicación Web para la Administración de Inmuebles**

2

**EL TITULO DE LA OBRA TRADUCIDA AL GUARANÍ**

Apohára: Cardozo Benítez Orlando Javier

Sambyhyhára: Ing Gabriela Cuba

Tembikuaareka rape:

**ÑEMOMBYKY**

Es la traducción en lengua Guaraní del resumen.

*ÑE'Ê TEKOTEVÊVA*: 4-6 palabras.



Cornisa: **Desarrollo de Aplicación Web para la Administración de Inmuebles**

3

**EL TITULO DE LA OBRA TRADUCIDA AL INGLÉS**

Author: .....

Advisor: .....

Research Line:.....

**SUMMARY**

Es la traducción en lengua extranjera del resumen.

*KEY WORDS:* 4-6 palabras.

## **Título del Trabajo**

Los verbos de acción del trabajo de investigación deben estar en tiempo pasado: “La investigación se realizó...; se aplicó un test de...”. (Reemplazar este párrafo por la introducción del trabajo, este párrafo debe quedar debajo del título del trabajo, en la página 4, eliminar las anotaciones de la página anterior).

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 trabajo practico o de campo comenzó desde el punto de desarrollo del software eligiendo el campo de aplicación es el de la ingeniería del software con resultado de tener una automatización de casi todos las tareas que requieren realizar manualmente por un administrador a través del sistema considerando un aporte a la comunidad inmobiliaria local y regional por sus reglas de negocio que están insertando al producto que por cierto no existen en el mercado local alguna herramienta adecuada a las necesidades de la comunidad.

## **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 las calle Ana Cheraniuk casi Avenida San Blas denominada Arrúa Emprendimientos con registro único de contribuyente número 1.448.520-6 se ha hallado en ella la gran 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 como ser la propia información de los inquilinos, sus estados de cuenta, la generación de recibos de pago, los cálculos por mora, informe de pagos hechos que en la gran mayoría de los administradores de edificios no cuentan con una herramienta que le pueda facilitar automatizando los procesos con lo cual hace que una tarea conlleve mayor dedicación de tiempo y más dificultad de registro mediando libros y/o bibliorátos , lo cual 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 tenemos sobre gestión administrativa de la organización que administra el inmueble?
2. ¿Cómo podríamos 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.** Implementar una Aplicación web para una organización dedicada a la administración de inmuebles.

#### **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 .
- Definir un servidor web donde alojar el proyecto final.

### **Justificación**

La importancia de la presente investigación tendrá una gran relevancia para los beneficiados , en este caso los administradores de los departamentos 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 despistado 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, el 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 software desarrollado es de código abierto (open source) cosa que ayudara a ahorrar la adquisición de licencias el cual permitirá ser modificado o ampliado de alguna manera

## Revisión de Literatura<sup>2</sup>

### Bases teóricas

#### 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”*

**Comentario [U1]:** Esto va dentro de Revisión de la Literatura  
Revisión de la literatura es el título principal, luego como subtítulo Antecedentes y posteriormente van las Bases teóricas

---

<sup>2</sup> APA (2010), p. 63: Cada nueva sección comienza en una nueva página, con el título de nivel 1. No se marca los encabezados ni con números ni con letras. El número de niveles (que en cada trabajo puede variar), depende del grado de complejidad con que se aborda la problemática investigada.

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 Ingeniería 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 (aulaFormativa).

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 (SQL Server , 2016).

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.

**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).

## **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.

### **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.

### **Software para administración de inmuebles en Encarnación.**

A través de la información captada en la ciudad de Encarnación 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.





## **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 dicho investigación hallados ya, en la investigación básica, puntualizando la aplicación directa al problema del sector de administración de inmuebles.

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 que se realizará en el proceso del proyecto partirán 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 utilizó en el proceso de recolección de datos todas las fuentes bibliográficas que se pudieran conseguir especialmente libros digitales, teniendo siempre la asesoría de la profesora actualmente de metodología de la investigación, también en tutoriales de páginas web dedicada exclusivamente a la enseñanza de lenguajes de programación.

### **Metodología de Desarrollo de Software**

La metodología a 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, estadía 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 comprobante de factura legal

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

## Resultados y Discusión

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**

Para el diseño del modelado de datos se utilizó la herramienta MySQL WorkBench que la misma brinda diseñar de forma visual la base 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

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

Como servidor web se utilizó 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 4 – Logo del servidor web

### 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. Cabe mencionar como ha sido la primera vez utilizando esta herramienta 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 así aplicar ajustes sobre lo ya realizado.



Figura 5 – Logo de Trello

The screenshot displays a login interface with a dark header bar at the top. Below the header, the title "Iniciar sesión" is centered. The form contains two input fields: "Email" with the placeholder "Ingrese su email" and "Contraseña" with the placeholder "Ingrese su contraseña". Below these fields is a blue button labeled "INGRESAR". Underneath the button are two links: "REGISTRATE" and "OLVIDASTE TU CONTRASEÑA?".

**Figura 6 — Módulo de autenticación**

En el módulo de login se pudo realizar a través de unas de las gemas que posee el framework RoR llamado device . Es una gema extraordinaria, 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 todos los print 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.

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.



## Cornisa: Desarrollo de Aplicación Web para la Administración de Inmuebles

27

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.

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

Para el 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 Edificio' (New Building) registration form within the DEPARSYS application. The form is titled 'Nuevo Edificio' and contains several input fields: 'Nombre edificio', 'Nombre propietario', and 'Direccion'. Below these fields is a blue button labeled 'Agregar Departamento'. At the bottom of the form, there is a table with three columns: 'Nombre', 'Precio', and 'Descripcion'. A blue button labeled 'CREAR EDIFICIO' is located at the bottom left of the form, and an orange button labeled 'Remover' is at the bottom right. The application's navigation bar at the top includes links for 'Edificios', 'Inquilinos', 'Contratos', 'Cajas', 'Recibos', and 'Gastos', along with a 'Salir' (Logout) button.

Figura 9 – Interfaz Registro de Edificio con sus departamentos



Figura 10 – Interfaz Listado Edificios

En el 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

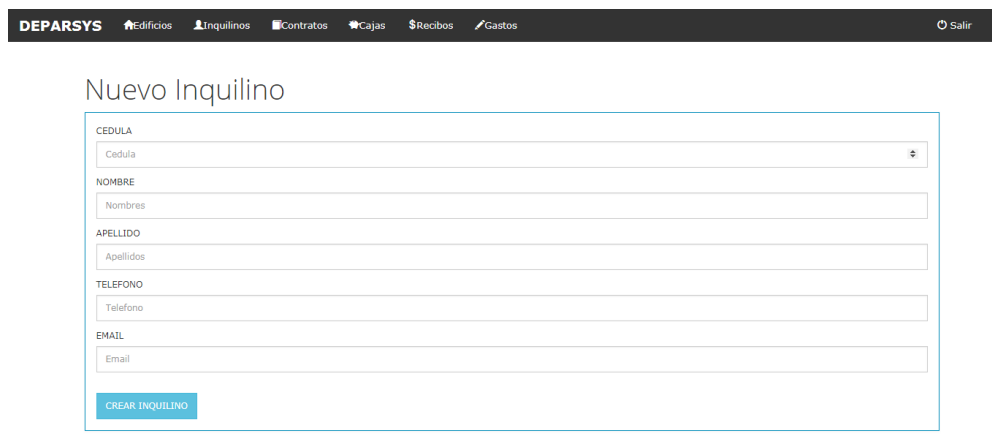


Figura 11 – Interfaz Registro de Inquilino

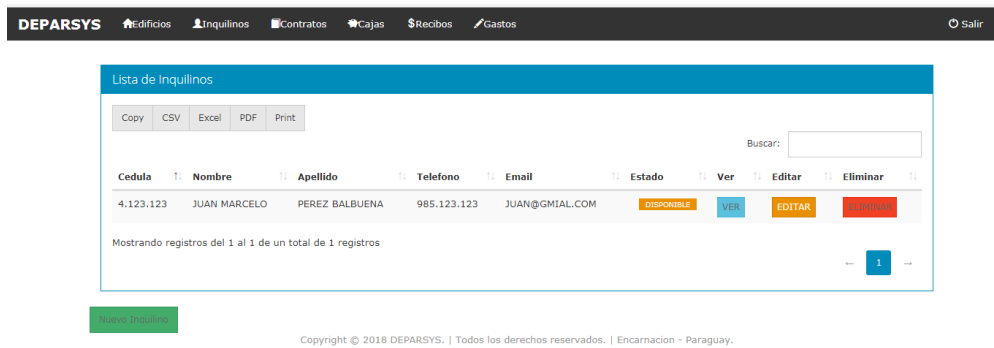


Figura 12 – Interfaz Listado de Inquilinos

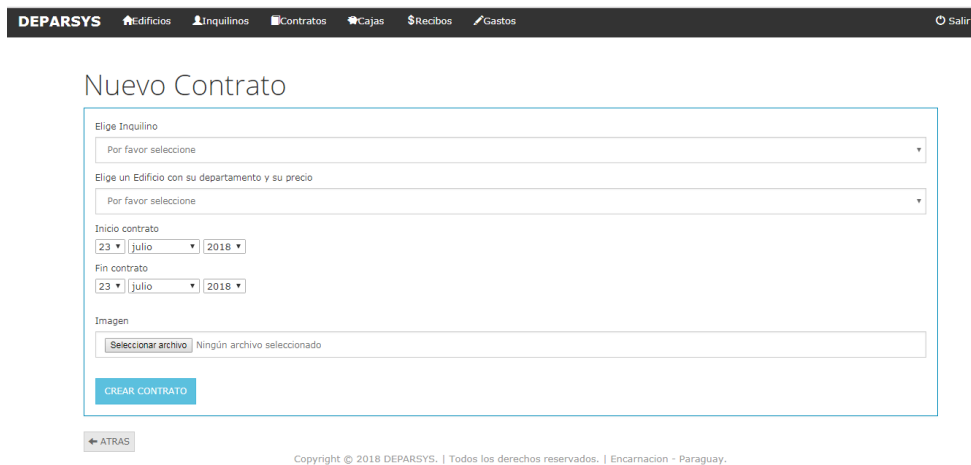



Figura 13 – Interfaz Registro de Contrato



The screenshot displays the 'Lista de Contratos' (List of Contracts) interface. At the top, there is a navigation bar with the DEPARSYS logo and menu items: Edificios, Inquilinos, Contratos, Cajas, Recibos, and Gastos. A 'Salir' (Logout) button is on the right. Below the navigation bar, the 'Lista de Contratos' section includes a toolbar with 'Copy', 'CSV', 'Excel', 'PDF', and 'Print' buttons. A search bar labeled 'Buscar:' is positioned to the right. The main area contains a table with the following columns: Inquilino, Departamento, Precio, Fecha contrato, Inicio contrato, Fin contrato, Contrato, Ver, Editar, and Eliminar. A single record is shown for 'JUAN MARCELO PEREZ BALBUENA' in department 'A1' with a price of '250.000', start date '23/07/2018', and end date '23/07/2019'. The 'Contrato' column contains a 'Descargar' button, and the 'Ver', 'Editar', and 'Eliminar' columns contain buttons labeled 'VER', 'EDITAR', and 'ELIMINAR' respectively. Below the table, it states 'Mostrando registros del 1 al 1 de un total de 1 registros'. A pagination control shows '1' of 1 pages. A green 'Nuevo Contrato' button is at the bottom left. The footer contains the copyright notice: 'Copyright © 2018 DEPARSYS. | Todos los derechos reservados. | Encarnacion - Paraguay.'

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

Figura 14 – Interfaz Lista de 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 6 o 12 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.

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 Encarnacion - 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

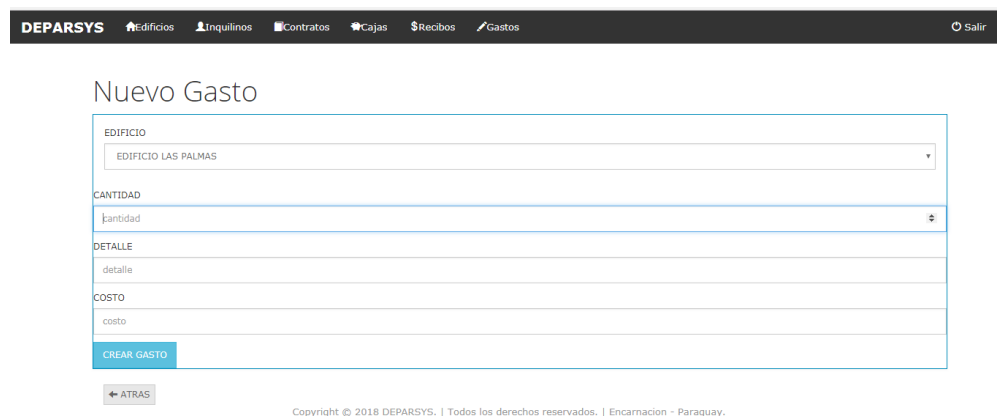
TOTAL A PAGAR EN GUARANIES:
DOSCIENTOS CINCUENTA MIL

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

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  
cantidad

DETALLE  
detalle

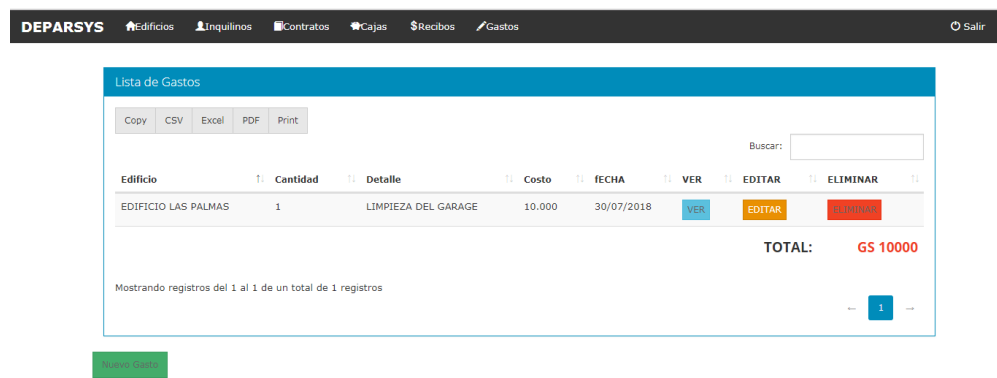
COSTO  
costo

CREAR GASTO

← ATRÁS

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**

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.



### **Conclusión**

En el contexto de la recolección de datos se describió esta problemática con el que contaba la organización, se ha dimensionado de alguna manera el impacto que tendrá el sistema web para inmuebles en la misma, como se citaba en anteriores párrafos la empresa cuenta con unas innumerables tareas que se deben realizar manualmente dando margen a errores que afectan el rendimiento en eficacia y tiempo dando lugar a la necesidad de acelerar los procesos para ser más productivos.

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.

Para la misma se investigó a profundidad en el mundo web relevando sobre consultoras de la ciudad que se dedican a la administración de inmuebles donde no se han encontrado sistemas con los requisitos expuestos a través de la recolección de datos que ayudaron a especificar las funcionalidades necesarias para ser desarrollada.

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 íntegros al ser resguardadas en una base de datos

### **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.
-

### **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-depiera.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>

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. (s.f.). *Residentia*. Obtenido de Residentia: <https://www.residentia.net/>

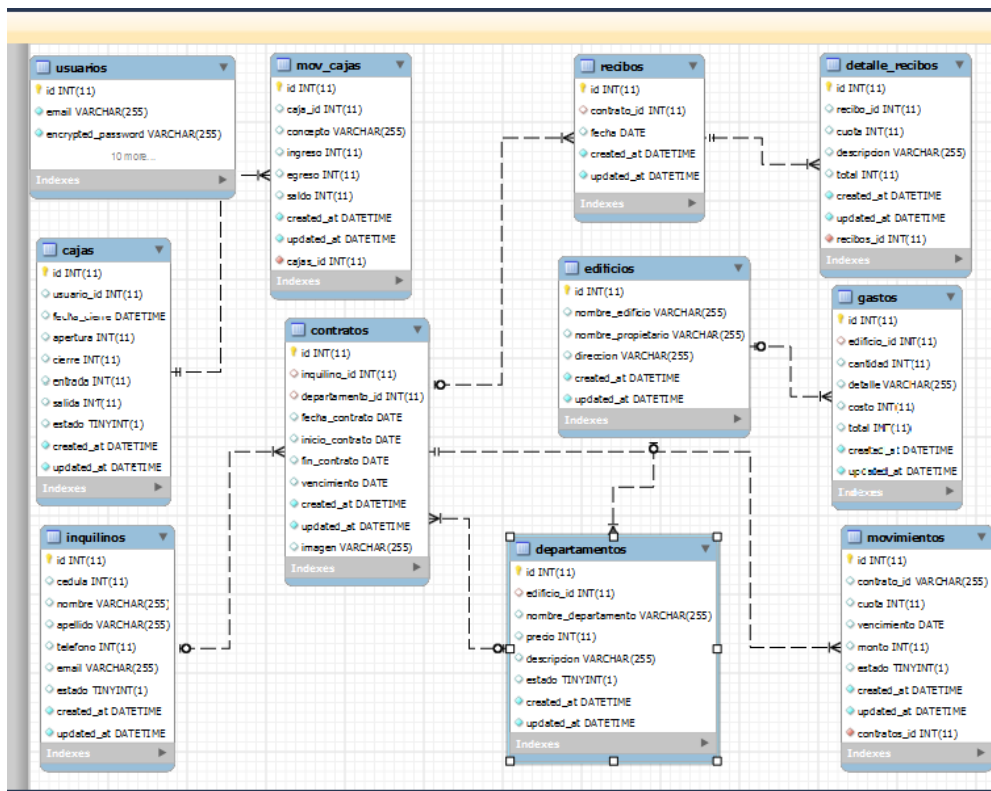
Spano, D. d. (2010). *El open source como facilitador del open access. Impacto y visibilidad de las revistas científicas*. Obtenido de [https://s3.amazonaws.com/academia.edu.documents/36153922/ponencia\\_spano\\_elis.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1503963768&Signature=%2F80MmzTLjgSoK%2Fl3NXS2jitD%2Fqc%3D&response-content-disposition=inline%3B%20filename%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_)

## Anexos

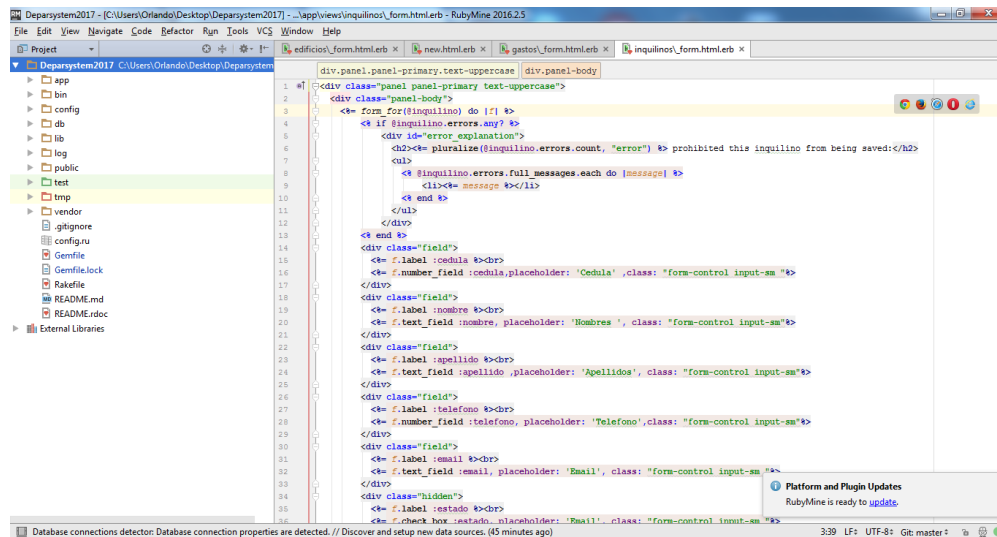
### Anexo 1: Enlace al proyecto

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

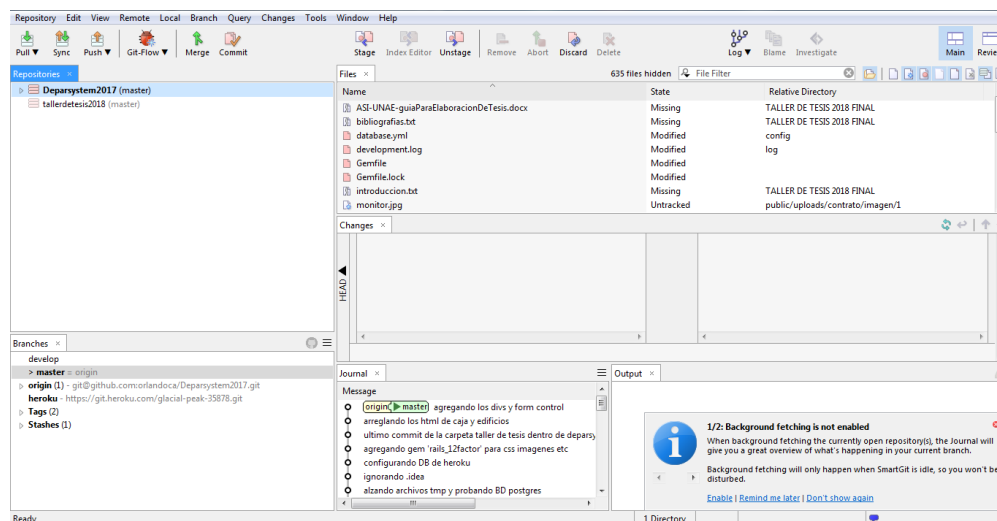
### Anexo 2: Modelado del Sistema



### Anexo 3: Estructura del proyecto



### Anexo 4: Repositorio local SmartGit



## Anexo 5: 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
 exitosadamente.' }
    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 6: Código del Proyecto – Controlador Edificios-Departamentos

```

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

  # 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 7: Código del Proyecto – Controlador Contratos

```

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
      @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 8: Código del Proyecto –Gastos

```

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
  end
end

```

```

    @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

  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

```

## Anexo 9: Código del Proyecto –Controlador Cajas

```

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
end

```

```

        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
end

```

## Anexo 10: Código del Proyecto –Controlador Movimiento Caja

```

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
 exitosadamente.' }
        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
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.' }
    }

    format.json { render :show, status: :ok, location: @mov_caja }
  else
    format.html { render :edit }
    format.json { render json: @mov_caja.errors, status: :unprocessable_entity }
  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
end

```

## Anexo 11: Código del Proyecto –Controlador Pagos

```

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

  # GET /pagos
  # GET /pagos.json
  def index
    @pagos = Pago.all
  end
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 12: 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
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 }
  end
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
end

```

### Anexo 13: 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
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 14: Código del Proyecto –Reporte pdf del recibo

Aquí es donde se diseña el reporte del pdf, las medidas de los campos, 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
    end
  end
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 14: Código del Proyecto –Routes

```

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
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
```