ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DEL SISTEMA DE CONTROL DE INVENTARIOS PARA LAS INSTITUCIONES PÚBLICAS DEL ECUADOR

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

JAIME DARÍO JIMÉNEZ TOSCANO
dr.jmnz@hotmail.com

DIRECTOR: ING. MARCOS RAÚL CÓRDOVA BAYAS
raul.cordova@epn.edu.ec

Quito, diciembre 2013

DECLARACIÓN

Yo, Jaime Darío Jiménez Toscano, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Jaime Darío Jiménez Toscano

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Jaime Darío Jiménez Toscano, bajo mi supervisión.

Ing. Raúl Córdova

DIRECTOR DE PROYECTO

AGRADECIMIENTO

A Dios, por su infinito amor y misericordia, por darme siempre la fortaleza para seguir adelante.

A mis padres, Luis y Fanny; y a mis hermanos: Pablo, Yessenia, Sandra, "Muñeca" (Gloria) y Paulina, porque siempre han estado incondicionalmente junto a mí en las buenas y en las malas.

A todas aquellas personas que me han respaldado en oración, en especial al Hno. Milton, Hna. Martita y Hna. Hilda.

A los amigos y compañeros que he tenido la oportunidad de conocer durante mi trayecto en la "poli", también son parte de esto.

A todos los profesores que de alguna forma han sabido transmitir sus conocimientos y experiencia, incluso más allá de lo académico.

Al "Viejo Lucho" (Rafa), por acolitar en la culminación de este proyecto.

A mi Director de Proyecto, el Ing. Raúl Córdova, por el tiempo y el apoyo brindado.

DEDICATORIA

A Dios, porque sin Él nada de esto tendría sentido.

A mi hermano Pablo Jiménez, por ser el instrumento que Dios utilizó para llevarme a sus caminos y por haber sido siempre un ejemplo de lucha.

A mi sobrina Raquel, por su muestra de cariño y afecto incondicional que siempre lo llevo presente.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En la actualidad, la mayoría de instituciones públicas realizan una gestión manual de los inventarios. El principal problema que esto representa es la demora que conlleva el cumplimiento de todo el proceso de solicitud y entrega de los bienes necesarios para el normal funcionamiento de una entidad, ya que se requiere la aprobación de diferentes personas que intervienen en el proceso. Por ello, es necesario un sistema que controle y optimice todo el proceso de gestión de inventarios, mejorando las interacciones que se dan entre los diferentes actores. Además, en el Ecuador está en vigencia un reglamento que regula el manejo de los bienes del sector público, el cual debe ser implementado para cumplir con las leyes respectivas. Por tal razón, en este proyecto se han elaborado 4 capítulos para poder describir y desarrollar una solución que pueda ser implementada en cualquier tipo de Institución Pública, dando como resultado el Sistema de Control de Inventarios para las Instituciones Públicas del Ecuador.

En el Capítulo 1 se hace una exposición de la manera en que se gestionan la mayoría de entidades del sector público, haciendo énfasis en los procesos que involucran el manejo de inventarios. Además, se justifica el uso de una metodología específica y las herramientas necesarias para el desarrollo del sistema.

A continuación, en el Capítulo 2, se tiene todo lo relacionado al diseño del sistema y a la implementación de dicho diseño, incluyendo las pruebas.

Luego, en el Capítulo 3, se describe de forma breve la Institución Pública en la que se realizó el caso de estudio y también se despliegan los resultados obtenidos luego de hacer la evaluación del sistema.

Finalmente, en el Capítulo 4, se puntualizan algunas conclusiones y recomendaciones luego de haber tenido la experiencia de trabajar en este proyecto.

PRESENTACIÓN

Este proyecto tiene como propósito automatizar el proceso que se sigue para el control y administración de los inventarios en las bodegas de una institución pública que cuente con varias dependencias que necesiten de los suministros necesarios para su funcionamiento, los que se almacenan en una o más bodegas que dependen de dicha institución. Esta automatización, a más de implementar los aspectos usuales de un sistema de control de inventarios, tomará en cuenta los reglamentos y procedimientos comunes que rigen a instituciones de este tipo. Por lo tanto, entre los objetivos que se pretenden alcanzar en este proyecto están los siguientes:

- Definir los conceptos fundamentales de la administración de inventarios en instituciones públicas.
- Establecer un procedimiento común que se ajuste a la forma habitual en que se controla el inventario en una institución pública del Ecuador.
- Aplicar el sistema en un caso de estudio.

En este proyecto no se considerará la posibilidad de acoplar el sistema desarrollado con otros sistemas, como es el caso de los sistemas de contabilidad o sistemas de facturación.

CAPÍTULO 1. PLANTEAMIENTO DEL PROBLEMA

1.1 ESTRUCTURA DE LAS INSTITUCIONES PÚBLICAS DEL ECUADOR

1.1.1 DEFINICIÓN DE INSTITUCIÓN PÚBLICA

Las instituciones públicas son entidades con personalidad jurídica propia, constituidas con capital estatal en su totalidad o de modo parcial. La misión de este tipo de organizaciones es la de "proveer bienes y servicios que promuevan, preserven, resguarden, orienten y estimulen las iniciativas de la sociedad compatibles con el interés general, y que contribuyan a paliar o compensar los eventuales desequilibrios emergentes de la conjunción de dichas iniciativas." [1]

Las instituciones públicas constituyen una gran parte de la economía del país, y son uno de los principales medios utilizados por el Gobierno para intervenir en ésta. Tienen una acentuada relevancia, no sólo en términos cuantitativos (por su volumen de negocios, su participación en el producto interior bruto – PIB, número de empleados, etc.), sino también al hecho de que se sitúan en sectores productivos clave.

El proceso de la toma de decisiones de las instituciones públicas difiere de aquellas que pertenecen al sector privado, debido a que el poder de iniciativa parte del Estado, que lo ejerce estableciendo sus objetivos y controlando su actividad.

1.1.2 DIFERENCIAS Y SIMILITUDES ENTRE INSTITUCIONES PÚBLICAS Y PRIVADAS

Lo que en la práctica caracteriza o diferencia a una institución o empresa pública de otra privada es su relación con los poderes públicos. A diferencia de la privada, la empresa pública no busca la maximización de los beneficios, las ventas o la cuota de mercado, sino que busca el interés general de la colectividad a la que pertenece, aunque este interés pueda, en ocasiones, ir en contra de los objetivos que rigen el desempeño de la empresa privada.

[1] INSTITUTO NACIONAL DE LA ADMINISTRACIÓN PÚBLICA. Organizaciones Públicas.

Por otra parte, la diferenciación entre instituciones públicas y privadas no es absoluta. Por un lado, existen empresas mixtas, cuyo capital es en parte público y en parte privado. Asimismo, una empresa privada puede convertirse en empresa pública si el Gobierno decide nacionalizarla. De forma análoga, una empresa pública puede pasar al sector privado tras un proceso de privatización.

Las principales diferencias que existen entre organizaciones públicas y privadas son las siguientes:

- Mientras que en las agendas de las organizaciones públicas predominan las cuestiones de interés público que trascienden la perspectiva de los intereses propios, las agendas de las organizaciones privadas incluyen, mayormente, asuntos concernientes a sus propios intereses.

- Las organizaciones públicas forman parte del sistema político; en cambio, para las organizaciones privadas, el sistema político constituye parte de un entorno al que consideran exógeno en la mayor parte de los casos, sin embargo, esto no quiere decir que las decisiones políticas no puedan influir notablemente en el ejercicio de sus actividades.

- Las organizaciones privadas están sometidas a una competencia creciente en los mercados nacionales e internacionales, mientras que las organizaciones públicas mantienen frecuentemente una condición monopólica en algunos mercados de bienes y servicios públicos.

- Las normas de comportamiento formal en las organizaciones de la Administración Pública están reglamentadas por el Derecho Administrativo, mientras que en las organizaciones privadas las normas son internas y exclusivas de cada organización, generadas en su propio ámbito.

- Los clientes de las organizaciones públicas actúan, en diferentes ocasiones, desempeñando el rol de ciudadanos (por ejemplo, ante el Registro Civil), de usuarios de bienes y servicios (por ejemplo, ante Correos del Ecuador), de beneficiarios de bienes y servicios (por ejemplo, respecto de un programa de desarrollo social), o de contribuyentes (por ejemplo, frente al Servicio de Rentas Internas). Los clientes de las

organizaciones privadas son los consumidores de los bienes y servicios que éstas producen.

Algunas de las similitudes que podemos encontrar entre ambos tipos de organizaciones son las siguientes:

- Ambos tipos de organización conforman sistemas socio-técnicos dirigidos a la consecución de ciertos objetivos.
- Se enfrentan, por igual, a entornos cambiantes con los cuales interactúan.
- Deben relacionarse con un conjunto de organizaciones competidoras, concurrentes, proveedoras y clientes, con las cuales las relaciones pueden tornarse conflictivas o cooperativas.
- La circulación de información y los procesos comunicativos son claves tanto en las organizaciones públicas como de las privadas.
- Los sistemas de planificación, decisión, coordinación y control son necesarios y existen en ambos tipos de organizaciones.

1.1.3 CARACTERIZACIÓN DE LA GESTIÓN DE LAS INSTITUCIONES PÚBLICAS

El artículo 227 de la Constitución expresa que "La administración pública constituye un servicio a la colectividad que se rige por los principios de eficacia, eficiencia, calidad, jerarquía, desconcentración, descentralización, coordinación, participación, planificación, transparencia y evaluación", [2] lo que evidencia que la administración pública está orientada a mejorar el rendimiento de las organizaciones antes que a buscar mejorar directamente las condiciones de igualdad social en la población. Es fundamental que las instituciones del Estado sean administradas en base a metodologías desarrolladas para el sector público; de igual manera, es importante conocer la forma cómo se están gestionando actualmente y a partir de este conocimiento poder definir los cambios necesarios para lograr instituciones que aporten al mejoramiento de las condiciones de vida de la sociedad.

[2] ASAMBLEA NACIONAL CONSTITUYENTE. Constitución Política de la República del Ecuador. 2008

En la Tabla 1.1 se resume el proceso general empleado en la gestión de instituciones públicas, los diferentes componentes que intervienen, sus competencias e instituciones responsables.

COMPONENTES	COMPETENCIAS	INSTITUCIONES RESPONSABLES
PLANIFICACIÓN	POLÍTICAS Y COORDINACIÓN DE: • PLANIFICACIÓN • ESTRATEGIA • PLANES PLURIANUALES • PLAN OPERATIVO ANUAL	Secretaría Nacional de Planificación y Desarrollo - SENPLADES
ORGANIZACIÓN	POLÍTICAS Y ASESORÍA TÉCNICA DE: • DESARROLLO INSTITUCIONAL • ADMINISTRACIÓN DE RECURSOS HUMANOS • REMUNERACIONES	Secretaría Nacional Técnica de Desarrollo de Recursos Humanos y Remuneraciones del Sector Público – SENRES
	POLÍTICAS Y APROBACIÓN DE: • PRESUPUESTO	Ministerio de Economía y Finanzas – MEF
DIRECCIÓN Y EJECUCIÓN	CUMPLIMIENTO DE OBJETIVOS CONTEMPLADOS EN LA MISIÓN, CONFORME SU ÁMBITO DE ACCIÓN Y NORMATIVA LEGAL	Las instituciones.
CONTROL	CONTROLAR Y VERIFICAR EL CUMPLIMIENTO DE LA NORMATIVIDAD, ÁMBITO DE COMPETENCIA Y POLÍTICAS	• CONTRALORÍA GENERAL DEL ESTADO • SENRES • SENPLADES • MEF • PROCURADURÍA GENERAL DEL ESTADO

Tabla 1.1 Proceso de Gestión Pública [3]

El Estado está inmerso en diferentes sectores de la sociedad, por lo que es necesario determinar una muestra representativa de las diferentes organizaciones que nos permita determinar los modelos de gestión administrativa que están

[3] Fuente: SENRES. Norma Técnica de Diseño de Reglamentos o Estatutos Orgánicos de Gestión Organizacional por Procesos.

utilizando; para la presente investigación se tomó como muestra a la Superintendencia de Compañías, Petroproducción, Ministerio de Salud Pública y Correos del Ecuador.

Para realizar la caracterización de la gestión de las organizaciones del sector público del Ecuador, es necesario realizar un análisis del espacio organizacional, el mismo que está constituido por las dimensiones:

1. **material**: edificios, máquinas, inventarios, etc.;
2. **humana**: conformada por seres motivados exclusivamente por compensaciones monetarias;
3. **tecnológica**: procesos, técnicas, sistemas operacionales, etc.; y
4. **estructural – jerárquica:** corresponde a la dimensión política del espacio, con la participación de la lógica de la división del poder en la organización, su departamentalización, la toma de decisiones, etc.

Con el fin de hacer una comparación entre las instituciones tomadas como muestra, estas cuatro dimensiones han sido disgregadas en las siguientes variables: Jerarquía de la autoridad, división del trabajo – especialización, procedimientos, profesionalización, burocratización, tecnología, recursos humanos, y origen e historia.

En la siguiente Tabla 1.2 se hace una comparación del espacio organizacional de las cuatro instituciones públicas, utilizando las variables antes mencionadas. Podemos evidenciar que existen entre estas organizaciones muchas más similitudes que diferencias: todas son organizaciones burocráticas, lo que se puede evidenciar a través de las estructuras organizacionales fundamentadas en una jerarquía administrativa; todas las organizaciones fundamentan su quehacer a partir de la división del trabajo; el Ministerio de Salud y Correos del Ecuador agrupan sus actividades por procesos y tienen definidos su documentación de acuerdo a los requerimientos de la Norma Técnica de Diseño de Reglamentos o Estatutos Orgánicos de Gestión Organizacional por Procesos.

Variable	Correos del Ecuador	Ministerio de Salud Pública	Petroproducción	Superintendencia de compañías
Jerarquía de la autoridad	Estructura jerárquica	Estructura jerárquica	Estructura jerárquica	Estructura jerárquica
División del trabajo	Gestión por procesos Sistema de gestión de la calidad	Gestión por procesos	Gestión por funciones	Gestión por funciones
Procedimientos	Manual de procedimientos	Manual de procedimientos	Manual de funciones	Manual de funciones
Profesionalización del nivel directivo	16 directivos 13 profesionales 4 con formación de cuarto nivel	28 directivos 25 profesionales 18 con formación de cuarto nivel	13 directivos 11 profesionales 1 con formación de cuarto nivel	26 directivos 20 profesionales 8 con formación de cuarto nivel
Burocratización	Burocracia profesional	Burocracia profesional	Burocracia profesional	Burocracia profesional
Tecnología	Sistemas informáticos acordes a sus requerimientos	Sistemas informáticos acordes a sus requerimientos	Sistemas informáticos acordes a sus requerimientos	Sistemas informáticos acordes a sus requerimientos
Recursos humanos	Antigüedad: 1 año	Antigüedad: 20 años Edad promedio: 51 años	Antigüedad: 20 años Edad promedio: 44 años	Antigüedad: 14 años Edad promedio: 44 años
Origen e historia	29 de septiembre de 1971	6 de junio de 1967	Corporación Estatal Petrolera Ecuatoriana (CEPE) 23 de junio de 1972 Petroproducción 26 de septiembre de 1989	1 de junio de 1967

Tabla 1.2 Espacio organizacional [4]

1.2 GESTIÓN DE INVENTARIOS EN LAS INSTITUCIONES PÚBLICAS DEL ECUADOR

En las instituciones públicas existen procedimientos que permiten la ejecución adecuada de las actividades. Es importante elaborar “rutinas de procedimientos” para demostrar en forma más sencilla como se ejecuta una actividad, quiénes son los responsables de cada fase y qué instrumentos requiere para su cumplimiento.

[4] Fuente: NARANJO, Efraín. Caracterización de la Gestión en las Empresas Públicas.

Cuando la institución dispone de rutinas de procedimientos para realizar su trabajo, el personal más calificado puede dedicar mayor tiempo a funciones de supervisión, dejando al personal subordinado las actividades de menor importancia o repetitivas.

Un manual de procedimientos administrativos incluye la secuencia en que deben efectuarse las actividades, los pasos en que deben llevarse a cabo las mismas, así como las formas de papelería que es necesario incluir, modificar o cambiar para la operación eficiente de los sistemas de procedimientos.

Una vez definido un procedimiento, es importante que se respeten todos sus pasos, ya que la omisión de alguno de ellos en lugar de favorecer, entorpece y obstaculiza todo el procedimiento administrativo.

Existen tres procedimientos básicos que intervienen o influyen en la gestión de los inventarios en las entidades del sector público ecuatoriano: el procedimiento para el plan de adquisiciones, el procedimiento de compras y el procedimiento para la provisión de artículos de almacén.

La Figura 1.1 expone algunos de los símbolos que serán utilizados para ilustrar estos procedimientos.

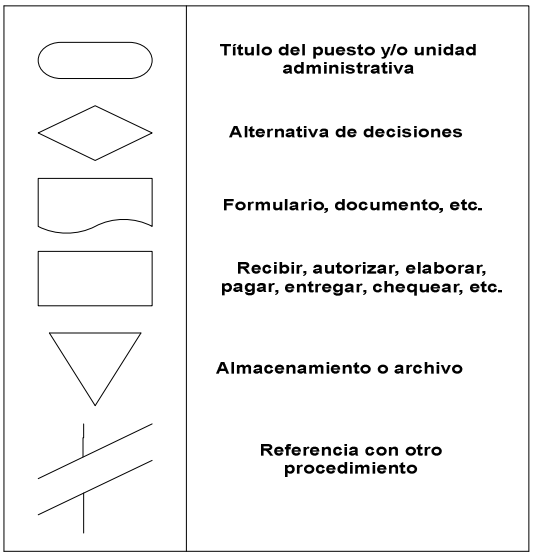


Figura 1.1 Símbolos utilizados para las rutinas de procedimientos [5]

[5] Fuente: VÁSQUEZ, Víctor Hugo. Organización Aplicada.

1.2.1 PROCEDIMIENTO PARA EL PLAN DE ADQUISISIONES

Cualquier organización, sea grande o pequeña, debe planear las adquisiciones de bienes que son necesarios para el cumplimiento de las funciones de la entidad. En las entidades del sector público ecuatoriano, se debe elaborar el Plan Anual de Adquisiciones en coordinación con la Dirección Financiera y Unidades Administrativas, de acuerdo a la prioridad y necesidades de cada dependencia y en base a las disponibilidades presupuestarias. Sin embargo, según el Artículo 6 del Reglamento General de Bienes del Sector Público, se pueden adquirir bienes que no consten en el respectivo Plan Anual de Adquisiciones, lo cual requerirá de la resolución de la más alta autoridad de la entidad.

El esquema de la Figura 1.2 tiene como objetivo mostrar la metodología para elaborar una programación de las compras que deben ser efectuadas por la entidad pública, para lo cual se recomienda la utilización de las técnicas de planeación.

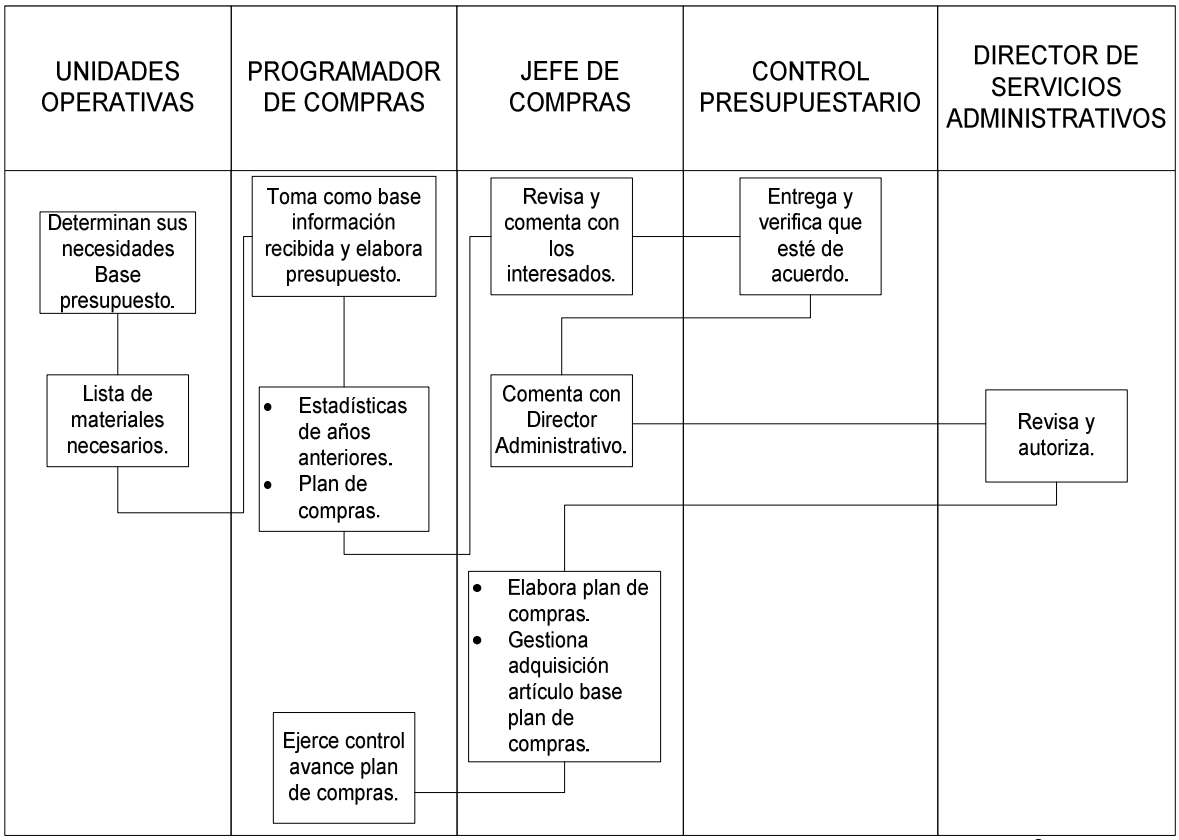


Figura 1.2 Rutina de procedimientos del plan de adquisiciones [6]

[6] Fuente: VÁSQUEZ, Víctor Hugo. Organización Aplicada.

1.2.2 PROCEDIMIENTO DE COMPRAS

Las entidades del sector público realizan sus compras en base a un plan previamente elaborado, generalmente el Plan Anual de Adquisiciones. Toda requisición de bienes se la efectúa previa legalización de la orden respectiva; de otra manera no se podrá atender ninguna petición. Las instituciones públicas, realizan sus compras en base a los aspectos normativos que regulan la acción, considerando que hoy existe un Sistema de Contratación Pública, el cual articula a todas las instituciones del Sector Público en ámbitos como la planificación, programación, presupuesto, control, administración y ejecución de las adquisiciones de bienes; por lo que la autorización de las requisiciones de bienes se cumple de conformidad con reglamentos especiales que en cada organización en particular se aplique de acuerdo a sus necesidades.

El procedimiento de compras en una entidad pública es el que se detalla en la Figura 1.3.

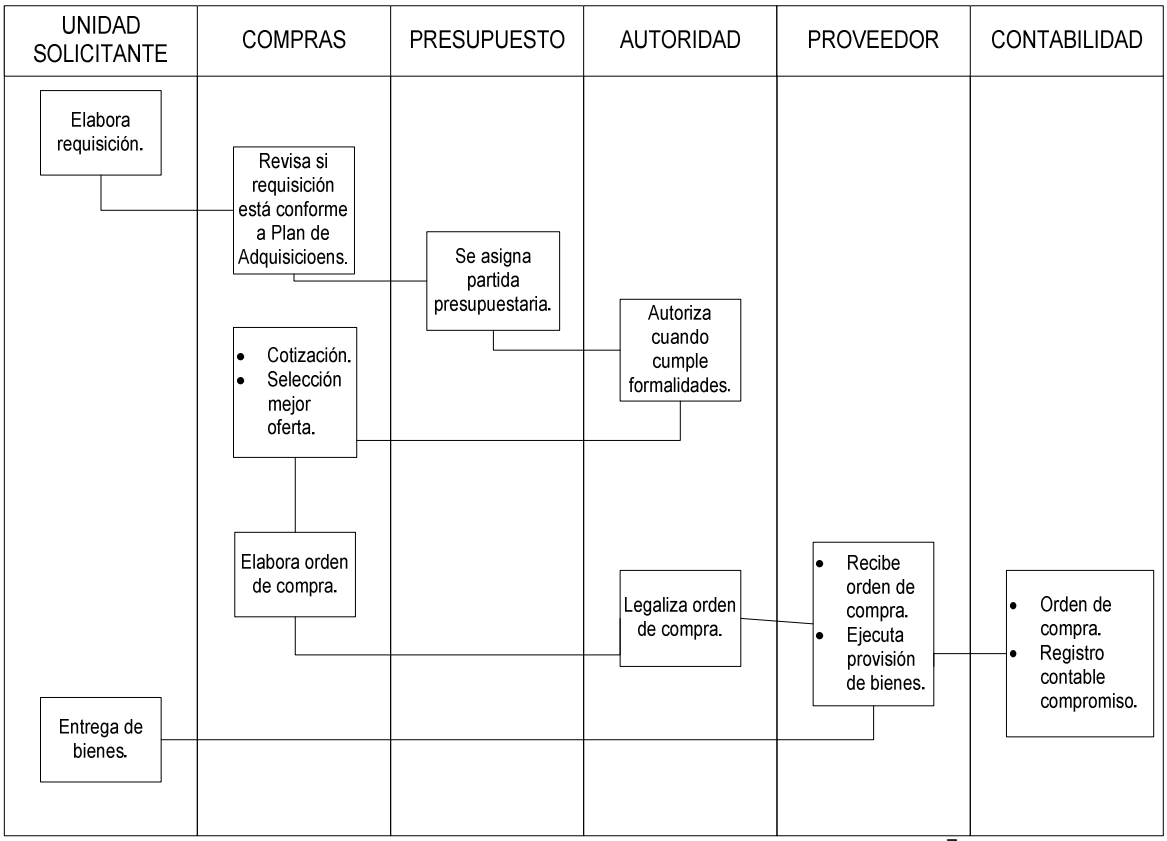


Figura 1.3 Rutina de procedimientos para compras [7]

[7] Fuente: VÁSQUEZ, Víctor Hugo. Organización Aplicada.

En la Figura 1.4 se tiene un ejemplo del documento utilizado para realizar una requisición.

SOLICITADO POR:…… UNIDAD ADMINISTRATIVA……	FECHA:	No.

REQUERIDO POR:……..	Autorizado por:	Partida:

CANTIDAD	DESCRIPCIÓN	MARCA	PRECIO/U.	PRECIO/T.	ORDEN DE COMPRA No.

Firma solicitante	Firma de autorizado	Fecha de entrega

Figura 1.4 Ejemplo de formulario utilizado para requisiciones [8]

1.2.3 PROCEDIMIENTO PARA LA PROVISIÓN DE ARTÍCULOS DE ALMÁCÉN

El empleado de una entidad que cumple funciones de Guardalmacén debe tener presente que cualquier demora en la entrega de los suministros solicitados por las unidades de la organización, puede ocasionar un descenso en la productividad por la falta de una oportuna utilización de los recursos materiales en relación al tiempo programado para el cumplimiento de la función.

[8] Fuente: VÁSQUEZ, Víctor Hugo. Organización Aplicada.

Por norma general, la institución debe propender a que los trámites administrativos relacionados con la provisión de suministros sean sencillos y ágiles, sin dejar de establecer los mecanismos y procedimientos de control necesarios para favorecer una adecuada administración de los recursos materiales.

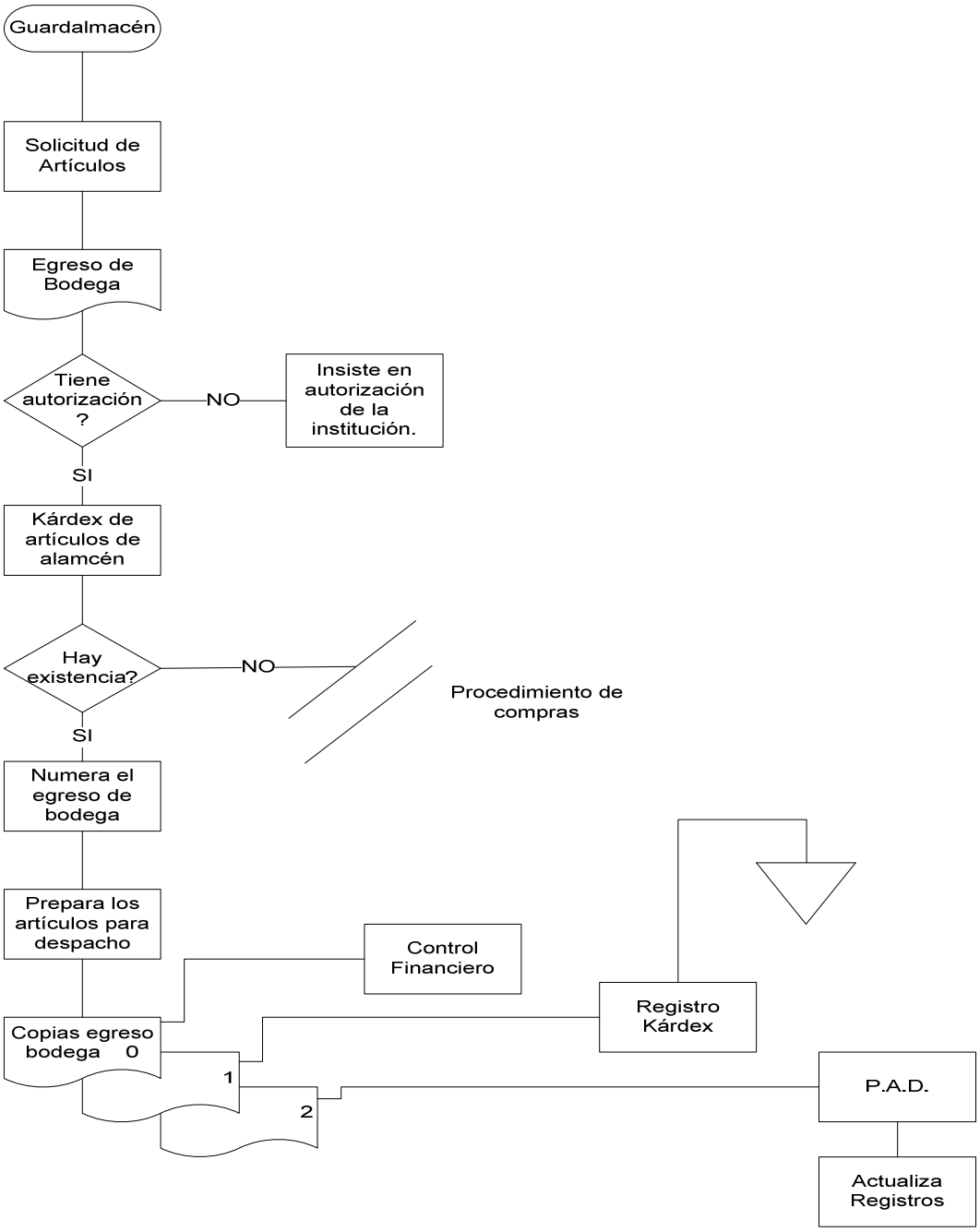


Figura 1.5 Rutina gráfica de procedimientos – Provisión artículos de almacén [9]

[9] Fuente: VÁSQUEZ, Víctor Hugo. Organización Aplicada.

Todo artículo para salir del almacén debe estar respaldado por una orden de movimiento de bodega, debidamente legalizado por una autoridad competente de la organización. Generalmente quien autoriza la salida de artículos del almacén son los jefes de los Departamentos Administrativos o de Servicios Generales a través de la utilización de formularios expresamente diseñados para el efecto. La Figura 1.5 ilustra el procedimiento para la provisión de artículos de almacén.

1.3 JUSTIFICACIÓN DE LA METODOLOGÍA DE DESARROLLO

El desarrollo de un buen sistema de software depende de un sinnúmero de actividades y etapas, por ello la elección de la metodología de desarrollo para un equipo en un determinado proyecto es vital para obtener un producto de calidad.

En la actualidad, las metodologías se basan en filosofías de desarrollo que se las puede dividir en dos grandes grupos: las metodologías tradicionales, que se basan en una fuerte planificación durante todo el desarrollo, y las metodologías ágiles, en las que el desarrollo de software es incremental, cooperativo, sencillo y adaptado.

1.3.1 METODOLOGÍAS TRADICIONALES

Las metodologías tradicionales son denominadas, a veces, de forma peyorativa, como metodologías pesadas y básicamente centran su atención en llevar una documentación exhaustiva de todo el proyecto y en cumplir con un plan de proyecto, el cual es definido en la fase inicial del desarrollo.

Otra de las características importantes dentro de este enfoque es la premisa universal de la ingeniería de software que dice que el costo de cambiar un programa aumenta exponencialmente en el transcurso del tiempo, como se muestra en la Figura 1.6.

Las metodologías tradicionales se focalizan en la documentación, planificación y procesos (plantillas, técnicas de administración, revisiones, etc.)

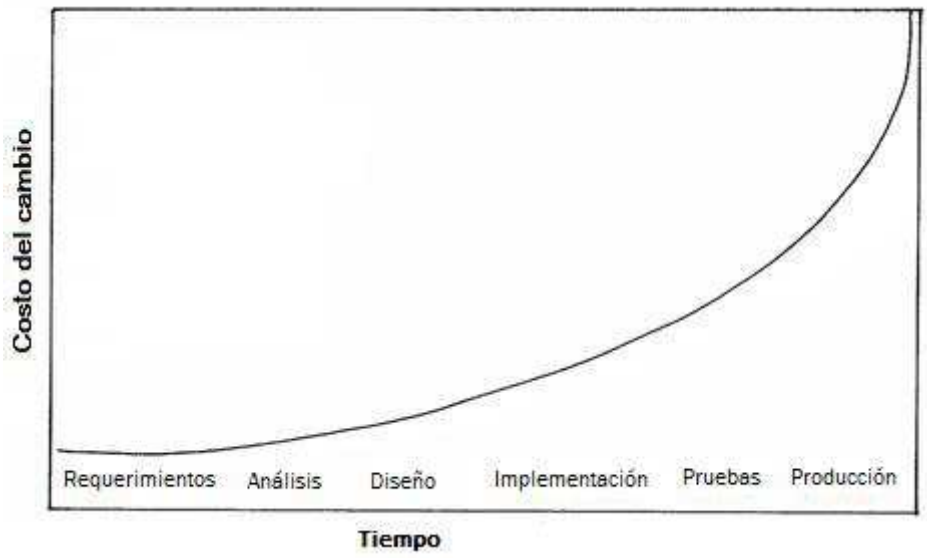


Figura 1.6 Costo del cambio a lo largo del tiempo

1.3.2 METODOLOGÍAS ÁGILES

Este enfoque nace como respuesta a los problemas que pueden surgir de las metodologías tradicionales y se basa en dos aspectos esenciales: retrasar las decisiones y utilizar la planificación adaptativa. Basan su fundamento en la adaptabilidad de los procesos de desarrollo. Estas metodologías priorizan la capacidad de responder ante un cambio en lugar de ajustarse al seguimiento de un plan estricto.

Tomando las ideas de la Tabla 1.3 se puede decir que las metodologías tradicionales presentan los siguientes problemas a la hora de abordar proyectos:

- Existen unas costosas fases previas de especificación de requisitos, análisis y diseño. La corrección durante el desarrollo de errores introducidos en estas fases será costosa, es decir, se pierde flexibilidad ante los cambios.
- El proceso de desarrollo está ceñido por documentos firmados.
- El desarrollo es más lento.
- Es difícil para los desarrolladores entender un sistema complejo en su globalidad.

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo)	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas / normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 1.3 Comparativa entre metodologías tradicionales y ágiles [10]

Las metodologías ágiles de desarrollo están especialmente indicadas en proyectos con requisitos poco definidos o cambiantes. Estas metodologías se aplican bien en equipos pequeños que resuelven problemas concretos, lo que no está reñido con su aplicación en el desarrollo de grandes sistemas, ya que con una correcta modularización de los mismos se puede llegar a una implantación exitosa. Las metodologías ágiles presentan diversas ventajas, entre las que se pueden destacar:

- Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos breves de software funcional.

[10] Fuente: LABORATORIO NACIONAL DE CALIDAD DEL SOFTWARE (INSTITUTO NACIONAL DE TECNOLOGÍA DE LA COMUNICACIÓN). Ingeniería del Software: Metodologías y Ciclos de Vida.

- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Importancia de la simplicidad, eliminando el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo.

Dentro de los procesos ágiles de desarrollo, la Programación Extrema (Extreme Programming o XP) es uno de los enfoques más destacados, y para el presente proyecto es la metodología que será utilizada. La principal razón por la que se utilizará XP es que esta metodología toma muy en cuenta que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo del software; lo cual encaja perfectamente con la realidad que se presenta al momento de desarrollar la mayoría de sistemas de software.

Se puede considerar a la Programación Extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto y aplicarlo de manera dinámica durante el ciclo de vida del software.

1.3.3 EXTREME PROGRAMMING - XP

La Programación Extrema es un enfoque de la ingeniería del software formulado por Kent Beck. Al igual que las otras metodologías ágiles, la Programación Extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Se cree que la capacidad de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en la retroalimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios.

XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes. Los principios y prácticas son de sentido común pero llevadas al extremo, de ahí proviene su nombre.

Además de estos valores, que podríamos denominar como los pilares de XP, tenemos un compendio de 12 buenas prácticas:

- **La planificación**: se utilizan las “Historias de Usuario” para realizar el análisis, son la forma en la cual el usuario entrega los requerimientos del sistema al equipo de trabajo. Son la descripción del sistema desde el punto de vista del usuario.

- **Versiones rápidas**: La primera versión contendrá el conjunto mínimo de requisitos más necesarios para el sistema.

- **Sistema metafórico**: Cada proyecto debe tener una metáfora asociada que nos ofrezca unos criterios para nombrar lo que vayamos haciendo de forma fácil.

- **Diseño simple**: Como los requerimientos cambian, o pueden hacerlo, diariamente, hay que utilizar los diseños más simples posibles para cumplir los requerimientos que tenemos en la actualidad.

- **Testeo continuo**: Antes de que se implemente cualquier característica de un sistema, se debe escribir un test para ella.

- **Refactoring**: Cuando tenemos que introducir una nueva característica del sistema, si esta tiene mucho en común con otra previa, lo mejor es eliminar el código duplicado, sin miedo a que falle, debido a que el test probará el correcto funcionamiento.

- **Programación en parejas**: Se trabaja en parejas, cada una utilizando un único ordenador. Así, el código se revisa mientras se desarrolla.

- **Propiedad colectiva del código**: Cualquiera puede modificar cualquier módulo en cualquier momento, nadie tiene la propiedad de ningún módulo.

- **Integración continua**: Todos los cambios se introducen en el sistema, al menos, una vez al día.
- **Semanas de 40 horas de trabajo**: Los programadores se deben ir a casa a su hora. No se debe exceder las 8 horas diarias de trabajo.
- **Cliente en su sitio**: Siempre hay un usuario del sistema que es accesible por los miembros del equipo de trabajo.
- **Estándares de codificación**: Todos deben usar los mismos criterios a la hora de programar. De esta forma, no sería posible determinar quién ha realizado una determinada parte de la implementación.

Con el objetivo de justificar la utilización de esta metodología de desarrollo, en la Tabla 1.4 se hace una comparación de las principales características de la Programación Extrema con las características del proyecto, de esta manera se puede evidenciar que la metodología seleccionada es la más adecuada para este tipo de proyecto.

Características de XP	Características del proyecto
Permite introducir nuevos requisitos o cambiar los anteriores de un modo dinámico.	Si bien el proyecto comienza con una serie de consideraciones ya expuestas y ratificadas en cuanto al alcance del sistema, se ha previsto que durante la etapa de desarrollo se puedan introducir nuevos requerimientos que no representen cambios drásticos a lo que ya esté desarrollado hasta el momento de implementar el nuevo requerimiento.
Se obtienen rápidamente versiones que implementan parte de los requisitos.	Los requerimientos del sistema están divididos en grupos, de tal forma que se puedan ir implementando partes del sistema totalmente funcionales antes de tener el sistema completo.
Es adecuado para proyectos pequeños y medianos.	De acuerdo al alcance y a la complejidad del sistema, se puede considerar al proyecto como de tamaño mediano.

Es adecuado para proyectos con alto riesgo.	Tomando en cuenta la complejidad, el desarrollo de este sistema no involucra mayores riesgos como para que el proyecto fracase. Sin embargo, al tratarse de un proyecto vinculado con el Estado, los tiempos de las entregas deben cumplirse estrictamente a lo estipulado para no tener ningún inconveniente legal que pueda significar la terminación del proyecto.
Su ciclo de vida es iterativo e incremental.	El desarrollo del proyecto se lo ha enfocado para que el producto sea entregado al cliente de forma incremental sobre un periodo de tiempo planificado.
Necesita una continua interacción con el cliente, y no con cualquiera, sino con alguien que conozca la institución y sus necesidades.	Se tiene el respaldo de una persona que está encargada de la recolección de los requerimientos y que conoce ampliamente de instituciones públicas y de sus necesidades.
No produce demasiada documentación acerca del diseño o la planificación.	Este proyecto no demanda la exhaustiva documentación de la planificación, ya que se requiere que el desarrollo del sistema sea lo más rápido posible.

Tabla 1.4 Comparación de XP con las características del proyecto

1.4 JUSTIFICACIÓN DE LAS HERRAMIENTAS DE DESARROLLO

En abril de 2008, el Presidente del Ecuador, Rafael Correa Delgado, firmó el decreto 1014, con el cual el uso de Software Libre pasa a ser una política de Estado, la cual debe empezar a ser aplicada en todas las entidades públicas; por tal razón, para el presente proyecto se va a utilizar una infraestructura LAMP[11]: Linux como sistema operativo, Apache como servidor web, MySQL como gestor de base de datos y PHP como lenguaje de programación.

[11] Acrónimo de **L**inux-**A**pache-**M**ySQL-**P**HP.

1.4.1 ARQUITECTURA LAMP

La arquitectura LAMP es uno de los casos de éxito más sobresalientes del software libre como arquitectura. Esta arquitectura, que consta de un sistema operativo, un servidor web, una base de datos y un lenguaje de programación, todos ellos distribuidos bajo una licencia de software libre, ofrece una manera tremendamente rápida para la construcción de una aplicación web a un bajo coste, además de venir normalmente incluida en la mayoría de distribuciones Linux. Siendo tecnologías sin dependencia alguna entre ellas (nunca fueron pensadas con el fin de interactuar entre sí), hace que resulte sorprendente los buenos resultados que se suelen obtener en producción, y siendo un gran ejemplo de la variedad y calidad de productos que ofrece el software libre.

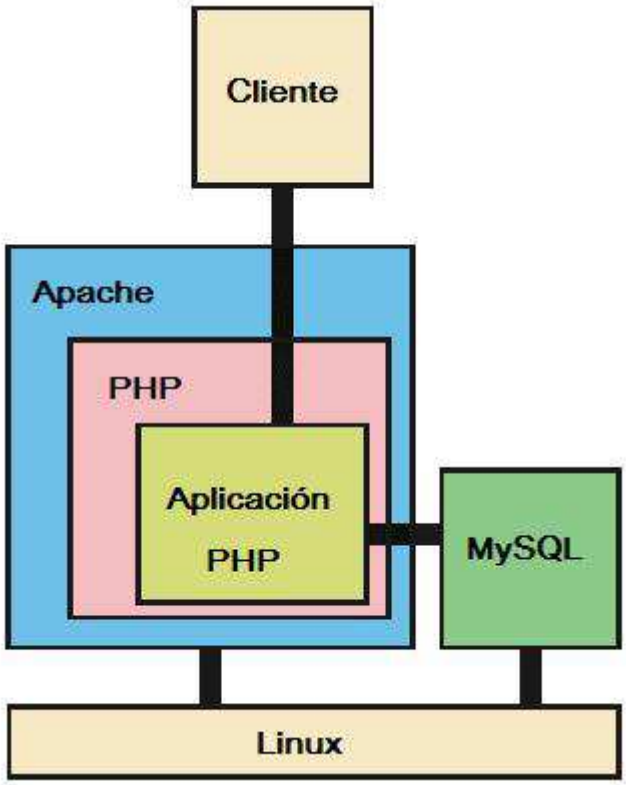


Figura 1.7 Arquitectura LAMP

1.4.1.1 Linux

Linux es un sistema operativo con dos características muy particulares que lo diferencian del resto de sistemas que podemos encontrar en el mercado; la primera, es que es libre, esto significa que no tenemos que pagar ningún tipo de

licencia a ninguna casa desarrolladora de software por el uso del mismo, la segunda, es que el sistema viene acompañado del código fuente. El sistema lo forman el núcleo del sistema (kernel) más un gran número de programas / bibliotecas que hacen posible su utilización. Muchos de estos programas y bibliotecas han sido posibles gracias al proyecto GNU, por esto mismo, muchos llaman a Linux, GNU/Linux, para resaltar que el sistema lo forman tanto el núcleo como gran parte del software producido por el proyecto GNU. La distribución Linux que se va a utilizar para levantar los servicios necesarios para la ejecución del sistema es Fedora en su versión 13, la cual fue liberada el 25 de mayo de 2010 y también se la conoce como Goddard.

1.4.1.2 Apache

Apache es un servidor web potente y flexible; es altamente configurable y extensible con módulos de terceros y módulos personalizados; provee completamente su código fuente y viene con una licencia no restrictiva; puede ser ejecutado en Windows 2000, Netware (desde versiones 5.x), OS/2, la mayoría de versiones Unix / Linux y en muchos otros sistemas operativos. Apache siempre está desarrollándose activamente.

1.4.1.3 MySQL

MySQL es un gestor de base de datos que ha llegado a ser la más popular de código abierto en el mundo debido a su alto rendimiento, alta confiabilidad y facilidad de uso. También es la base de datos de preferencia para una nueva generación de aplicaciones construidas en la arquitectura LAMP. Muchas de las organizaciones más grandes del mundo incluyendo Facebook, Google, Adobe, Alcatel ponen su confianza en MySQL para ahorrar tiempo y dinero en el fortalecimiento de sus sitios web de alto volumen de contenido. MySQL es muy flexible, pudiéndose ejecutar en más de 20 plataformas incluyendo Linux, Windows, Mac, Solaris e IBM. No importa si se trata de alguien nuevo en la tecnología de bases de datos, de un desarrollador experimentado o de un administrador, MySQL ofrece las herramientas y el soporte necesario para alcanzar el éxito.

1.4.1.4 PHP

PHP es un lenguaje de programación interpretado o framework para HTML, diseñado originalmente para la creación de páginas web dinámicas. Se usa principalmente para la interpretación del lado del servidor, pero actualmente puede ser utilizado desde una interfaz de línea de comandos o en la creación de otros tipos de programas incluyendo aplicaciones con interfaz gráfica. PHP es compatible con la mayoría de servidores web, incluyendo Apache. PHP permite embeber fragmentos de código en páginas HTML normales – código que es interpretado mientras las páginas son servidas a los usuarios. PHP también sirve como un lenguaje de "enlace", haciendo más fácil conectar páginas web a bases de datos del lado del servidor. Algunas de las razones para utilizar PHP son las siguientes: es libre, es de código abierto, gran funcionalidad, multiplataforma, es estable, rápido y es fácil de aprender.

A más del lenguaje de programación PHP, se van a utilizar dos herramientas que complementan la funcionalidad que se puede obtener al utilizar solo el lenguaje de lado del servidor. Estás herramientas son JQuery y CSS, las cuales proveen las utilidades necesarias para desarrollar una aplicación web de forma más profesional y con interfaces mucho más entendibles y fáciles de utilizar por parte de los usuarios finales. Según el sitio oficial de JQuery, "JQuery es una rápida y concisa librería de Javascript que simplifica el recorrido por un documento HTML, el manejo de eventos, animaciones e interacciones Ajax[12] para un desarrollo web rápido. JQuery está diseñado para cambiar la manera en que se escribe Javascript"[13]. Por otra parte, CSS, Hojas de Estilo en Cascada (Cascading Style Sheets), es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. CSS se utiliza para dar estilo a documentos HTML y XML, separando el contenido de la presentación. CSS permite a los desarrolladores controlar el estilo y el formato de múltiples páginas web al mismo tiempo.

[12] Acrónimo de Asynchronous JavaScript And XML (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas.
[13] Fuente: THE JQUERY FOUNDATION. JQuery is a new kind of JavaScript Library. http://jquery.com/

1.4.2 HERRAMIENTAS DE DESARROLLO

Una vez definida la infraestructura sobre la cual se va a trabajar, se han seleccionado algunas herramientas para llevar adelante el desarrollo de este proyecto, las cuales se describen a continuación.

1.4.2.1 CakePHP

CakePHP es un framework de desarrollo rápido para PHP, el cual usa patrones de diseño conocidos. Es un proyecto de código abierto y es gratis para cualquiera que quiera utilizarlo. Un framework PHP es una colección de código, librerías, clases y entorno de ejecución que ayuda a los desarrolladores a construir aplicaciones web más rápidamente. La idea principal detrás del uso de frameworks es la de proveer a los desarrolladores las funcionalidades comúnmente utilizadas y una estructura básica sobre la cual se puede construir una aplicación.

Un patrón de diseño es una solución general a un problema que se presenta con frecuencia en el desarrollo web. Un patrón de diseño no es un código completo, más bien es una descripción de cómo resolver un problema que puede ser utilizada en diferentes situaciones. En desarrollo web, hay muchos patrones de diseño que son utilizados para resolver problemas repetitivos y comunes. Uno de estos patrones que implementa CakePHP es el patrón MVC, que es la sigla de Modelo Vista Controlador y es el núcleo de CakePHP.

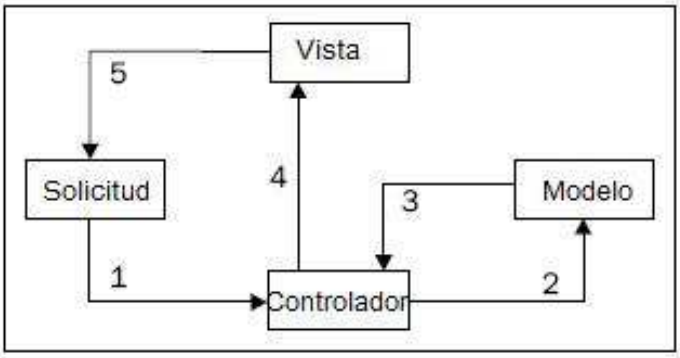


Figura 1.8 Núcleo de CakePHP - Modelo Vista Controlador[14]

[14] Fuente: BARI, Ahsanul. SYAM, Anupom. CakePHP Application Development.

El diagrama de la Figura 1.8 muestra cómo trabaja todo junto cuando se da una petición de usuario:

1. La petición es despachada al controlador, con datos de usuario.
2. El controlador procesa la petición y llama al modelo para acceder a los datos.
3. El modelo responde a la llamada del controlador enviando o guardando datos.
4. Luego, el controlador envía los datos de salida a la vista.
5. La vista despliega los datos en el formato adecuado.

La integración de patrones de diseño comunes en CakePHP significa que los desarrolladores no necesitan desperdiciar tiempo tratando de resolver problemas que comúnmente están presentes en casi todos los proyectos web. Estos ya están resueltos en CakePHP. Como resultado, el desarrollador se puede enfocar en la lógica del negocio específica de la aplicación. Esto se deriva en un desarrollo de la aplicación mucho más rápido.

1.4.2.2 SimpleTest

Una de las características fundamentales de la metodología de desarrollo seleccionada para este proyecto es la realización de pruebas unitarias continuas. Para esto se aconseja escribir primero el código de las pruebas antes de la codificación en sí. Una herramienta que nos ayuda con este objetivo es SimpleTest.

SimpleTest es el entorno ideal para realizar pruebas unitarias en proyectos desarrollados en CakePHP, ya que se integra perfectamente con este. SimpleTest proporciona dos cosas, un framework que permite la fácil escritura de las pruebas y también la funcionalidad para ejecutar las pruebas cómodamente desde un navegador web y poder analizar sus resultados.

1.4.2.3 NetBeans

NetBeans es un Ambiente de Desarrollo Integrado de código abierto. Tiene todas las herramientas necesarias para crear aplicaciones de escritorio, empresariales, web y móviles; con plataformas como Java, C/C++, PHP, Javascript y Groovy. Posee un buen rendimiento, brinda una excelente experiencia en la codificación y también soporta la integración con múltiples frameworks PHP. NetBeans ofrece plantillas y también generación automática de código, refactorización (renombramiento instantáneo), ayudas textuales y completación inteligente de código. El editor reconoce código HTML en archivos JavaScript y código JavaScript en archivos HTML, también reconoce código HTML en archivos PHP. Se integra con populares librerías JavaScript como JQuery y Prototype, además de soportar el lenguaje para hojas de estilo CSS.

1.4.3 JUSTIFICACIÓN DE LA INFRAESTRUCTURA Y HERRAMIENTAS UTILIZADAS

Hay que recordar que al tratarse de un sistema para instituciones públicas, la principal razón en la que se basa la selección de las herramientas es que se debe utilizar software libre, debido al decreto ejecutivo mencionado anteriormente. Partiendo de eso, en la Tabla 1.5 se resume y se resalta también las características por las cuales se ha escogido la infraestructura y las herramientas para el presente proyecto.

Herramienta	Función	Justificación
Linux Fedora 13	Sistema operativo	Es un sistema operativo optimizado para que sobre él se implementen varios tipos de servidores, entre ellos: servidores web como Apache y de bases de datos como MySQL.
Apache 2.2.15	Servidor web	Permite montar un potente servidor web y es altamente configurable.
MySQL 5.1.45	Gestor de base de datos	Permite configurar un servidor de base de datos muy confiable, ya que cuenta con herramientas que facilitan la administración y la seguridad de los datos.

PHP 5.3.6	Lenguaje de programación	Es un lenguaje de programación que posibilita la creación de sistemas web robustos, gracias a la amplia funcionalidad que ofrece. Es compatible con el servidor web Apache y también puede interactuar con una base de datos MySQL.
CakePHP 1.3.4	Framework de desarrollo	Ayuda a la rápida creación de aplicaciones web desarrolladas con el lenguaje PHP y utiliza estándares de programación, lo cual es una recomendación de la metodología de desarrollo utilizada para este proyecto.
SimpleTest 1.0.1	Entorno para pruebas	Ofrece toda la funcionalidad necesaria para realizar pruebas unitarias constantemente, lo cual también es una disposición de XP. Además, se complementa muy bien con el framework CakePHP.
NetBeans 6.8	Ambiente de desarrollo (IDE)	Posee herramientas suficientes que permiten organizar, visualizar, corregir y ejecutar el código fuente de manera efectiva. Se integra perfectamente con todas las anteriores herramientas.

Tabla 1.5 Justificación de las herramientas de desarrollo

CAPÍTULO 2. ANÁLISIS Y DISEÑO DEL SISTEMA

2.1 ESPECIFICACIÓN DE REQUERIMIENTOS

2.1.1 HISTORIAS DE USUARIO

Cuando se utiliza Extreme Programming como metodología de desarrollo, la forma en la que se especifican los requerimientos, por así decirlo, es mediante las Historias de Usuario, las cuales consisten en una serie de documentos cortos (tarjetas) escritos por los usuarios. Algunas de sus características son las siguientes:

- Son parecidas a los casos de uso de UML, pero las escriben los usuarios.
- Son cortas (dos ó tres líneas aproximadamente) y sin vocabulario técnico.
- Cada una especifica un requisito del sistema.
- Pueden usarse para:
    - Estimar el tiempo de desarrollo.
    - Hacer un test de aceptación a partir de la Historia de Usuario.
- Los desarrolladores estiman cuanto tiempo es necesario para implementar cada una.
- El tiempo para implementar una Historia de Usuario se lo mide en días o semanas de tiempo ideal. Se define como tiempo ideal al tiempo de trabajo sin interrupción, en el cual una pareja de programadores puede concentrar toda su atención a las tareas de desarrollo.

Una vez definidas las características de las Historias de Usuario, se procedió a la escritura de las mismas, para lo cual se han tomado en cuenta tres aspectos o módulos del sistema:

- **Administración del sistema –** Manejo de datos base.
- **Movimientos externos –** Relaciones con los proveedores.

- **Movimientos internos –** Relaciones con las dependencias de la institución pública.

2.1.1.1 Historias de Usuario para la Administración del Sistema

Desde la Tabla 2.1 hasta la Tabla 2.8 se tienen las Historias de Usuario que han sido definidas para la Administración del Sistema.

Historia de Usuario No: 1	**Título:** Autenticación de usuario
Descripción: Todo usuario del sistema debe poder ingresar únicamente mediante la validación de un nombre de usuario y una contraseña.	
Prioridad: Alta (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 1

Tabla 2.1 Historia de Usuario “Autenticación de usuario”

Historia de Usuario No: 2	**Título:** Crear departamento
Descripción: Como usuario Administrador debo registrar los datos de un nuevo departamento y poder ver una lista de todos los departamentos existentes.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 3

Tabla 2.2 Historia de Usuario “Crear departamento”

Historia de Usuario No: 3	**Título:** Crear usuario
Descripción: Como usuario Administrador debo registrar los datos de un nuevo usuario y poder ver una lista de todos los usuarios existentes.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 3

Tabla 2.3 Historia de Usuario “Crear usuario”

Historia de Usuario No: 4	**Título:** Crear proveedor
Descripción: Como usuario Administrador debo registrar los datos de un nuevo proveedor y poder ver una lista de todos los proveedores existentes.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 3

Tabla 2.4 Historia de Usuario “Crear proveedor”

Historia de Usuario No: 5	**Título:** Crear categoría de productos
Descripción: Como usuario Administrador debo registrar los datos de una nueva categoría de productos y poder ver una lista de todas las categorías existentes.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 3

Tabla 2.5 Historia de Usuario “Crear categoría de productos”

Historia de Usuario No: 6	**Título:** Crear producto
Descripción: Como usuario Administrador debo registrar los datos de un nuevo producto y poder ver una lista de todos los productos existentes.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 3

Tabla 2.6 Historia de Usuario “Crear producto”

Historia de Usuario No: 7	**Título:** Perfil de usuario
Descripción: Cada usuario debe tener la posibilidad de ver la información relacionada a su perfil, donde se incluirá la opción para poder cambiar su contraseña. Sólo el Administrador puede cambiar la contraseña de cualquier usuario desde su cuenta.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 3 días	**Iteración asignada:** 3

Tabla 2.7 Historia de Usuario “Perfil de usuario”

Historia de Usuario No: 8	**Título:** Comunicaciones internas
Descripción: Como usuario Supervisor debo poder publicar comunicaciones de interés común a todos los usuarios del sistema, por lo que estarán disponibles para cualquier usuario sin importar su perfil.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 4 días	**Iteración asignada:** 3

Tabla 2.8 Historia de Usuario “Comunicaciones internas”

2.1.1.2 Historias de Usuario para los Movimientos Externos

En lo concerniente a las relaciones externas, las Historias de Usuario que se han obtenido se muestran desde la Tabla 2.9 hasta la Tabla 2.12.

Historia de Usuario No: 9	**Título:** Pedido a proveedor
Descripción: Como usuario Bodeguero debo ingresar un nuevo pedido a proveedor, el cual será enviado para ser aprobado o anulado por un usuario Supervisor.	
Prioridad: Alta (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 5 días	**Iteración asignada:** 1

Tabla 2.9 Historia de Usuario “Pedido a proveedor”

Historia de Usuario No: 10	**Título:** Listar pedidos a proveedores
Descripción: Como usuario Supervisor y como usuario Bodeguero debo poder ver una lista de todos los pedidos a proveedores con el estado actual del pedido.	
Prioridad: Alta (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 1

Tabla 2.10 Historia de Usuario “Listar pedidos a proveedores”

Historia de Usuario No: 11	**Título:** Aprobar / Anular pedido a proveedor
Descripción: Como usuario Supervisor debo aprobar o anular aquellos pedidos a proveedor que se encuentren en un estado de “enviado”.	
Prioridad: Alta (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 1

Tabla 2.11 Historia de Usuario “Aprobar / Anular pedido a proveedor”

Historia de Usuario No: 12	**Título:** Confirmar entrega de pedido a proveedor
Descripción: Como usuario Bodeguero debo confirmar la entrega de aquellos pedidos a proveedor que se encuentren en un estado de “aprobado”, ó bien dejar constancia de cualquier anomalía en el momento de la entrega.	
Prioridad: Alta (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 3 días	**Iteración asignada:** 1

Tabla 2.12 Historia de Usuario “Confirmar entrega de pedido a proveedor”

2.1.1.3 Historias de Usuario para los Movimientos Internos

Finalmente, las Historias de Usuario correspondientes a las relaciones internas, se pueden observar desde la Tabla 2.13 hasta la Tabla 2.21.

Historia de Usuario No: 13	**Título:** Pedido a bodega
Descripción: Como usuario Solicitante debo ingresar un nuevo pedido a Bodega, el cual será enviado para ser aprobado o anulado por un usuario Supervisor.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 5 días	**Iteración asignada:** 2

Tabla 2.13 Historia de Usuario “Pedido a bodega”

Historia de Usuario No: 14	**Título:** Listar todos los pedidos a bodega
Descripción: Como usuario Supervisor debo poder ver una lista de todos los pedidos a bodega con el estado actual del pedido. Como usuario Solicitante debo poder ver una lista de mis pedidos a bodega con su estado actual.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 4 días	**Iteración asignada:** 2

Tabla 2.14 Historia de Usuario “Listar todos los pedidos a bodega”

Historia de Usuario No: 15	**Título:** Aprobar / Anular pedido a bodega
Descripción: Como usuario Supervisor debo aprobar o anular aquellos pedidos a bodega que se encuentren en un estado de “enviado”.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 2

Tabla 2.15 Historia de Usuario “Aprobar / Anular pedido a bodega”

Historia de Usuario No: 16	**Título:** Listar pedidos a bodega aprobados
Descripción: Como usuario Bodeguero debo poder ver una lista de todos los pedidos a bodega que tengan un estado de “aprobado”.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 2 días	**Iteración asignada:** 2

Tabla 2.16 Historia de Usuario “Listar pedidos a bodega aprobados”

Historia de Usuario No: 17	**Título:** Confirmar entrega de pedido a bodega
Descripción: Como usuario Bodeguero debo confirmar la entrega de aquellos pedidos a bodega que se encuentren en un estado de “aprobado”.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Medio (Alto / Medio / Bajo)
Tiempo estimado: 3 días	**Iteración asignada:** 2

Tabla 2.17 Historia de Usuario “Confirmar entrega de pedido a bodega”

Historia de Usuario No: 18	**Título:** Stock
Descripción: Como usuario Supervisor y como usuario Bodeguero debo poder ver una lista de productos y categorías con su stock actual.	
Prioridad: Media (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 4 días	**Iteración asignada:** 2

Tabla 2.18 Historia de Usuario “Stock”

Historia de Usuario No: 19	**Título:** Resumen de pedidos a bodega
Descripción: Como usuario Supervisor debo poder ver un resumen de pedidos a bodega, consolidados por departamento y luego por estado.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 4 días	**Iteración asignada:** 3

Tabla 2.19 Historia de Usuario "Resumen de pedidos a bodega"

Historia de Usuario No: 20	**Título:** Resumen de pedidos a proveedor y a bodega
Descripción: Como usuario Bodeguero debo poder ver un resumen de pedidos tanto a proveedor como a bodega, consolidados por estado.	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 3 días	**Iteración asignada:** 3

Tabla 2.20 Historia de Usuario "Resumen de pedidos a proveedor y a bodega"

Historia de Usuario No: 21	**Título:** Reportes
Descripción: Como usuario Supervisor debo poder ver los reportes necesarios para tomar decisiones y realizar los trámites administrativos pertinentes. (mínimo 10, máximo 20)	
Prioridad: Baja (Alta / Media / Baja)	**Riesgo:** Bajo (Alto / Medio / Bajo)
Tiempo estimado: 5 días	**Iteración asignada:** 3

Tabla 2.21 Historia de Usuario "Reportes"

2.1.2 PLAN DE ENTREGAS

Se procede a hacer un resumen de la estimación de las Historias de Usuario para poder planificar las entregas, para lo cual se van a considerar semanas de 5 días y días de 8 horas, que es lo que recomienda XP. La Tabla 2.22 muestra los tiempos estimados para cada Historia.

La estimación de las Historias de Usuario en un esfuerzo de equipo. El equipo estudia la Historia y considera cuánto tiempo tomaría implementarla. Podría ser que los miembros del equipo no lleguen a un acuerdo sobre la estimación. Algunos pueden pensar que la Historia es difícil y que tomaría un largo tiempo desarrollarla. Otros pueden pensar que es fácil y que tomaría poco tiempo su desarrollo. Por eso, es recomendable seguir la regla del más optimista. Es decir, si después de una discusión razonable el desacuerdo persiste, se escoge el tiempo estimado más corto. Hay que recordar que los tiempos estimados no son imperativos y que un par de malas estimaciones no representan un desastre. A lo que se apunta con esto es mejorar continuamente la habilidad del equipo para estimar tiempos.

Al escoger el tiempo estimado más optimista se consiguen dos cosas: se mantiene tensión en la estimación de los tiempos para que estos no sean exageradamente largos; y se mantiene tensión en el equipo para que todos los miembros aprendan a no ser tan optimistas, de esta forma, aquellos miembros del equipo, cuyo optimismo fue abrumador, aprenderán a moderar ese optimismo.

Una vez hecha la estimación de las Historias de Usuario, se procede a calcular el Esfuerzo de Desarrollo y de esta forma obtener el Tiempo Calendario que se asignará a cada iteración y a cada entrega, para lo cual consideramos lo siguiente:

- Esfuerzo por horas de desarrollo: 1 persona = 8 horas
- Esfuerzo por días de desarrollo: 1 persona = 5 días
- Esfuerzo por semanas de desarrollo: 1 persona = 1 semana

MÓDULO	HISTORIA DE USUARIO	No.	TIEMPO ESTIMADO		
			SEMANAS	DÍAS	HORAS
TAREAS DE ADMINISTRACIÓN	Autenticación de usuario	1	0.4	2	16
	Crear departamento	2	0.4	2	16
	Crear usuario	3	0.4	2	16
	Crear proveedor	4	0.4	2	16
	Crear categoría de productos	5	0.4	2	16
	Crear producto	6	0.4	2	16
	Perfil de usuario	7	0.6	3	24
	Comunicaciones internas	8	0.8	4	32
RELACIONES EXTERNAS	Pedido a proveedor	9	1	5	40
	Listar pedidos a proveedores	10	0.4	2	16
	Aprobar / Anular pedido a proveedor	11	0.4	2	16
	Confirmar entrega de pedido a proveedor	12	0.6	3	24
RELACIONES INTERNAS	Pedido a bodega	13	1	5	40
	Listar todos los pedidos a bodega	14	0.8	4	32
	Aprobar / Anular pedido a bodega	15	0.4	2	16
	Listar pedidos a bodega aprobados	16	0.4	2	16
	Confirmar entrega de pedido a bodega	17	0.6	3	24
	Stock	18	0.8	4	32
	Resumen de pedidos a bodega	19	0.8	4	32
	Resumen de pedidos a proveedor y a bodega	20	0.6	3	24
	Reportes	21	1	5	40
TIEMPO TOTAL ESTIMADO			12.6	63	504

Tabla 2.22 Estimación de las Historias de Usuario

Como en el equipo de desarrollo se cuenta con una sola persona, el esfuerzo de desarrollo y el tiempo calendario serán iguales, tal como se muestra en la Tabla 2.23. Hay que tomar en cuenta que la metodología utilizada estima los tiempos en tiempo ideal y como ya se mencionó anteriormente, el tiempo ideal considera el trabajo de una pareja de programadores, es decir, aunque se tuviera 2 personas en el equipo de desarrollo, el esfuerzo y el tiempo calendario seguirían siendo iguales si se sigue las recomendaciones de XP al pie de la letra.

Para escoger las Historias de Usuario que se van a desarrollar en una iteración u otra se puede tomar en cuenta el riesgo y la prioridad que tienen cada una de ellas. Si existe alguna Historia que conlleva un gran riesgo debería realizarse ésta primero, ya que sería inútil desarrollar las Historias de menor riesgo si más tarde nos vamos a encontrar en un punto que puede determinar el fin del proyecto al no poder implementar alguna de las Historias.

En cuanto a las prioridades, es obvio que se debería desarrollar primero aquellas Historias que tienen una alta prioridad para el usuario; por ello, para el presente proyecto se va a ubicar las Historias de Usuario en cada iteración tomando en cuenta solo las prioridades, ya que ninguna de ellas representa un gran riesgo para el desarrollo del sistema. Por esta razón es que podemos notar en la Tabla 2.23 que las Historias de Usuario pertenecientes a las Tares de Administración, exceptuando la Autenticación de usuario, se las realizará en la tercera y última iteración, ya que ninguna de ellas es indispensable desde un inicio porque los datos necesarios para el funcionamiento del sistema se los puede introducir directamente en la base de datos y de esta manera se tiene la capacidad de empezar a implementar otras Historias de Usuario que tienen una mayor prioridad.

En cuanto a las entregas, se ha dispuesto hacer una de ellas luego de las dos primeras iteraciones, puesto que ya se tendría implementado completamente todo lo concerniente a las relaciones internas y externas, lo cual ya representa algo de valor para el cliente. Y una segunda entrega se la realizará después de añadir lo relacionado a las tareas de administración, con lo cual se completaría el desarrollo del sistema al finalizar la tercera iteración.

MÓDULO	HISTORIA DE USUARIO	No.	ESFUERZO DE DESARROLLO			TIEMPO CALENDARIO			ITERACIÓN ASIGNADA			ENTREGA ASIGNADA	
			SEMANAS IDEALES	DÍAS IDEALES	HORAS IDEALES	SEMANAS	DÍAS	HORAS	1	2	3	1	2
TAREAS DE ADMINISTRACIÓN	Autenticación de usuario	1	0.4	2	16	0.4	2	16	X			X	
	Crear departamento	2	0.4	2	16	0.4	2	16			X		X
	Crear usuario	3	0.4	2	16	0.4	2	16			X		X
	Crear proveedor	4	0.4	2	16	0.4	2	16			X		X
	Crear categoría de productos	5	0.4	2	16	0.4	2	16			X		X
	Crear producto	6	0.4	2	16	0.4	2	16			X		X
	Perfil de usuario	7	0.6	3	24	0.6	3	24			X		X
	Comunicaciones internas	8	0.8	4	32	0.8	4	32			X		X
RELACIONES EXTERNAS	Pedido a proveedor	9	1	5	40	1	5	40	X			X	
	Listar pedidos a proveedores	10	0.4	2	16	0.4	2	16	X			X	
	Aprobar / Anular pedido a proveedor	11	0.4	2	16	0.4	2	16	X			X	
	Confirmar entrega de pedido a proveedor	12	0.6	3	24	0.6	3	24	X			X	
RELACIONES INTERNAS	Pedido a bodega	13	1	5	40	1	5	40		X		X	
	Listar todos los pedidos a bodega	14	0.8	4	32	0.8	4	32		X		X	
	Aprobar / Anular pedido a bodega	15	0.4	2	16	0.4	2	16		X		X	
	Listar pedidos a bodega aprobados	16	0.4	2	16	0.4	2	16		X		X	
	Confirmar entrega de pedido a bodega	17	0.6	3	24	0.6	3	24		X		X	
	Stock	18	0.8	4	32	0.8	4	32		X		X	
	Resumen de pedidos a bodega	19	0.8	4	32	0.8	4	32			X		X
	Resumen de pedidos a proveedor y a bodega	20	0.6	3	24	0.6	3	24			X		X
	Reportes	21	1	5	40	1	5	40			X		X
TIEMPO CALENDARIO ESTIMADO TOTAL (SEMANAS)									2.8	4	5.8	6.8	5.8

Tabla 2.23 Plan de entrega

2.2 DISEÑO

2.2.1 ARQUITECTURA DEL SISTEMA

La arquitectura que ha sido implementada para el sistema es una arquitectura en tres capas, con lo cual, las distintas lógicas de la aplicación han sido separadas y poseen una estructura bien definida, tal como se puede apreciar en la Figura 2.1.

Las capas que conforman el sistema son las siguientes:

- **Capa de datos (capa 1).** Es la capa donde residen los datos y la encargada de acceder a los mismos. El gestor de base de datos utilizado es MySQL, el cual recibe las solicitudes de almacenamiento o recuperación de datos desde la capa de negocio.

- **Capa de negocio (capa 2).** Aquí es donde se establece toda la lógica del negocio, ya que se definen todas las reglas que deben cumplirse para responder adecuadamente a las peticiones del usuario. Es una capa intermedia entre la capa de datos y la capa de presentación, puesto que se relaciona con ambas. La aplicación reside en un servidor web Apache, el cual tiene instalado un intérprete del lenguaje de programación PHP, debido a que la aplicación ha sido completamente desarrollada con la ayuda del framework CakePHP con el objetivo de implementar el patrón MVC (Modelo, Vista, Controlador).

- **Capa de presentación (capa 3).** Esta capa se encarga de proveer una interfaz entre el sistema y el usuario. Básicamente, se responsabiliza de que se le comunique información al usuario por parte del sistema y viceversa, manteniendo una comunicación exclusiva con la capa de negocio. Para ello, se utiliza un navegador web (Mozilla Firefox, por ejemplo) que interpreta el código html generado por la capa de negocio y presenta los resultados de forma adecuada. Además del código html, también se ejecuta en el lado del cliente todos los scripts JQuery (lenguaje javascript), los cuales se utilizan para una mejor presentación de la interfaz de la aplicación y optimizar su funcionalidad.

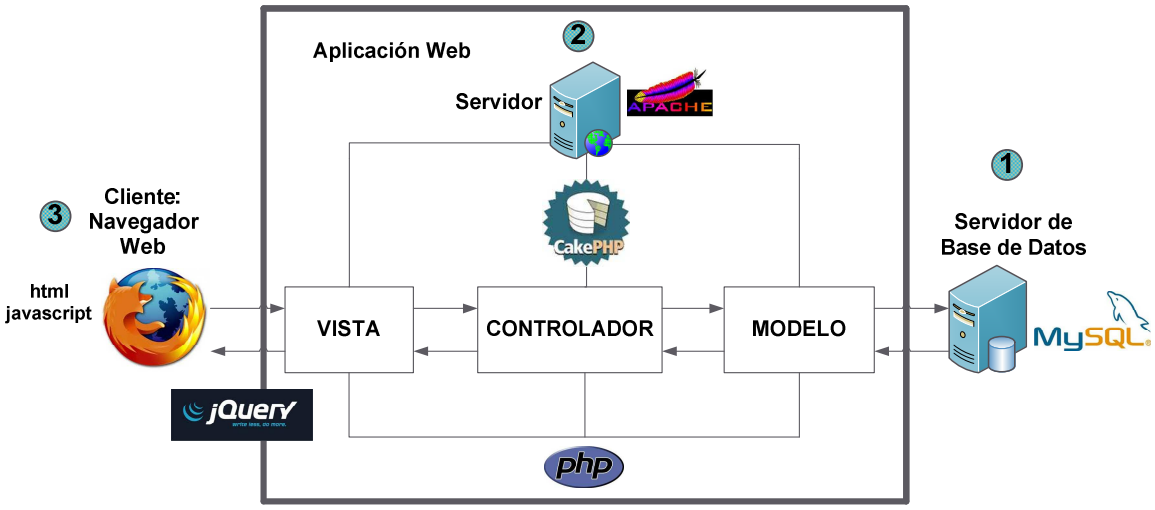


Figura 2.1 Arquitectura del sistema

2.2.2 METÁFORA DEL SISTEMA

En la programación extrema el sistema es definido mediante una metáfora o un conjunto de metáforas compartidas por el cliente y el equipo de desarrollo. Una metáfora es una historia que describe cómo debería funcionar el sistema, es decir un conjunto de nombres que actúen como vocabulario para hablar sobre el dominio del problema, ayudando a la nomenclatura de clases y métodos del sistema.

Por lo tanto, la metáfora que se ha definido para el Sistema de Inventarios es la siguiente:

El Sistema de Inventarios para las Instituciones Públicas del Ecuador manejará dos aspectos primordiales, a los cuales se los ha denominado Movimientos Externos y Movimientos Internos:

- Los Movimientos Externos se refieren a la interacción entre la Unidad Administrativa encargada del manejo de los bienes de la Institución Pública y los proveedores.

- Los Movimientos Internos hacen referencia a la interacción entre la Unidad Administrativa encargada del manejo de los bienes de la Institución Pública y otras dependencias de la misma institución.

A más de las cuestiones anteriores, el Sistema también ejecuta actividades correspondientes a la manipulación de los datos base, tales como usuarios, proveedores y productos, a las que en su conjunto se las ha denominado Administración del Sistema, siendo éste el tercer módulo que forma parte del Sistema.

Además, el Sistema deberá manejar los siguientes perfiles de usuario:

- **Administrador –** Maneja los datos básicos del sistema.
- **Supervisor –** Aprueba o rechaza pedidos hechos a proveedores y a bodega.
- **Solicitante –** Realiza pedidos a bodega.
- **Bodeguero –** Ingresa y controla la entrega de pedidos a proveedores y pedidos a bodega.

También, los pedidos que se realizan, tanto a bodega como a proveedores, pueden tener los siguientes estados:

- **Enviado –** Cuando el pedido ha sido ingresado al sistema.
- **Aprobado –** Cuando el pedido ha sido aprobado para su realización.
- **Anulado –** Cuando el pedido ingresado ha sido negado y no se puede hacer efectivo.
- **Entregado –** Cuando los bienes de un pedido has sido entregados físicamente.
- **Rechazado –** Este estado puede ser otorgado solo a pedidos a proveedor y se lo hace cuando ha habido alguna anomalía en la recepción de los productos en bodega.

2.2.3 MODELOS DE DATOS

A nivel de los datos, se va a hacer una diferenciación en el nombramiento de los pedidos, de esta forma, a los pedidos realizados a proveedores, se los va a denominar simplemente “pedidos” y a los pedidos realizados internamente a

bodega, se los denominará “encargos”. Esto no se verá reflejado en la capa de presentación, es solo una convención utilizada a nivel de datos.

Algo que se puede notar en ambos modelos de datos, tanto en el conceptual (Figura 2.2), como en el físico (Figura 2.3), es que todas las entidades o tablas tienen un atributo llamado *id* y otro llamado *name* (exceptuando las tablas *notas*, *detalle_pedidos* y *detalle_encargos*). Esto se debe a que se están siguiendo las recomendaciones del framework CakePHP para poder generar código fuente automáticamente a partir del modelo de datos, lo cual no es obligatorio, pero sí de gran utilidad si se quiere generar rápidamente un “esqueleto” de la aplicación.

Las nuevas tablas que se generan en el modelo físico de datos se deben a que existen relaciones de cardinalidad de “muchos a muchos” entre algunas entidades; por ejemplo, la relación entre la entidad *pedidos* y la entidad *productos* en el modelo conceptual, da como resultado una nueva tabla denominada *detalle_pedidos*. También existen atributos compuestos que dan lugar a la formación de nuevas tablas en el modelo físico, todos estos atributos tienen un nombre que termina con el prefijo “id”, como es el caso del atributo *departamento_id* de la entidad *users*, por ejemplo; lo cual da como resultado la aparición de la tabla *departamentos* en el modelo físico.

2.2.3.1 Modelo Conceptual

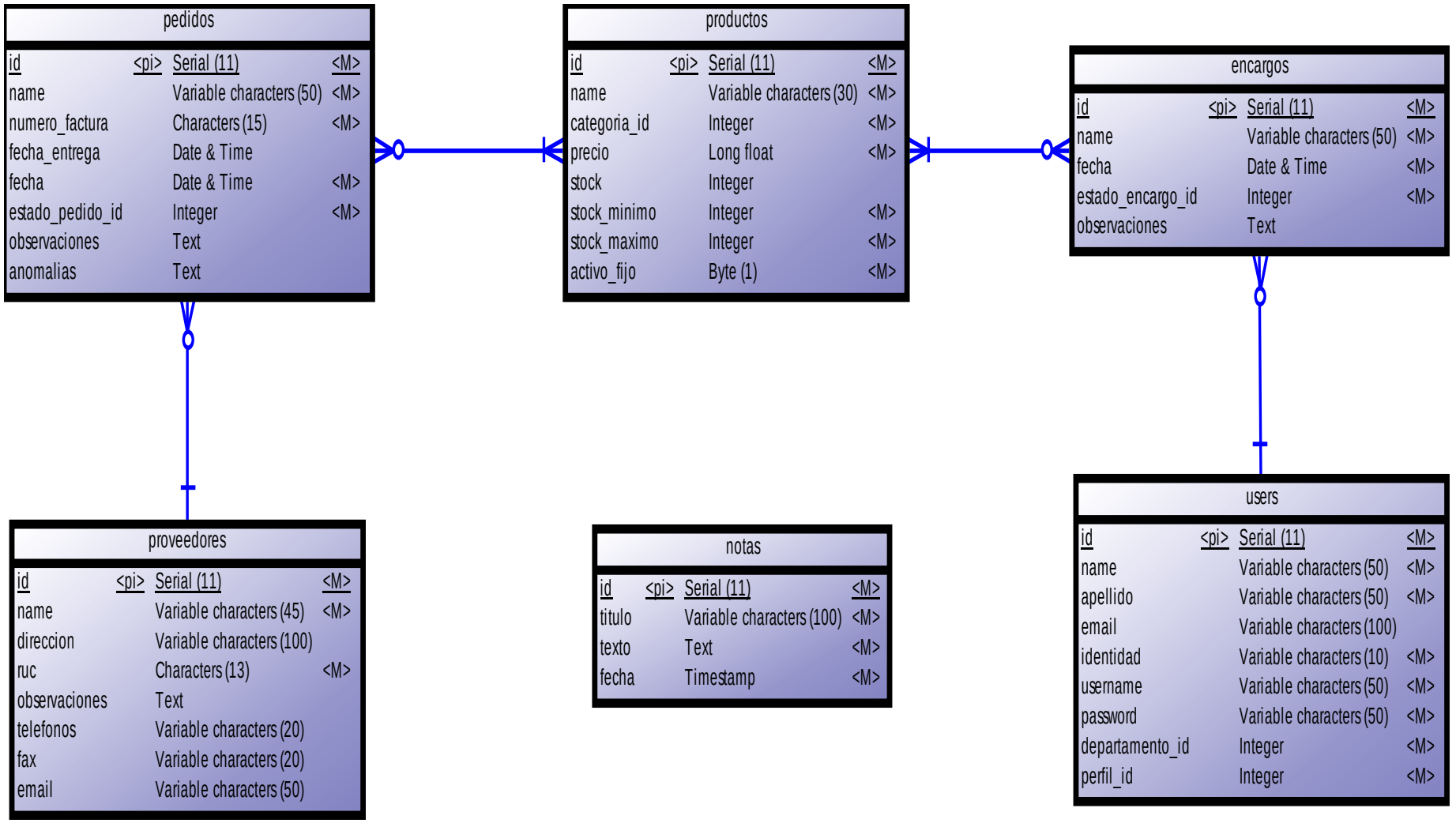


Figura 2.2 Modelo Conceptual de Datos

2.2.3.2 Modelo Físico

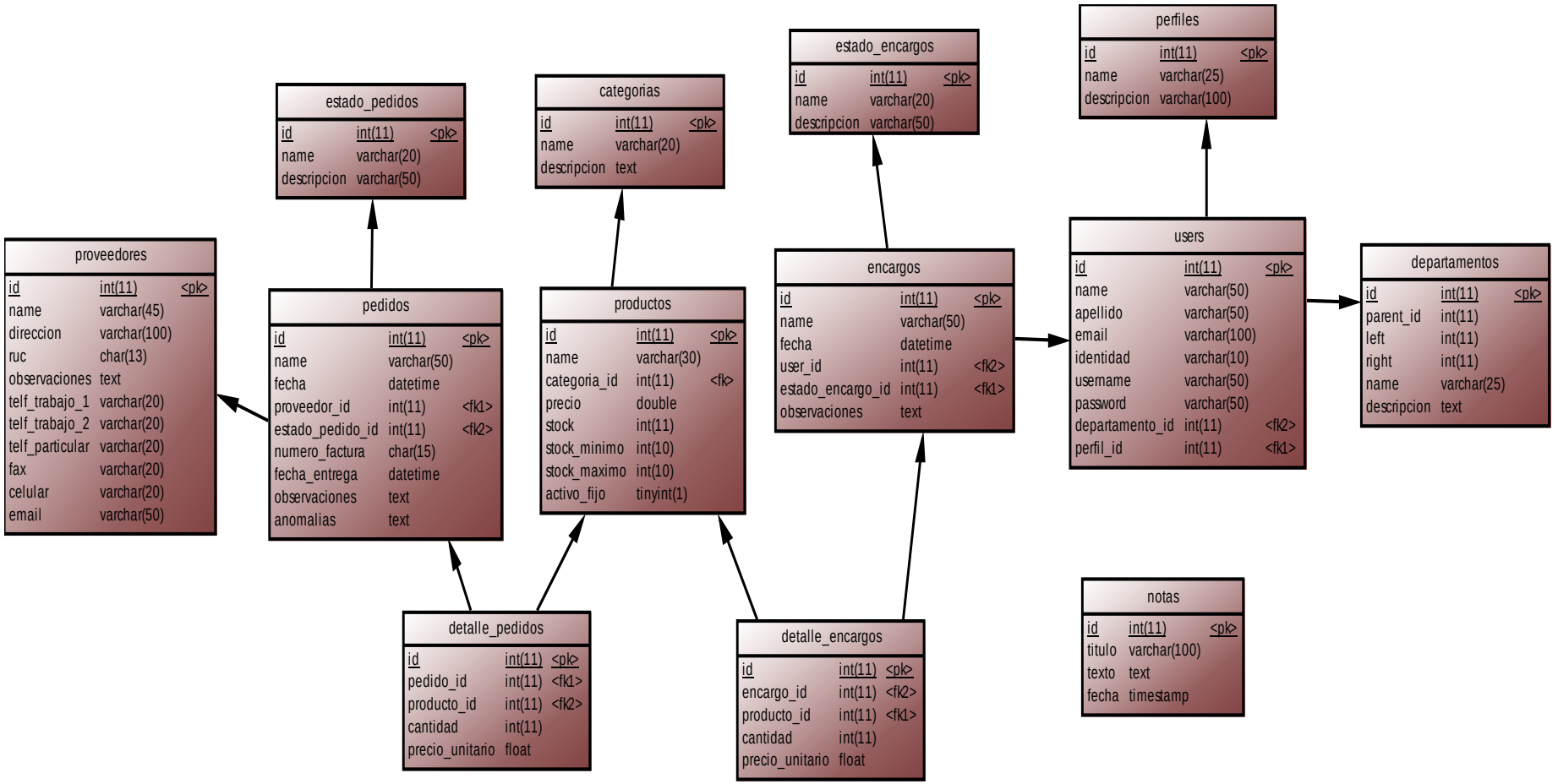


Figura 2.3 Modelo Físico de Datos[15]

[15] También existen tablas que son usadas con propósitos de auditoría, las cuales se las puede apreciar en el Anexo 1.

2.2.4 DISEÑO DE INTERFACES

Para utilizar el sistema, todo usuario debe ingresar con sus respectivas credenciales; para ello, se dispondrá de un pequeño formulario como el que se muestra en la Figura 2.4

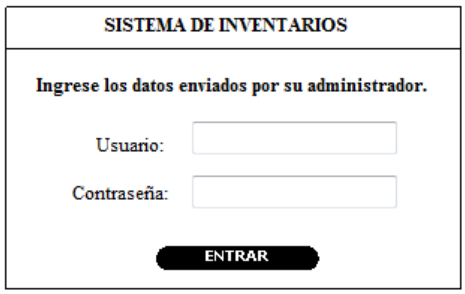


Figura 2.4 Formulario para autenticación de Usuario

Una vez que el usuario ingresa al sistema, se presentará una pantalla como la que se puede ver en la Figura 2.5. Esta es una plantilla que variará de acuerdo al perfil del usuario, es decir, las pestañas y los accesos directos serán diferentes dependiendo de los procesos que puede manejar cada tipo de usuario. En la zona para formularios y reportes aparecerá lo relacionado a cada pestaña.

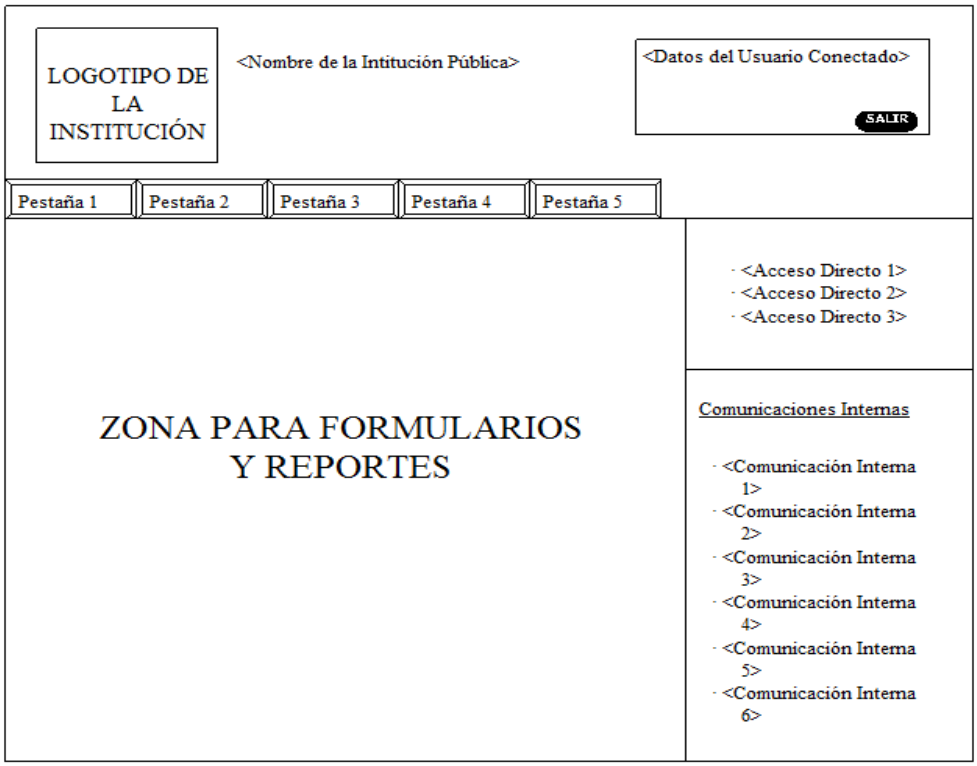


Figura 2.5 Plantilla del entorno del sistema

A continuación, a manera de ejemplo, se presentan dos formularios elementales en el manejo de inventarios.

El primero, Figura 2.6, es un formulario para ingresar pedidos, el cual se lo puede utilizar tanto para las relaciones internas como las externas.

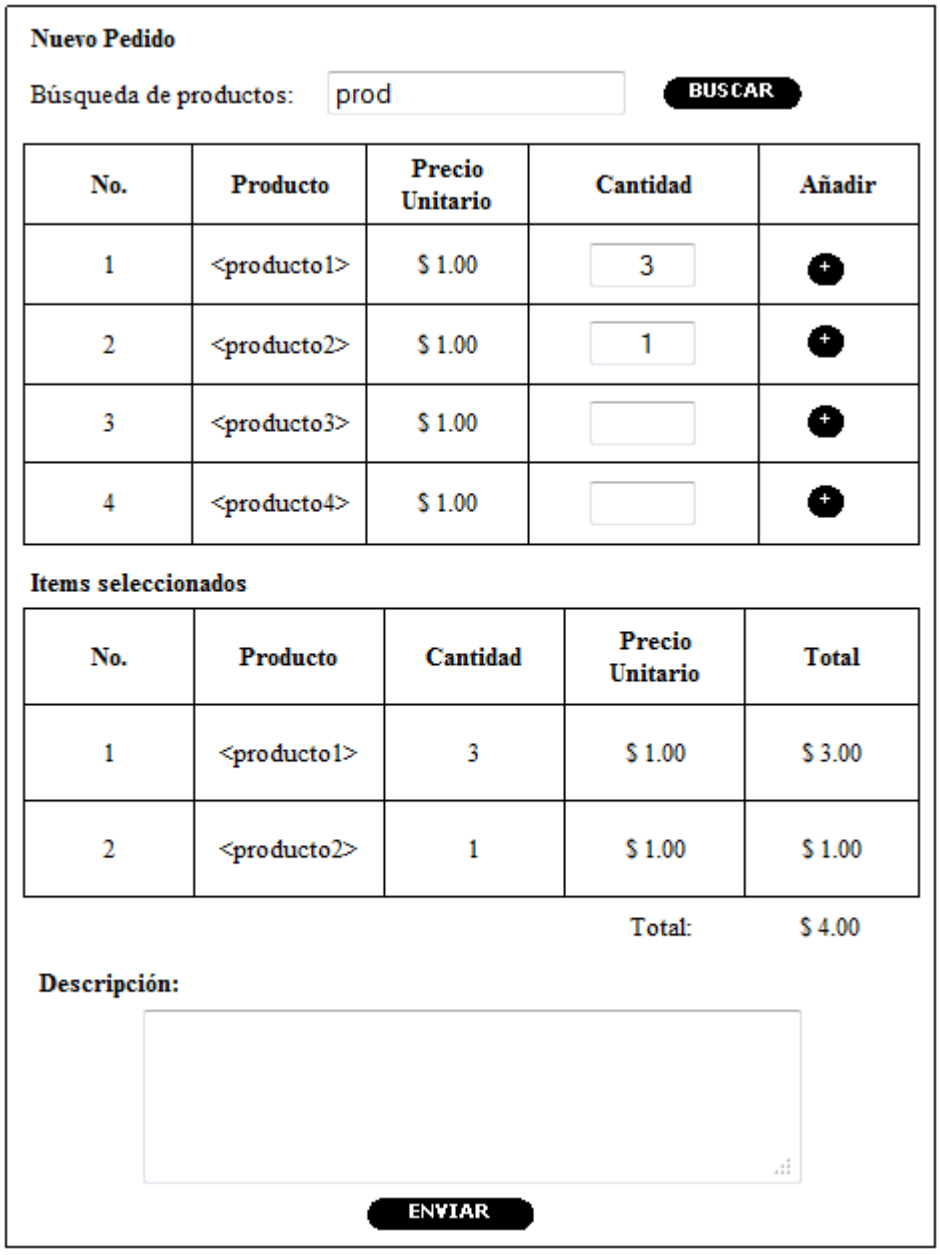


Figura 2.6 Formulario para ingreso de pedido

Y el segundo, Figura 2.7, es un formulario que lo utiliza el usuario con el perfil adecuado, para aprobar o anular los pedidos enviados. De igual manera, este formulario se lo puede utilizar en los dos tipos de relaciones: internas y externas.

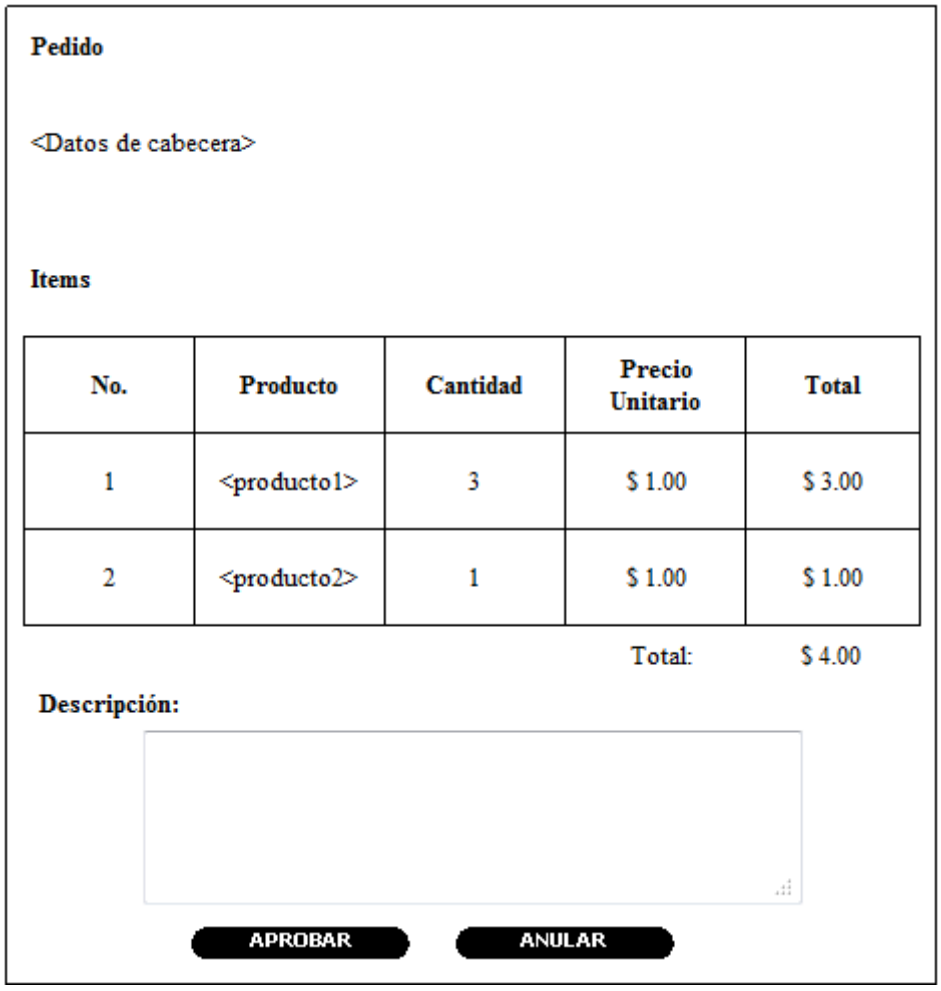


Figura 2.7 Formulario para aprobación o anulación de pedido

2.2.5 MAPA DEL SITIO

La estructura del sitio dependerá del perfil de cada usuario, ya que cada uno maneja diferentes procesos dentro del Sistema de Inventarios. En las figuras 2.8, 2.9, 2.10 y 2.11 se ilustra la navegabilidad para cada uno de los perfiles de usuario involucrados en el sistema.

2.2.5.1 Navegabilidad para el Usuario Administrador

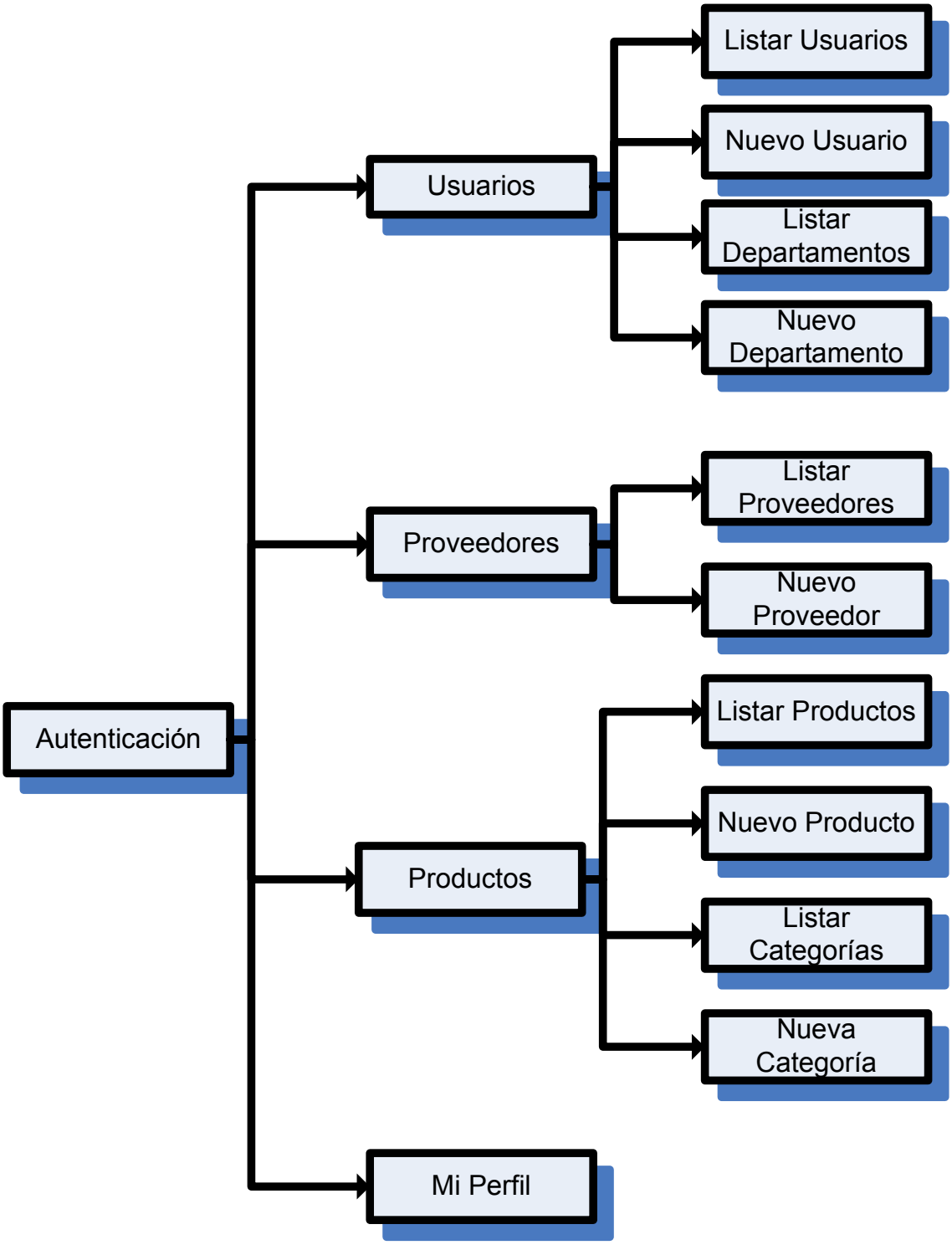


Figura 2.8 Mapa del sitio para el usuario Administrador

2.2.5.2 Navegabilidad para el Usuario Supervisor

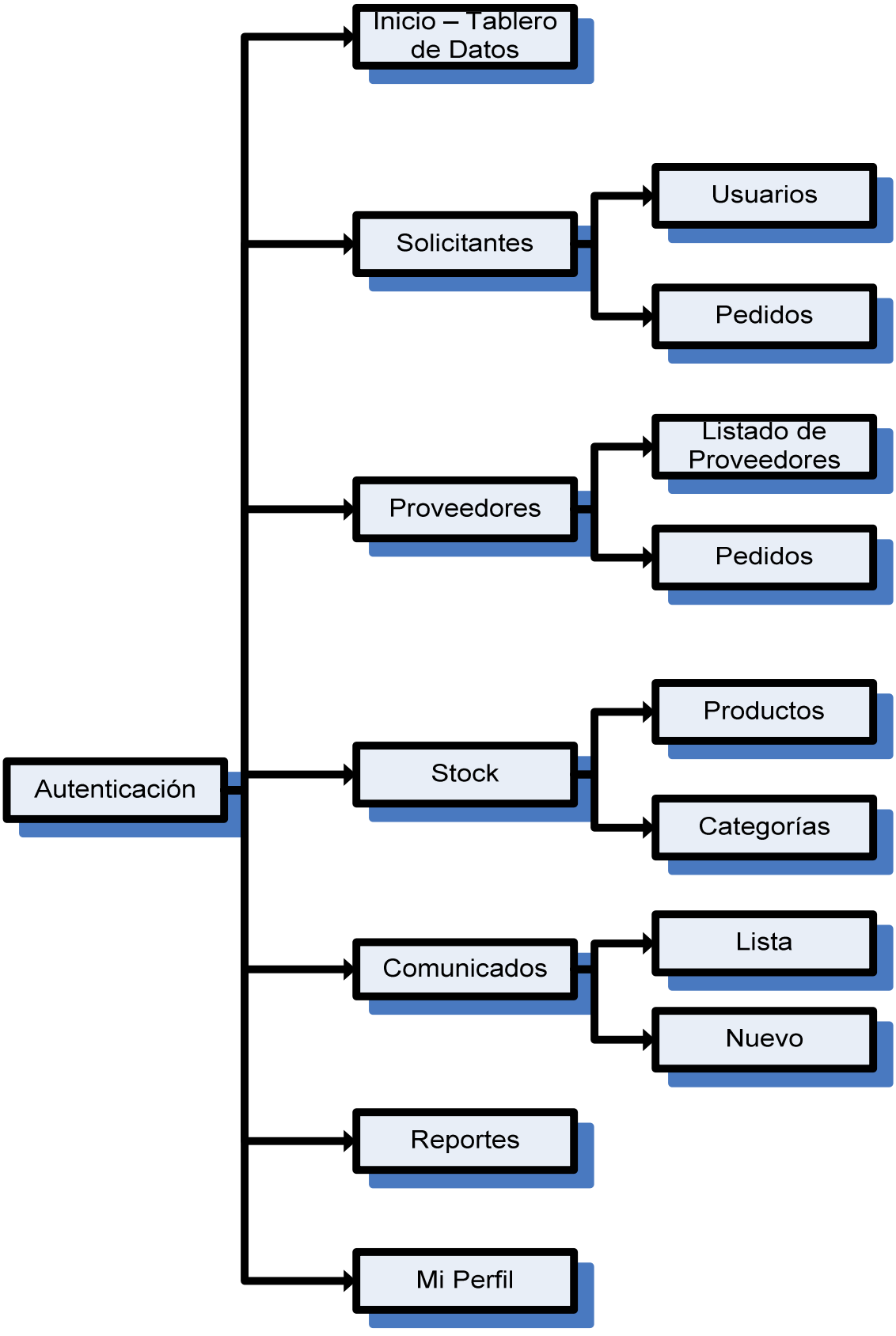


Figura 2.9 Mapa del sitio para el usuario Supervisor

2.2.5.3 Navegabilidad para el Usuario Bodeguero

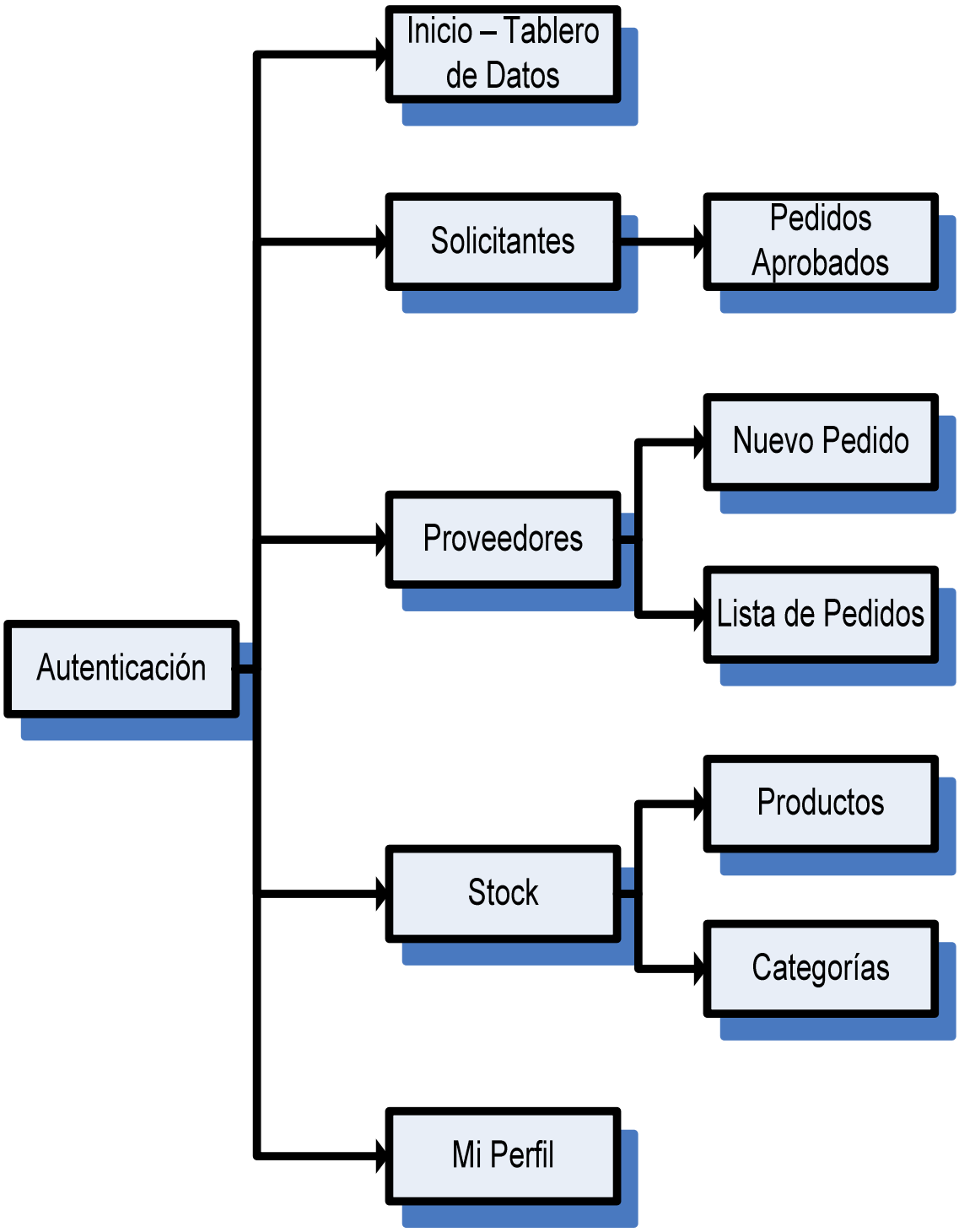


Figura 2.10 Mapa del sitio para el usuario Bodeguero

2.2.5.4 Navegabilidad para el Usuario Solicitante

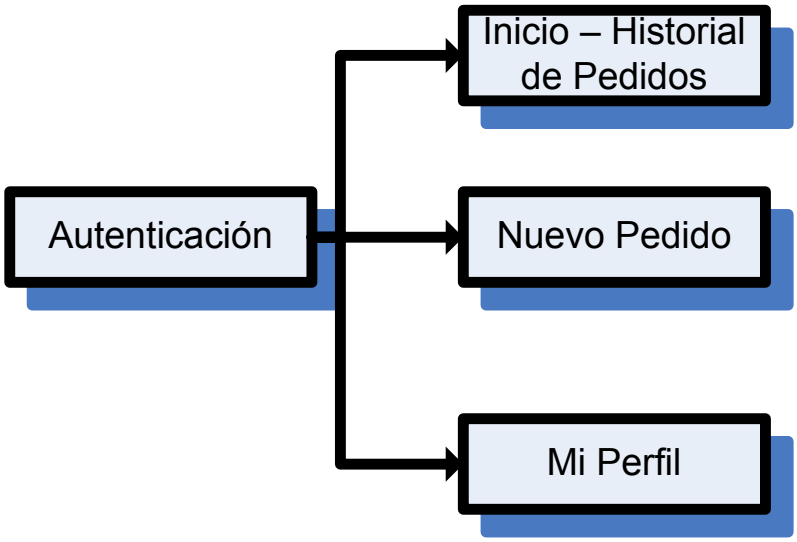


Figura 2.11 Mapa del sitio para el usuario Solicitante

2.3 IMPLEMENTACIÓN

2.3.1 ESTÁNDARES EN LA IMPLEMENTACIÓN

Los estándares son fundamentales cuando los programadores cambian de pareja o hacen *refactoring* del código de otros, por tal motivo hay que aspirar conseguir un código con el mismo estilo, homogéneo y legible. De hecho, al utilizar un framework de desarrollo como CakePHP, los estándares no se ciñen solamente al código fuente de la aplicación, sino que se debe seguir una serie de convenciones que comienzan desde la capa de datos, llegando hasta el nombramiento de ficheros y su ubicación dentro del sistema de archivos. Las convenciones de nombramiento en CakePHP tienen reglas específicas que le posibilitan ensamblar las diferentes piezas de la aplicación sin la necesidad de escribir mucho código; de esta forma, cuando el framework busca algún recurso solicitado, este puede ser encontrado y ejecutado rápidamente sin presentar ningún mensaje de error.

Las convenciones en CakePHP comienzan con la base de datos. Si se diseña en primer lugar la forma en la cual la aplicación almacenará y trabajará con los datos, entonces es más fácil aplicar las convenciones al resto de elementos, por lo que si en algún momento, se obtienen errores en la ejecución de la aplicación, lo más probable es que se deberá a alguna disconformidad con las reglas que establece

el framework. A continuación se detallan algunas de estas reglas, las cuales se originaron de las mejores prácticas para el nombramiento de los distintos elementos de una aplicación.

2.3.1.1 Nombramiento de Controladores

El nombre de cada controlador debe ajustarse al nombre de una tabla de la base de datos. Tanto los nombres de las tablas como los de los controladores deben estar en minúsculas y en plural. Por ejemplo, para la tabla *productos* de la base de datos, el archivo del controlador debe tomar el mismo nombre de la tabla seguido de un subguión más la palabra *controller* y la extensión del archivo *php*. Entonces, el nombre del archivo del controlador para la tabla *productos* sería *productos_controller.php*, el cual debe ser ubicado en la carpeta *app/controllers*.

Dentro del archivo, lo primero que se debe hacer es heredar de la clase *AppController*, la cual es llamada por el framework cuando la aplicación es ejecutada. Para ello, se debe crear una nueva clase cuyo nombre es igual al de la tabla de la base de datos, pero comenzando con mayúscula y seguido de la palabra *Controller*. Es decir, la clase quedaría de la siguiente forma:

```
<?

class ProductosController extends AppController {

}

?>
```

Si se siguen estas convenciones en todos los controladores, CakePHP será capaz de despachar una petición al controlador correcto y ejecutar la acción correcta.

2.3.1.2 Nombramiento de Modelos

Los modelos se nombran igual que la tabla de la base de datos, pero en singular, ya que los modelos representan la interacción con tan solo una instancia de la tabla a la vez. De esta forma, el nombre del archivo del modelo para la tabla *productos* sería *producto.php* y se ubicará en la ruta *app/models*. Hay que notar

que en los modelos no se añade un subguión, ni la palabra *model* en el nombre del archivo.

La estructura de las clases que se debe crear para los modelos es similar a la de los controladores. Hay que crear una nueva clase que hereda de la clase *AppModel*, su nombre debe comenzar con mayúscula, debe estar en forma singular (en concordancia con el nombre de la respectiva tabla de la base de datos) y nada más. Así, para la tabla *productos*, la clase del modelo sería:

```
<?

class Producto extends AppModel {

}

?>
```

2.3.1.3 Nombramiento de Vistas

Cuando el controlador genera vistas, CakePHP automáticamente buscará un archivo con el mismo nombre que la acción. Las vistas corresponden a acciones contenidas en el script del controlador y son almacenadas en una carpeta que se creará después de crear el controlador. El primer paso para crear una vista a ser usada como salida para un script de controlador es crear la carpeta en *app/views* que se ajuste al nombre del controlador. Los nombres de las carpetas de las vistas están en minúsculas y en plural, tal como la convención usada para el nombramiento de tablas de la base de datos. Para el archivo *productos_controller.php*, se deberá crear la carpeta *app/views/productos.*

Los archivos de las vistas se emparejan con acciones en el controlador, por eso éstos deben ser nombrados adecuadamente. Cuando una acción del controlador es llamada, el framework automáticamente busca la vista correspondiente que tiene un esquema específico de nombramiento. Por ejemplo, en la clase *ProductosController*, se podría tener una acción que sirva para editar los datos de un producto. Entonces, se crea una acción en el archivo *productos_controller.php* llamada *edit*. En esta acción (la cual es una función PHP en el script del controlador), se tienen algunas variables que se envían a la vista siguiendo la estructura MVC. CakePHP automáticamente buscará una vista llamada

app/views/productos/edit.ctp (*.ctp* es la extensión por defecto para todas las vistas en CakePHP, no *.html*, ni *.php*).

2.3.1.4 Resumen

En la Tabla 2.24 se tiene una síntesis de las convenciones utilizadas en el nombramiento y ubicación de los diferentes elementos que componen la aplicación, tomando como ejemplo la tabla *productos* de la base de datos.

Tipo	Nombre del archivo	Nombre de la clase	Directorio donde se almacena
Modelo	producto.php	Producto	app/models
Controlador	productos_controller.php	ProductosController	app/controllers
Vista	{concuerda con el nombre de la acción en el controlador}.ctp	----------	app/views/productos

Tabla 2.24 Elementos MVC y convenciones de nombramiento

Si hay la necesidad de nombrar una tabla de la base de datos con más de una palabra, se debe separar cada palabra con un subguión, como en el caso de la tabla *detalle_pedidos* y luego se siguen las mismas reglas para el resto de elementos, como se puede observar en la Tabla 2.25.

Tipo	Nombre del archivo	Nombre de la clase	Directorio donde se almacena
Modelo	detalle_pedido.php	DetallePedido	app/models
Controlador	detalle_pedidos_controller.php	DetallePedidosController	app/controllers
Vista	edit.ctp	----------	app/views/detalle_pedidos

Tabla 2.25 Convenciones de nombramiento para tablas con más de una palabra

2.3.2 DICCIONARIO DE DATOS

Desde la Tabla 2.26 hasta la Tabla 2.38 se muestra el diccionario de datos del sistema. Todas las tablas contienen 8 columnas que detallan las características de cada uno de los atributos. Estas columnas son las siguientes:

- **Atributo –** Se refiere al nombre del atributo en la base de datos.
- **Descripción –** Es la explicación de lo que representa el atributo.
- **Tipo de dato –** Es la especificación del tipo de dato que puede almacenar el atributo.

- **Tamaño –** Es la longitud máxima en bytes que puede tener el atributo.
- **PK –** Es la sigla de Primary Key y especifica si el atributo es o no una clave primaria. Si se marca con un punto, significa que el atributo está definido como tal.
- **FK –** Es la sigla de Foreign Key y especifica si el atributo es o no una clave foránea. Si se marca con un punto, significa que el atributo está definido como tal; aunque las relaciones existentes entre las tablas de la base de datos en realidad no se las implementa en el nivel mismo de los datos, sino a nivel de la aplicación utilizando la funcionalidad que provee el framework.
- **NN –** Es la sigla de No Nulo y define si el atributo debe o no tener obligatoriamente un valor. Si se marca con un punto, significa que el atributo nunca puede tener un valor nulo.
- **Único –** Define si el valor del atributo puede o no repetirse en las diferentes instancias. Si se marca con un punto, significa que el valor del atributo debe ser único.

Tabla: categorias							
Descripción: Guarda los datos de las categorías de productos.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar la categoría.	integer	11	●		●	●
name	Es el nombre de la categoría.	varchar	20				●
descripcion	Es la descripción de la categoría.	text					

Tabla 2.26 Estructura de la tabla *categorías*

Tabla: departamentos							
Descripción: Guarda los datos de los departamentos de la entidad pública.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar el departamento.	integer	11	●		●	●
parent_id	Es un atributo utilizado por el framework para reflejar en la capa de presentación la estructura de árbol que se necesita para los departamentos.	integer	11				
left	IDEM	integer	11				
right	IDEM	integer	11				
name	Es el nombre del departamento.	varchar	25				●
descripcion	Es la descripción del departamento.	text					

Tabla 2.27 Estructura de la tabla *departamentos*

Tabla: detalle_encargos							
Descripción: Guarda los datos de los detalles de pedidos hechos a bodega.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un ítem del pedido.	integer	11	●		●	●

encargo_id	Es el identificador del pedido al que pertenece el detalle.	integer	11		●	●	
producto_id	Es el identificador de un producto incluido en el pedido.	integer	11		●	●	
cantidad	Es el número de unidades de un producto.	integer	11			●	
precio_unitario	Es el precio de cada unidad del producto en dólares	float				●	

Tabla 2.28 Estructura de la tabla *detalle_encargos*

Tabla: detalle_pedidos							
Descripción: Guarda los datos de los detalles de pedidos hechos a proveedores.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un ítem del pedido.	integer	11	●		●	●
pedido_id	Es el identificador del pedido al que pertenece el detalle.	integer	11		●		
producto_id	Es el identificador de un producto incluido en el pedido.	integer	11		●		
cantidad	Es el número de unidades de un producto.	integer	11				
precio_unitario	Es el precio de cada	float					

	unidad del producto en dólares						

Tabla 2.29 Estructura de la tabla *detalle_pedidos*

Tabla: encargos							
Descripción: Guarda los datos de pedidos hechos a bodega.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar el pedido.	integer	11	●		●	●
name	Es un título que sirve para identificar el pedido.	varchar	50				
fecha	Es la fecha en la que se ingresa el pedido al sistema.	datetime				●	
user_id	Es el identificador del usuario que hace el pedido.	integer	11		●		
estado_encargo_id	Es el identificador del estado actual del pedido.	integer	11		●		
observaciones	Son las observaciones hechas al	text					

	momento de aprobar o anular un pedido.						

Tabla 2.30 Estructura de la tabla *encargos*

Tabla: estado_encargos							
Descripción: Guarda los datos de los diferentes estados que pueden tener los pedidos hechos a bodega.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar el estado del pedido.	integer	11	●		●	●
name	Es el nombre del estado del pedido.	varchar	20				
descripcion	Es la descripción del estado del pedido.	varchar	50				

Tabla 2.31 Estructura de la tabla *estado_encargos*

Tabla: estado_pedidos							
Descripción: Guarda los datos de los diferentes estados que pueden tener los pedidos hechos a proveedores.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar el estado del pedido.	integer	11	●		●	●
name	Es el nombre del estado del pedido.	varchar	20				

descripcion	Es la descripción del estado del pedido.	varchar	50				

Tabla 2.32 Estructura de la tabla *estado_pedidos*

Tabla: notas							
Descripción: Guarda los datos de las comunicaciones enviadas internamente parar todos los usuarios del sistema.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar la comunicación interna.	integer	11	●		●	●
titulo	Es el asunto de la comunicación.	varchar	100			●	
texto	Es el cuerpo de la comunicación.	text				●	
fecha	Es la fecha de publicación de la comunicación.	datetime				●	

Tabla 2.33 Estructura de la tabla *notas*

Tabla: pedidos							
Descripción: Guarda los datos de pedidos hechos a proveedores.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un pedido a proveedor.	integer	11	●		●	●
name	Es un título que sirve para	varchar	50				

	identificar el pedido.						
fecha	Es la fecha en la que se ingresa el pedido al sistema.	datetime					
proveedor_id	Es el identificador del proveedor del pedido.	integer	11		●	●	◕[16]
estado_pedido_id	Es el identificador del estado actual del pedido.	integer	11		●		
numero_factura	Es el número de la factura correspondiente al pedido.	char	15				◕[17]
fecha_entrega	Es la fecha en la que el pedido ingresa físicamente a bodega. También se refiere a la fecha de rechazo en caso de haber anomalías.	datetime					
observaciones	Son las observaciones	text					

[16] Se creó una restricción llamada *factura_unica_por_proveedor*, la cual involucra que la combinación de los campos *proveedor_id* y *numero_factura* debe ser única.
[17] IDEM

	hechas al momento de aprobar o anular un pedido.						
anomalias	Es el detalle de los problemas que se presentaron al momento de entregar el pedido y la razón por la que se lo rechazó.	text					

Tabla 2.34 Estructura de la tabla *pedidos*

Tabla: perfiles							
Descripción: Guarda los datos de los diferentes tipos de usuarios que pueden ingresar al sistema.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar el perfil de usuario.	integer	11	●		●	●
name	Es el nombre del perfil de usuario.	varchar	25				
descripcion	Es la descripción del perfil de usuario.	varchar	100				

Tabla 2.35 Estructura de la tabla *perfiles*

Tabla: productos							
Descripción: Guarda los datos de los productos de bodega.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un producto.	integer	11	●		●	●
name	Es el nombre del producto.	varchar	30			●	●
categoria_id	Es el identificador de la categoría a la que pertenece le producto.	integer	11		●	●	
precio	Es el precio en dólares del producto.	double				●	
stock	Es el número actual de unidades del producto en bodega.	integer	11				
stock_minimo	Es el número mínimo de unidades del producto que deben existir en bodega.	integer	10			●	
stock_maximo	Es el número máximo de unidades del producto que deben existir en bodega.	integer	10			●	
activo_fijo	Este atributo permite conocer si el bien existente se trata de	boolean	1			●	

	un activo fijo. (Un valor de 1 significa que lo es y un valor de 0 significa que no)						

Tabla 2.36 Estructura de la tabla *productos*

Tabla: proveedores							
Descripción: Guarda los datos de los proveedores.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un proveedor.	integer	11	●		●	●
name	Es el nombre del proveedor.	varchar	45				
direccion	Es la dirección del domicilio comercial del proveedor.	varchar	100				
ruc	Es el número de Registro Único de Contribuyentes.	char	13				●
observaciones	Es cualquier tipo de observación hecha al proveedor.	text					
telf_trabajo_1	Es un número telefónico comercial del proveedor.	varchar	20				
telf_trabajo_2	Es un número telefónico comercial del proveedor.	varchar	20				
telf_particular	Es un número	varchar	20				

	telefónico fijo de un representante de la empresa proveedora.						
fax	Es el número de fax del proveedor.	varchar	20				
celular	Es el número telefónico móvil de un representante de la empresa proveedora.	varchar	20				
email	Es la dirección de correo electrónico del proveedor.	varchar	50				

Tabla 2.37 Estructura de la tabla *proveedores*

Tabla: users							
Descripción: Guarda los datos de los usuarios del sistema.							
Atributo	**Descripción**	**Tipo de dato**	**Tamaño**	**PK**	**FK**	**NN**	**Único**
id	Es un número secuencial que sirve para identificar un usuario del sistema.	integer	11	●		●	●
name	Es el nombre o nombres reales del usuario.	varchar	50			●	
apellido	Es el apellido o apellidos reales del usuario.	varchar	50			●	
email	Es la dirección de	varchar	100				

	correo electrónico del usuario.						
identidad	Es el número del documento de identificación del usuario.	varchar	10			●	◕[18]
username	Es el nombre con el que se identifica al usuario dentro del sistema.	varchar	50			●	●
password	Es la contraseña del usuario encriptada.	varchar	50			●	
departamento_id	Es el identificador del departamento al cual pertenece el usuario.	integer	11		●	●	◕[19]
perfil_id	Es el identificador del perfil del usuario.	integer	11		●	●	◕[20]

Tabla 2.38 Estructura de la tabla *users*

2.3.3 DIAGRAMAS DE CLASES

Utilizando el comando *cake bake* que proporciona el framework, las clases del sistema se pueden generar automáticamente a partir del modelo de datos. Para esto se debió seguir rigurosamente las convenciones de nombramiento que dicta CakePHP.

Para cada una de las tablas del modelo de datos, se crearán dos clases: una para el manejo precisamente de los datos (modelo) y otra para el manejo de la lógica del negocio (controlador). Así, por ejemplo, tenemos que para la tabla *productos*

[18] Se creó una restricción llamada *identidad_perfil_departamento_combinacion_unica*, la cual involucra que la combinación de los campos *identidad*, *departamento_id* y *perfil_id* debe ser única.
[19] IDEM
[20] IDEM

se generan las clases *Producto* y *ProductosController*, que pertenecen a los modelos (Figura 2.12) y a los controladores (Figura 2.13), respectivamente. Esto se puede hacer no solo con tablas, sino también con vistas, como es el caso de la vista *v_solicitantes*.

Las clases generadas para los controladores, en primera instancia, contienen algunos métodos básicos, tales como:

- **index –** que es la acción que se ejecuta por defecto cuando se llama al controlador al cual pertenece.
- **view –** que permite visualizar un registro de la base de datos,
- **add –** el cual permite añadir un registro en la base de datos,
- **edit –** que sirve para modificar un registro de la base de datos, y
- **delete –** que permite eliminar un registro de la base de datos.

Estos métodos generados automáticamente, pueden ser modificados más adelante para ajustarlos a necesidades específicas, o bien se los puede obviar para crear otros métodos personalizados.

En cuanto a las clases referentes a los modelos, se puede observar que básicamente tienen atributos mediante los cuales se definen las diferentes relaciones de cardinalidad que se tiene entre las tablas de la base de datos y también se tiene un atributo que interviene en la validación de los datos.

Como se observa en las figuras 2.12 y 2.13, todas las clases, tanto en los controladores como en los modelos, heredan de otras clases que constituyen el núcleo de CakePHP. En las clases de los controladores, se puede notar que heredan de la clase AppController, ésta a su vez hereda de la clase Controller y ésta por último hereda de la clase Object; algo similar se puede ver en las clases de los modelos. Todas estas clases del núcleo tienen funcionalidades básicas que un controlador o un modelo, dependiendo del caso, necesitan ejecutar. Como resultado de heredar de estas clases, todos los controladores o modelos tendrán estas funcionalidades y no necesitarán definirlas nuevamente.

2.3.3.1 Diagrama de Clases para los Modelos

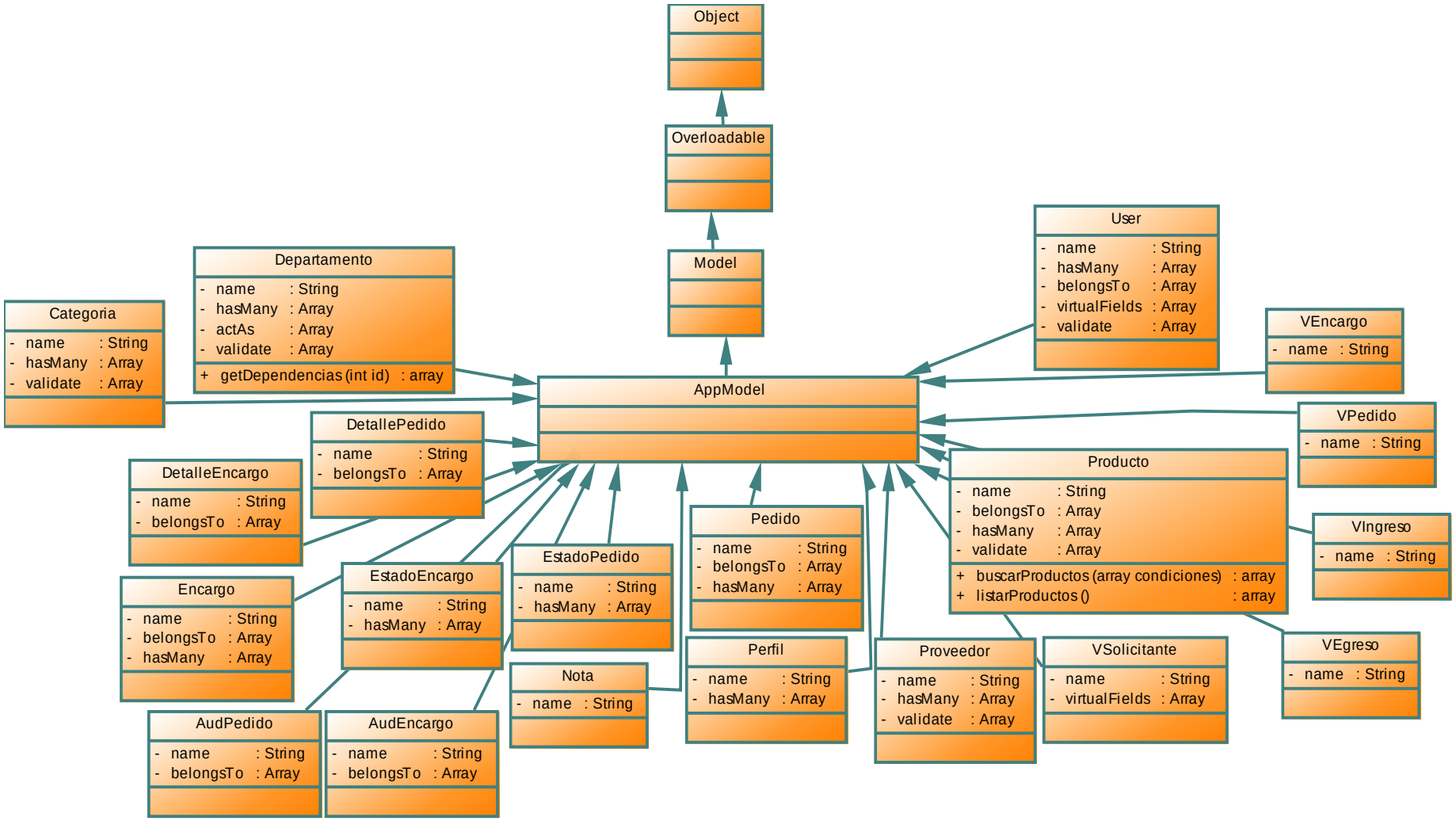


Figura 2.12 Clases de los Modelos

2.3.3.2 Diagrama de Clases para los Controladores

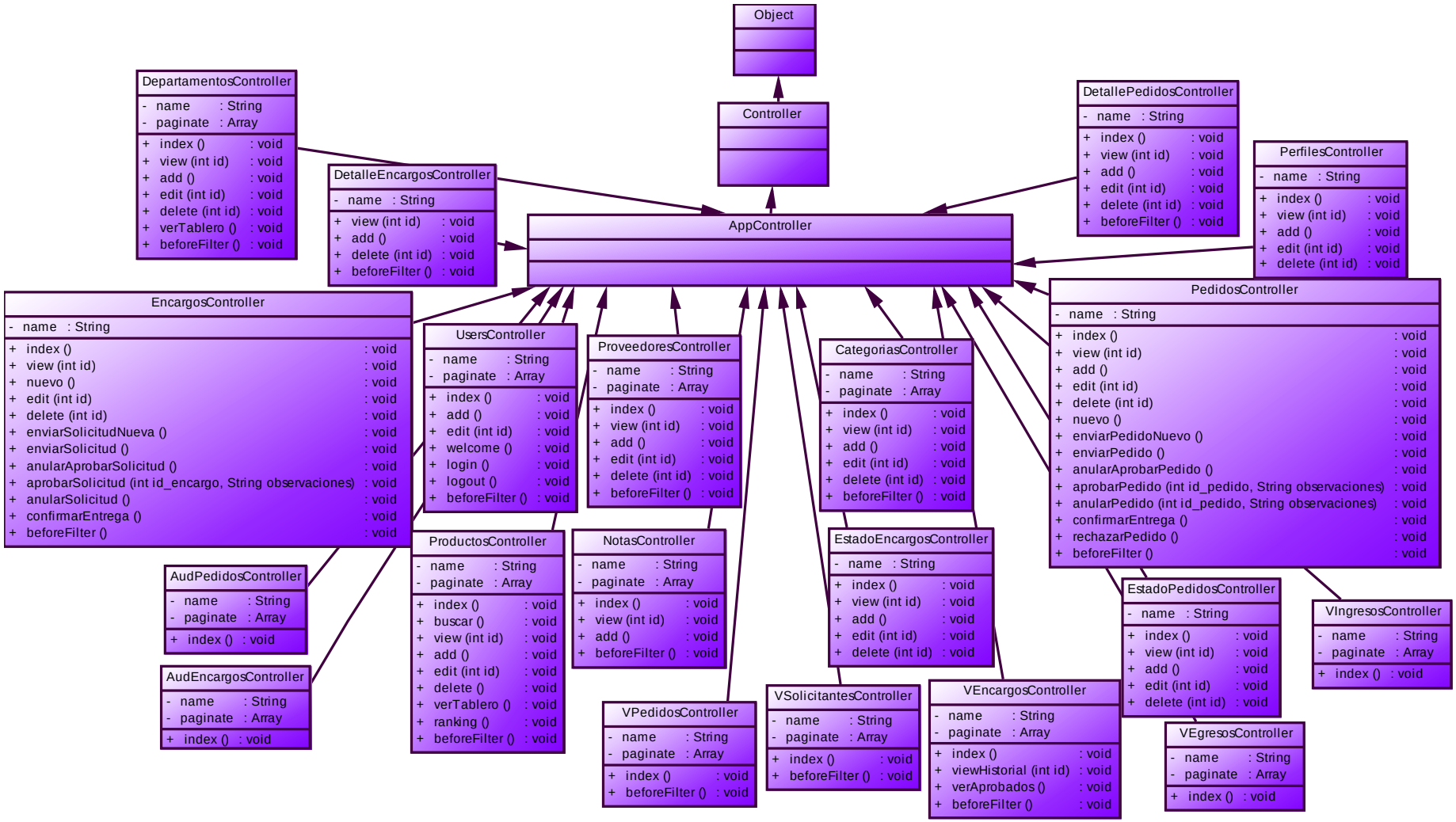


Figura 2.13 Clases de los Controladores

2.3.3.3 Diagrama de Clases utilizadas para las Vistas (Helpers)

Los Ayudantes o Helpers son clases a modo de componentes para la capa de presentación de la aplicación. Contienen lógica de presentación que puede ser compartida por muchas vistas, elementos y layouts (plantillas).

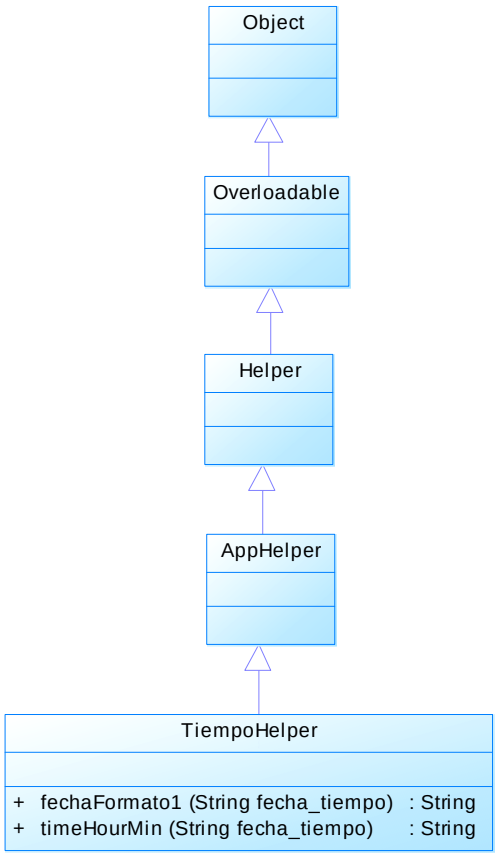


Figura 2.14 Helpers

Al igual que los controladores y los modelos, los helpers heredan de otras clases que constituyen el núcleo de CakePHP y para este proyecto el único que helper que se ha definido es uno denominado TiempoHelper, el cual permite controlar el formato de presentación para todo lo relacionado a horas y fechas.

2.3.4 TAREAS DE PROGRAMACIÓN

Cada Historia de Usuario presentada en el Plan de Entregas es transformada en Tareas de Programación (también conocidas como Tareas de Ingeniería o Tareas de Desarrollo). Generalmente, las tareas son más pequeñas que la historia global,

pero en el presente proyecto se va a considerar una sola Tarea de Programación por cada Historia de Usuario, ya que la complejidad de las Historias de Usuario no amerita dividirlas en varias tareas.

Lo ideal es que durante las reuniones de deliberación, el cliente pueda responder todas las consultas por parte de los desarrolladores, para lo cual se puede utilizar la técnica denominada “tormenta de ideas”; de esta forma, los desarrolladores tienen una base para convertir las Historias de Usuario en Tareas de Programación.

En la Tabla 2.39 se presenta una tarjeta que puede ser utilizada como plantilla por los miembros del equipo de desarrollo para escribir Tareas de Programación, la cual debe ser llenada con la información correspondiente.

Tarea de Programación	
Número de tarea:	**Número de historia:**
Título:	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción:	
Anotaciones respecto al sistema:	

Tabla 2.39 Plantilla para elaborar Tareas de Programación

Los campos de la tarjeta son los siguientes:

- **Número de tarea:** es un identificador secuencial, el cual puede tomar el orden en el que se van a implementar las tareas.

- **Número de historia:** es el identificador de la Historia de Usuario a la que pertenece la Tarea de Programación.
- **Titulo:** corresponde al nombre que se le ha otorgado a la tarea, comúnmente es el mismo nombre de la historia que se implementa (si es que una historia implica el desarrollo de una tarea, como en el caso de este proyecto).
- **Tipo de tarea:** especifica el tipo de actividad que se realiza, puede ser desarrollo, corrección, mejora u otra.
- **Descripción:** aquí se debe anotar la descripción de la tarea de tal forma que pueda ser comprendida por todos los miembros del equipo.
- **Anotaciones respecto al sistema**: corresponden a detalles relevantes que los miembros consideren importantes para la implementación de la tarea.

Desde la Tabla 2.40 hasta la Tabla 2.60 se tienen las tarjetas de las Tareas de Programación correspondientes a cada Historia de Usuario, las cuales están en orden de implementación, tomando en cuenta la prioridad de cada Historia de Usuario. Además, las tareas que se presentan aquí conciernen sólo a tareas de desarrollo.

Tarea de Programación	
Número de tarea: 1	**Número de historia:** 1
Título: Autenticación de usuario	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe crear un formulario en el cual se tenga dos cajas de texto, una para ingresar el nombre de usuario y otra para ingresar la contraseña. La contraseña debe mostrarse solo como puntos al momento de escribirse. Para ingresar se debe hacer clic en un botón o se debe presionar la tecla “Enter”. Si los datos ingresados son correctos, se obtiene la pantalla de bienvenida del sistema,	

caso contrario se muestra un mensaje de error.
Anotaciones respecto al sistema: Ninguna.

Tabla 2.40 Tarea de Programación “Autenticación de usuario”

Tarea de Programación	
Número de tarea: 2	**Número de historia:** 9
Título: Pedido a proveedor	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Primero se debe realizar una búsqueda de productos, ingresando una palabra o parte de ésta. Los productos que coinciden con la búsqueda deben aparecer con su respectivo precio y una caja de texto para ingresar la cantidad requerida. Cada producto que aparece en la lista debe tener un botón para poder hacer clic sobre él y añadir el producto al pedido. El detalle del pedido debe irse actualizando conforme se van añadiendo los productos. Una vez que se tenga el pedido completo, se ingresa una descripción, se escoge el proveedor y se hace clic en un botón para enviarlo. El pedido quedará con estado de “Enviado”.	
Anotaciones respecto al sistema: CakePHP trabaja por defecto con la librería javascript Prototype, cuya funcionalidad es necesaria para implementar esta tarea, pero para este proyecto también se está utilizando JQuery. Estas dos librerías generan un conflicto que tiene que ser resuelto.	

Tabla 2.41 Tarea de Programación “Pedido a proveedor”

Tarea de Programación	
Número de tarea: 3	**Número de historia:** 10
Título: Listar pedidos a proveedores	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra

Descripción: Se debe presentar una lista ordenada descendentemente según el número del pedido, con sus respectivos datos de cabecera como proveedor, fecha, estado, etc. Y también se debe poder ordenar la lista en forma ascendente o descendente según éstos campos. En cada entrada debe haber un botón para hacer clic y poder ver el detalle del pedido seleccionado.
Anotaciones respecto al sistema: Ninguna.

Tabla 2.42 Tarea de Programación “Listar pedidos a proveedores”

Tarea de Programación	
Número de tarea: 4	**Número de historia:** 11
Título: Aprobar / Anular pedido a proveedor	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Al ver el detalle de un pedido que está con estado de “Enviado”, a más del detalle, se debe tener una caja de texto para poder ingresar cualquier observación y bajo ésta se tendrán 2 botones, uno para anular el pedido y otro para aprobarlo, quedando el pedido con estado “Anulado” o “Aprobado” según sea el caso.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.43 Tarea de Programación “Aprobar / Anular pedido a proveedor”

Tarea de Programación	
Número de tarea: 5	**Número de historia:** 12
Título: Confirmar entrega de pedido a proveedor	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra

Descripción: Al ver el detalle de un pedido que está con estado de “Aprobado”, a más del detalle, se debe tener una caja de texto para ingresar el número de RUC del proveedor y así poder confirmar la entrega de los productos haciendo clic en un botón, con lo que el pedido queda con un estado de “Entregado”. También se debe tener una caja de texto para ingresar cualquier observación en caso de que haya inconvenientes al momento de la entrega, de esta forma al hacer clic en otro botón se rechaza el pedido, quedando el pedido con un estado de “Rechazado”.
Anotaciones respecto al sistema: Ninguna.

Tabla 2.44 Tarea de Programación “Confirmar entrega de pedido a proveedor”

Tarea de Programación	
Número de tarea: 6	**Número de historia:** 13
Título: Pedido a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Primero se debe realizar una búsqueda de productos, ingresando una palabra o parte de ésta. Los productos que coinciden con la búsqueda deben aparecer con su respectivo precio y una caja de texto para ingresar la cantidad requerida. Cada producto que aparece en la lista debe tener un botón para poder hacer clic sobre él y añadir el producto al pedido. El detalle del pedido debe irse actualizando conforme se van añadiendo los productos. Una vez que se tenga el pedido completo, se ingresa una descripción y se hace clic en un botón para enviarlo. El pedido quedará con estado de “Enviado”.	
Anotaciones respecto al sistema: CakePHP trabaja por defecto con la librería javascript Prototype, cuya funcionalidad es necesaria para implementar esta tarea, pero para este proyecto también se está utilizando JQuery. Estas dos librerías generan un conflicto que tiene que ser resuelto.	

Tabla 2.45 Tarea de Programación “Pedido a bodega”

Tarea de Programación	
Número de tarea: 7	**Número de historia:** 14
Título: Listar todos los pedidos a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe presentar una lista ordenada descendentemente según el número del pedido, con sus respectivos datos de cabecera como usuario, fecha, estado, etc. Y también se debe poder ordenar la lista en forma ascendente o descendente según éstos campos. En cada entrada debe haber un botón para hacer clic y poder ver el detalle del pedido seleccionado. Si el usuario es Supervisor, éste puede hacer previamente un filtrado de pedidos por usuario, para lo cual, se debe hacer una búsqueda de usuarios por nombre y que al momento de seleccionar una entrada se pueda ver todos los pedidos de ese usuario. Si el usuario es Solicitante y el pedido aún no ha sido aprobado por un usuario Supervisor, al momento de ver el detalle del pedido debo tener la opción de anular el pedido haciendo clic en un botón, con lo que el pedido queda con un estado de "Anulado".	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.46 Tarea de Programación "Listar todos los pedidos a bodega"

Tarea de Programación	
Número de tarea: 8	**Número de historia:** 15
Título: Aprobar / Anular pedido a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Al ver el detalle de un pedido que está con estado de "Enviado", a más del detalle, se debe tener una caja de texto para poder ingresar cualquier observación y bajo ésta se tendrán 2 botones, uno para anular el pedido y otro para aprobarlo, quedando el pedido con estado "Anulado" o "Aprobado" según sea el caso.	

Anotaciones respecto al sistema: Ninguna.

Tabla 2.47 Tarea de Programación “Aprobar / Anular pedido a bodega”

Tarea de Programación	
Número de tarea: 9	**Número de historia:** 16
Título: Listar pedidos a bodega aprobados	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe presentar una lista solo de pedidos aprobados, ordenada descendentemente según el número del pedido, con sus respectivos datos de cabecera como usuario, fecha, etc. Y también se debe poder ordenar la lista en forma ascendente o descendente según éstos campos. En cada entrada debe haber un botón para hacer clic y poder ver el detalle del pedido seleccionado.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.48 Tarea de Programación “Listar pedidos a bodega aprobados”

Tarea de Programación	
Número de tarea: 10	**Número de historia:** 17
Título: Confirmar entrega de pedido a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Al ver el detalle de un pedido que está con estado de “Aprobado”, a más del detalle, se debe tener un botón para poder confirmar la entrega del pedido al usuario solicitante, quedando el pedido con estado de “Entregado”.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.49 Tarea de Programación “Confirmar entrega de pedido a bodega”

Tarea de Programación	
Número de tarea: 11	**Número de historia:** 18
Título: Stock	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe presentar una lista de todos los productos ordenados en orden alfabético, con información sobre su stock actual, stock mínimo, stock máximo y otros datos relevantes. Se debe tener una caja de texto para ingresar una palabra clave y poder realizar una búsqueda de productos que coincidan con esa palabra. Para ver la información del stock, también se debe poder hacerlo ingresando desde las categorías de productos, para ver los productos que pertenecen a una categoría específica.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.50 Tarea de Programación “Stock”

Tarea de Programación	
Número de tarea: 12	**Número de historia:** 2
Título: Crear departamento	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para ingresar los datos de un nuevo departamento y haciendo clic en un botón guardarlos en la base de datos. Se debe poder ver una lista con todos los departamentos creados, en la cual se pueda realizar búsquedas por nombre y al seleccionar una entrada se pueda ver el detalle de los usuarios que pertenecen a dicho departamento. También se podrá ver las dependencias de éste departamento y al hacer clic sobre el nombre de una de ellas, se podrá ver su información.	
Anotaciones respecto al sistema: Se debe configurar correctamente la tabla correspondiente de la base de datos con el fin de poder utilizar la funcionalidad	

que ofrece el framework para crear la estructura de árbol de los departamentos.

Tabla 2.51 Tarea de Programación “Crear departamento”

Tarea de Programación	
Número de tarea: 13	**Número de historia:** 3
Título: Crear usuario	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para ingresar los datos de un nuevo usuario y haciendo clic en un botón guardarlos en la base de datos. Se debe poder ver una lista con todos los usuarios creados, en la cual se pueda realizar búsquedas por nombre y al seleccionar una entrada se pueda ver el detalle de la información del usuario.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.52 Tarea de Programación “Crear usuario”

Tarea de Programación	
Número de tarea: 14	**Número de historia:** 4
Título: Crear proveedor	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para ingresar los datos de un nuevo proveedor y haciendo clic en un botón guardarlos en la base de datos. Se debe poder ver una lista con todos los proveedores creados, en la cual se pueda realizar búsquedas por nombre y al seleccionar una entrada se pueda ver el detalle de la información del proveedor.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.53 Tarea de Programación “Crear proveedor”

Tarea de Programación	
Número de tarea: 15	**Número de historia:** 5
Título: Crear categoría de productos	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para ingresar los datos de una nueva categoría de productos y haciendo clic en un botón guardarlos en la base de datos. Se debe poder ver una lista con todas las categorías creadas, en la cual se pueda realizar búsquedas por nombre y al seleccionar una entrada se pueda ver el detalle de los productos que pertenecen a dicha categoría.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.54 Tarea de Programación “Crear categoría de productos”

Tarea de Programación	
Número de tarea: 16	**Número de historia:** 6
Título: Crear producto	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para ingresar los datos de un nuevo producto y haciendo clic en un botón guardarlos en la base de datos. Se debe poder ver una lista con todos los productos creados, en la cual se pueda realizar búsquedas por nombre.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.55 Tarea de Programación “Crear producto”

Tarea de Programación	
Número de tarea: 17	**Número de historia:** 7
Título: Perfil de usuario	**Tipo de tarea:** Desarrollo / Corrección /

	Mejora / Otra
Descripción: Al ver la información del perfil de usuario, el usuario Administrador tendrá la posibilidad de cambiar la contraseña de cualquier cuenta de usuario, pero si se trata de otro tipo de usuario, sólo podrá cambiar la contraseña de su propia cuenta.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.56 Tarea de Programación “Perfil de usuario”

Tarea de Programación	
Número de tarea: 18	**Número de historia:** 8
Título: Comunicaciones internas	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un formulario para crear una nueva comunicación interna (tipo correo electrónico), la cual al ser publicada, podrá ser observada por todos los usuarios del sistema en la parte derecha de la pantalla. Sólo se presentarán las 7 últimas comunicaciones y al hacer clic sobre el título de una de ellas, se podrá observar el comunicado completo, incluyendo la hora y fecha de la publicación.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.57 Tarea de Programación “Comunicaciones internas”

Tarea de Programación	
Número de tarea: 19	**Número de historia:** 19
Título: Resumen de pedidos a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un tablero de datos que presente un consolidado de los pedidos hechos a bodega según su estado. Esto se lo debe hacer por cada	

departamento. También se mostrará el número de usuarios de cada departamento, además de una sumatoria del número de productos que están en pedidos con estado “Enviado”, con su respectivo total en dinero.
Anotaciones respecto al sistema: Ninguna.

Tabla 2.58 Tarea de Programación “Resumen de pedidos a bodega”

Tarea de Programación	
Número de tarea: 20	**Número de historia:** 20
Título: Resumen de pedidos a proveedor y a bodega	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Se debe tener un tablero de datos que presente un consolidado de los pedidos hechos a bodega y a proveedores, según su estado.	
Anotaciones respecto al sistema: Ninguna.	

Tabla 2.59 Tarea de Programación “Resumen de pedidos a proveedor y a bodega”

Tarea de Programación	
Número de tarea: 21	**Número de historia:** 21
Título: Reportes	**Tipo de tarea:** Desarrollo / Corrección / Mejora / Otra
Descripción: Un reporte gráfico de los productos más pedidos, tanto en cantidad como en valores. Un reporte de existencias. Un reporte de acciones realizadas por los usuarios. Todos los reportes deben tener un botón para imprimir.	
Anotaciones respecto al sistema: Para los reportes gráficos se debe tener instalada la librería GD de PHP y la librería de terceros *libchart*.	

Tabla 2.60 Tarea de Programación “Reportes”

2.4 PRUEBAS

Toda característica de la aplicación debe ser probada. Los programadores escriben pruebas para chequear el correcto funcionamiento del código fuente: pruebas unitarias; mientras que los clientes realizan pruebas funcionales: pruebas de aceptación. El resultado es una aplicación más segura y de calidad.

2.4.1 PRUEBAS DE UNIDAD

Básicamente, los elementos de la aplicación que se han ido probando durante el desarrollo son los modelos y los controladores, para lo cual se utilizó la herramienta SimpleTest. A continuación se tiene un ejemplo de caso de prueba para cada uno de estos elementos[21].

2.4.1.1 Modelos

Para hacer pruebas de fragmentos de código que dependen de los modelos y de los datos, se debe crear primero algo denominado “fixtures”. Los *fixtures* son en realidad clases que sirven para generar tablas en una base de datos de prueba. En estas clases se define principalmente cómo debe ser creada la tabla (qué campos son parte de la tabla) y qué registros van a poblar inicialmente dicha tabla. En la Figura 2.15 se tiene un ejemplo del fixture creado para hacer pruebas del modelo *Departamento*. La estructura de la tabla se la importa directamente de la base de datos mediante la variable *$import*, de esta forma no se tiene que definirla manualmente (ver definición de la tabla *departamentos* en la Tabla 2.27). Los datos que se utilizarán en la prueba también pueden ser importados desde la tabla correspondiente en la base de datos, pero es preferible definirlos dentro del fixture para tener un mayor control sobre ellos, lo cual se lo hace con la variable *$records*.

[21] Todos los casos de pruebas unitarias se encuentran almacenados en el directorio de la aplicación, en la ruta *inventario/app/test/cases*.

```php
<?php
 class DepartamentoFixture extends CakeTestFixture {
      var $name = 'Departamento';

      var $import = 'Departamento';

      var $records = array(
          array ('id' => 1, 'parent_id' => null, 'left' => null, 'right' => null,
           'name' => 'No especificado', 'descripcion' => 'No especificado'),
          array ('id' => 2, 'parent_id' => 0, 'left' => 1, 'right' => 12,
           'name' => 'Direccion Administrativa', 'descripcion' => 'No especificado'),
          array ('id' => 3, 'parent_id' => 1, 'left' => 2, 'right' => 5,
           'name' => 'Logistica', 'descripcion' => 'No especificado'),
          array ('id' => 4, 'parent_id' => 2, 'left' => 3, 'right' => 4,
            'name' => 'Bodega', 'descripcion' => 'No especificado'),
          array ('id' => 5, 'parent_id' => 1, 'left' => 6, 'right' => 7,
            'name' => 'Finanzas', 'descripcion' => 'No especificado'),
          array ('id' => 6, 'parent_id' => 1, 'left' => 8, 'right' => 9,
            'name' => 'sistemas', 'descripcion' => 'No especificado'),
          array ('id' => 7, 'parent_id' => 1, 'left' => 10, 'right' => 11,
            'name' => 'R.R.H.H.', 'descripcion' => 'No especificado')
      );
 }
 ?>
```

Figura 2.15 Fixture para el modelo *Departamento*

Una vez creado el fixture para el modelo, se puede seguir con la creación de la clase para el respectivo caso de prueba. Como se puede observar en el ejemplo de la Figura 2.16, en la variable *$fixtures* se deben especificar todos los fixtures necesarios para realizar la prueba, es decir, todos los fixtures de los modelos que se relacionan directa o indirectamente con el modelo a probar.

```php
<?php
  App::import('Model', 'Departamento');

  class DepartamentoTestCase extends CakeTestCase {

      var $fixtures = array('app.departamento','app.user','app.perfil','app.encargo',
                            'app.detalle_encargo','app.estado_encargo','app.producto',
                            'app.categoria','app.detalle_pedido','app.pedido','app.estado_pedido',
                            'app.proveedor');

      function testGetDependencias() {
          $this->Departamento =& ClassRegistry::init('Departamento');

          $result = $this->Departamento->getDependencias(3);
          $expected = array();
          $this->assertEqual($result, $expected);
      }
  }

 ?>
```

Figura 2.16 Caso de prueba para el modelo *Departamento*

En este ejemplo, la única prueba que se realiza es sobre el método *getDependencias* del modelo *Departamentos*, el cual pasa un parámetro que representa el departamento del cual se quiere obtener las dependencias. En este caso, se le pasa el identificador de un departamento que no tiene dependencias, por lo que se espera que el arreglo que se obtiene de la ejecución de este método sea un arreglo vacío. El resultado de las pruebas se las puede ver desde un navegador web, ingresando la ruta del archivo que permite ejecutar todos los casos de prueba, que para este proyecto es *http://localhost/inventario/app/webroot/test.php*. Allí, ingresando a *App->Test Cases*, se tiene la lista de todos los casos de prueba creados para la aplicación. En la Figura 2.17 se tiene el resultado de ejecutar el caso de prueba para el modelo *Departamento*.

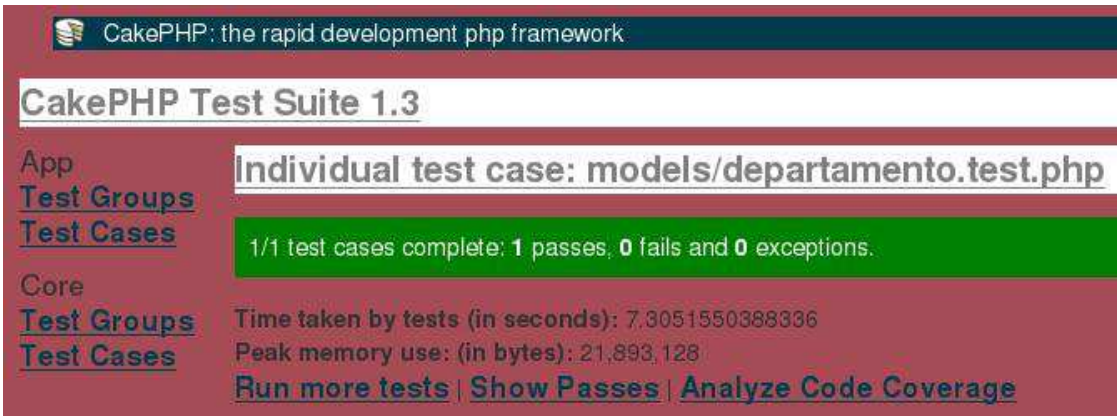


Figura 2.17 Ejecución del caso de prueba para el modelo *Departamento*

2.4.1.2 Controladores

Para el caso de los controladores, se debe crear una sola clase por cada controlador a probar. En la Figura 2.18, se tiene el ejemplo del caso de prueba para el controlador *Departamentos*. En esta clase se escriben los diferentes métodos que servirán para probar el funcionamiento del controlador.

```php
<?php
class DepartamentosControllerTest extends CakeTestCase {
    function startCase() {
      echo '<h1>Comenzando caso de prueba</h1>';
    }
    function endCase() {
      echo '<h1>Terminando caso de prueba</h1>';
    }
    function startTest($metodo) {
      echo '<h3>Comenzando método ' . $metodo . '</h3>';
    }
    function endTest($metodo) {
      echo '<h3>Terminando método ' . $metodo . '</h3>';
    }

    function testIndexGetRenderedHtml() {
      $result = $this->testAction('/departamentos/index',
      array('return' => 'render'));
      debug(htmlentities($result));
    }
    function testIndexGetRenderedView() {
      $result = $this->testAction('/departamentos/index',
      array('return' => 'render'));
      debug($result);
    }
    function testIndexGetViewVars() {
      $result = $this->testAction('/departamentos/index',
      array('return' => 'vars'));
      debug($result);
    }
    function testIndexNoVars() {
       $result = $this->testAction('/departamentos/index',
      array('return' => 'vars'));
      $this->assertNotEqual($result, array());
    }
}
?>
```

Figura 2.18 Caso de prueba para el controlador *Departamentos*

Los cuatro primeros métodos (*startCase*, *endCase*, *startTest* y *endTest*) solo sirven para obtener un mensaje al inicio y al fin de la ejecución de todo el caso de prueba y de cada método en particular, luego se tienen todos los métodos que sirven para realizar las pruebas propiamente dichas sobre las acciones de los controladores. En este ejemplo se utiliza la acción *index* del controlador *Departamentos*, y se tienen cuatro métodos que sirven para realizar algunas de las pruebas más comunes, cuyos resultados se detallan a continuación:

- testIndexGetRenderedHtml – permite visualizar el código html generado.

Figura 2.19 Resultado de la ejecución del método *testIndexGetRenderedHtml*

- testIndexGetRenderedView – permite ver la vista generada para una acción del controlador.

Figura 2.20 Resultado de la ejecución del método *testIndexGetRenderedView*

- testIndexGetViewVars – genera un arreglo con todas las variables que están disponibles en la vista.

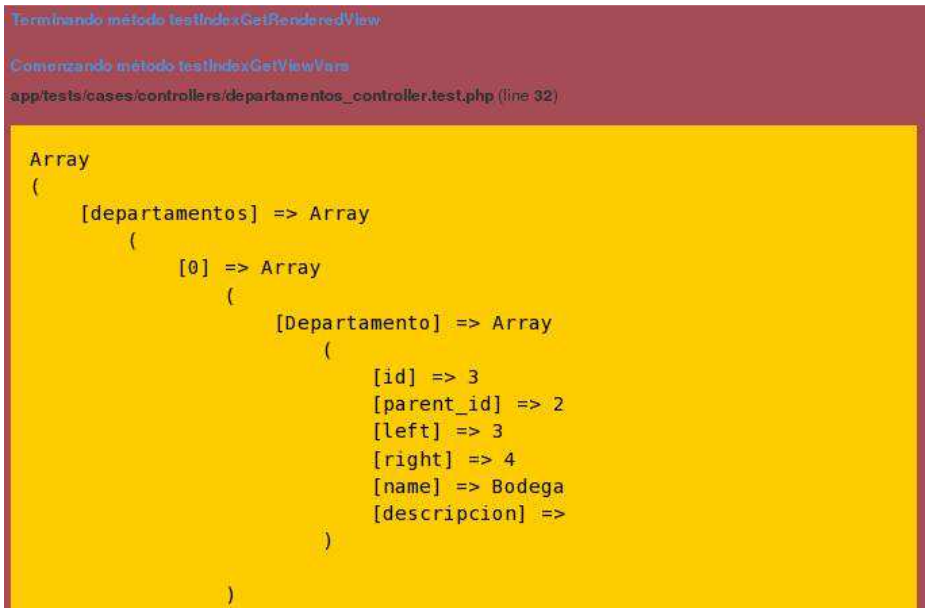


Figura 2.21 Resultado de la ejecución del método *testIndexGetViewVars*

- testIndexNoVars – es un ejemplo de prueba que utiliza un método *assert*, tal como se lo hizo en el caso de prueba para el método *Departamento*. En este caso, se comprueba que el arreglo de variables disponibles en la vista no sea vacío, y como ya se vio que en la ejecución del método anterior sí existen variables, la prueba debe pasar. Los resultados se obtienen al final de la ejecución de todo el caso de prueba, como se observa en la Figura 2.22.

Figura 2.22 Resultado de la ejecución del método *testIndexNoVars*

2.4.2 PRUEBAS DE ACEPTACIÓN

Las pruebas de aceptación se las realizan conjuntamente con el usuario. Con la finalidad de llevar un registro de lo realizado en cada una de estas pruebas, se pueden utilizar tarjetas como la de la Tabla 2.61.

Caso de Prueba	
Número de caso:	**Número de historia:**
Título:	
Precondiciones:	
Datos de Entrada:	
Resultado esperado:	
Evaluación:	

Tabla 2.61 Plantilla para Pruebas de Aceptación

En donde:

- **Número de caso:** es el número de caso de prueba.
- **Número de historia:** representa el número de Historia de Usuario que se está evaluando.
- **Título:** es el título o nombre que se da al caso de prueba, puede ser el mismo de la Historia de Usuario.
- **Precondiciones:** son los escenarios que se deben haber cumplido previamente para que la prueba pueda llevarse a cabo satisfactoriamente.
- **Datos de entrada:** son los datos que se ingresan (mediante un formulario) para la ejecución y prueba de la respectiva Historia de Usuario.
- **Resultado esperado:** es la descripción de lo que se espera que haga el sistema luego de ejecutar el caso de prueba.

- **Evaluación:** es la descripción de lo que realmente hizo el sistema, en relación al resultado esperado, luego de ejecutar el caso de prueba.

A continuación se tienen los casos de prueba para cada una de las Historias de Usuario:

Caso de Prueba	
Número de caso: 1	**Número de historia:** 1
Título: Autenticación de usuario	
Precondiciones: Debe haber por lo menos un usuario registrado en la base de datos.	
Datos de Entrada: Usuario (varchar 50), Contraseña (varchar 50)	
Resultado esperado: Al ingresar los datos correctos de un usuario registrado, el sistema debe permitir ver la pantalla de bienvenida, caso contrario se presenta un mensaje de error.	
Evaluación: Se ingresó datos válidos y se permitió ingresar al sistema. Con datos incorrectos se presentó el siguiente mensaje “No existe el usuario o la contraseña es incorrecta”.	

Tabla 2.62 Caso de prueba “Autenticación de usuario”

Caso de Prueba	
Número de caso: 2	**Número de historia:** 9
Título: Pedido a proveedor	
Precondiciones: Debe haber por lo menos un proveedor y un producto registrado en la base de datos.	
Datos de Entrada: Cantidad (integer 11), Descripción (varchar 50)	
Resultado esperado: Se guarda los datos del pedido con un estado de “Enviado”.	

Evaluación: Los datos se guardaron exitosamente.

Tabla 2.63 Caso de prueba “Pedido a proveedor”

Caso de Prueba	
Número de caso: 3	**Número de historia:** 10
Título: Listar pedidos a proveedores	
Precondiciones: Debe haber por lo menos un pedido a proveedor registrado en la base de datos.	
Datos de Entrada: Ninguno.	
Resultado esperado: Se presenta una lista de todos los pedidos a proveedor ingresados.	
Evaluación: Todo los pedidos a proveedor existentes se presentan en la lista.	

Tabla 2.64 Caso de prueba “Listar pedidos a proveedores”

Caso de Prueba	
Número de caso: 4	**Número de historia:** 11
Título: Aprobar / Anular pedido a proveedor	
Precondiciones: Debe haber por lo menos dos pedidos a proveedor en la base de datos con un estado de “Enviado”.	
Datos de Entrada: Observaciones (text)	
Resultado esperado: Al aprobar un pedido se debe cambiar el estado de dicho pedido a “Aprobado” en la base de datos. De igual forma, si se anula, debe cambiar a “Anulado”.	
Evaluación: Los pedidos cambiaron exitosamente de estado.	

Tabla 2.65 Caso de prueba “Aprobar / Anular pedido a proveedor”

Caso de Prueba	
Número de caso: 5	**Número de historia:** 12
Título: Confirmar entrega de pedido a proveedor	
Precondiciones: Debe haber registrado por lo menos dos pedidos con estado de "Aprobado".	
Datos de Entrada: Número de factura (char 15), Observaciones (text)	
Resultado esperado: Al confirmar la entrega, el pedido cambia de estado a "Entregado". Si se lo rechaza, el estado cambia a "Rechazado".	
Evaluación: Los pedidos cambiaron exitosamente de estado.	

Tabla 2.66 Caso de prueba "Confirmar entrega de pedido a proveedor"

Caso de Prueba	
Número de caso: 6	**Número de historia:** 13
Título: Pedido a bodega	
Precondiciones: Debe haber por lo menos un usuario con perfil de "Solicitante" y un producto registrado en la base de datos con suficiente stock.	
Datos de Entrada: Cantidad (integer 11), Descripción (varchar 50)	
Resultado esperado: Se guarda los datos del pedido con un estado de "Enviado".	
Evaluación: Los datos se guardaron exitosamente.	

Tabla 2.67 Caso de prueba "Pedido a bodega"

Caso de Prueba	
Número de caso: 7	**Número de historia:** 14
Título: Listar todos los pedidos a bodega	

Precondiciones: Debe haber por lo menos un pedido a bodega registrado en la base de datos.
Datos de Entrada: Ninguno.
Resultado esperado: Se presenta una lista de todos los pedidos a bodega ingresados.
Evaluación: Todos los pedidos a bodega se presentan en la lista.

Tabla 2.68 Caso de prueba “Listar todos los pedidos a bodega”

Caso de Prueba	
Número de caso: 8	**Número de historia:** 15
Título: Aprobar / Anular pedido a bodega	
Precondiciones: Debe haber por lo menos dos pedidos a bodega en la base de datos con un estado de “Enviado”.	
Datos de Entrada: Observaciones (text)	
Resultado esperado: Al aprobar un pedido se debe cambiar el estado de dicho pedido a “Aprobado” en la base de datos. De igual forma, si se anula, debe cambiar a “Anulado”.	
Evaluación: Los pedidos cambiaron exitosamente de estado	

Tabla 2.69 Caso de prueba “Aprobar / Anular pedido a bodega”

Caso de Prueba	
Número de caso: 9	**Número de historia:** 16
Título: Listar pedidos a bodega aprobados	
Precondiciones: Debe haber por lo menos un pedido a bodega que esté con estado de “Aprobado”.	

Datos de Entrada: Ninguno.
Resultado esperado: Se presenta una lista de todos los pedidos a bodega aprobados
Evaluación: Todos los pedidos a bodega aprobados se presentan en la lista.

Tabla 2.70 Caso de prueba “Listar pedidos a bodega aprobados”

Caso de Prueba	
Número de caso: 10	**Número de historia:** 17
Título: Confirmar entrega de pedido a bodega	
Precondiciones: Debe haber por lo menos un pedido a bodega con estado de “Aprobado”.	
Datos de Entrada: Ninguno.	
Resultado esperado: Al confirmar la entrega, el pedido debe cambiar su estado a “Entregado”.	
Evaluación: El estado del pedido cambia exitosamente.	

Tabla 2.71 Caso de prueba “Confirmar entrega de pedido a bodega”

Caso de Prueba	
Número de caso: 11	**Número de historia:** 18
Título: Stock	
Precondiciones: Debe haber por lo menos un producto registrado.	
Datos de Entrada: Ninguno.	
Resultado esperado: Se presenta una lista de todos los productos existentes, con su stock actual.	
Evaluación: Todos los productos existentes se presentan en la lista con la respectiva	

información de stock.

Tabla 2.72 Caso de prueba “Stock”

Caso de Prueba	
Número de caso: 12	**Número de historia:** 2
Título: Crear departamento	
Precondiciones: Ninguna.	
Datos de Entrada: Departamento padre (integer 11), Nombre (varchar 25), Descripción (text)	
Resultado esperado: Todos los datos ingresados se guardan en la base de datos.	
Evaluación: Los datos quedan registrados exitosamente.	

Tabla 2.73 Caso de prueba “Crear departamento”

Caso de Prueba	
Número de caso: 13	**Número de historia:** 3
Título: Crear usuario	
Precondiciones: Debe haber por lo menos un departamento registrado.	
Datos de Entrada: Nombres (varchar 50), Apellidos (varchar 50), Cédula (varchar 10), E-mail (varchar 100), Departamento (integer 11), Perfil (integer 11), Nombre de Usuario (varchar 50), Contraseña (varchar 50), Repetir (varchar 50)	
Resultado esperado: Todos los datos ingresados se guardan en la base de datos.	
Evaluación: Los datos quedan registrados exitosamente.	

Tabla 2.74 Caso de prueba “Crear usuario”

Caso de Prueba	
Número de caso: 14	**Número de historia:** 4
Título: Crear proveedor	
Precondiciones: Ninguna.	
Datos de Entrada: Nombre (varchar 45), Dirección (varchar 100), RUC (char 13), Teléfono 1 (varchar 20), Teléfono 2 (varchar 20), Tseléfono Particular (varchar 20), Fax (varchar 20), Celular (varchar 20), E-Mail (varchar 50), Observaciones (text)	
Resultado esperado: Todos los datos ingresados se guardan en la base de datos.	
Evaluación: Los datos quedan registrados exitosamente.	

Tabla 2.75 Caso de prueba “Crear proveedor”

Caso de Prueba	
Número de caso: 15	**Número de historia:** 5
Título: Crear categoría de productos	
Precondiciones: Ninguna.	
Datos de Entrada: Nombre (varchar 20), Descripción (text)	
Resultado esperado: Todos los datos ingresados se guardan en la base de datos.	
Evaluación: Los datos quedan registrados exitosamente.	

Tabla 2.76 Caso de prueba “Crear categoría de productos”

Caso de Prueba	
Número de caso: 16	**Número de historia:** 6
Título: Crear producto	

Precondiciones: Debe haber por lo menos una categoría de productos registrada.
Datos de Entrada: Nombre (varchar 30), Categoría (integer 11), Precio (double), Stock mínimo (integer 10), Stock máximo (integer 10), Tipo (boolean)
Resultado esperado: Todos los datos ingresados se guardan en la base de datos.
Evaluación: Los datos quedan registrados exitosamente.

Tabla 2.77 Caso de prueba “Crear producto”

Caso de Prueba	
Número de caso: 17	**Número de historia:** 7
Título: Perfil de usuario	
Precondiciones: Debe haber por lo menos un usuario con perfil de “Administrador” y otro usuario con un perfil cualquiera.	
Datos de Entrada: Nueva contraseña (varchar 50), Repetir (varchar 50)	
Resultado esperado: Se debe poder ingresar al sistema con la nueva contraseña.	
Evaluación: Se ingresó al sistema exitosamente.	

Tabla 2.78 Caso de prueba “Perfil de usuario”

Caso de Prueba	
Número de caso: 18	**Número de historia:** 8
Título: Comunicaciones internas	
Precondiciones: Ninguna	
Datos de Entrada: Asunto (varchar 100), Contenido (text)	
Resultado esperado: Se publica una nueva comunicación interna.	

Evaluación: La comunicación se publicó exitosamente.

Tabla 2.79 Caso de prueba “Comunicaciones internas”

Caso de Prueba	
Número de caso: 19	**Número de historia:** 19
Título: Resumen de pedidos a bodega	
Precondiciones: Debe haber por lo menos un pedido a bodega registrado.	
Datos de Entrada: Ninguno.	
Resultado esperado: Se presenta el tablero con datos coherentes.	
Evaluación: Los datos se presentan coherentemente en el tablero de resumen.	

Tabla 2.80 Caso de prueba “Resumen de pedidos a bodega”

Caso de Prueba	
Número de caso: 20	**Número de historia:** 20
Título: Resumen de pedidos a proveedor y a bodega	
Precondiciones: Debe haber por lo menos un pedido a bodega y un pedido a proveedor registrados.	
Datos de Entrada: Ninguno.	
Resultado esperado: Se presenta el tablero con datos coherentes.	
Evaluación: Los datos se presentan coherentemente en el tablero de resumen.	

Tabla 2.81 Caso de prueba “Resumen de pedidos a proveedor y a bodega”

Caso de Prueba	
Número de caso: 21	**Número de historia:** 21
Título: Reportes	
Precondiciones: Debe haber por lo menos un pedido a bodega y un pedido a proveedor registrados.	
Datos de Entrada: Ninguno.	
Resultado esperado: Los reportes se presentan con datos coherentes y se imprimen correctamente.	
Evaluación: Los datos de los reportes se presentan coherentemente y se pueden imprimir sin ningún problema.	

Tabla 2.82 Caso de prueba “Reportes”

CAPÍTULO 3. APLICACIÓN AL CASO DE ESTUDIO

3.1 DESCRIPCIÓN DE LA INSTITUCIÓN PÚBLICA

3.1.1 MINISTERIO DE DESARROLLO URBANO Y VIVIENDA

El Ministerio de Desarrollo Urbano y Vivienda (MIDUVI) es una Institución Pública cuyo objetivo es el de "contribuir al desarrollo del País a través de la formulación de políticas, regulaciones, planes, programas y proyectos, que garanticen un Sistema Nacional de Asentamientos Humanos, sustentado en una red de infraestructura de vivienda y servicios básicos que consoliden ciudades incluyentes, con altos estándares de calidad, alineados con las directrices establecidas en la Constitución Nacional y el Plan Nacional de Desarrollo."[22] El edificio principal de esta Cartera de Estado se encuentra ubicado en la ciudad de Quito, en la Av. 10 de Agosto y Luis Cordero.

Algunos de los objetivos estratégicos de esta institución son:

- "Facilitar las condiciones que hagan posible que las familias con menores ingresos puedan acceder a una vivienda digna, o mejorar la vivienda precaria que poseen.
- Mejorar y ampliar la dotación de agua potable y saneamiento a las áreas urbano-marginales y rurales, donde existen altos índices de pobreza, déficit y alta densidad demográfica.
- Promover e incentivar la participación del sector privado, tanto en el financiamiento como en la construcción de programas de vivienda social y proyectos de agua potable, saneamiento y residuos sólidos.
- Apoyar a los municipios para que desarrollen los mecanismos e instrumentos que les permitan administrar en forma planificada el uso y ocupación del suelo, en forma tal que se mejoren las condiciones sociales y ambientales de los pueblos y ciudades.

[22] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Objetivos.

- Incentivar la participación de las comunidades organizadas, para facilitar la atención a la demanda de Vivienda, Agua Potable, Saneamiento y Residuos sólidos"[23]

3.1.2 VALORES

Los valores que la Institución cree que son los más importantes en su diaria labor y por lo tanto los promueve, son valores tales como:

- Equidad
- Solidaridad
- Transparencia
- Responsabilidad
- Corresponsabilidad
- Lealtad
- Honestidad

3.1.3 MISIÓN

"Formular normas, políticas, directrices, planes, programas y proyectos de hábitat, vivienda, agua potable, saneamiento y residuos sólidos, a través de una gestión eficiente, transparente y ética para contribuir al buen vivir de la sociedad ecuatoriana."[24]

3.1.4 VISIÓN

"Ser el eje estratégico del desarrollo social a nivel nacional, a través de la conformación de un Sistema Nacional de Asentamientos Humanos y ciudades incluyentes, solidarias, participativas y competitivas, para garantizar un hábitat sustentable de la sociedad ecuatoriana."[25]

[23] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Objetivos.
[24] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Valores / Misión / Visión.
[25] IDEM

3.1.5 ESTRUCTURA ORGÁNICA – FUNCIONAL

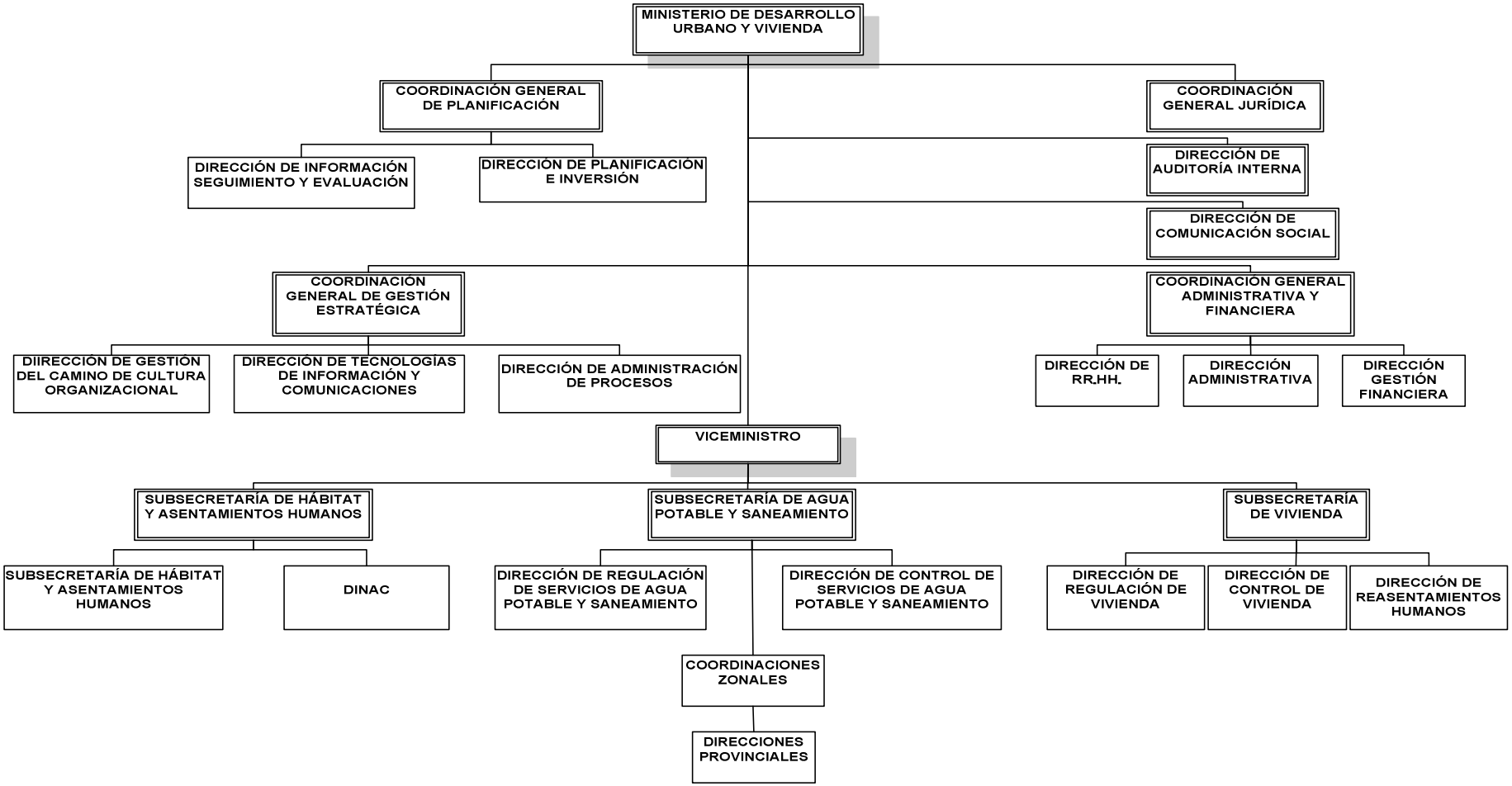


Figura 3.1 Organigrama del Ministerio de Desarrollo Urbano y Vivienda[26]

[26] Fuente: MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Organigrama del Ministerio de Desarrollo Urbano y Vivienda.

3.2 IMPLEMENTACIÓN DEL SISTEMA

3.2.1 REQUERIMIENTOS MÍNIMOS

3.2.1.1 Software

En la Tabla 1.5 del Capítulo 1 – Justificación de las herramientas de desarrollo – se tienen las versiones del software con el cual se ha desarrollado y se ha probado el Sistema de Control de Inventarios. Por lo tanto, las versiones que se deberían utilizar en la implementación del sistema son las siguientes:

Software	Versión	Función
Fedora	13	Sistema Operativo
Apache	2.2.15	Servidor Web
MySQL	5.1.45	Gestor de Base de Datos
PHP	5.3.6	Lenguaje de Programación
CakePHP	1.3.4	Framework de Desarrollo

Tabla 3.1 Requerimientos mínimos de software

3.2.1.2 Hardware

A excepción del sistema operativo, los requerimientos de hardware para las herramientas utilizadas no son muy grandes; por ejemplo, para instalar un Servidor Web Apache se puede utilizar tan solo una máquina con un procesador de 1GHz, 512 MB de RAM y 50 MB de disco. Por tal razón, los requerimientos mínimos de hardware estarán supeditados a los requerimientos del sistema operativo, en este caso Fedora 13 con entorno gráfico, y según la página oficial del proyecto fedora (http://fedoraproject.org), los requerimientos de instalación recomendados son:

- PC compatible con procesadores Intel de 64 bits
- 1GB de memoria (RAM)
- 10GB espacio en el disco / almacenamiento [27]

[27] RED HAT, INC. Y OTROS. Requerimientos de instalación recomendados.

Por supuesto, para una posible implantación del sistema se debe tomar en cuenta el volumen de datos que se espera almacenar en disco y la cantidad de usuarios y el tiempo que estarán conectados al sistema.

3.2.2 PROCEDIMIENTO PARA LA IMPLEMENTACIÓN

Debido a que en la Institución Pública no se cuenta con máquinas que cumplan con los requisitos de software necesarios para levantar el Sistema, se procedió a implementar el servidor desde una ubicación externa al edificio donde funciona el Ministerio de Vivienda. Como el Sistema de Inventarios está creado para ser utilizado vía Web, el único requisito que se debe cumplir es el acceso a Internet desde dentro de la Institución, lo cual es un hecho.

El servidor fue montado en la residencia del autor, donde también se dispone de acceso a Internet. Las características del servidor son las siguientes:

- Procesador AMD Turion Dual Core de 2.1 GHz
- 4 GB de RAM
- Disco Duro de 250 GB

Esta máquina funciona como Servidor Web y también como Servidor de Base de Datos.

Para poder acceder a la Aplicación desde cualquier ubicación, se debe configurar el router para que la dirección IP asignada al servidor esté como una Zona Desmilitarizada (DMZ), tal como se aprecia en la Figura 3.2.

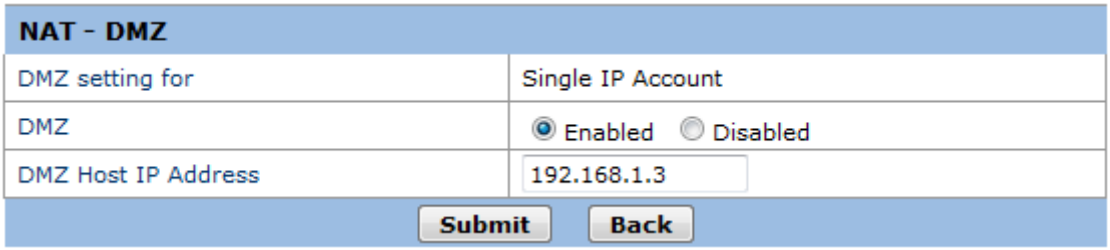


Figura 3.2 Configuración de Zona Desmilitarizada

La IP asignada al servidor dentro de la red doméstica es 192.168.1.3, pero al estar configurado como una DMZ, se puede acceder a la Aplicación desde la Web

utilizando la IP pública asignada al router, por ejemplo: 186.46.48.173. De esta forma, al ingresar desde un navegador la dirección *http://186.46.48.173/inventario*, se obtendrá la pantalla para autenticarse en el Sistema.

3.3 ANÁLISIS DE RESULTADOS

Para realizar la evaluación del Sistema por parte de los usuarios, se ha preparado una encuesta para medir la usabilidad del software. Esta encuesta está basada en el método SUMI (Software Usability Measurement Inventory), el cual es un método rigurosamente probado para medir la calidad del software desde el punto de vista del usuario final. Este método fue desarrollado por un grupo de investigación denominado *Human Factors Research Group* de la Universidad *College Cork* en la República de Irlanda y es mencionada en el estándar ISO 9241 como un reconocido método para medir la satisfacción del usuario. El método originalmente consta de 50 preguntas[28], pero para este proyecto solo se han seleccionado algunas de las preguntas más relevantes (En el Anexo 4 se tienen todas las encuestas realizadas).

La encuesta se realizó a cuatro usuarios, uno por cada perfil: Administrador, Bodeguero, Solicitante y Supervisor.

El primer punto que se puede analizar es la percepción que tienen los usuarios acerca de la rapidez con la que se ejecutan las operaciones dentro del Sistema, para ello se ha tomado en consideración los resultados obtenidos en las preguntas 1, 4 y 13.

[28] En el Anexo 3 se tiene la encuesta en inglés utilizada por el método SUMI.

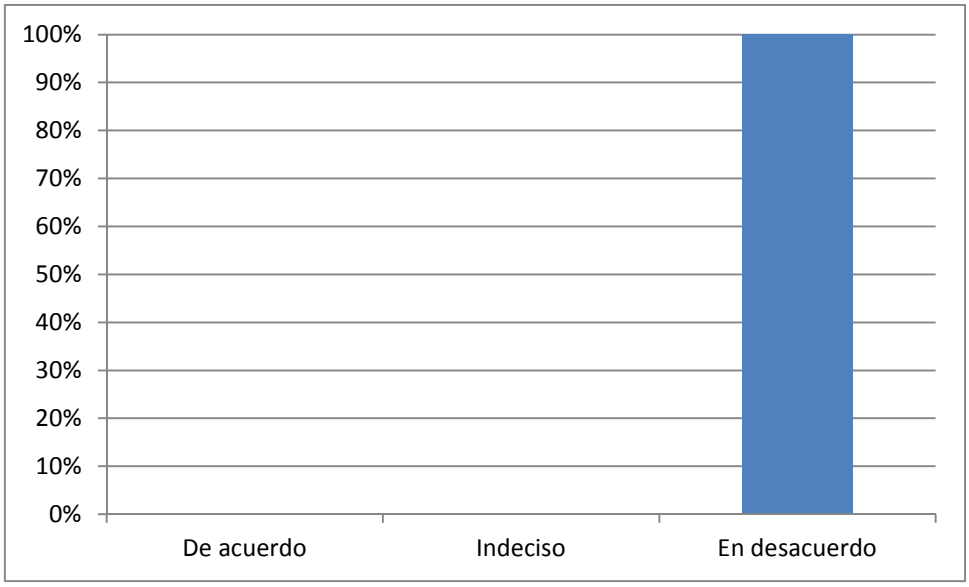


Figura 3.3 Pregunta 1 - Este software se ejecuta demasiado lento

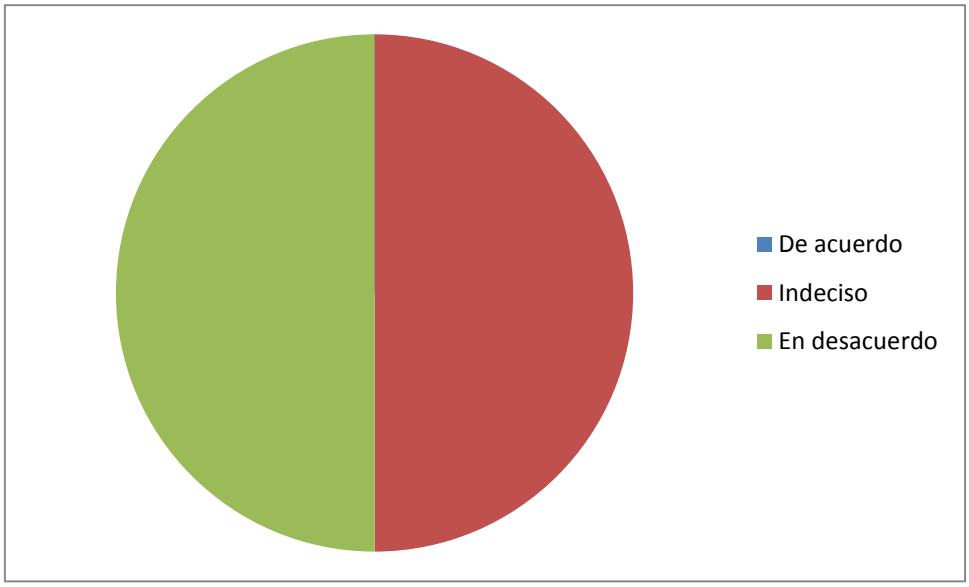


Figura 3.4 Pregunta 4 – Este software se ha detenido inesperadamente en algún momento

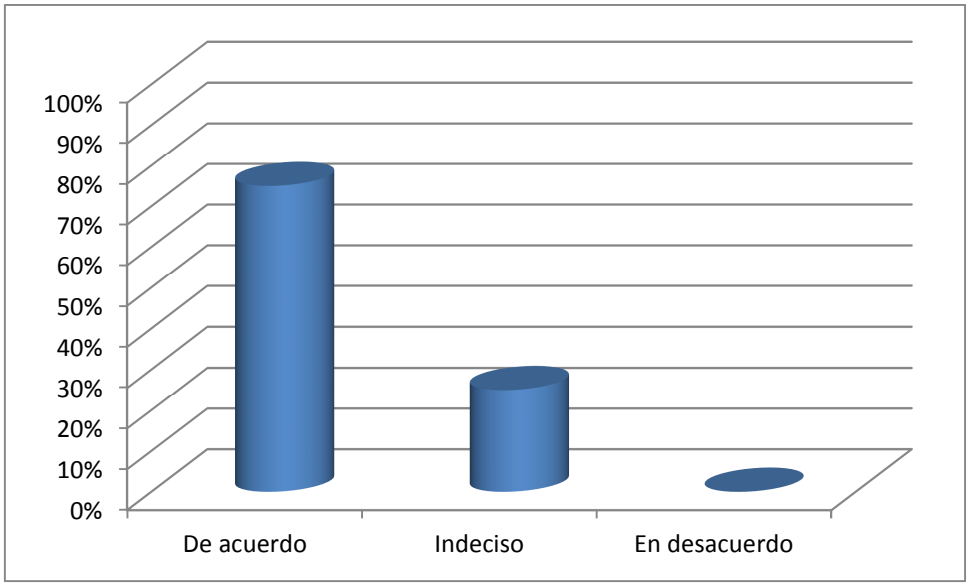


Figura 3.5 Pregunta 13 – La velocidad de este software es los suficientemente rápida

Como se puede observar en las Figuras 3.3, 3.4 y 3.5; las valoraciones obtenidas en cuanto a la velocidad son mayoritariamente positivas.

Otro punto que se ha analizado es la facilidad de uso del Sistema y para esto se han evaluado los resultados de las preguntas 5, 6 y 11 (Figuras 3.6, 3.7 y 3.8 respectivamente), las cuales hacen hincapié en este aspecto.

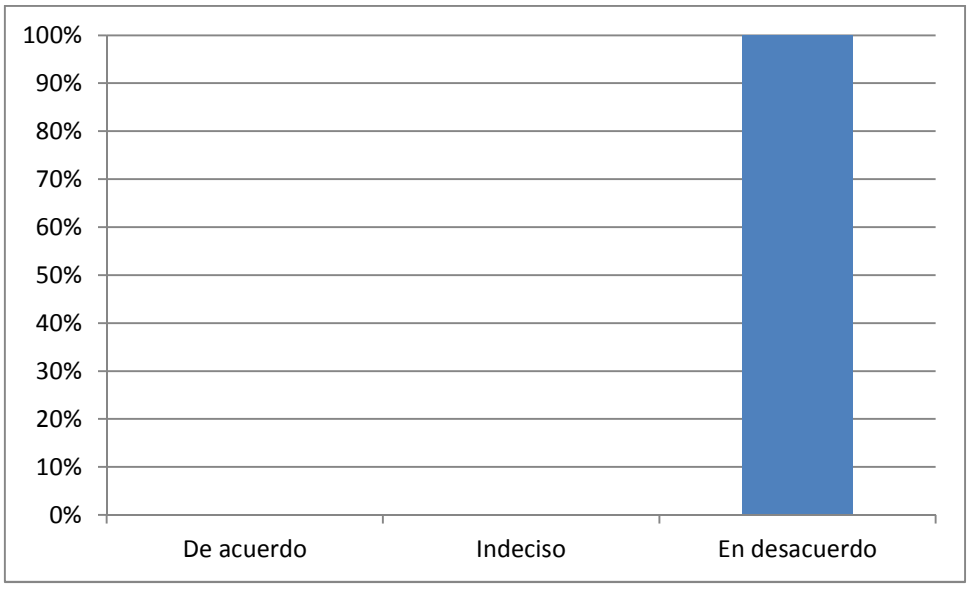


Figura 3.6 Pregunta 5 – Toma demasiado tiempo aprender la funcionalidad de este software

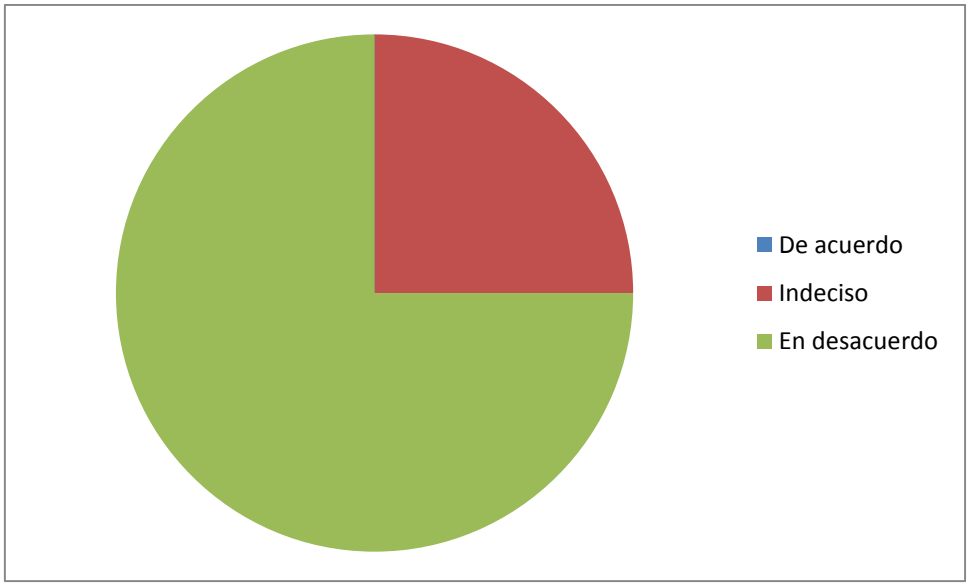


Figura 3.7 Pregunta 6 – A veces llego a un punto que no sé qué hacer con este software

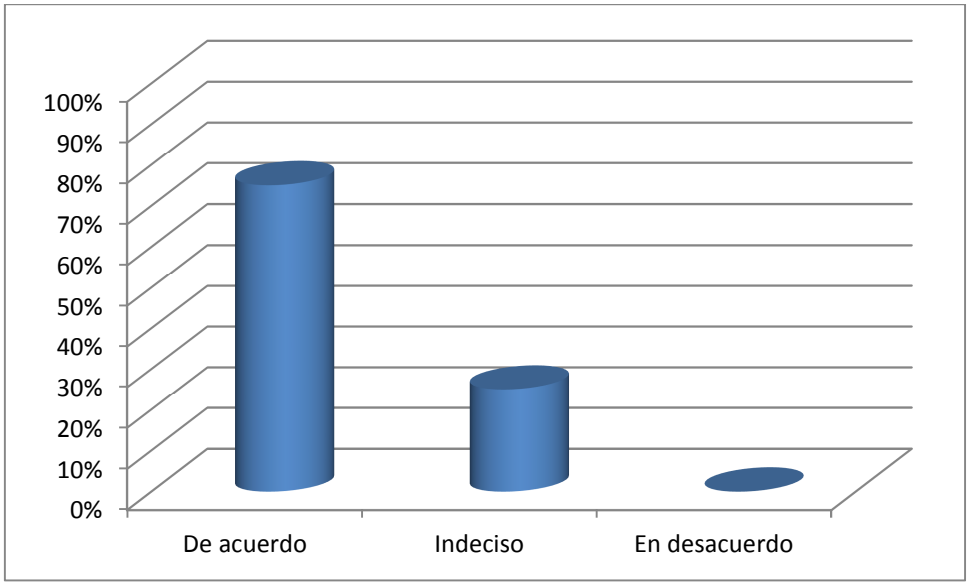


Figura 3.8 Pregunta 11 – Las tareas se pueden realizar de una manera simple utilizando este software

De igual manera, se tienen resultados positivos en cuanto a la facilidad de uso que presenta el software para los usuarios, ya que consideran que es muy simple su manipulación y se pueden aprender rápidamente las tareas que se pueden realizar con él.

Un aspecto muy importante que también se debe analizar es la satisfacción que tiene el usuario con respecto al diseño del sistema en lo referente a su interfaz y a la manera en que presenta la información. Para este punto se han considerado las preguntas 3, 8 y 15 como las más relevantes para indagar en ello.

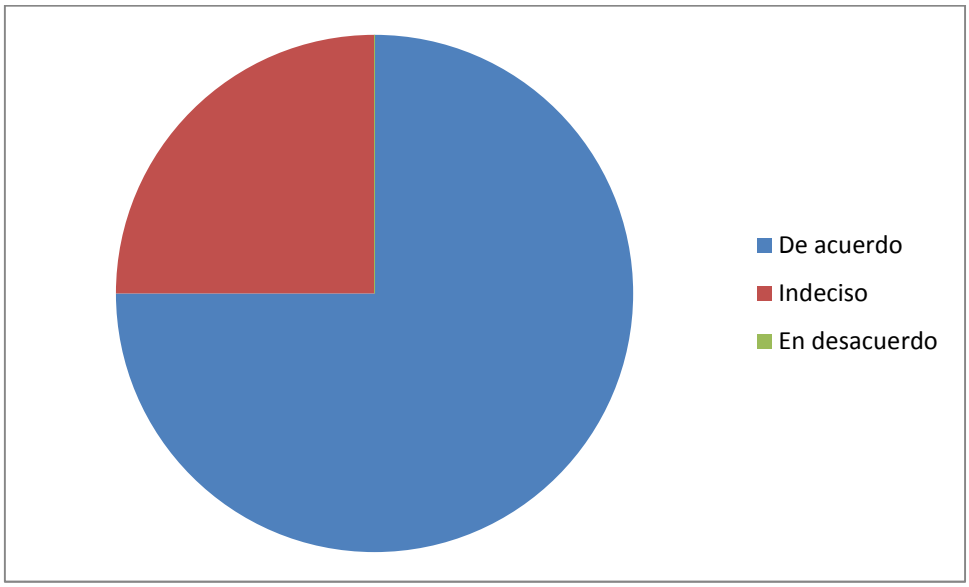


Figura 3.9 Pregunta 3 – Las instrucciones y los avisos que se presentan son de ayuda

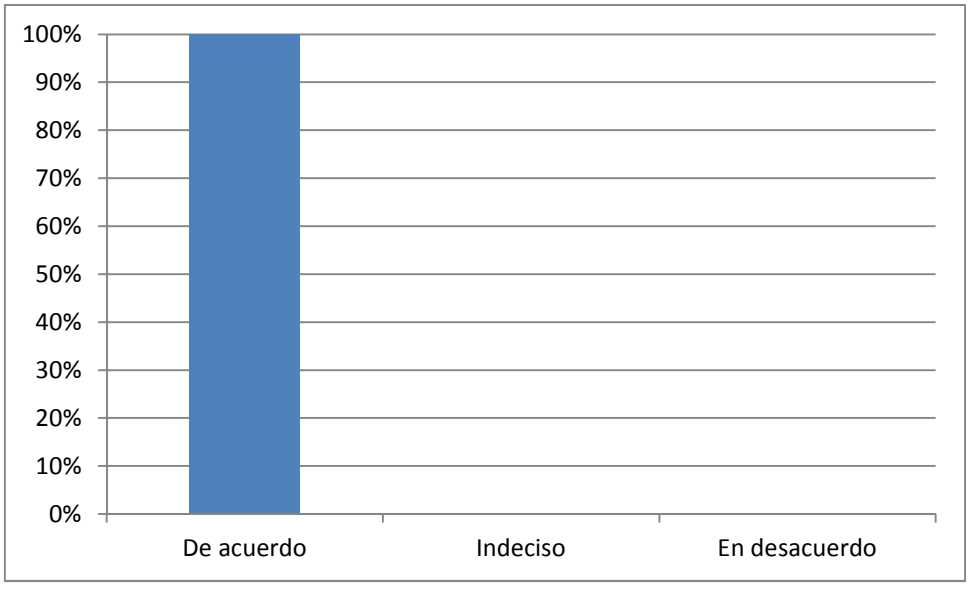


Figura 3.10 Pregunta 8 – La información presentada es clara y comprensible

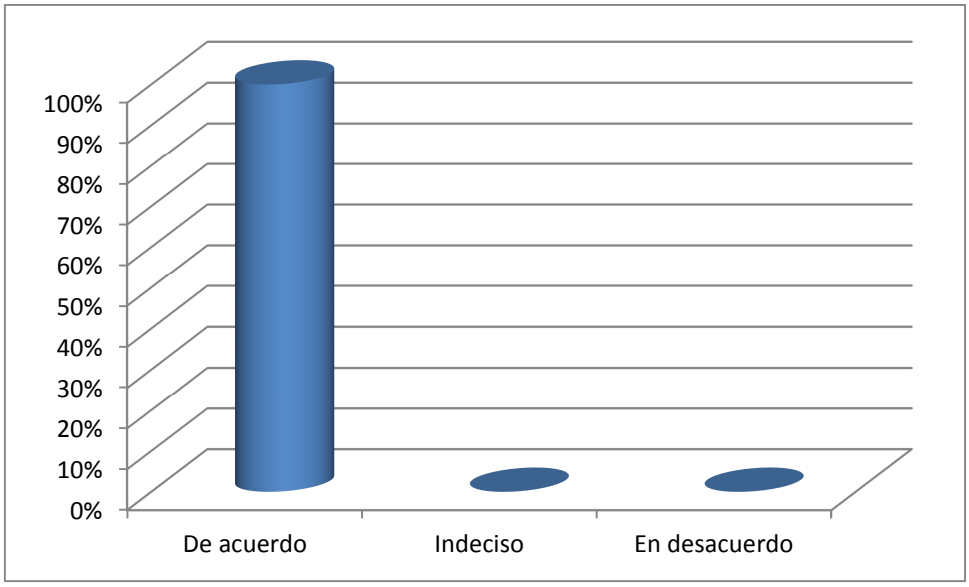


Figura 3.11 Pregunta 15 – La organización de los menús parece bastante lógica

Los gráficos de las figuras 3.9, 3.10 y 3.11 muestran claramente que a los usuarios encuestados les parece que los elementos de la Aplicación tienen una organización adecuada y que la información presentada es inteligible.

Por último, no se puede dejar de lado el análisis de la importancia que tiene el Sistema de Inventarios para los usuarios que manejan este proceso dentro de la Institución. En este caso, se debe tomar en cuenta la pregunta 27 de la encuesta, cuyo enunciado y resultados se los muestra en la Figura 3.12.

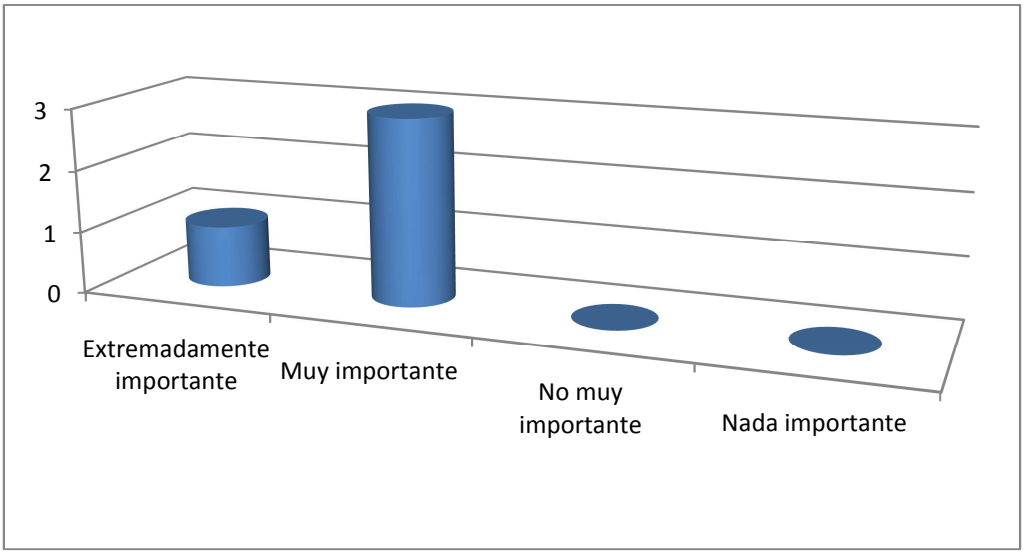


Figura 3.12 Pregunta 27 - ¿Qué tan importante es para usted el tipo de software que acaba de calificar?

De esta forma se demuestra que el Sistema de Inventarios es de gran importancia para la mayoría de usuarios, por lo que se espera que en un futuro pueda ser implantado dentro de la Institución.

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- La mayoría de Instituciones Públicas poseen manuales y rutinas de procedimientos similares en cuanto al manejo de los inventarios, por lo que el Sistema de Inventarios para las Instituciones Públicas del Ecuador se ajusta convenientemente a las necesidades de dichas Instituciones.

- La metodología de desarrollo escogida, Programación Extrema, es la más adecuada para este tipo de proyecto, ya que posee la flexibilidad necesaria para poder introducir requerimientos que no estuvieron claros desde un principio y sirve para llevar adelante el desarrollo de un sistema que no representa demasiada complejidad.

- Las pruebas continuas y la refactorización del código son dos aspectos fundamentales de la metodología de desarrollo que ayudan a mantener la calidad del software.

- La utilización de un framework de desarrollo simplifica mucho el trabajo de codificación, aportando con funcionalidad que permite implementar fácilmente aquellos aspectos del sistema que no forman parte de las reglas del negocio, pero que son importantes para la seguridad e integridad de la información.

- Los tres documentos básicos con los que se puede llevar adelante con éxito un proyecto desarrollado en XP son tarjetas donde se definen las Historias de Usuario, las Tareas de Programación y los Casos de Prueba.

- La utilización del Sistema de Inventarios simplifica el proceso de entrada y salida de bienes en las Instituciones Públicas, permite almacenar información que genera reportes automáticamente y de esta forma tomar decisiones que sirvan para depurar la forma en que se realizan las adquisiciones de bienes.

- El Sistema de Inventarios también podría ser utilizado en otro tipo de Instituciones, no necesariamente públicas, pero en este proyecto se ha

hecho énfasis en las necesidades de las Instituciones que pertenecen al Estado.

4.2 RECOMENDACIONES

- Llevar un historial comparativo de los tiempos estimados y los tiempos reales de implementación de las Historias de Usuario para ir afinando la estimación de los tiempos de proyectos futuros.

- Al utilizar una metodología que minimiza la generación de documentación en el diseño, es importante seguir estrictos estándares de codificación para que el mismo código fuente de la aplicación se convierta en el principal instrumento para comprender el funcionamiento del sistema.

- Distribuir el Sistema de Inventarios para las Instituciones Públicas del Ecuador como software libre, para que de esta forma se pueda modificar el código fuente y mejorarlo de acuerdo a las experiencias de las distintas instituciones, así se puede compartir este conocimiento que beneficie al perfeccionamiento de los procesos de manejo de inventarios.

- Se recomienda que se haga la implementación de firma electrónica para que los documentos e informes generados en el Sistema de Inventarios para las Instituciones Públicas del Ecuador tengan validez legal.

BIBLIOGRAFÍA

[1] ÁLVAREZ, José. ARIAS, Manuel. Método extreme programming. http://www.ia.uned.es/ia/asignaturas/adms/GuiaDidADMS/node61.html, 24 de septiembre de 2012.

[2] AMARO, Sarah. VALVERDE, Jorge. Metodologías Ágiles. Universidad Nacional de Trujillo. Trujillo, Perú. 2007

[3] ANÓNIMO. Instituciones Públicas del Ecuador. http://www.buenastareas.com/ensayos/Instituciones-Publicas-Del-Ecuador/2547232.html, 10 de abril de 2012

[4] APACHE SOFTWARE FOUNDATION. Frequently Asked Questions. http://wiki.apache.org/httpd/FAQ#What_is_Apache.3F, 6 de septiembre de 2012

[5] ASAMBLEA NACIONAL CONSTITUYENTE. Constitución Política de la República del Ecuador. 2008

[6] ASAMBLEA NACIONAL CONSTITUYENTE. Ley Orgánica del Sistema Nacional de Contratación Pública. 2008

[7] BARI, Ahsanul. SYAM, Anupom. CakePHP Application Development. Packt Publishing. Birmingham - Mumbai. 2008

[8] BECK, Kent. Extreme Programming Explained. Addison-Wesley Professional. 1999

[9] BECK, Kent. FOWLER, Martin. Planning Extreme Programming. Addison-Wesley Professional. 2000

[10] CAKE SOFTWARE FOUNDATION. Testing (Cookbook 1.3). http://book.cakephp.org/1.3/en/The-Manual/Common-Tasks-With-CakePHP/Testing.html, 4 de febrero de 2013

[11] CONTRALORÍA GENERAL DEL ESTADO. Manual General de Administración y Control de los Activos Fijos del Sector Público. 1996

[12] CONTRALORÍA GENERAL DEL ESTADO. Reglamento General de Bienes del Sector Público. 2006

[13] CONVERSE, Tim. MORGAN, Clark. PARK, Joyce. PHP5 and MySQL Bible. Wiley Publishing, Inc. Indianapolis. 2004

[14] CORTIZO Pérez, José Carlos. RUIZ Leyva, Miguel. EXPÓSITO, Diego. eXtremme Programming. AINetSolutions Technical Report No. 01. Madrid. 2003

[15] FERNÁNDEZ, Jorge. Arquitectura LAMP. http://softwarelibrejorge.blogspot.com/2011/02/arquitectura-lamp.html, 18 de enero de 2013

[16] GOLDING, David. Beggining CakePHP: From Novice to Professional. Apress. 2008

[17] HUMAN FACTORS RESEARCH GROUP. What is SUMI? http://sumi.ucc.ie/whatis.html, 2 de abril de 2013

[18] INSTITUTO NACIONAL DE LA ADMINISTRACIÓN PÚBLICA. Organizaciones Públicas. Tomo 1. Guía Temática Básica. Buenos Aires. 1997

[19] LABORATORIO NACIONAL DE CALIDAD DEL SOFTWARE (INSTITUTO NACIONAL DE TECNOLOGÍA DE LA COMUNICACIÓN). Ingeniería del Software: Metodologías y Ciclos de Vida. León, España. 2009

[20] LETELIER, Patricio. Programación Extrema. http://www.slideshare.net/guest82ea27/seminario-metodologas-giles-y-xp-tema-3-extreme-programming, 24 de enero de 2013

[21] MARTÍNEZ, Rafael. Sobre Linux. http://www.linux-es.org/sobre_linux, 6 de septiembre de 2012

[22] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Objetivos. http://www.habitatyvivienda.gob.ec/objetivos/, 22 de marzo de 2013

[23] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Organigrama del Ministerio de Desarrollo Urbano y Vivienda. http://www.habitatyvivienda.gob.ec/organigrama-del-ministerio-de-desarrollo-urbano-y-vivienda/, 28 de marzo de 2013

[24] MINISTERIO DE DESARROLLO URBANO Y VIVIENDA. Valores / Misión / Visión. http://www.habitatyvivienda.gob.ec/valores-mision-vision/, 28 de marzo de 2013

[25] NARANJO, Efraín. Caracterización de la Gestión en las Empresas Públicas. Revista Politécnica. Vol. 29(1): 18–28. 2010

[26] ORACLE CORPORATION. Why MySQL? http://www.mysql.com/why-mysql, 6 de septiembre de 2012

[27] RED HAT, INC. Y OTROS. Requerimientos de instalación recomendados. http://fedoraproject.org/es/, 28 de febrero de 2013

[28] ORACLE CORPORATION. NetBeans IDE 7.2 Features. http://netbeans.org/features/, 12 de septiembre de 2012

[29] SECRETARÍA NACIONAL TÉCNICA DE DESARROLLO DE RECURSOS HUMANOS Y REMUNERACIONES DEL SECTOR PÚBLICO – SENRES. Norma Técnica de Diseño de Reglamentos o Estatutos Orgánicos de Gestión Organizacional por Procesos. 2006

[30] THE JQUERY FOUNDATION. JQuery is a new kind of JavaScript Library. http://jquery.com/, 11 de septiembre de 2012

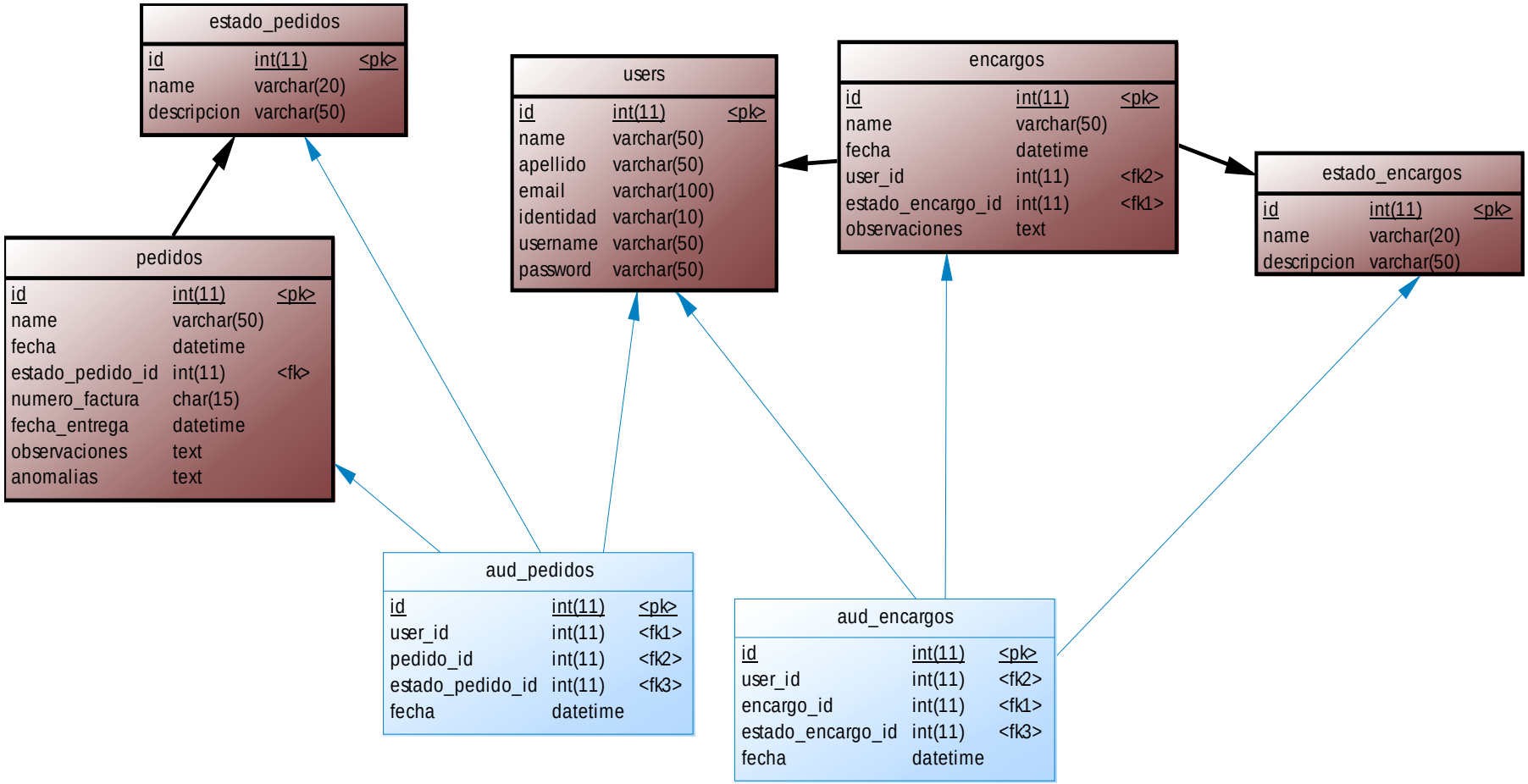
[31] VÁSQUEZ, Víctor Hugo. Organización Aplicada. Quito. 1985

[32] WORLD WIDE WEB CONSORTIUM (W3C). Guía Breve de CSS. http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo, 11 de septiembre de 2012

ANEXOS

ANEXO 1. TABLAS PARA AUDITORÍA

Las tablas utilizadas para auditoría están de color celeste. Aquí se puede apreciar la relación que tienen con las otras tablas.



ANEXO 2. SCRIPT DE BASE DE DATOS

Anexo Digital. Se encuentra en la ruta

Anexos/ScriptBD/estructura_datos_iniciales.sql

ANEXO 3. SOFTWARE USABILITY MEASUREMENT INVENTORY (SUMI)

Anexo Digital. Se encuentra en la ruta *Anexos/Encuestas/sumi.pdf*

ANEXO 4. ENCUESTAS

ANEXO 5. CERTIFICADO DE LA INSTITUCIÓN

ANEXO 6. MANUAL DE USUARIO

Anexo Digital. Se encuentra en la ruta *Anexos/Manual/Manual de Usuario.pdf*