Facultad de Ciencias de la Computación

Benemérita Universidad
Autónoma de Puebla

Sistema de Administración de Existencia de un Negocio

T E S I S

para obtener el titulo de:

INGENIERO en Ciencias de la computación

P r e s e n t a:

Daninayaloani Antolín López Nicolás

Asesora:

M.C. Meliza Contreras González

Puebla, Puebla.
2013

KUTAURI NUU:

TAARI

NAARI

KUA'ARI JINÁ'A

Contenido

Introducción

La adopción que ha tenido la computación ha sido muy rápida, llegando a introducirse en los ámbitos sociales, económicos de los individuos y de las empresas, de manera directa o indirecta.

Es ya muy común que las grandes empresas hagan uso de software para el manejo de distintas tareas para ayudar a ser más eficiente. Las pequeñas empresas son las que comienzan a interesarse en hacer usos de sistemas de información, con la idea de mejorar el tratamiento de su información y explotación de estos datos, que los ayude a tomar mejores decisiones.

Este proyecto tiene como propósito desarrollar un sistema que permita administrar la existencia de productos que ofrece un negocio, mostrando gráficamente los productos que están por agotarse haciendo uso de indicadores de urgencia.

El sistema es una aplicación web la cual fue desarrollada siguiendo la metodología del Proceso Unificado de Rational , codificado en python y haciendo uso del framework Django.

Este documento esta dividido en 5 capítulos: Capitulo 1 describe el problema en general que se aborda; Capitulo 2 se muestra el marco de referencia de los conceptos mínimos necesarios para el desarrollo del sistema; Capitulo 3 consta del analísis y diseño del sistema; Capitulo 4 se describe los puntos importantes del desarrollo del sistema; Por último en el Capitulo 5 se muestran las conclusiones y posibles mejoras del sistema.

Capitulo 1: Presentación

La gestión del stock en una empresa es una de las tareas principales de cualquier clase de negocio que se dedique a la compra y venta de bienes o servicios, el correcto manejo del stock permite a la empresa mantener el control fidedigno de los productos en su haber.

Dada la complejidad de manejar el stock surge la necesidad de contar con software que permita obtener inventarios, estos a su vez ayuden a gestionar las necesidades y requerimientos de los clientes, y de la habilidad de los proveedores de suministrar nuevo producto.

El sistema de administración de existencias de un negocio, es una propuesta que busca ayudar a manejar de manera fácil y clara el stock de una pequeña empresa.

1.1 **Planteamiento del Problema**

El sistema de administración de existencias de un negocio, busca ayudar al usuario final en la toma de decisiones al momento de decidir que productos abastecer nuevamente en su almacén; Este sistema también hará más sencillo que el usuario obtenga reportes de sus productos en su almacén.

La información respecto a las ventas y reportes de productos en stock, suele ser delicada, es por que ello que se debe garantizar en todo momento el que solo ciertos usuarios con privilegios sean los que pueden visualizar esa información, el sistema será el encargado de garantizar estos privilegios.

1.2 **Objetivo General**

Desarrollar un sistema para administrar el stock de una pequeña empresa, donde se integre un tablero que combina datos e indicadores gráficos para visualizar de manera rápida la información del stock; Reporte de aquellos productos que se encuentren escasos, por agotarse y agotados.

1.3 **Requerimientos**

El sistema deberá cubrir los siguiente requisitos:

1.3.1 **Funcionales**

Reporte por cada uno de estas condicionantes:

- Mostrar los Productos en el almacén.
- Mostrar los Productos por existencia.
- Mostrar los Productos inexistentes (0 en stock).
- Mostrar los Productos escasos.
- Exportar reportes a formato PDF (portable document format)

- Acceso al sistema mediante login.
- Tres niveles diferentes de permisos para el acceso al sistema.
  - Administrador
  - Gerente
  - Dependiente
- Tablero con gráficos (circulo de color) de los productos que se requiera resurtir.

1.3.2 **No funcionales**

- Hacer que el sistema sea liviano.
- Sistema fácil de utilizar.

- Sistema rápido.
- El front-end debe tener un manejo simple e intuitivo.
- La interfaz se realizará con HTML5.
- El sistema debe visualizarse y funcionar en el Navegador Web Safari®.

1.4 **Alcances y Limitaciones**

Alcances

El software desarrollado permitira a los usuarios recuperar información de los datos de los productos almacenados en una base de datos, para que por medio de gráficos se represente el estado individual de cada producto..

El software será independiete de la plataforma utilizada.

Limitaciones

Las pruebas no se realizarán en un ambiente 100% real, ya que no se cuenta con un conjunto grande y fiel de datos.

Capitulo 2: Marco Teórico

2.1 **Ingeniería de software**

La ingenería de Software ofrece métodos y técnicas, para desarrollar y mantener sistemas de software, para que se comporten de forma fiable y eficiente, logrando así satisfacer todas las necesidades de los clientes. Hoy día es cada vez más frecuente la consideración de la Ingeniería del Software como una nueva área de la ingeniería, y el ingeniero del software comienza a ser una profesión implantada en el mundo laboral internacional, con derechos, deberes y responsabilidades que cumplir. [1]

En la construcción y desarrollo de proyectos se aplican métodos y técnicas para resolver los problemas, la informática aporta herramientas y procedimientos sobre los que se apoya la Ingeniería de Software:

- Mejoran la calidad de los productos de software.
- Aumentar la productividad y el trabajo de los ingenieros de software.
- Facilitar el control del proceso de desarrollo de software.
- Suministrar a los desarrolladores las bases para construir software de alta calidad en forma eficiente.
- Definir una disciplina que garantice la producción y el mantenimiento de los productos de software desarrollados en el plazo fijado y dentro del costo estimado.

Los métodos de la Ingeniería de Software indican cómo construir técnicamente el software, abarcan una gran gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Los métodos de Ingeniería de Software dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Entre los objetivos que abarca la ingeniería de software pueden describirse en base a las: “Las cinco C”:

Capacidad: las actividades de la organización están influenciadas por la capacidad de esta para procesar transacciones con rapidez y eficiencia. Los sistemas de información mejoran esta capacidad en tres formas:

- Aumentan la velocidad de procesamiento.
- Aumento en el volumen.
- Recuperación más rápida de la información.

Comunicación: la falta de comunicación es una fuente común de dificultades que afectan tanto al cliente como al empleado. Sin embargo, los sistemas de información bien desarrollados amplían la comunicación y facilitan la integración de funciones individuales.

Costo: para determinar si la compañía evoluciona en la forma esperada, de acuerdo con lo presupuestado, se debe de llevar a cabo el seguimiento de los costos de mano de obra, bienes y gastos generales.

Control: este se obtiene al tomar en cuenta que debemos tener mayor seguridad de información y un menor margen de error, ya que algunas veces el hecho de que los datos pueden ser guardados de una manera adecuada para su lectura por medio de una máquina, es una seguridad difícil de alcanzar en un medio ambiente donde no existen computadoras.

Competitividad: los sistemas de información computacionales son un arma estratégica, capaces de cambiar la forma en que la compañía compite en el mercado, en consecuencias estos sistemas mejoran la organización y le ayudan a ganar “ventaja competitiva”.

2.2 **RUP**

El Proceso Unificado de Rational es un *proceso de ingeniería de software*. Proporciona un enfoque para la asignación de tareas y responsabilidades dentro de un grupo de desarrollo. Su objetivo es asegurar que la producción de software de gran calidad que satisfaga las necesidades de los usuarios finales dentro de un calendario predecible y presupuesto. [2]

El Proceso Unificado de Rational (Rational Unified Process en inglés, habitualmente resumido como RUP) junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de procesos adaptables al contexto y necesidades de cada organización.

También se conoce por este nombre al software desarrollado por Rational, hoy propiedad de IBM, el cual incluye información entrelazada de diversos artefactos y descripciones de las diversas actividades. Está incluido en el Rational Method Composer (RMC), que permite la personalización de acuerdo con las necesidades.

Originalmente se diseñó un proceso genérico y de dominio público, el Proceso Unificado, y una especificación más detallada, el Rational Unified Process, que se vendiera como producto independiente.

Principales Características

El Proceso Unificado de Rational es una metodología de desarrollo de software orientada a objetos creada por *Rational Software Corporation.* Los creadores de la metodología son los mismos que los del UML: Ivar Jacobson, Grady Booch y James Rumbaugh, que respectivamente eran autores de las metodologías: *Process Objectory,* el método Booch y la metodología OMT. Como toda metodología de desarrollo software su finalidad es convertir las especificaciones que da el cliente en un sistema software. Las características que tiene el R.U.P. son:

- Está basado en componentes. Estos componentes a su vez están conectados entre sí a través de interfaces.
- Utiliza el UML como notación básica.
- Dirigido por casos de uso.
- Centrado en la arquitectura.
- Ciclo de vida iterativo e incremental.

Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos y roles. Otras características esenciales del Proceso Unificado de Rational son:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Pretende implementar las mejores prácticas en Ingeniería de Software.
- Desarrollo iterativo.
- Administración de requisitos.
- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.

Principios de desarrollo

El RUP está basado en 6 principios clave que son los siguientes:

Adaptar el proceso

El proceso deberá adaptarse a las necesidades del cliente ya que es muy importante interactuar con él. Las características propias del proyecto u organización. El tamaño del mismo, así como su tipo o las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto en un área subformal.

Equilibrar prioridades

Los requisitos de los diversos participantes pueden ser diferentes, contradictorios o disputarse recursos limitados. *Debe encontrarse un equilibrio que satisfaga los deseos de todos*. Gracias a este equilibrio se podrán corregir desacuerdos que surjan en el futuro.

Demostrar valor iterativamente

Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados

Colaboración entre equipos

El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requisitos, desarrollo, evaluaciones, planes, resultados, etc.

Elevar el nivel de abstracción

Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o marcos de referencia (frameworks) por nombrar algunos. Esto evita que los ingenieros de software vayan directamente de los requisitos a la codificación de software a la medida del cliente, sin saber con certeza qué codificar para satisfacer de la mejor manera los requisitos y sin comenzar desde un principio pensando en la reutilizar el código. Un alto nivel de abstracción también permite discusiones sobre diversos niveles y soluciones

arquitectónicas. Éstas se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con el lenguaje UML.

Enfocarse en la calidad

El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción. El aseguramiento de la calidad forma parte del proceso de desarrollo y no de un grupo independiente.

Ciclo de Vida con sistemas RUP

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. El ciclo de vida organiza las tareas en fases e iteraciones.

RUP divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en las distintas actividades. En la Figura muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del proyecto, la eliminación de los riesgos críticos, y al establecimiento de una baseline (Línea Base) de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requisitos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la baseline de la arquitectura, abarcan más los flujos de trabajo de requisitos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la baseline de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realiza una pequeña cascada para cada ciclo. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios.

Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía, en la Figura 1.1 se muestra el ciclo de vida RUP.

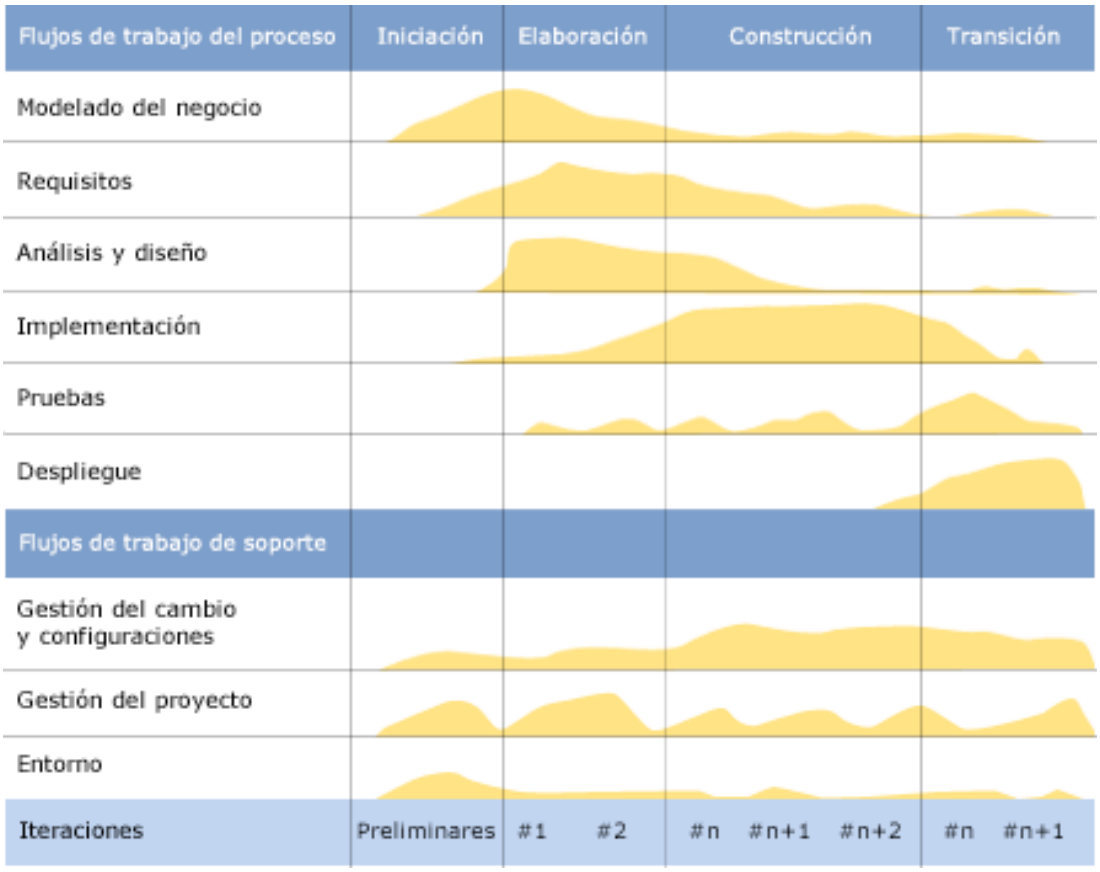

Figura 1.1. Esfuerzo dedicado en cada actividad, según fase del proyecto.

2.3 **UML**

Que es UML

Lenguaje Unificado de Modelado (LUM o UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido, basado en el sistema de unificación de Grady Booch, Ivar Jacobson y James Rumbaugh, un método de modelado orientado a objetos.

El Lenguaje Unificado de Modelado (UML) es un lenguaje gráfico para la visualización, especificación, construcción y documentación de los artefactos de un sistema de software.

El modelado es un parte esencial de proyecto de software extenso, cual ayuda en el desarrollo de medianos y pequeños proyectos. UML hace posible describir sistemas con palabras e imágenes. Puede ser usado para modelar distintos sistemas: sistemas de software, sistemas de negocios, o cualquier otro sistema. [3]

2.3.1 **Modelos de UML**

UML esta conformado por diferentes elementos gráficos para conformar gráficos, también tiene una serie de reglas para combinar tales gráficos.

Los diagramas tienen como objetivo presentar diversas perspectivas de un sistema. A esto se le llama *Modelo*.

Un modelo UML representa lo que hará el sistema, pero no como se realizara la implementación.

2.3.1.1 Elementos de UML

- Diagrama de casos de uso
  - Caso de uso
    - Actor
    - Descripción de casos de uso
- Diagrama de clases
- Clase
  - Atributos
  - Operaciones
  - Plantillas
  - Asociaciones de clases
    - Generalización
    - Asociaciones
    - Acumulación
    - Composición
- Otros componentes de los diagramas de clases
  - Interfaces
  - Tipo de datos
  - Enumeraciones
  - Paquetes
- Diagramas de secuencia
- Diagramas de colaboración
- Diagrama de estado

- Listo
- Escuchando
- Trabajando
- Detenido

- Diagrama de actividad
  - Actividad

- Elementos de ayuda
  - Línea de texto
  - Notas de texto y enlaces
  - Cajas

- Diagramas de relación de entidad
  - Entidad
  - Atributos de la entidad
  - Restricciones

- Conceptos del diagrama de relaciones de entidades extendido (EER)
  - Especialización
  - Especialización disjunta
  - Especialización de solapamiento
  - Categoría

2.4 **Modelo para procesos de software**

2.4.1 **Proceso Unificado**

El Proceso Unificado (PU) es un proceso de desarrollo de software que describe “el conjunto de actividades necesarias para transformar los requisitos del usuario en un sistema software”. Está dirigido por casos de uso, centrado en la arquitectura del sistema, y es iterativo e incremental. [4]

Dirigido por casos de uso

- Un caso de uso es un fragmento de funcionalidad del sistema que proporciona un resultado de valor a un usuario. Los casos de uso modelan los requerimientos funcionales del sistema.
- Todos los casos de uso juntos constituyen el modelo de casos de uso.
- Los casos de uso también guían el proceso de desarrollo (diseño, implementación, y prueba). Basándose en los casos de uso los desarrolladores crean una serie de modelos de diseño e implementación que llevan a cabo los casos de uso. De este modo los casos de uso no solo inician el proceso de desarrollo sino que le proporcionan un hilo conductor, avanza a través de una serie de flujos de trabajo que parten de los casos de uso.

Centrado en la arquitectura

La arquitectura de un sistema software se describe mediante diferentes vistas del sistema en construcción. El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles de lado.

Los casos de uso y la arquitectura están profundamente relacionados. Los casos de uso deben encajar en la arquitectura, y a su vez la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, actualmente y a futuro.

El arquitecto desarrolla la forma o arquitectura a partir de la comprensión de un conjunto reducido de casos de uso fundamentales o críticos (usualmente no más del 10 % del total). En forma resumida, podemos decir que el arquitecto:

- Crea un esquema en borrador de la arquitectura comenzando por la parte no específica de los casos de uso (por ejemplo la plataforma) pero con una comprensión general de los casos de uso fundamentales.
- A continuación, trabaja con un conjunto de casos de uso, claves o fundamentales. Cada caso de uso es especificado en detalle y realizado en términos de subsistemas, clases, y componentes.
- A medida que los casos de uso se especifican y maduran, se descubre más de la arquitectura, y esto a su vez lleva a la maduración de más casos de uso. Este proceso continúa hasta que se considere que la arquitectura es estable.

Iterativo e Incremental

Es práctico dividir el esfuerzo de desarrollo de un proyecto de software en partes más pequeñas (mini proyectos). Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo, y los incrementos a crecimientos en el producto. Las iteraciones deben estar controladas. Esto significa que deben seleccionarse y ejecutarse de una forma planificada.

Los desarrolladores basan la selección de lo que implementarán en cada iteración en dos cosas: el conjunto de casos de uso que amplían la funcionalidad, y en los riesgos más importantes que deben mitigarse.

En cada iteración los desarrolladores identifican y especifican los casos de uso relevantes, crean un diseño utilizando la arquitectura seleccionada como guía, para implementar dichos casos de uso. Si la iteración cumple sus objetivos, se continúa con la próxima, en otro caso deben revisarse las decisiones previas y probar un nuevo enfoque.

Beneficios del enfoque iterativo

- La iteración controlada reduce el riesgo a los costos de un solo incremento.
- Reduce el riesgo de retrasos en el calendario atacando los riesgos mas importantes primero.
- Acelera el desarrollo. Los trabajadores trabajan de manera más eficiente al obtener resultados a corto plazo.
- Tiene un enfoque más realista al reconocer que los requisitos no pueden definirse completamente al principio.

2.5 **Base de Datos**

Una **base de datos** es un almacén que permite guardar grandes cantidades de información de forma organizada pertenecientes a un mismo contexto y almacenados sistemáticamente para su uso posterior.

Una base de datos se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada.

Las bases de datos no son un fenómeno nuevo. El término de base de datos fue escuchado por primera vez en 1963, en un simposio celebrado en California, USA.

Herman Hollerith mecanizó el almacenamiento del Censo de EEUU de 1890 en lo que de alguna forma se considera la primera base de datos significativa “computarizada”.

En la actualidad, y gracias al desarrollo tecnológico de campos como la informática y la electrónica, se puede ofrecer un amplio rango de soluciones al problema de almacenar datos.

Desde el punto de vista informático, la base de datos es un sistema formado por un conjunto de datos almacenados en discos que permiten el acceso directo a ellos y un conjunto de programas que manipulen ese conjunto de datos.

2.5.2 **Sistema Gestor de Base de Datos**

Un sistema gestor de bases de datos o SGBD (aunque se suele utilizar más a menudo las siglas DBMS procedentes del inglés, Data Base Management System) es el software que permite a los usuarios procesar, describir, administrar y recuperar los datos almacenados en una base de datos.

En estos Sistemas se proporciona un conjunto coordinado de programas, procedimientos y lenguajes que permiten a los distintos usuarios realizar sus tareas habituales con los datos, garantizando además la seguridad de los mismos. [5]

El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente.

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La gestión de datos implica tanto la definición de estructuras para almacenar la información como proporcionar mecanismos para la manipulación de la información. Además, los sistemas de bases de datos deben de garantizar la fiabilidad de la información almacenada, a pesar de la caída del sistema o de los accesos no autorizados. Si los datos van a ser compartidos entre diferentes usuarios, el sistema debe evitar posibles resultados anormales.

Para los SGBD existen distintos objetivos que deben cumplir:

- Abstracción de la información. Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario. Así, se definen varios niveles de abstracción.
- Independencia. La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- Consistencia. En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. Por otra parte, la base de datos representa una realidad determinada que tiene determinadas condiciones, por

ejemplo que los menores de edad no pueden tener licencia de conducir. El sistema no debería aceptar datos de un conductor menor de edad. En los SGBD existen herramientas que facilitan la programación de este tipo de condiciones.

- Seguridad. La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.
- Manejo de Transacciones. Una transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.
- Tiempo de respuesta. Lógicamente, es deseable minimizar el tiempo que el SGBD tarda en darnos la información solicitada y en almacenar los cambios realizados.

2.5.3 **Modelo de bases de datos**

Un modelo de base de datos o esquema de base de datos es la estructura o el formato de una base de dato, básicamente es una descripción de algo conocido como contenedor de datos (algo en donde se guarda la información), así como de los métodos para almacenar y recuperar información de esos contenedores.

Los modelos de Base de Datos se clasifican de acuerdo al modelo de administración de datos.

Algunos de los modelos de Base de Datos que con más frecuencia se utilizan, son los siguientes:

Bases de datos jerárquicas

Estas base de datos almacenan su información en una estructura jerárquica. En este modelo los datos se organizan en una forma similar a un árbol (visto al revés), en donde un nodo padre de información puede tener varios hijos. El nodo que no tiene padres es llamado raíz, y a los nodos que no tienen hijos se los conoce como hojas. Este tipo de base de datos

son muy útiles en el caso de aplicaciones que manejan un gran volumen de información y datos muy compartidos permitiendo crear estructuras estables y de gran rendimiento.

Una de las principales limitaciones de este modelo es su incapacidad de representar eficientemente la redundancia de datos.

Base de datos de red

La principal diferencia de este modelo frente al jerárquico, es la modificación del concepto de nodo: Este modelo permite que un mismo nodo tenga varios padres (posibilidad no permitida en el modelo jerárquico).

Esta mejora permite ofrecer una solución eficiente al problema de redundancia de datos; pero, aun así, la dificultad que significa administrar la información en una base de datos de red ha significado que sea un modelo utilizado en su mayoría por programadores más que por usuarios finales.

Bases de datos transaccionales

Son bases de datos cuyo único fin es el envío y recepción de datos a grandes velocidades, estas bases son muy poco comunes y están dirigidas por lo general al entorno de análisis de calidad, datos de producción e industrial, es por ello que tanto la redundancia y duplicación de información no es un problema como con las demás bases de datos, por lo general para poder aprovechar al máximo, permiten algún tipo de conectividad a bases de datos relacionales.

En general el único fin de las Base de Datos transaccionales es: recolectar y recuperar los datos a la mayor velocidad posible.

Bases de datos relacionales

Este modelo es el más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente, ya que utiliza tablas bidimensionales para la representación lógica de los datos y sus relaciones.

Su idea fundamental es el uso de "relaciones", estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "tuplas". Cada relación puede visualizarse como una tabla que está compuesta por registros (las filas de una tabla), que representarían las tuplas, y campos (las columnas de una tabla).

El lenguaje más habitual para construir las consultas a bases de datos relacionales es SQL, (*Structured Query Language*) Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

Algunas de sus principales características son:

- Puede ser entendido y usado por cualquier usuario.
- Permite ampliar el esquema conceptual sin modificar las aplicaciones de gestión.
- Los usuarios no necesitan saber donde se encuentran los datos físicamente.

Bases de datos multidimensionales

Son bases de datos ideadas para desarrollar aplicaciones muy concretas, como creación de Cubos OLAP. Básicamente no se diferencian demasiado de las bases de datos tes (una tabla en una base de datos relacional podría serlo también en una base de datos multidimensional), la diferencia está más bien a nivel conceptual; en las bases de datos multidimensionales los campos o atributos de una tabla pueden ser de dos tipos, o bien representan dimensiones de la tabla, o bien representan métricas que se desean estudiar.

2.5.4 **Normalización**

La normalización es un concepto que hace referencia a la relaciones. Básicamente el principio de **normalización** indica que las tablas de las base de datos eliminarán las incoherencias y redundancias, y minimizarán la ineficacia.

Las bases de datos se describen como incoherentes cuando sus datos se introducen de forma incoherente o cuando los datos de una tabla no coinciden con los datos introducidos en otra tabla.

Una base de datos *ineficaz* no permite aislar los datos exactos que desea. Una base de datos que almacena todos sus datos en una tabla obligará a pasar por innumerables campos simplemente para recuperar un dato. Por otra parte, una base de datos completamente normalizada almacena cada información en su propia tabla e identifica cada información con su propia clave principal.

La falta de normalización puede ser intencionada o resultado de una falta de experiencia o cuidado del diseñador original de la base de datos. En cualquier caso, si se elige normalizar una base de datos existente, se debe hacer en una etapa temprana del desarrollo de las aplicaciones, ya que este depende de la estructura de las tablas.

La normalización es una serie de reglas que involucran análisis y transformación de las estructuras de datos en relaciones que exhiben propiedades únicas de consistencia, mínima redundancia y máxima estabilidad. [6]

Regla 1: *Unicidad de campo*: Cada campo de una tabla debe contener un único tipo de información.

Regla 2: *Clave Principal*: Cada tabla debe tener un único identificador, o clave principal, que esté formado por uno o más campos de la tabla.

En un buen diseño de base de datos relacional, cada uno de los registros de cualquier tabla debe de ser identificado de forma única.

Regla 3: *Dependencia Funcional*: Para cada valor único de la clave principal, los valores de las columnas de datos deben estar relacionados y deben describir completamente el contenido de la tabla.

Regla 4: I*ndependencia de los campos*: Debe ser posible realizar cambios en cualquier campo que no forme parte de la clave principal sin que para ello se vea afectado cualquier otro campo.

2.5.4.1 Formas Normales

Se dice que un esquema de relación está en una determinada forma normal si satisface un conjunto determinado de restricciones sobre los atributos. Cuantas más restricciones existan, menor será el número de relaciones que las satisfagan. Y cuanto más alta sea la forma normal en la que se encuentran los esquemas de relación, menores serán los problemas en el mantenimiento de la base de datos.

2.5.4.1.1 Primera Forma Normal

Una relación R se encuentra en primera forma normal (1FN) si y sólo si todos los dominios simples subyacentes contienen solo valores atómicos. Es decir, que el cruce de una fila con una columna sólo tiene un dato (no existen grupos repetitivos). [7]

1.5.4.1.2 Segunda Forma Normal

Una relación R se encuentra en segunda forma normal (2FN) si y sólo si está en primera forma normal y todos los atributos no clave, dependen funcionalmente de forma completa de la clave primaria. [7]

2.6.4.1.2 Tercera Forma Normal

Se dice que una relación está en tercera forma normal (3FN) si y sólo si está en segunda forma normal y todos los atributos no clave dependen de manera no transitiva de la clave primaria. [7]

2.5 **Arquitectura Cliente – Servidor**

EL termino Cliente-Servidor fue usado por primera vez en la década de 1980 haciendo referencia a las computadoras personales en una red. En la actual el modelo Cliente-Servidor a empezó a ganar aceptación a finales de los 80's. El termino Cliente-Servidor es usado para describir un modelo computacional para el desarrollo de sistemas. Este modelo esta basado en la distribución de funciones entre dos tipos de entidades independientes y autónomas : Servidor y Cliente. Un cliente es cualquier procesos que solicita servicios específicos de los procesos del servidor. Un servidor es un proceso que provee servicios solicitados por los clientes.

Una única maquina puede ser ambos cliente y servidor dependiendo de la configuración del software. Los procesos de Cliente y Servidor puede estar en la misma computadora o en diferentes computadoras conectadas por una red.

En general, Cliente-Servidor es un sistema. No solo es hardware o software. No es necesariamente un programa el cual viene en una caja para ser instalado en el disco duro de una computadora. Cliente-Servidor es un conglomerado de equipos de computo, infraestructura, y programas de software trabajando juntos para llevar a cabo las tareas que permiten a los usuarios ser mas eficientes y productivos. Las aplicaciones Cliente-Servidor pueden distinguirse por la naturaleza de servicios o tipos de soluciones que proveen.

Cuando los procesos del Cliente y Servidor residen en dos o más computadoras independientes en una red, el Servidor puede proporcionar servicios a más de un Cliente. Además, un Cliente puede solicitar los servicios de varios Servidores de la red sin tener en cuenta la ubicación o las características físicas de la computadora en la que se encuentran alojados los procesos del Servidor. La red une al Servidor y Cliente junto, proporcionando el medio a través del cual los clientes y el servidor se comunican. [8]

Características

En la arquitectura Cliente - Servidor el remitente de una solicitud es conocido como *cliente*.

Sus características son:

- Es quien inicia solicitudes o peticiones, tienen por tanto un papel activo en la comunicación.
- Espera y recibe las respuestas del servidor.
- Por lo general, puede conectarse a varios servidores a la vez.
- Normalmente interactúa directamente con los usuarios finales mediante una interfaz gráfica de usuario.

Al receptor de la solicitud enviada por el cliente se conoce como servidor.

Sus características son:

- Al iniciarse esperan a que lleguen las solicitudes de los clientes, desempeñan entonces un papel pasivo en la comunicación (dispositivo esclavo).
- Tras la recepción de una solicitud, la procesan y luego envían la respuesta al cliente.
- Por lo general, aceptan conexiones desde un gran número de clientes (en ciertos casos el número máximo de peticiones puede estar limitado).
- No es frecuente que interactúen directamente con los usuarios finales.

2.5.1 **Apache**

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.11 y la noción de sitio virtual. [9]

El servidor Apache se desarrolla dentro del proyecto HTTP Server de la Apache Software Foundation. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA HTTPd 1.3, pero más tarde fue reescrito por completo.

El nombre de “Apache” fue elegido por el respecto que le tienen a los Nación de Nativos Americanos Apache, conocidos por sus grandes habilidades en estrategias de guerra y su resistencia inagotable. Apache consistía solamente de conjunto de parches, por ello el nombre en inglés de “*a patchy server*” (un servidor parchado), pero este no es el origen del por que el nombre. [10]

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración.

Apache tiene amplia aceptación en la red: desde 1996, Apache, es el servidor HTTP más usado. Alcanzó su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70% de los sitios web en el mundo, sin embargo ha sufrido un descenso en su cuota de mercado en los últimos años. (Estadísticas históricas y de uso diario proporcionadas por Netcraft3).

La mayoría de las vulnerabilidades de la seguridad descubiertas y resueltas tan sólo pueden ser aprovechadas por usuarios locales y no remotamente. Sin embargo, algunas se pueden accionar remotamente en ciertas situaciones, o explotar por los usuarios locales malévolos en las disposiciones de recibimiento compartidas que utilizan PHP como módulo de Apache.

Apache es usado principalmente para enviar páginas Web estáticas y dinámicas en la World Wide Web. Muchas aplicaciones Web están diseñadas asumiendo como ambiente de implantación a Apache, o que utilizarán características propias de este servidor Web.

2.6 **Lenguaje de programación Python**

¿Qué es Python?

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible.

Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos. [11]

- Orientado a objetos

  La orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos.

  Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

- Multiplataforma

  El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.

Python tienen varias ventajas, de las cuales sobresalen:

- Lenguaje Multipropósito.
- Sintaxis simple.
- Multiplataforma.
- Puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos.

2.7 **Framework Web Django**

Genéricamente, un framework es un conjunto de herramientas, librerías, convenciones y buenas prácticas que pretenden encapsular las tareas repetitivas en módulos genéricos fácilmente reutilizables.

"Django es un framework web de código abierto escrito en *Python* que permite construir aplicaciones web más rápido y con menos código." [12]

Django abstrae en una colección de librerías escritas en Python, la mayor parte del proceso de desarrollo web, permitiendo a los desarrolladores implementar rápidamente respaldados por Bases de Datos sitios web que utilizan páginas web dinámicas.

Mediante un sistema dinámico, nos referimos a las páginas web que se construyen sobre la marcha usando los datos de la petición del navegador, la URL y una base de datos.

El uso de páginas dinámicas resuelven distintos problemas. El mayor problema que resuelve es tratar de mantener numerosas páginas estáticas esparcidos por todo un sitio web clásico.

Django junto con otros Frameworks, se clasifica (Figura 1.4) como parte de la tercera generación del desarrollo Web .

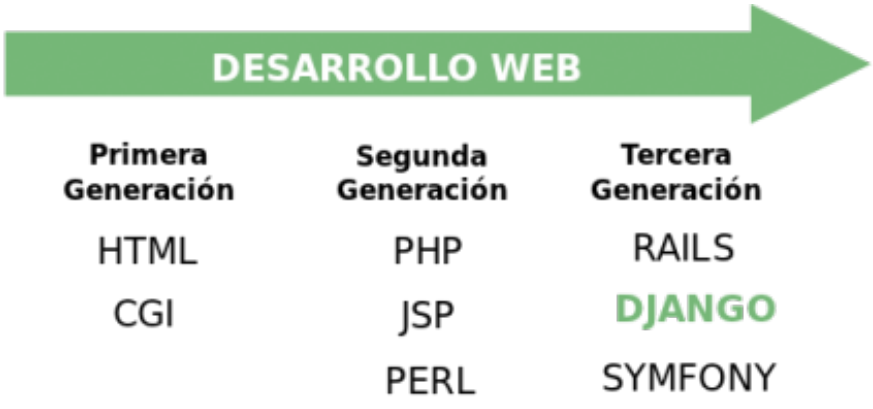

Figura 1.4 Clasificación de Frameworks.

2.7.1 **MTV y Django**

Django parcialmente sigue el patrón MVC por lo que se le puede llamar un Framework MVC. A continuación a grandes rasgos como se descompone el M, V, C en Django:

- M, la parte de acceso a datos, es manejado por la capa de base de datos por Django.
- V, la parte que selecciona cuales son los datos que serán mostrados y como deberán ser desplegados, es manejado por las vistas (views) y plantillas (templates).
- C, la parte de los delegados para una vista (view) dependiendo de las entrada de un usuario, es manejado por el framework Django, siguiendo su URLconf y llamando la apropiada función de Python para la URL dada.

Debido a que la "C" es manejado por el propio Framework y la mayor parte de la emoción de usar Django sucede en los modelos, plantillas y vistas, Django ha sido referido como un Framework MTV. [13]

MTV define una forma de desarrollar software en la que el código para definir y acceder a los datos (el modelo) esta separado del pedido lógico de asignación de ruta (el controlador), que a su vez está separado de la interfaz del usuario (vista).

Teniendo así dos principales ventajas, una de ellas es que sea fácil por un lado desarrollar la base de datos y otro desarrollar las páginas web. Otra es que el código y las direcciones URL son mucho más limpias, más fácil de mantener, y más elegante.

A continuación, en la Figura 1.5 se muestra el funcionamiento del MTV en Django.

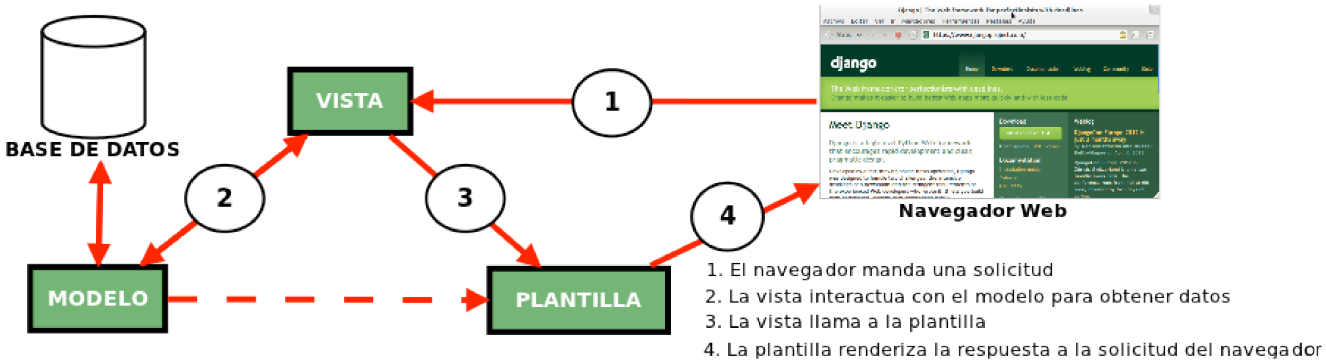

Figura 1.5 Funcionamiento del modelo MTV de Django.

A continuación se explica a que refieren: El Modelo, la Vista y el Template dentro del Framework Web Django.

2.7.1.1 Modelo

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

2.7.1.2 Vista

La vista se presenta en forma de funciones en Python, su propósito es determinar que datos serán visualizados La vista también se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y la validación de datos a través de formularios. Se encarga sólo de la manipulación de los datos, y no la presentación de estos, que llega a confundir por el nombre.

2.7.1. 3 Template

Template o plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, no solamente crea contenido en HTML (también XML, CSS, Javascript, CSV, etc).

La plantilla recibe los datos de la vista y luego los organiza para la presentación al navegador web. Las etiquetas que Django usa para las plantillas permiten que sea flexible para los diseñadores del Frontend.

2.7.2 **Despachador de URL**

Django posee un mapeo de URLs que permite controlar el despliegue de las vistas, esta configuración es conocida como URLConf.

El trabajo del URLConf es leer la URL que el usuario solicitó, encontrar la vista apropiada para la solicitud y pasar cualquier variable que la vista necesite para completar su trabajo. El URLConf esta construido con expresiones regulares en Python. Este URLConf permite que las rutas que maneje Django sean agradables y fácil de entender para el usuario.

Considerando al URLConf en el esquema anterior, como se muestra en Figura 1.6, se tendría un resultado más completo de la manera de como funciona de Django.

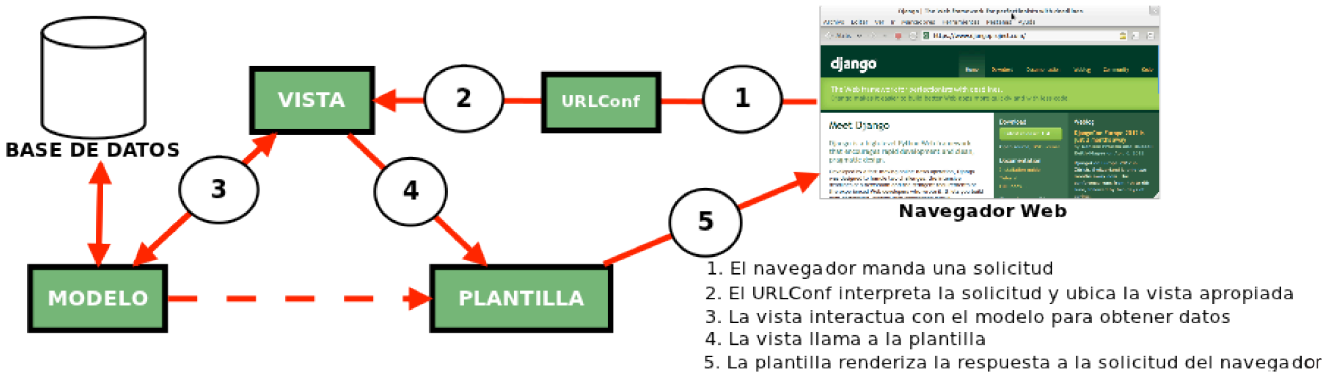

Figura 1.6 Funcionamiento del modelo MTV de Django.

2.7.3 **Archivos Predeterminados de Django**

Django hace uso de una serie de archivos para lograr usar el MVT, los cuales a continuación se describe:

Archivos del Proyecto

- __init__.py: Este es un archivo vacío que le dice a Python que debe considerar este directorio como un paquete de Python.
- manage.py: Este archivo contiene una porción de código que permite interactuar con el proyecto de Django.
- settings.py: Este archivo contiene todas las configuraciones para el proyecto.
- urls.py: Contiene las rutas que están disponibles en el proyecto, manejado por URLConf.

Archivos de la Aplicación

- __init__.py
- models.py: En este archivo se declaran las clases del modelo.
- views.py: En este archivo se declaran las funciones de la vista.
- test.py: En este archivo se declaran las pruebas necesarias para la aplicación

2.7.4 **Ventajas**

Django tiene varias ventajas, de las más conocidas y utilizadas son:

- Potente ORM (mapeo objeto-relacional).
- Muy flexible.
- Eficiencia SQL.
- Sintaxis Concisa.
- Opción de escribir SQL crudo.
- Permite el desarrollo en equipo.
- Uso de MTV.

2.7.5 **Componentes más usados de Django**

Django cuenta con muchas librerías, que logran hacer que el desarrollo sea más simple y rápido, de las cuales las mas utilizadas son las siguientes.

- Librería ORM.
- Despachador URL.
- Administrador.
- Gestor de Usuarios.
- Internacionalización.
- Manejo de sesiones.
- Cache.

Capitulo 3: Análisis y Diseño

3.1 **Diagrama de casos de uso**

Una vez analizado los requerimientos del sistema se deduce claramente los casos de uso, los cuales se muestran en la Figura 2.1.

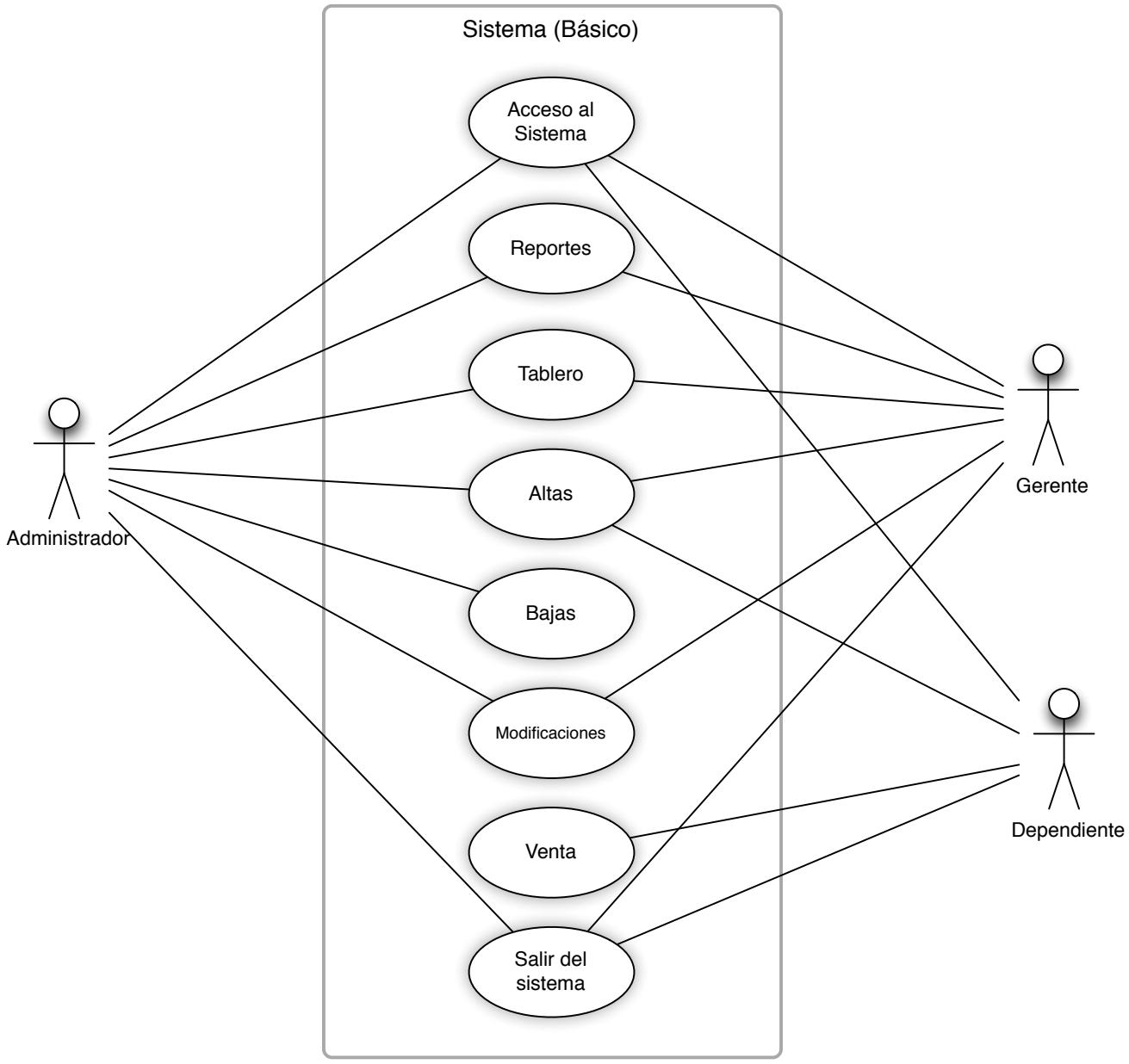

Figura 2.1. Diagrama de Casos de uso

3.2 **Especificación de casos de uso**

Conocido los distintos casos de uso, a continuación se explica individualmente cada caso para obtener un perspectiva más amplia del objetivo de cada uno de ellos y la manera en como se interactúa hacia con ellos; Se describe los diferentes roles dentro del sistema.

Indicadores de Stock	
Función:	Mostar una representación gráfica de productos escasos , agotados y suficientes.
Descripción:	Permite al administrador y Gerente, ver gráficos representando en 3 estados el estado de cada producto dentro del stock.
Entrada:	
Fuentes:	
Salida:	Se muestra en pantalla, gráficos (circulo de color) y en el, el nombre del producto.
Proceso:	El sistema recupera los datos almacenados en la tabla Producto de los campos: *nombre*, *cantidad* e *indicador*. De acuerdo al valor de *indicador* se recupera de la tabla *Indicador* los campos *minimo* y *maximo*, para tomar las siguientes consideraciones: 1.- Si el valor del *cantidad* esta por debajo del valor de *minimo*, muestra un circulo rojo. 2.- Si el valor del *cantidad* está entre el valor de *minimo* y *maximo*, muestra un circulo amarillo. 3.- Si el valor del *cantidad* es mayor a *maximo*, muestra un circulo verde.
Restricciones:	
Precondición:	Haber ingresado al Sistema con Credencial de Administrador o Gerente.
Poscondición	

Acceso al sistema	
Función:	Permitir el acceso de sistema, para su administración.
Descripción:	Permite al administrador, Gerente y Dependiente, dependiendo de su nivel, acceder a diferentes funciones del sistema.
Entrada:	Usuario y Contraseña.
Fuentes:	Teclado y Mouse.
Salida:	Pantalla con botones.
Proceso:	El sistema muestra en pantalla dos campos a llenar, una vez llenados, lo datos se cotejan en una base datos para obtener el nivel de permiso que tiene asignado, y de acuerdo a esto, se selecciona que funciones tiene permitido utilizar, estas funciones serán las cuales se mostraran en pantalla.
Restricciones:	
Precondición:	
Poscondición	

Reporte de productos sin stock (0 en existencia)	
Función:	Mostrar aquellos productos que tenga 0 en existencia.
Descripción:	Permite al administrador y Gerente, ver aquellos productos que tienen en existencia 0.
Entrada:	Credencial de administrador o Gerente.
Fuentes:	Teclado y Mouse.
Salida:	Se muestra en pantalla, aquellos productos que en el almacén existan cero.
Proceso:	El sistema de acuerdo al valor *cantidad* de cada producto, mostrara en pantalla a todos aquellos productos que su valor sea 0.
Restricciones:	
Precondición:	Credencial de administrador o Gerente.
Poscondición	

Reporte de productos escasos	
Función:	Mostrar aquellos productos, que están apunto de agotarse.
Descripción:	Permite al administrador y Gerente, ver aquellos productos que son escasos (Dependiendo del Indicador del producto).
Entrada:	Credencial de administrador o Gerente o Dependiente.
Fuentes:	Teclado y Mouse.
Salida:	Se muestra en pantalla, aquellos productos que en el almacén estén en un rango de su indicador.
Proceso:	El sistema de acuerdo al indicador del producto, validara si el valor de *cantidad* del producto se encuentra por debajo del valor mínimo de su indicador, para posteriormente decidir si mostrarlo o no en pantalla.
Restricciones:	
Precondición:	Credencial de administrador o Gerente
Poscondición	

Reporte de ventas Total del día	
Función:	Mostrar un historial de los productos vendidos.
Descripción:	Permite al administrador y Gerente, ver aquellos productos que fueron vendidos.
Entrada:	Credencial de administrador o Gerente.
Fuentes:	Teclado y Mouse.
Salida:	Reporte de la ventas, realizadas de acuerdo al día de la consulta del reporte.
Proceso:	
Restricciones:	
Precondición:	Haber entrado al sistema con credencial de Administrador o Gerente
Poscondición	

Reporte de ventas por rango de fechas a mostrar.	
Función:	Mostrar un historial de los productos vendidos, en un rango de fecha determinado por el usuario.
Descripción:	Permite obtener un reporte de los productos que se hallan vendido en un rango de fechas.
Entrada:	Rango de fechas.
Fuentes:	Teclado y Mouse.
Salida:	Se muestra en pantalla, aquellos productos que han sido vendidos.
Proceso:	El sistema muestra en pantalla dos campos a rellenar, para el rango de fecha. Después de ingresar el rango de fecha, el sistema mostrara un historial de productos vendidos en el rango de fechas ingresado.
Restricciones:	
Precondición:	Haber entrado al sistema con credencial de administrador o Gerente.

Alta de productos	
Función:	Ingresar nuevos productos a la base de datos.
Descripción:	Permite al administrador, agregar nuevos productos a la base de datos.
Entrada:	Credencial de administrador. Datos: ID Indicador, Nombre, Marca, Precio, Cantidad.
Fuentes:	Teclado y Mouse.
Salida:	Confirmación en pantalla, mostrando los datos ingresados. Mensaje en pantalla : Se añadió con éxito el Producto.
Proceso:	El sistema desplegara un formulario para dar de alta un nuevo producto, en el cual debe ingresar los Datos (descritos en la Entrada). Se validan que no falten datos y se agrega un nuevo producto a la base de datos.
Restricciones:	Si falta algún dato, el sistema notifica la ausencia de este.
Precondición:	Credencial de administrador.
Poscondición	El producto queda dado de alta y puede mostrarse en los reportes y venderse.

3.3 **Diseño de la Base de Datos**

El diseño de la Base de Datos, es una de las partes más importantes en el desarrollo de la aplicación, puesto que un mal diseño puede llevar a que el sistema contenga incongruencias y/o ambigüedades en el manejo de la información y por tanto el desempeño de la aplicación se vea mermado.

Analizado el problema, para que el sistema pueda cubrir los requerimientos funcionales, es necesario tener una base de datos, la cuál contendrá una serie de tablas, donde se almacenará la información con la cual la aplicación interactuará.

3.3.1 **Diagrama Entidad-Relación del Sistema**

De acuerdo al análisis del sistema, se identificó las entidades, sus atributos y la forma en como se relacionan entre ellas, en la Figura *2.2* se muestra a grandes rasgos un bosquejo de como se implementara la base de datos.

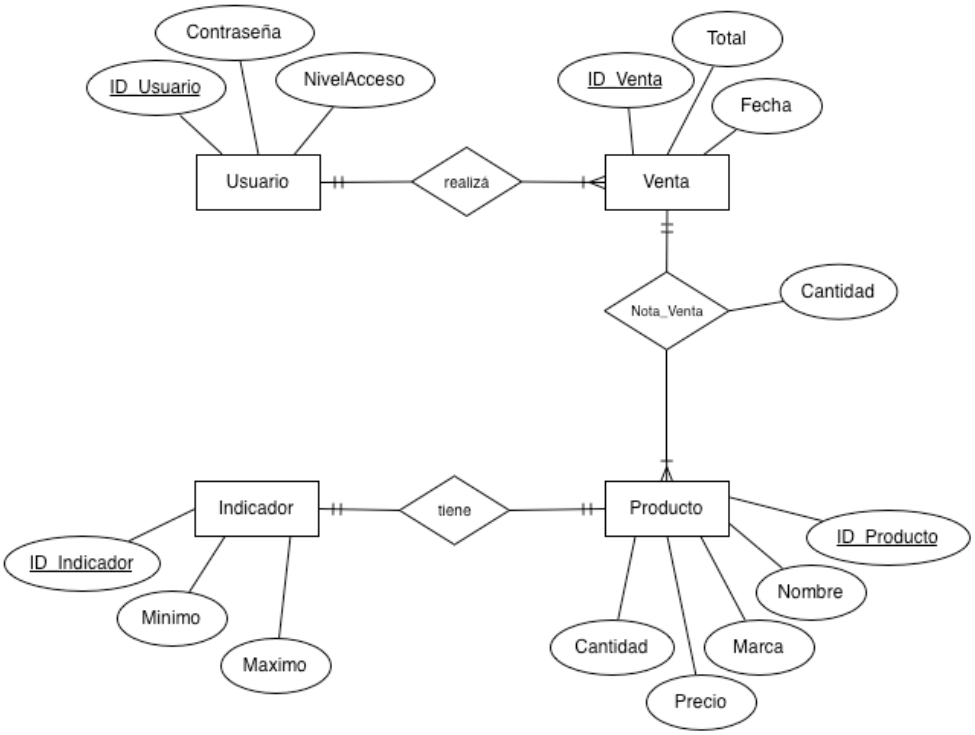

Figura 2.2 *Diagrama Entidad - Relación del sistema*

3.3.2 **Diagrama Modelo Relacional**

Haciendo uso de la aplicación MySQL Workbench 5.2.38, y con como base el Modelo Entidad Relación, se hizo el Modelo relacional.

En la figura 2.3 se muestra el modelo, en ellas se definen el tipo de dato para cada campo, las llaves primarias y las llaves foráneas a partir de la relación.

A partir del diagrama entidad relación, de la relación entre las entidades Venta yProducto, se genero una nueva tabla que se nombro NotaVenta , puesto que era una relación Muchos a Muchos.

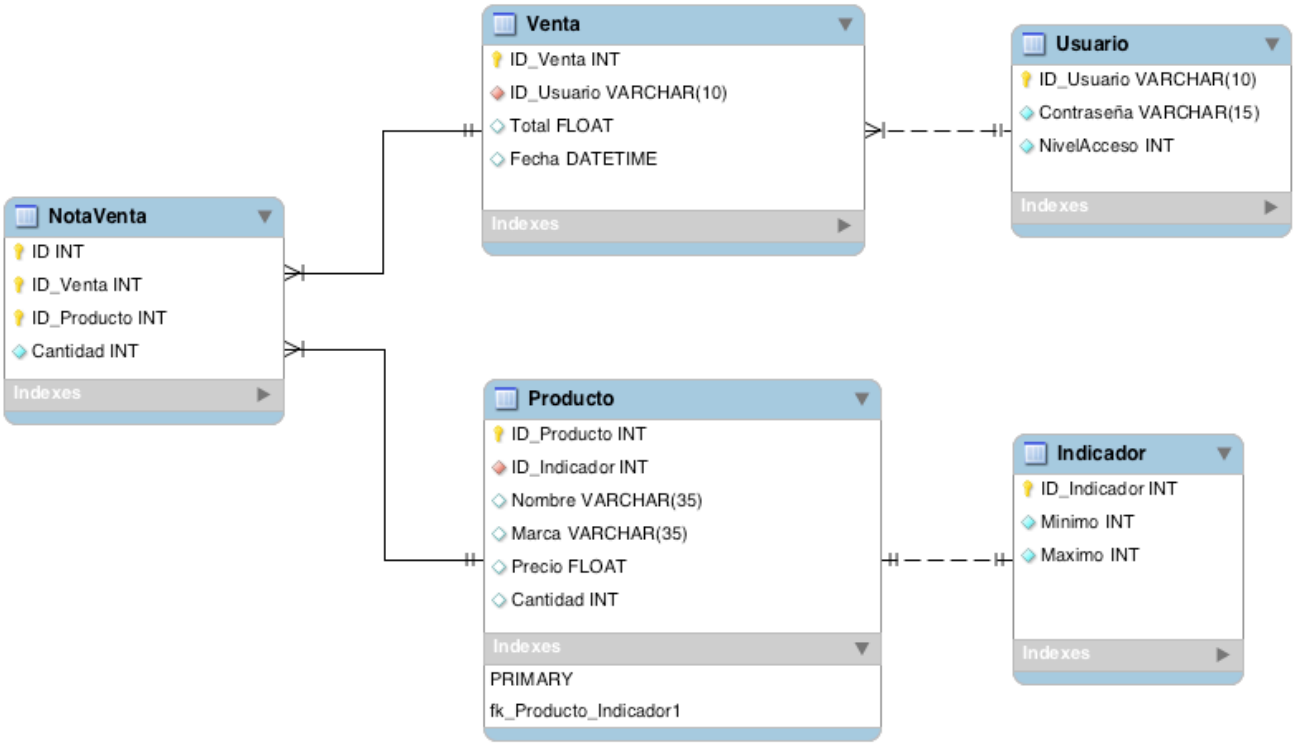

Figura 2.3. *Diagrama Modelo Relacional del sistema.*

Haciendo uso de la misma Aplicación, se genero las sentencia en MySQL, que posteriormente se usaron para crear la Base de Datos en un servidor de MySQL.

3.3.3 **Normalización**

Las tablas creadas a partir de los modelos Entidad Relación y Relacional, cada una de ellas cumplen la tercera Forma Normal (3FN) de normalización.

A continuación se muestra la forma que toman las tablas al cumplir la tercera Forma Normal.

Usuario

ID_Usuario	Contraseña	NivelAcceso

Venta

ID_Venta	ID_Usuario	Total	Fecha

NotaVenta

ID	ID_Venta	ID_Producto	Cantidad

Producto

ID_Producto	ID_Indicador	Nombre	Marca	Precio	Cantidad

Indicador

ID_Indicador	Minimo	Maximo

Hecha la normalización se nota que el diseño podrá satisfacer los requerimientos de la aplicación y tomando en consideración esto se creo la Base de Datos usando las sentencias SQL hechas con MySQL Workbench 5.2.38.

Capitulo 4: Desarrollo del Sistema

La aplicación web se desarrollo en lenguaje Python y haciendo uso del framework Django, para la codificación de la lógica del sistema.

La interfaz de usuario fue creada en HTML y haciendo uso de CSS.

4.1 **Herramientas de Desarrollo**

Para el desarrollo del sistema se hizo uso de distintas aplicaciones que a continuación se explican:

- Para la creación y modificación de código de la lógica del sistema se utilizó la aplicación “Sublime Text 2 Beta, build: 2181”, esto puesto que aplicación maneja una serie de opciones para manejar de manera más fácil el lenguaje Python.

- La interfaz del usuario se desarrollo haciendo uso de aplicación “Espresso Versión 2.0.3”.

- Para el manejo de la base de datos se empleo “MAMP PRO Versión 2.0.5”.

- Para visualizar la interfaz y probar el funcionamiento y apariencia de la aplicación web, se hizo uso de los navegadores web “Safari Versión 5.1.7” y “Google Chrome Versión 19.0.1084.53”.

4.2 **Entorno de trabajo**

La aplicación web fue desarrollada en Mac OS X 10.7.4, por ello lo que a continuación se explica es para este Sistema Operativo.

Como base para el desarrollo y funcionamiento de la aplicación, es prioritario el que se debe tener instalado lo siguiente:

- Python.

La instalación para poder hacer uso de Python es bastante fácil, basta con descargar el instalador desde *http://www.python.org/download/* , tomando como única consideración el sistema operativo en el que se utilizara.

Para el desarrollo de la aplicación se hizo uso de Python versión 2.7.3.

- Django

El Framework Django se puede descargar desde *https://www.djangoproject.com/download/*, se utilizo la versión 1.4; La instalación es simple, basta con desde la terminal ejecutar el siguiente comando:

sudo python setup.py install

- MySQL

Para hacer uso del sistema gestor de Base de Datos de MySQL, se utilizó la aplicación MAMP que se puede obtener desde *http://www.mamp.info/en/index.html*

La instalación de esta aplicación es sumamente simple basta con arrastrar la aplicación a la carpeta *"Aplicaciones"*.

Esta aplicación tiene de dos versiones una de paga (la cual puede ser proba por 14 días) y una gratuita.

- Conector Django MySQL

Para que Django pueda acceder a una Base de Datos en MySQL es necesario instalar un conector, el cual puede ser descargado desde *http://www.mysql.com/products/connector/*

Para su instalación basta con ejecutar en la terminal estos comandos:

python setup.py build

sudo python setup.py install

4.3 **Creación del Proyecto con Django**

Para comenzar el desarrollo de una aplicación con Django y Python, se realizan dos simples pasos:

I. Crear un proyecto.

Como se muestra en la figura 3.1, para crear el proyecto en Django basta con desde la terminal escribir el comando:

django-admin.py startproject "nombre_del_proyecto"

Figura 3.1 *Creación de proyecto desde la terminal*

II. Crear una aplicación.

Una vez realizado el proyecto se crea la aplicación, en la figura 3.2 se muestra que para ello es necesario acceder a la carpeta del proyecto y posteriormente crear la aplicación con el comando:

python manage.py startapp "nombre_de_la_aplicación"

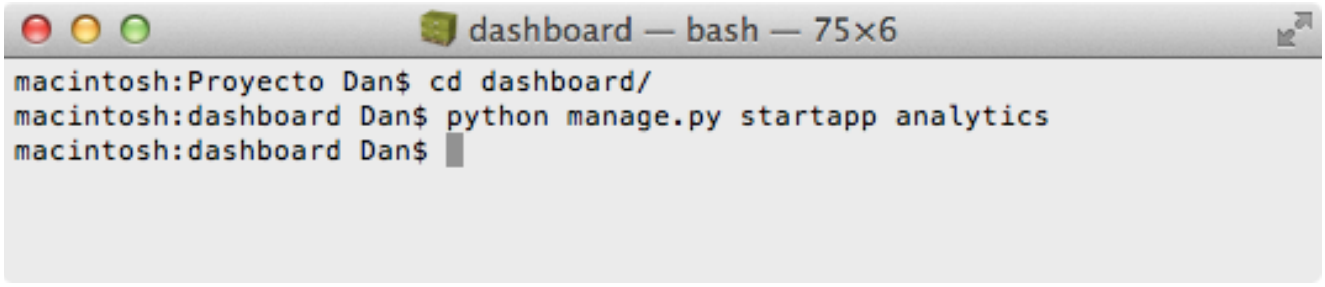

Figura 3.2 *Creación de la aplicación desde la terminal*

Una vez creado el proyecto y la aplicación, Django como base crea dos carpetas, una tiene el nombre del proyecto y otra el nombre de la aplicación, en ellas se crea distintos archivos con extensión ***py***, que son archivos con código en lenguaje Python, los cuales son modificados para satisfacer nuestros requerimientos.

En la figura 3.3 se muestra la estructura de las carpetas y los archivos creados.

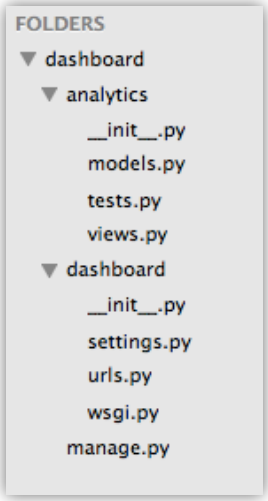

Figura 3.3 *Estructura de los archivos creados por Django.*

Realizado lo anterior, solo se tiene que iniciar el servidor (Figura 3.4) del proyecto y se esta listo para empezar el desarrollo.

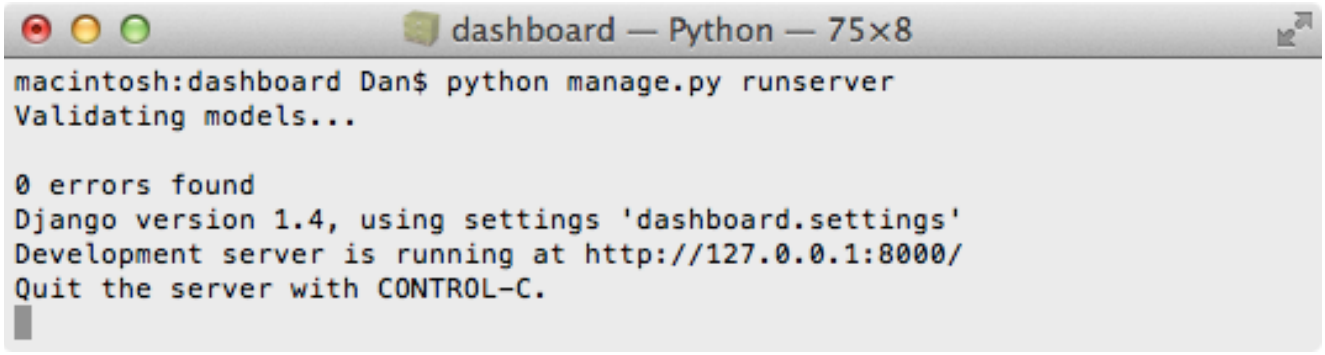

Figura 3.4 *Inicio del Servidor del Proyecto*

4.3.1 **Modelo Template Vista en Django**

Django tiene como principal característica usar su Modelo-Template-vista, para manejar toda la aplicación haciendo uso de tres bloques.

4.3.1.1 **Modelo del Sistema**

Un modelo en Django, es la manera de como está estructurada una tabla dentro de una base de datos.

La definición de modelos se hace dentro del archivo "***models.py***", en la figura 3.4, se muestra la manera en como están definidos los modelos para la aplicación.

models.py — dashboardProject

models.py

```
1  from django.db import models
2  from django.contrib.auth.models import User
3
4  class Producto(models.Model):
5      ID_Indicador = models.ForeignKey('Indicador')
6      Nombre = models.CharField(max_length=50)
7      Marca = models.CharField(max_length=50)
8      Precio = models.FloatField()
9      Cantidad = models.IntegerField()
10
11 class Venta(models.Model):
12     ID_Usuario = models.ForeignKey(User)
13     Total = models.FloatField()
14     Fecha = models.DateField()
15
16 class NotaVenta(models.Model):
17     ID_Venta = models.ForeignKey('Venta')
18     ID_Producto = models.ForeignKey('Producto')
19     Cantidad = models.IntegerField()
20
21 class Indicador(models.Model):
22     Minimo = models.IntegerField()
23     Maximo = models.IntegerField()
```

Figura 3.4 *Definición de modelos de la aplicación.*

4.3.1.2 **Vista del Sistema**

Una vista (view), es una función en Python que hace una solicitud Web y devuelve una respuesta, esta respuesta puede ser el contenido de una pagina, un error 404, una imagen, entre otras más.

La vista contiene la Lógica necesaria para devolver una **respuesta**; Todas las respuestas se encuentras en el archivo "***views.py***".

La figura 3.5 muestra solo una de las vistas creada para la aplicación; La vista realiza una conexión a la Base de Datos para recuperar una serie de datos, los cuales se procesan para después enviar estos datos a un *Template*, el cual se encarga de desplegar la información en una pagina web, con un formato visual entendible.

views.py — dashboardProject

views.py

```
# -*- encoding: utf-8 -*-
from django.contrib.auth import *
#####################################################
# ⬇⬇⬇Función que ocupa la vista "circles"⬇⬇⬇
def get_data_circle():
    cursor = connection.cursor()

    cursor.execute('SELECT ID_Indicador_id, Nombre, Cantidad FROM analytics_Producto')
    rowsProduct = cursor.fetchall()

    cursor.execute('SELECT id, Minimo, Maximo FROM analytics_indicador')
    rowsIndicador = cursor.fetchall()

    cursor.close()


    values =[]
    for i in range(0,len (rowsProduct)):
        for j in range(0,len(rowsIndicador)):
            if rowsProduct[i][0] == rowsIndicador[j][0]:
                if  rowsProduct[i][2] >= rowsIndicador[j][1]  and rowsProduct[i][2] <= rowsIndicador[j][2] :
                    values.append ([rowsProduct[i][1],1,rowsProduct[i][2]])
                elif rowsProduct[i][2] < rowsIndicador[j][1]:
                    values.append ([rowsProduct[i][1],0,rowsProduct[i][2]])
                elif rowsProduct[i][2] > rowsIndicador[j][2]:
                    values.append ([rowsProduct[i][1],2,rowsProduct[i][2]])

    values.sort(key=lambda b: b[1])
    return values
# ⬇⬇⬇ Vista ⬇⬇⬇
def circles(request):
    return render_to_response('circles/circles.html',{'datos': get_data_circle(), 'name':'circles'})
```

Figura 3.5 *Vista de la aplicación*

4.3.1.3 **Template del Sistema**

Plantilla o Template, es donde se define la apariencia de la interfaz y la manera en que se mostrará la información y además con la que el usuario interactuará.

Es un archivo HTML, donde contiene etiquetas HTML y etiquetas propias del Framework Django; En la figura 3.6 se muestra la plantilla (*Template)* para la administración de la aplicación.

base.html — dashboardProject

base.html

```
44      {% block breadcrumbs %}
45      <div class="breadcrumbs">
46      <a href="{% url 'admin:index' %}">{% trans 'Home' %}</a>
47      {% if title %} &rsaquo; {{ title }}{% endif %}
48      </div>
49      {% endblock %}
50      {% endif %}
51
52      {% block messages %}
53          {% if messages %}
54          <ul class="messagelist">{% for message in messages %}
55            <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>
56          {% endfor %}</ul>
57          {% endif %}
58      {% endblock messages %}
59
60      <!-- Content -->
61      <div id="content" class="{% block coltype %}colM{% endblock %}">
62          {% block pretitle %}{% endblock %}
63          {% block content_title %}{% if title %}<h1>{{ title }}</h1>{% endif %}{% endblock %}
64          {% block content %}
65          {% block object-tools %}{% endblock %}
66          {{ content }}
67          {% endblock %}
68          {% block sidebar %}{% endblock %}
69          <br class="clear" />
70      </div>
71      <!-- END Content -->
72
73      {% block footer %}<div id="footer"></div>{% endblock %}
74  </div>
75  <!-- END Container -->
76
77  </body>
78  </html>
```

Figura 3.6 *Plantilla de Administración.*

La imagen solo muestra una parte del template, para que se fácil de visualizar que es lo que hace el template.

4.4 **Interfaz del Sistema**

La interfaz es una de las partes importantes de la Aplicación Web, ya que es con lo que interactúa el usuario, esta tiene un funcionamiento simple e intuitivo, que facilita el uso de la aplicación.

En el desarrollo de las *plantillas* se hizo uso de HTML, CSS y etiquetas de Django, estas etiquetas son utilizadas puesto que el contenido final, es generado dinámicamente, dependiendo de los usuarios y las respuestas que envía las vistas.

Dentro de la Aplicación Web, existen tres roles principales, los cuales cada uno de ellos tiene diferentes privilegios, esto como consideración de los requerimientos Funcionales de la Aplicación.

De acuerdo al rol de usuario, se tiene diferentes opciones a la cuales pueden acceder para hacer uso.

Dentro del *template* que genera y muestra de manera dinámica la pagina web con las opciones acorde a cada rol, se hace uso de etiquetas de Django y de HTML para el despliegue de esta información.

En la Figura 3.7 se muestra parte del código necesario, que se ocupa para mostrar las distintas opciones a las que puede tener el usuario dependiendo de sus privilegios de acuerdo al rol del usuario, la *vista* envía una serie de datos al *template* en el que se especifican las opciones disponibles que tiene el usuario para utilizar.

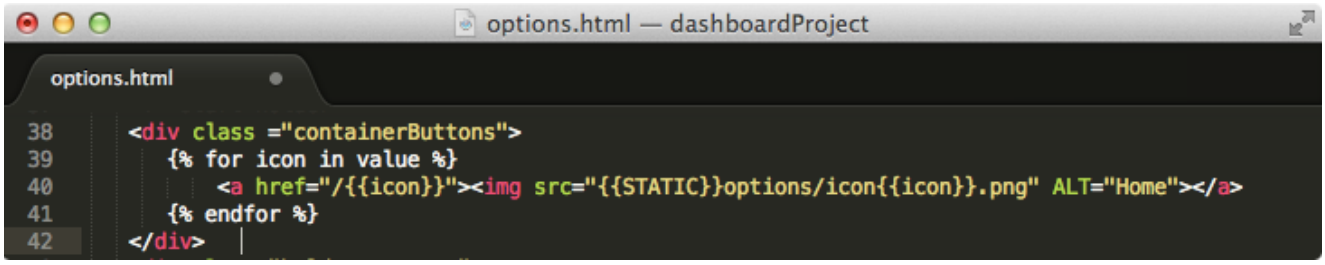

Figura 3.7 *Código para el despliegue de opciones, de los usuarios de acuerdo a su rol.*

Lo privilegios son manejados por el ***Gestor de Usuarios*** propio del Framework Django y una pequeña adaptación para que funcione de acuerdo a lo que se requiere mostrar.

En la Figura 3.8 se muestra el código que se utiliza para tomar la decisión de los datos a enviar como respuesta al *template*, para que este genere y despliegue las distintas opciones al usuario.

views.py — dashboardProject

options.html

```
303  def options(request):
304      if request.user.is_anonymous():
305          print request.user.is_anonymous()
306          return HttpResponseRedirect('/login')
307      else:
308          current_user = request.user
309          group = request.user.groups.values_list('name',flat=True)
310          print current_user
311          print group[0]
312          if (group[0]== 'Administrador'):
313              opt= get_stack(0)
314          if (group[0]== 'Gerente'):
315              opt= get_stack(1)
316          if (group[0]== 'Dependiente'):
317              opt= get_stack(2)
318          print opt
319          return render_to_response('options/options.html',{'value' : opt})
320
```

Figura 3.8 *Código para la toma de decisión de la respuesta a enviar al usuario.*

En la Figura 3.9 se muestra lo que es visualizado por el usuario, dependiendo de sus privilegios.

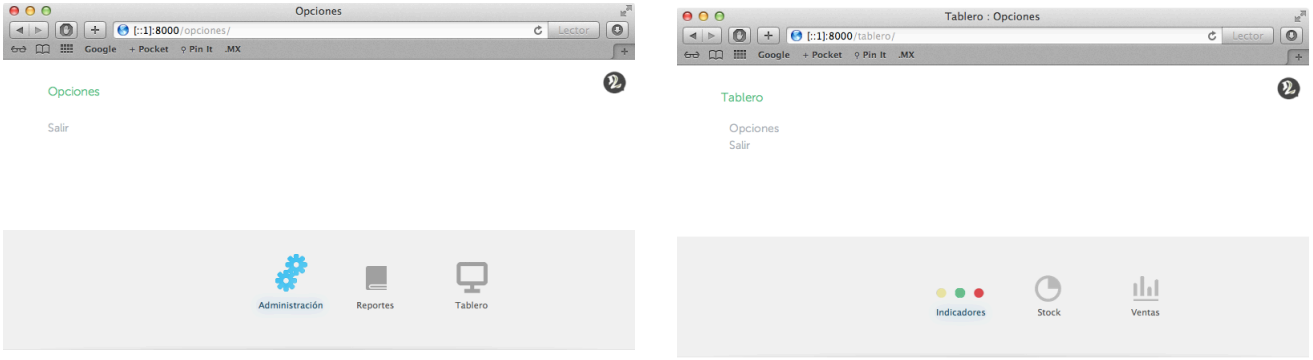

Figura 3.9 *Interfaz para Usuarios con mayores privilegios*

4.4.1 **Interfaz de los Usuarios**

Para que un usuario pueda hacer uso de las funciones de la aplicación, es necesario que este pase un proceso de autenticación.

Como se muestra en la Figura 3.10, para el proceso de autenticación la aplicación tiene una interfaz de *Login* donde es necesario introducir el *"nombre de usuario"* y *"contraseña"* por parte del usuario, para después de esto validar los datos y desplegar las opciones disponibles para el usuario.

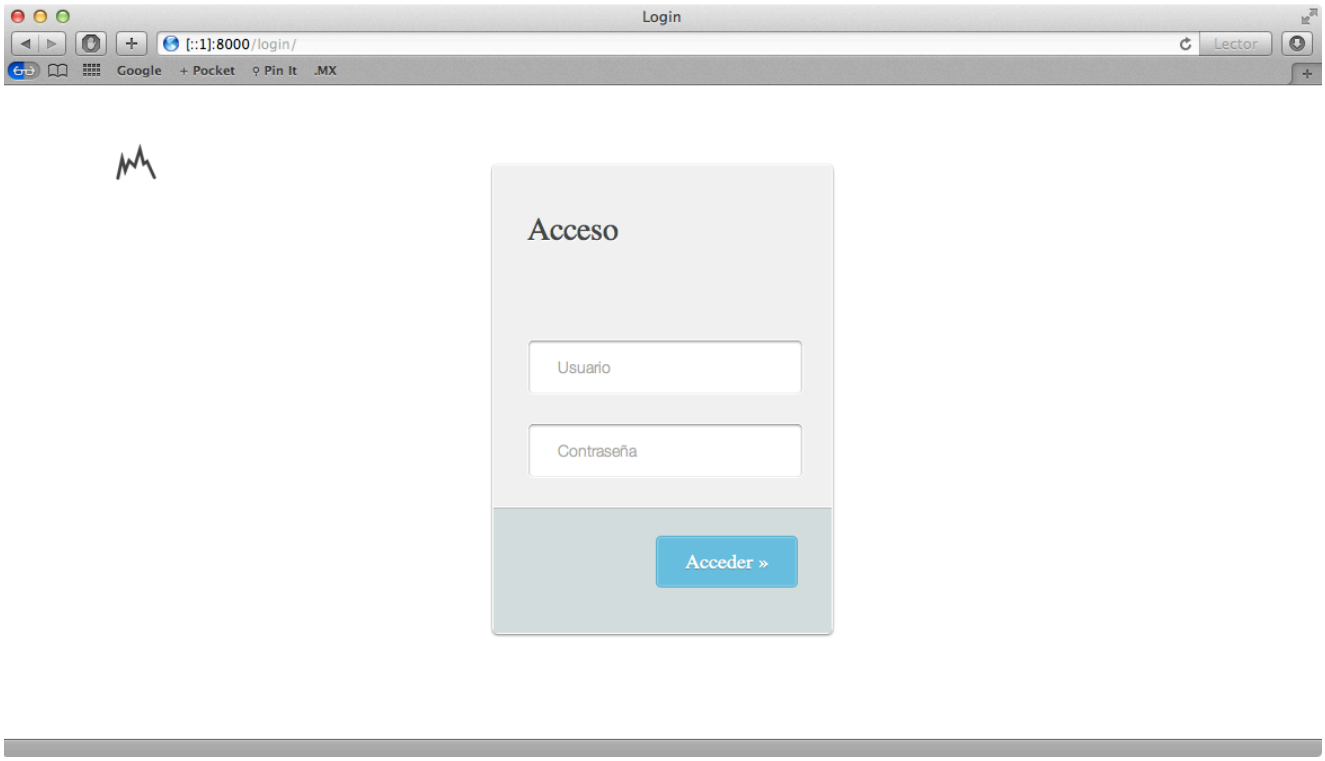

Figura 3.10 *Interfaz de Login para proceso de Autenticación.*

Una vez hecho el proceso de autenticación, de acuerdo a los privilegios del usuario, como se muestra en la Figura 3.9 a este se le despliega las distintas opciones que tiene a su disposición para utilizar.

Todo usuario que esta registrado dentro de la Aplicación, tiene una interfaz donde puede realizar las tareas básicas de Crear, Recuperar, Actualizar y Borrar los datos almacenados dentro de la Base de Datos.

En la Figura 3.11 y Figura 3.12 se muestra la interfaz a la que tienen acceso los usuarios de la Aplicación, con la particularidad que de acuerdo al rol de usuario, este tenga algunas tareas básicas denegadas, de acuerdo a sus privilegios.

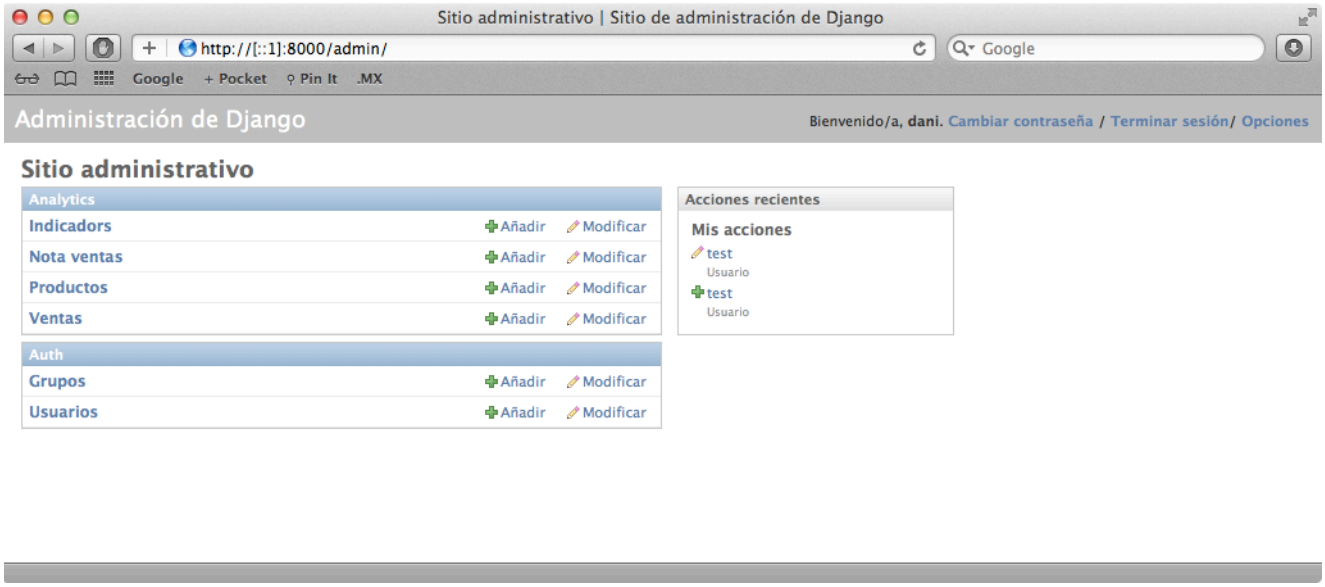

Figura 3.11 *Interfaz para Crear, Recuperar, Actualizar y Eliminar Datos de la Base de Datos para usuario con mayores Privilegios*

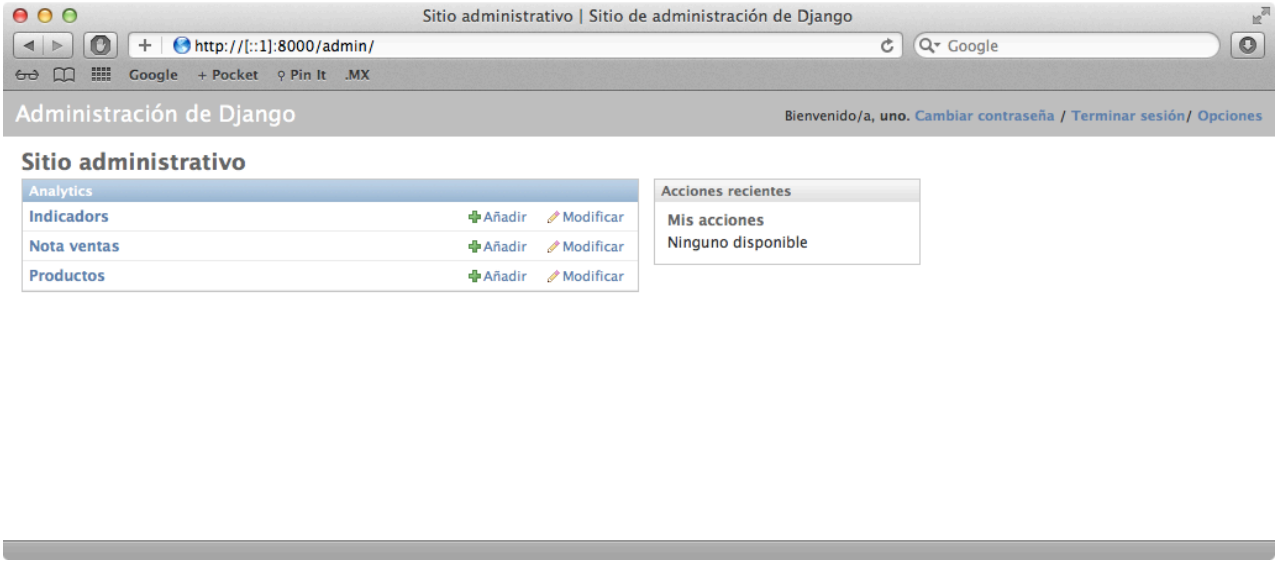

Figura 3.12 *Interfaz para Crear, Recuperar, Actualizar y Eliminar Datos de la Base de Datos para usuario con menores Privilegios*

Una de las interfaces principales del sistema a la que tiene acceso el usuario, es la que se muestra en la Figura 3.13, donde son representado los indicadores individuales de los productos en stock.

Dependiendo de los valores del indicador individual del producto y la cantidad existente del producto (datos que están almacenados en la Base de Datos), es representado por un círculo de color con tres opciones, Verde, Rojo o Amarillo esto depende del estado en stock en que se encuentra el producto; El Rojo representa cantidad escasa del producto, Amarillo cantidad aceptable y Verde cantidad optima del producto.

Figura 3.13 *Representación de indicadores productos almacenados en stock.*

En la Figura 3.14 se muestra la Interfaz con la que interactúa el usuario, donde se muestra una gráfica circular que representa de forma general el estado del stock vigente.

Haciendo uso del estado individual de cada producto (Verde, Amarillo, Rojo), se calcula el porcentaje, para lograra representar el total por estado de los productos, buscando poder tener una visión más clara del estado en general en el que se encuentra el Stock (esto se puede comparar con la Figura 3.13)

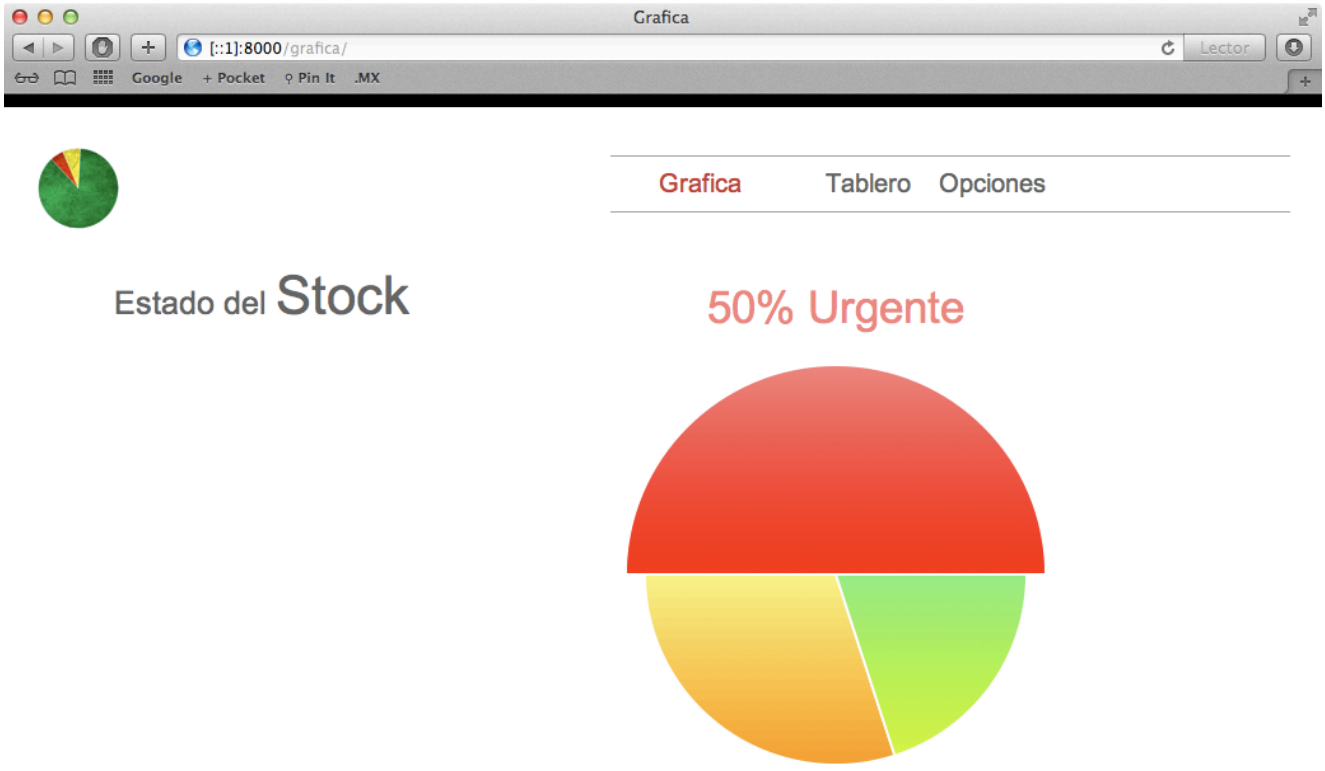

Figura 3.14 *Representación del estado global del Stock.*

Cabe destacar el hecho de que el despliegue de los círculos se genera dinámicamente, a partir de los datos enviados como respuesta por la *vista*.

El manejo de errores es controlado por el framework Django, como se muestra en la Figura 3.15 el error o errores es desplegado solo en la consola del servidor nunca el usuario visualiza estos errores, salvo el Error 404.

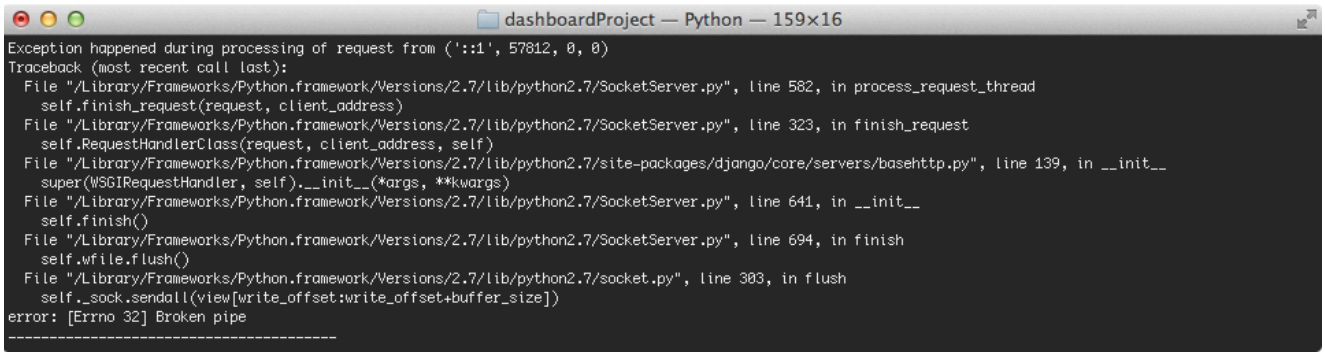

Figura 3.13 *Error controlado por el framework Django, mostrado en la consola del servidor.*

4.5 **Implantación de la aplicación**

La aplicación se desarrollo en una computadora con las siguientes características:

Sistema operativo: Mac OS X Lion 10.7.4
Procesador: 2.4 GHz Intel Core 2 Duo
Memoria: 5 GB 1067 MHz DDR3

Localmente la aplicación funciono correctamente respecto a los requerimientos funcionales.

La aplicación se instaló en una computadora con Sistema Operativo Windows 7 (las características se mencionan a continuación) respondiendo correctamente, con pequeñas diferencias en cuanto a la forma de que como se visualizan los datos, pero cumpliendo totalmente los requerimientos funcionales.

Sistema operativo: Windows 7 Ultimate
Procesador: 2.4 GHz Intel Core 2 Duo
Memoria: 1.5 GB

Capitulo 5: Conclusiones

- El uso de la aplicación desarrollada, puede ayudar a los pequeños contribuyentes en sus tareas diarias, con un funcionamiento simple pero muy eficiente, que ayude a mejorar la eficiencia del manejo de inventario del negocio, permitiendo tomar mejores decisiones al momento de especificar precios de los productos y el abastecer de los mismos, para así poder hacer más rentable su negocio.

- El análisis y diseño del sistema, es parte fundamental durante el ciclo de vida del software, ayuda a asegurar el éxito del proyecto, además de poder acelerar el trabajo y aumentar la calidad de los resultados. Es por ello, que el no realizar un análisis o realizarlo de manera insuficiente, provocará que en el equipo de trabajo no siempre tenga el mejor razonamiento para resolver los problemas; Por el contrario realizar un buen análisis y diseño, da ventajas al equipo de trabajo para que tengan la posibilidad de tomar mejores decisiones al momento de codificar los distintos componentes del sistema y así cumplir con los requerimientos del sistema en tiempo y forma

- La ausencia de una organización bien establecida durante el desarrollo de un sistema puede llevar al equipo de desarrollo a perder el rumbo de como se quiere alcanzar la meta del proyecto, es por ello que se debe hacer uso de alguna de las metodologías existentes; Este es uno de los puntos más importante que se debe tomar en cuenta para el desarrollo de sistemas, ya que ayuda a organizar las ideas y sobre todo guiar a todos los involucrados en el desarrollo del sistema, para alcanzar el cometido del sistema. De acuerdo a la complejidad del sistema y de entre las varías metodologías existentes, se debe tomar la decisión de que metodología es la optima a utilizar, para ayudar en la planificación del ciclo de vida para el sistema, ayudara a establecer o vislumbrar los pasos necesarios así como el tiempo necesario para realizar cada uno de estos, logrando así poder aproximar el tiempo necesario para el desarrollo ya que el tiempo es un factor inherente, y el cual siempre es de suma importancia para el usuario final.

- El usar el framework Django para el desarrollo de el Sistema, trae consigo posibilidad de distribuir el desarrollo de tareas especificas (conexión a la base de datos, lógica de la aplicación, presentación de la información, entre otras) para el funcionamiento del sistema, haciendo más ordenado el trabajo para el equipo de desarrollo de la aplicación, además de ya tener desarrolladas distintas tareas básicas típicas que en la mayoría de los sistema es implementado, logrando hacer que el desarrollo sea más simple, rápido y eficiente.

5.1 **Aportaciones**

La documentación del desarrollo del sistema sirva de base para entender a grandes ragos el funcionamiento y utilización del framework Django, y las ventas que se obtienen al hacer uso del framework.

5.2 **Trabajo a futuro**

El Sistema de administración de existencias de un negocio es una aplicación base que puede ser utilizada por cualquier clase de negocio que se dedique a la compra y venta de bienes o servicios de distintos rubros.

El sistema como tal puede puede ser ampliado, diseñando e implementando distintos componentes que puede ayudar en la automatización de los distintos procesos básicos (lector de código de barras, compras online, etc) del negocio de un pequeño contribuyente.

La aplicación almacena el historial de la información manejada dentro del negocio, gracias a esto podría ser posible implementar y aplicar Inteligencia de Negocios, para que los propietarios puedan ser ayudados por la aplicación a tomar mejores decisiones, disminuir el margen de error y obtener mejores ingresos.

Bibliografía

[1] Roger S. Pressman, "**INGENIERÍA DEL SOFTWARE. Un enfoque práctico.** 5º Ed.", McGRAW-HIL, 2002.

[2] Philippe Kruchten, "**The Rational Unified Process An Introduction**, Second Edition", Addison Wesley, 2000, ISBN: 0-201-70710-1.

[3] Patrick Grässle, Henriette Baumann, Philippe Baumann, "**UML 2.0 in Action A Project-Based Tutorial**", Packt Publishing, 2005, ISBN: 1-904811-55-8.

[4] Fernando Alonso Amo, Loïc Martinez, Fco. Javier Segovia, "**INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE. Modelo de desarrollo de programas.**", DELTA, 2005, ISBN: 84-96477-00-2.

[5] Ma. Victoria Nevado Cabello, "I**NTRODUCCION A LAS BASES DE DATOS RELACIONALES**", Vision Libros, 2010, ISBN: 978-84-9886-809-8.

[6] Fray León Osorio Rivera, "**Base de datos relacionales Teoría y práctica**", Fondo Editorial ITM, 2008, ISBN: 978-958-8351-42-1.

[7] Ma. Victoria Nevado Cabello, "**Introducción a Las Bases de Datos Relacionales**" , Visio Libros, ISBN: 978-84-9886-809-8

[8] Subhash Chandra Yadav, Sajay Kumar Singh, "**An Introduction to CLIENT/SERVER COMPUTING**", New Age International Publishers, 2009, ISBN(13): 978-81-224-2861-2.

[9] Servidor HTTP Apache - Wikipedia, la enciclopedia libre, [en línea]. [octubre 2012] http://es.wikipedia.org/wiki/Servidor_HTTP_Apache

[10] Frequently Asked Questions, [en línea]. [octubre 2012] http://www.apache.org/foundation/faq.html

[11] Raúl Gonzales Duque, “**Python para todos**”, Creative Commons Reconocimiento 2.5 España.

[12] Django Software Foundation, Django web framework | Django en Español, [en línea]. [junio 2012] http://django.es/

[13] Adrian Holovaty and Jacob Kaplan-Moss. "**The Definitive Guide to Django Web Development Done Right**", Apress, 2008.

[14] Brad Dayley, "**Sams teach yourself Django in 24 hours**", Sams Publishing, 2008, ISBN-13: 978-0-672-32959-3.

[15] Scott Newman, "**Django 1.0 Template Development**", Packt Publishing, 2008, ISBN 978-1-847195-708.

[16] Ayman Hourieh, "**Learning Website Development with Django**", Packt Publishing, 2008, ISBN 978-1-847193-35-3.