ESCUELA POLITÉCNICA DEL EJÉRCITO

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CARRERA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

"DESARROLLO DE UN SISTEMA VÍA WEB PARA CONTROL DE PRODUCCIÓN EN LA GRANJA AVÍCOLA MARCO ANTONIO VIVANCO ÁLVAREZ, APLICANDO LA METODOLOGÍA FDD CON HERRAMIENTAS LIBRES"

Previo a la obtención del Título de:

INGENIERO EN SISTEMAS E INFORMÁTICA

POR: MAYRA ALEJANDRA SAMANIEGO PALLAROSO

SANGOLQUÍ, 07 de mayo de 2012

CERTIFICACIÓN

Certifico que el presente trabajo fue realizado en su totalidad por la Srta. MAYRA ALEJANDRA SAMANIEGO PALLAROSO como requerimiento parcial a la obtención del título de INGENIERO EN SISTEMAS E INFORMÁTICA.

Sangolquí, 04 de mayo de 2012

_____

Ing. César Villacis

DEDICATORIA

Este trabajo está dedicado a mis padres que con su apoyo y fortaleza me han hecho sentir que están a mi lado en todo momento a pesar de la distancia, cada palabra de amor arraigó en mi el deseo de seguir adelante y ser su motivo de orgullo.

Mayra Alejandra Samaniego Pallaroso

AGRADECIMIENTO

A Dios mi guía y soporte en todo, que conoce mis anhelos y bendice mi vida.

A mis padres por ser ejemplo de constancia y perseverancia, y por confortarme en los momentos de angustia.

A toda mi familia que de una u otra forma siempre estuvieron pendientes de mi crecimiento profesional.

Al mis directores del proyecto por ayudarme a conseguir un trabajo de excelente calidad.

Mayra Alejandra Samaniego Pallaroso

ÍNDICE DE CONTENIDO

Contenido

LISTADO DE TABLAS

LISTADO DE FIGURAS

RESUMEN

La Granja “Avícola Marco Antonio Vivanco Álvarez”, es una de las pioneras en el negocio de cría de pollos de engorde en la provincia de Manabí, desde 1992 hasta la actualidad, brindando un servicio de calidad y entrega de producción cárnica de excelencia a sus clientes. El producto de este desarrollo tiene como objetivo principal, automatizar su proceso de producción a través de un sistema web atractivo, intuitivo y eficaz que le permita gestionar eficientemente parte de la información de la granja.

El sistema web cuenta con los módulos de control de producción, ingresos, insumos y reportes. Esta aplicación web ha sido desarrollada utilizando la metodología de desarrollo ágil FDD (Feature Driven Development – Desarrollo Basado por Características), en conjunto con UML (Lenguaje Unificado de Modelado) para su diseño lógico.

Se usó la tecnología EJB, Enterprise Java Beans que permite el desarrollo simplificado de aplicaciones distribuidas, transaccionales, seguras y portables basadas en la tecnología Java. Todo esto ha sido construido e implementado sobre plataforma de software libre, reduciendo de este modo los costos de desarrollo y permitiendo al cliente obtener un retorno de la inversión en un tiempo no mayor a un año.

El software construido ha generado aceptación entre los usuarios de la granja, puesto que es una herramienta atractiva, completa y agradable, que facilita el seguimiento de cada uno de los parámetros de control de producción.

CAPÍTULO 1: INTRODUCCIÓN

1.1. Tema

Desarrollo de un sistema vía web para el control de producción en la granja avícola Marco Antonio Vivanco Álvarez, aplicando la metodología FDD con herramientas libres.

1.2. Introducción

La información y el conocimiento tecnológico se han convertido en fuente de progreso económico y de productividad; sin importar el área de trabajo en que se mire, cualquier persona que realice algún tipo de actividad económica está obligada cada vez más a utilizar la tecnología en sus negocios, esto debido a que las tecnologías de información (TI) impactan directamente en el flujo de trabajo y en el nivel de competitividad de una empresa.

Desde un email hasta un sistema de control especializado (ERP) marcan la diferencia a la hora de producir, y hoy en día resultaría casi imposible imaginar alguna actividad económica en donde no intervengan las tecnologías de información. El uso de herramientas de software libre para la implantación de soluciones informáticas es una de las opciones que más acogida ha tenido, debido al abaratamiento de costos y a su confiabilidad, permitiendo que las empresas decidan realizar inversiones en tecnología, que les brinde la oportunidad de mejorar su actividad productiva diaria y ofrecer mejores servicios a sus clientes.

En el Ecuador se está trabajando activamente en esta era de desarrollo tecnológico, cada vez son más los profesionales que buscan estudiar y especializarse para poder ofrecer soluciones informáticas prácticas y de calidad. Las pequeñas y medianas empresas del sector avícola no son la excepción, este sector está invirtiendo en herramientas que faciliten el control de producción, y les permitan darse a conocer en el mercado.

Siguiendo esta línea de avance tecnológico y para estar al mismo nivel de competitividad que sus similares, la gerencia de la granja avícola "Marco Antonio Vivanco Álvarez" tiene la necesidad de implementar un software que permita manejar, revisar la información relevante y llevar un control real de la producción, tomando en cuenta factores personalizados, que satisfagan las perspectivas del nivel administrativo y apoyen al nivel medio; por este motivo se hace indispensable la construcción de una herramienta informática, que permita estructurar un sistema de control de producción eficaz y eficiente.

Este proyecto está orientado a proporcionar esa solución informática, utilizando métodos, técnicas de ingeniería de software, y herramientas de software libre que logren el producto en funcionalidad, oportunidad y costo.

1.3. Planteamiento del Problema

Actualmente la granja avícola Marco Antonio Vivanco Álvarez no cuenta con una herramienta de software para llevar el control de sus procesos de producción y que sirva para la toma de decisiones. Los reportes requeridos por

el nivel gerencial son realizados manualmente. No hay acceso en línea a la información de producción actualizada.

1.4. Justificación

La producción avícola en Manabí es una de las áreas en que todavía el monitoreo de los procesos de control se los realiza de forma manuscrita, en algunos casos con herramientas computacionales, pero muy limitadas; lo que ocasiona que no se tengan datos estadísticos de las producciones para la toma de decisiones a nivel administrativo, y que no se puedan implementar mejoras en los procesos.

Actualmente en el mercado existen algunas opciones de sistemas computacionales para el área avícola, pero son muy genéricos, y no se adaptan a las necesidades de las granjas dedicadas a áreas específicas de producción como: ponedoras, pollos de engorde, incubadoras, etc., quedando módulos subutilizados, de tal manera que la implementación de uno de estos sistemas no es considerado por los gerentes como una inversión, sino como un elevado gasto. Por otro lado, los sistemas que se ofertan trabajan sobre plataformas privativas, haciendo que se incurra en un gasto adicional, al tener que adquirir licencias para el sistema operativo y base de datos.

La granja avícola “Marco Antonio Vivanco Álvarez”, es una de las tantas en Manabí que se ha quedado rezagada frente a sus similares de otras provincias, debido a la falta de implementación de tecnología en sus procesos.

El presente proyecto va dirigido al control de producción en la cría de pollos de engorde, y busca automatizar el flujo de procesos mediante herramientas de software libre, proveyendo una opción muy factible de implementar en términos monetarios.

La realización de este sistema en la institución auspiciante será el punto de partida para su implementación en el resto de la provincia, incrementando el nivel de competitividad de las granjas de este sector frente al resto del país.

1.5. Objetivos

1.5.1. Objetivo General

Desarrollar un sistema vía web para control de producción en la granja avícola “Marco Antonio Vivanco Álvarez”, aplicando la metodología FDD con herramientas libres, que ofrezca a los usuarios soluciones para optimizar procesos.

1.5.2. Objetivos Específicos

Diseñar una herramienta administrativa que facilite el acceso rápido y actualización de la información, a través de manejo de contenidos.

Desarrollar una aplicación que permita registrar y manipular la información generada durante todo el proceso de producción.

Proporcionar una herramienta que facilite la generación de reportes en los diferentes aspectos de la producción.

Aplicar la metodología de desarrollo ágil FDD - Feature Driven Development, en el desarrollo del sistema.

Elaborar el documento de especificación de requerimientos de software utilizando la norma IEEE830.

1.6. Alcance

El sistema de Control de producción de la granja avícola “Marco Antonio Vivanco Álvarez” considerará el siguiente alcance:

- *Ingresos,* que permitirá registrar la información de galpones, lotes, empleados y medicinas.
- *Control de Producción*, en donde se registrarán los diferentes parámetros zootécnicos generados durante el proceso de producción: índice de mortalidad, ganancia de peso promedio, control de medicaciones, control alimenticio y gastos.
- *Insumos,* que permitirá ingresar la información de las diferentes provisiones de alimento y medicinas en la granja.
- *Reportes*, que brindará una interfaz personalizada para revisar la información de acuerdo a los requerimientos del área administrativa.

CAPÍTULO 2: MARCO TEÓRICO

2.1. Agilísmo

Agilísmo es entregar valor al cliente, trabajando en equipo, haciendo que el cliente sea parte de ese equipo y participe activamente para llegar al objetivo.

El Agilísmo no busca trabajadores perfectos sino personas con actitud para lograr convertirse en un equipo ágil, en el que se maneje colaboración y confianza, de tal manera, que los procesos no solo sean buenos hoy, sino que sean mejores mañana y no queden obsoletos.

2.1.1. Origen del Agilísmo

El Agilísimo nace como alternativa a las metodologías formales a las que se consideraba excesivamente “pesadas” y rígidas por su carácter normativo. En marzo de 2001 diecisiete críticos de los modelos de mejora del desarrollo de software basado en procesos, convocados por Kent Beck: *Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas*, se reunieron en Salt Lake City para tratar sobre técnicas y procesos para desarrollar software, en la reunión se acuñó el término “Métodos Ágiles” aplicados al desarrollo de software.

Su objetivo fue proponer los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo de la construcción del proyecto. De allí nació *The Agile*

Alliance 3, una organización, sin ánimo de lucro, dedicada a promover los conceptos relacionados con el desarrollo ágil de software y ayudar a las organizaciones para que adopten dichos conceptos.[1]

2.2. Manifiesto Ágil

El manifiesto ágil es un documento que resume en cuatro valores y doce principios las mejores prácticas para el desarrollo de software, basados en la experiencia de los 17 industriales del software antes mencionados.

A continuación se presenta en detalle los valores y principios del manifiesto ágil:

2.2.1. Valores

1. *Individuos e interacciones sobre procesos y herramientas.*
2. *Software funcionando sobre documentación extensiva.*
3. *Colaboración con el cliente sobre negociación contractual.*
4. *Respuesta ante el cambio sobre seguir un plan.*

Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.[2]

2.2.2. Principios

1. *La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.*

[1] Agile Aliance 3, Recuperado 28 de noviembre, 2011, de www.agilealliance.org
[2] Manifiesto Ágil, Recuperado 28 de noviembre, 2011, de http://agilemanifesto.org

2. *Se acepta que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.*
3. *Se entrega software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.*
4. *Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.*
5. *Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.*
6. *El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.*
7. *El software funcionando es la medida principal de progreso.*
8. *Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.*
9. *La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.*
10. *La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.*
11. *Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.*

12. *A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.*[3]

2.3. Metodologías Ágiles en el Desarrollo de Software

Una metodología ágil es una colección de valores, principios y prácticas para modelar procesos, que pueden ser aplicados de manera simple y ligera. Las metodologías ágiles y las técnicas que van alrededor de ellas, promueven un cambio a la hora de afrontar proyectos, aportando calidad y productividad en tiempos de trabajo relativamente cortos.

Actualmente las metodologías ágiles son pensadas por los clientes para conseguir un retorno de la inversión (ROI) temprano. Las estadísticas indican que a las empresas que emprenden la creación de un producto online, no les interesa el cómo los desarrolladores lo construirán, solo piensan en recibir el producto terminado.

Mike Cohn en su libro *Succeeding with Agile,* indica algunos datos de la aplicación de las metodologías ágiles:

- El coste de los proyectos agiles es un 26% menor, en promedio. De los proyectos estudiados, el de menos ahorro logrado rebajó un 10% sus costes y el de mayor ahorro hasta un 70%.
- Los proyectos agiles salen al mercado hasta un 37% más rápido que los proyectos tradicionales.

[3] Principios del Manifiesto Ágil, Recuperado 28 de noviembre, 2011, de http://agilemanifesto.org

- La calidad de los proyectos agiles mejora un 63%, en promedio. Un 10% en el peor de los casos.
- Un 47% de los stakeholders consideran que, con los proyectos agiles, están más satisfechos. Un 31% se siente mucho más satisfecho.

Los métodos y prácticas ágiles persiguen la eficacia evitando el re-trabajo, realizando entregas frecuentes y manteniendo estrictos controles de calidad interna.

2.4. Metodologías Ágiles Vs. Metodologías Formales

No existe contradicción entre las posturas de desarrollo ágil y desarrollo formal, pero actualmente ya existe mayor evidencia de la gran utilidad de las metodologías ágiles, habiendo también campos en los que son necesarios los métodos formales.

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un producto más eficiente. Las metodologías ágiles motivan más a los equipos de trabajo, ya que se centran en las actividades más relevantes para la programación, que normalmente coinciden con las de mayor interés para los programadores, que son diseñar y programar.

Los procesos ágiles son una buena elección cuando se trabaja con requisitos desconocidos o variables, mientras que las metodologías

tradicionales son de alto uso cuando se asume que los requerimientos no tienen un alto grado de variación.

Para las metodologías formales si no existen requisitos estables, no cabe una gran posibilidad de tener un diseño constante y de seguir un proceso totalmente planificado. Los métodos ágiles, por el contrario, se predisponen a enfrentarse a cambios en los requerimientos en el proceso de desarrollo, haciendo que su proceso adaptativo sea mucho más efectivo que un proceso predictivo.

Las metodologías ágiles proporcionan una serie de pautas y principios que permiten la entrega del proyecto de manera menos complicada y más satisfactoria, tanto para los clientes como para los equipos de entrega, puesto que permite detectar errores en cada iteración, mientras que en las metodologías formales los errores no son detectados sino hasta la fase de pruebas, que es una de las finales.

En la Tabla 2.4.1 se muestran aspectos puntuales en los que difieren las metodologías formales de las ágiles.

Metodologías Formales	Metodologías Ágiles
Hacen énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto de	Se planifican iteraciones y se va desarrollando, probando y entregando cada una de ellas, a fin de que el cliente reciba resultados

software.	en periodos cortos.
Presta principal atención en los procesos, documentación y planificación.	Se da más importancia a la entrega de productos funcionales que a la documentación en exceso que al final puede no resultar útil
Los desarrolladores deben seguir el esquema de procesos planificado.	Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados.
Muestran cierta resistencia ante la presencia de cambios en el plan.	La capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan.

Tabla 2.4.1 Metodologías Formales Vs. Metodologías Ágiles
Fuente: Autor

2.5. Metodología de Desarrollo Ágil FDD

Se utilizará la metodología de desarrollo ágil FDD *(Feature Driven Development – Desarrollo Basado en Características),* en una arquitectura multicapas, que mezcla innovadoras combinaciones de prácticas reconocidas en el desarrollo ágil de software, propuestas en la perspectiva de la funcionalidad valorada por el cliente.

FDD, se estructura alrededor de la definición de características *(features)*, que representan la funcionalidad que debe contener el sistema, tienen un alcance lo suficientemente corto como para ser implementadas en pocas semanas y pueden ser agrupadas en diferentes jerarquías para representar aspectos en común del negocio.

Al igual que todos los procesos de desarrollo de software definidos, FDD se basa en un conjunto básico de "Buenas Prácticas". Cada práctica complementa y refuerza a las demás.[4]

A continuación se explica el ciclo de desarrollo de FDD basado en el libro *A Practical Guide to Feature-Driven Development,* Stephen R. Palmer – John M. Felsing, FDD Driven Development, 2002.

Las buenas prácticas que componen FDD son:

- Modelado global
- Desarrollo por características
- Propiedad individual sobre las clases (código)
- Equipos de trabajo por características
- Inspecciones
- Desarrollo continuo
- Gestión de Configuración
- Visibilidad de resultados.

[4] Feature-Driven Development. Recuperado: Noviembre 08, 2011, de http://www.pearsonhighered.com/assets/hip/us/hip_us_pearsonhighered/samplechapter/0130676152.pdf

FDD puede sin duda adaptarse a un conjunto de herramientas en particular y a equipos de personas con diferentes niveles de experiencia; y es este nivel de flexibilidad que hace que esta metodología sea relativamente fácil de adoptar dentro de una organización.

2.5.1. Modelo Global

El modelo global consiste en la construcción de diagramas de clases que representan los objetos importantes dentro del dominio del problema y las relaciones entre ellos, con la diferencia de que en este diagrama de clases se debe incluir las relaciones de generalización/especialización y operaciones que especifican el comportamiento de los objetos. Además es usual complementar los diagramas de clase con una serie de diagramas de secuencia de alto nivel que representen en forma explícita cómo los objetos interactúan entre ellos para cumplir con sus responsabilidades.

2.5.2. Desarrollo por Características

Una vez que se han identificado las clases en el modelo global, se puede diseñar e implementar cada una a la vez. La mayoría de metodologías utilizan casos de uso, descripción de casos de uso e historias de usuario, para redactar sus documentos de especificación funcional; al contrario, en FDD se sustituye todo lo anterior por una lista características funcionales de valor para el cliente, asegurándose de que los requisitos se expresen en un lenguaje de fácil entendimiento. A este documento se le denomina client-valued functions *(funciones valoradas por el cliente),* o features *(características).*

Las características son suficientemente pequeñas como para ser exitosamente implementadas en máximo dos semanas o menos, de tal manera que los clientes vean avances en forma frecuente.

2.5.3. Propiedad Individual sobre las Clases

La propiedad sobre una clase denota quién es el responsable del contenido de esa clase *(código)*. En FDD un desarrollador es designado propietario sobre un conjunto de clases del modelo global de objetos de dominio.

2.5.4. Equipos de trabajo por Características

La implementación de una característica involucra más de una clase, con sus respectivos propietarios de clase, por lo tanto, el responsable de cada característica necesitará la coordinación y esfuerzo de varios desarrolladores *(un trabajo de líder de equipo).* Alrededor de cada líder se van formando los equipos de trabajo, pudiendo cambiar los miembros de los equipos cada vez que se demande, de tal manera que el grupo esté conformado por los responsables de las clases que se necesitan para cumplir con la característica.

2.5.5. Inspecciones

El principal objetivo de las inspecciones es la detección de defectos; su aplicación conlleva beneficios como la transferencia de conocimiento y la aplicación de estándares de programación.

La combinación de los grupos de trabajo con las inspecciones, hace que la responsabilidad recaiga en todos los miembros del grupo. El nivel de

formalidad de cada inspección depende del grado de impacto del diseño y el código hacia afuera.

2.5.6. Construcción Continua

Significa tener intervalos en los que se unan todas las clases y características desarrolladas por los diferentes grupos así como también las librerías y componentes de las cuales dependen, y construir el sistema completo.

Realizar la construcción regularmente ayuda a detectar con tiempo los posibles errores de integración y asegura que se tenga siempre un sistema actualizado para mostrar al cliente.

2.5.7. Gestión de Configuración

La gestión de configuración depende de la naturaleza y la complejidad del software que se está desarrollando, FDD recomienda que no solo el código tenga versiones, sino también los documentos de requerimientos, diagramas de diseño y análisis, los casos de prueba, y en fin cualquier artefacto que pueda ser usado y mantenido durante el proceso de desarrollo. Además cualquier cambio en los procesos o ajustes que puedan suscitarse en la etapa de desarrollo y mantenimiento también deben ser versionados.

2.5.8. Visibilidad de Resultados

El contar con reportes de avance, permite a los líderes de grupo saber la rapidez con la que los desarrolladores añaden nuevas funcionalidades, y les provee el conocimiento suficiente para dirigir el proyecto correctamente.

El principal objetivo de la metodología FDD, es la entrega concreta de módulos, basándose en el desarrollo de software iterativo e incremental.[5]

Aplicando las buenas prácticas antes mencionadas, se realizaran los siguientes pasos secuenciales durante los cuales se diseña y se construye el sistema:

1. Desarrollo de un modelo global.
2. Construcción de una lista de características.
3. Planeación por característica.
4. Diseño por característica.
5. Construcción por característica
6. Pruebas por características

Las iteraciones se deciden en base a funcionalidades, que son pequeñas partes del software con significado relevante para el negocio. Una de las ventajas de centrarse en las funcionalidades del software es el poder formar un vocabulario común que fomente el diálogo fluido con los clientes, desarrollando entre ambos un modelo igual del negocio.

Al ser una metodología ágil, FDD ayuda a contrarrestar situaciones como: exceso en el presupuesto, errores en el software, o el hecho de no satisfacer los requerimientos del cliente en la entrega final del producto.

Espacio en Blanco puesto para completar la página

En la Figura 2.5.1 se muestra el marco de trabajo de FDD.

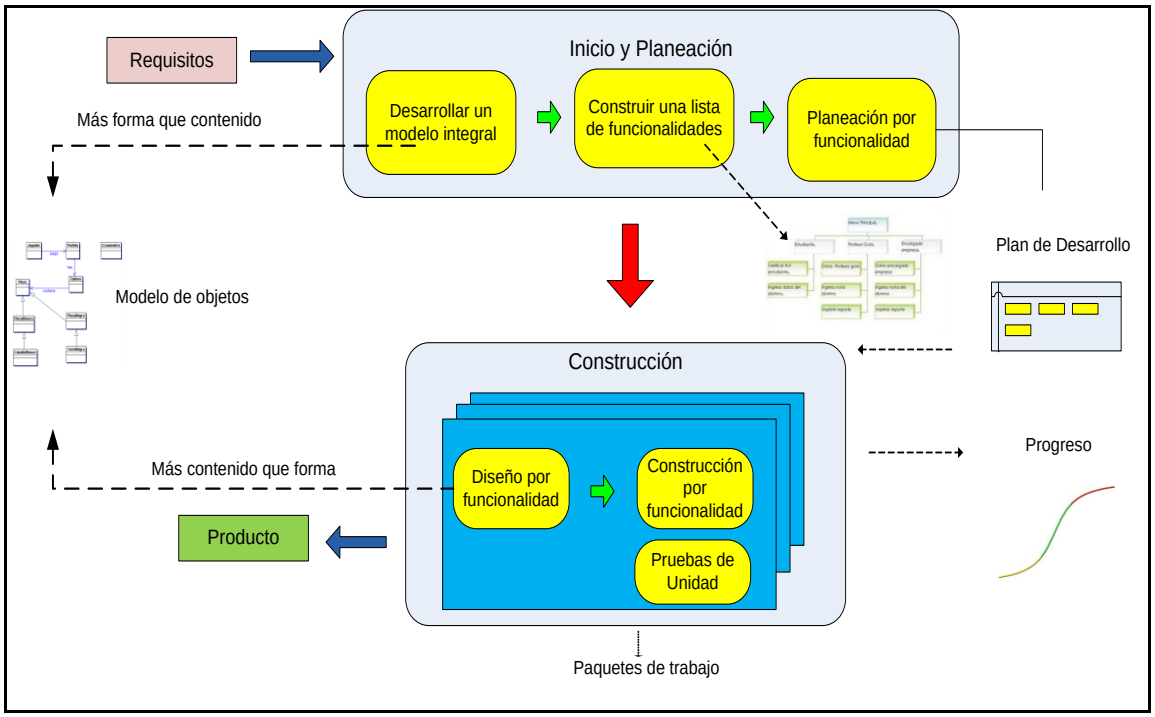


Figura 2.5.1 Marco de Trabajo - Metodología FDD
Fuente: http://www.koiosoft.com/en_us/wiki/agile-development

2.5.9. Ventajas y Desventajas

2.5.9.1. Ventajas

Dinamismo en la creación de grupos de trabajo, que agilita el desarrollo entregando resultados rápidos al cliente.

El control de versiones de todos los artefactos generados en el desarrollo, ayuda a mantener informado al equipo, sobre quién hizo el último cambio en el código.

El diseño de un modelo global al inicio del desarrollo, elimina posibles confusiones entre los miembros de los grupos y ayuda a comprender de mejor manera y con más rapidez las características que deben ser implementadas.

El hecho de no utilizar casos de uso, elimina el desperdicio de tiempo en documentar requerimientos que posiblemente cambien en la siguiente inspección del usuario.

La características son lo más pequeñas posible, de modo que puedan ser entregadas en máximo dos semanas.

Las inspecciones continuas y la participación activa del cliente permiten la detección de defectos y su corrección a tiempo.

El cliente percibe resultados en tiempos cortos.

2.5.10. Desventajas

Es necesario tener un conocimiento muy bien acentuado sobre los procesos del negocio y contar con un profesional de alta experiencia, para poder diseñar el modelo global de domino.

Muchas veces los propietarios de clases serán requeridos para el cumplimento de más de una característica al mismo tiempo en diferentes grupos.

Se necesita de experiencia y un exhaustivo análisis para la asignación de clases a los desarrolladores y para la formación de los grupos iniciales de trabajo.

2.6. Software Libre

El software libre se refiere a la libertad de copiar un programa y redistribuirlo a los demás; y a la libertad de modificar un programa de tal

manera que el usuario sea quien lo controle y pueda disponer del código fuente.[6]

Una vez obtenido, el software libre le permite al usuario gozar de cuatro libertades esenciales:

1. *La libertad de ejecutar el programa, para cualquier propósito.*
2. *La libertad de estudiar cómo trabaja el programa, y cambiarlo para que haga lo que el usuario quiera.*
3. *La libertad de redistribuir copias para que pueda ayudar al prójimo.*
4. *La libertad de distribuir copias de sus versiones modificadas a terceros. Si lo hace, puede dar a toda la comunidad una oportunidad de beneficiarse de sus cambios. El acceso al código fuente es una condición necesaria para ello.*[7]

2.7. Sistemas Distribuidos

Es un conjunto de computadoras separadas físicamente pero conectados entre sí por una red centralizada, se comunican a través de un protocolo prefijado por un esquema cliente servidor, coordinando sus acciones y el paso de mensajes para presentarse ante el usuario final como una gran máquina que ofrece alto rendimiento y velocidad.

2.8. W3C

El Consorcio World Wide Web (*Red amplia mundial, en español) (W3C)* es un consorcio internacional donde las organizaciones miembro, personal a tiempo completo y el público en general, trabajan conjuntamente para

[6] Definición de Sotware Libre, Recuperado 26 de noviembre de 2011, de GNU's Bulletin, Volume 1 Number 1, page 8
[7] Definición Software Libre, Recuperado Julio 14, 2011, de http://www.gnu.org/philosophy/free-sw.es.html.

desarrollar estándares Web. La misión del W3C es: Guiar la Web hacia su máximo potencial a través del desarrollo de protocolos y pautas que aseguren el crecimiento futuro de la Web. *(W3C, 2004).* [8]

2.9. Servicios Web

Un servicio web es un recurso de software constituido por un conjunto de protocolos y estándares; permite establecer comunicación e interoperabilidad entre diferentes máquinas de una red, con diferentes plataformas y entre programas distintos, esta comunicación se logra a través de la adopción de diversos estándares abiertos, como por ejemplo SOAP, WSDL, entre otros.

2.9.1. WSDL

WSDL son las siglas de Web Services Description Language *(Lenguaje de Descripción de Servicios, en español),* es un lenguaje basado en XML *(Lenguaje de Marcas Extensible)* para describir servicios web. Permite describir la interfaz pública de los servicios web; eso significa que detalla los protocolos y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo. WSDL se utiliza a menudo junto con SOAP y XML Schema. Un programa cliente se conecta a un servicio web y puede leer el WSDL, determinando así las funciones disponibles en el servidor.[9]

2.9.2. SOAP

SOAP al principio significaba Simple Object Access Protocol, luego fue Service Oriented Architecture Protocol, pero actualmente es simplemente

[8] W3C, Recuperado 25 de noviembre, 2011, de http://www.w3c.es/Consorcio/
[9] Definición de WSDL, Recuperado 26 de noviembre, 2011, de http://www.w3.org

SOAP, y es un protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML.[10]

El intercambio de mensajes *(datos XML)* se lo realiza sobre un ambiente descentralizado y distribuido. Actualmente, la especificación SOAP es mantenida por el XML Protocol Working Group de la W3C. Los mensajes SOAP, son independientes del sistema operativo, y pueden transportarse en varios protocolos de internet como SMTP, MIME y HTTP, siendo el más utilizado HTTP.

2.10. Java

Lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. Su sintaxis se basa en C y C++. Cuenta con un modelo de objetos simple que elimina la manipulación directa de herramientas de bajo nivel. [11]

El éxito de Java reside en que es un lenguaje sencillo e independiente de la plataforma, por lo que un programa hecho en Java se ejecutará igual en un PC con Windows que en una estación de trabajo basada en Unix. También hay que destacar su seguridad, su capacidad multihilo, su robustez y su integración con el protocolo TCP/IP, lo que lo hace un lenguaje ideal para Internet.

Espacio en Blanco puesto para completar la página.

[10] Definición de SOAP, Recuperado 25 de noviembre, 2011, de http://www.w3.org

[11] Definición de Java, Recuperado noviembre 26, 2011, de http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)#Entornos_de_funcionamiento

2.10.1. Java Virtual Machine

Máquina virtual de proceso nativo *(ejecutable en una plataforma específica)* que interpreta y ejecuta programas escritos en Java, específicamente puede interpretar el bytecode *(archivo .class)* generado al compilar en Java; puede estar implementada en software, hardware, una herramienta de desarrollo o un Web browser.

Lo que hace la máquina virtual de java es terminar de compilar el bytecode en lenguaje máquina para que la aplicación Java pueda ser ejecutada en un dispositivo específico. Básicamente se sitúa en un nivel superior al hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode, como el sistema sobre el que se pretende ejecutar.[12]

Las tareas principales de la máquina virtual de java son las siguientes:

- Reservar espacio en memoria para los objetos creados
- Liberar la memoria no usada
- Asignar variables a registros y pilas
- Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos
- Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java

Espacio en Blanco puesto para completar la página

[12] Definición de JVM, Recuperado noviembre 25, 2011, de http://es.wikipedia.org/wiki/M%C3%A1quina_virtual_Java

En la Figura 2.10.1 se muestra el funcionamiento de la Máquina Virtual de Java.

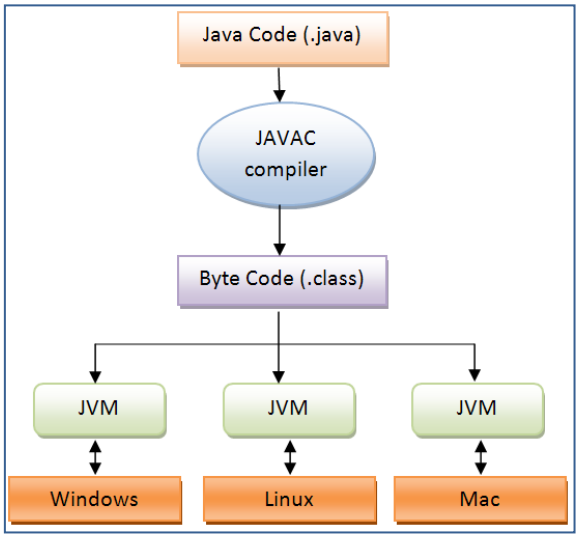


Figura 2.10.1 Funcionamiento de la Máquina Virtual de Java
Fuente: http://viralpatel.net/blogs/2008/12/java-virtual-machine-an-inside-story.html

2.10.2. JDK

El JDK *(Java Development Kit – Kit de Desarrollo de Java)* es un producto que permite crear aplicaciones en Java. Este paquete incluye un conjunto de herramientas para compilar, depurar, generar documentación e interpretar código escrito en Java.

2.10.3. JRE

El JRE (Java *RunTime Environment – Entorno de Desarrollo de Java)* es el software que se necesita tener instalado para poder ejecutar cualquier aplicación Java en cualquier sistema operativo. El JRE se compone de herramientas necesarias como la máquina virtual de java *(java.exe)* y el conjunto de librerías estándar de Java. El JDK incluye al JRE.

El usuario final usa el JRE como parte de paquetes de software o pluggins en un navegador Web.

2.10.4. JDBC

JDBC son las siglas *Java Data Base Connectivity*, es un paquete de Java que provee clases e interfaces que interactúan con bases de datos relacionales enviando sentencias SQL y procesando sus resultados.

Proporciona un mapeo directo de las tablas de la base de datos, en donde cada tabla es representada por una clase, cada fila de la tabla se convierte en una instancia de la clase, y cada columna corresponde a un atributo de dicha instancia.

En la Figura 2.10.2 se muestra la arquitectura JDBC.

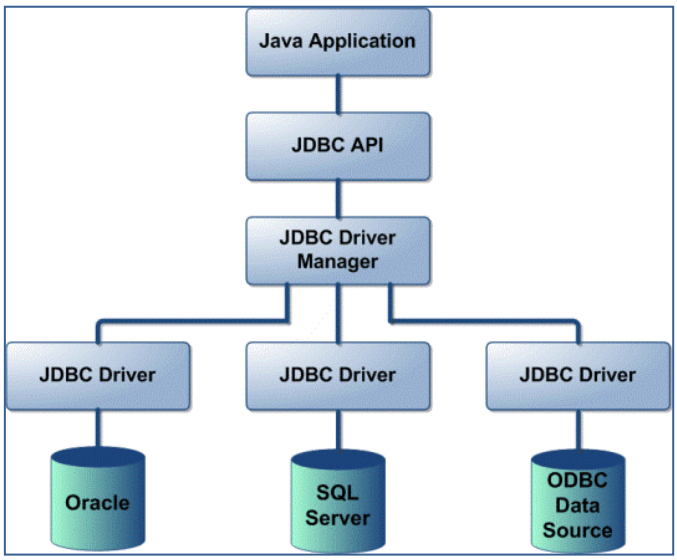


Figura 2.10.2 Arquitectura JDBC
Fuente: http://www.developersbook.com/jdbc/interview-questions/jdbc-interview-questions-faqs.php

La arquitectura JDBC está formada por dos capas:

1. *JDBC API*, que utiliza un controlador de base de datos específico para proveer una conexión transparente.
2. *JDBC Driver,* que gestiona una lista de controladores de base de datos y coordina las peticiones de conexión de una aplicación java con el controlador correcto para acceder a una base de datos específica.

Los componentes principales de JDBC son:

- *Controlador*: Link de comunicación hacia la base de datos, maneja toda la comunicación con la base de datos.
- *Conexión*: Interface que tiene todos los métodos para manipular la base de datos, representa el contexto de la comunicación a través de objetos.
- *Sentencia*: Encapsula una sentencia SQL que es pasada a la base de datos para su análisis, compilación y ejecución.
- *ResultSet***:** Conjunto de filas que representan el resultado obtenido de la ejecución de una sentencia.

2.10.5. EJB

EJB son las siglas de Enterprise Java Bean, es un componente desarrollado por Sun Microsystem que proporciona funcionalidades que trabajan del lado del servidor en sistemas multicapa. Un EJB se ejecuta en un contenedor de EJB y ofrece al desarrollador servicios que no necesita programar, como por ejemplo persistencia y seguridad.

Los EJB encapsulan la lógica de negocio en forma total y se constituyen especialmente para integrar la lógica de las empresas en sistemas distribuidos, cubriendo la necesidad de intermediar entre la capa web y diversos sistemas empresariales. Además permiten desarrollar componentes (*enterprise beans*) reutilizables y ensamblables en distintas aplicaciones. Los componentes EJB funcionan eficientemente gracias al trabajo del contendor EJB.

2.10.6. Contenedor EJB

Un contenedor EJB es un entorno en tiempo de ejecución que sirve de interfaz entre el bean y el servidor: Gestiona los componentes EJB, y proporciona una serie de servicios para que estos los utilicen.

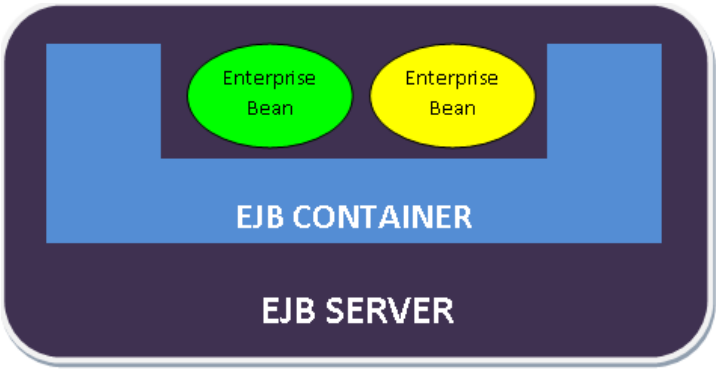


Figura 2.10.3 Arquitectura EJB
Fuente: htp://www.java.com

El servidor EJB, proporciona servicios y gestiona 2 recursos:

1. EJB container, entorno de ejecución
2. Enterprise Bean, almacena la lógica del negocio.

2.11. Action Script 3

Lenguaje orientado a objetos desarrollado originalmente por Macromedia Inc. (En la actualidad propiedad de Adobe Systems). Se basa en ECMAScript, el lenguaje internacional de programación estándar para scripting, desarrollado por el cuerpo de normas ECMA *(European Computer Manufacturers Association).* Se utiliza principalmente para el desarrollo de sitios web y software dirigido a la plataforma Adobe Flash Player, se lo implementa en las páginas web en forma de archivos SWF incrustados.

Con el uso de este lenguaje de programación, los desarrolladores pueden lograr una productividad y un rendimiento excelente con el contenido y las aplicaciones que se dirigen a Flash Player. ActionScript 3.0 está diseñado para abordar los siguientes objetivos:

- *Seguridad*: Soporta la seguridad para que los desarrolladores puedan escribir inequívoca y fácilmente código mantenible.
- *Simplicidad*: Es lo suficientemente intuitivo para que los desarrolladores puedan leer y escribir programas sin estar constantemente consultando un manual de referencia.
- *Rendimiento*: Permite a los desarrolladores escribir programas complejos que cumplan objetivos de manera eficiente y responsable.[13]

2.12. RTMP

RTMP *(Real-Time Messaging Protocol, Protocolo de Mensajería en Tiempo Real)* es un protocolo TCP que permite conexiones persistentes con baja latencia, fue diseñado para obtener un alto rendimiento en la transmisión de audio, vídeo y datos entre las tecnologías de la plataforma Adobe Flash, incluyendo Adobe Flash Player y Adobe AIR.

Inicialmente fue desarrollado por Macromedia, actualmente está bajo el dominio de Adobe, quién liberó una especificación para uso público *(RTMP Specification).* Para lograr transmitir tanta información como sea posible, RTMP divide los datos en fragmentos cuyo tamaño se negocia de forma

[13] Definición de Action Script 3, Recuperado 15 de marzo, 2012, de http:// http://www.adobe.com

dinámica entre el cliente y el servidor; otras veces se mantiene el tamaño sin cambios: 64 octetos de datos de audio, y 128 para datos de vídeo y otros tipos de datos. La codificación de la conexión y los comandos de control, RTMP la realiza utilizando AMF.[14]

2.13. AMF

AMF *(Action Message Format)* es un formato binario que sirve para serializar objetos gráficos de Action Script. Los objetos serializados se utilizan para conservar y recuperar el estado público de una aplicación a través de sesiones.

AMF codifica llamadas a procedimientos remotos en una representación binaria compacta que puede ser transferida a través de los protocolos: HTTP/HTTPS o RTMP/RTMPS usados por Flash Media Server. El formato en el que son convertidos los objetos y datos es mucho más compacto que otras representaciones como XML. AMF permite la creación de canales de mensajería basados en los servicios de red nativa de Flash Player y AIR. Los canales representan la conexión física cliente – servidor, son compartidos de forma predeterminada lo que permite comunicar diferentes destinos utilizando el mismo canal.

2.14. UML

UML (*Unified Modeling Language*, *Lenguaje Unificado de Modelado),* es el estándar más utilizado para representar y modelar la información con la que

[14] RTMP, Recuperado 16 de marzo, 2012, de http://www.adobe.com

se trabaja en las fases de análisis y, especialmente, de diseño de software. Sirve para el modelado completo de sistemas complejos, tanto en el diseño como para la arquitectura de hardware donde se ejecute.

Dentro del marco de trabajo de FDD se requieren al menos tres diagramas UML, reemplazando los casos de uso por una Lista de Características Evaluadas por el Cliente. Los diagramas UML que se van a modelar en el presente proyecto son: Diagrama de Clases y Diagramas de Secuencia.

2.14.1. Diagrama de Clases

Es el diagrama principal para el análisis y diseño, representa las entidades existentes en cualquier sistema orientado a objetos y ayuda a visualizar las relaciones involucradas entre ellas, logrando así la descripción de la vista estática del modelo o parte del modelo.

Un diagrama de clases está compuesto por los siguientes elementos:

- Clase: Elemento que encapsula toda la información de características y comportamientos que un objeto puede generar. En UML la clase se representa por un rectángulo con tres divisiones:

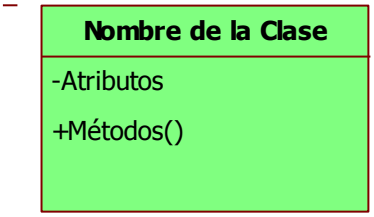


Figura 2.14.1 Estructura de una Clase
Fuente: Autor

- Atributos, Métodos y Visibilidad:

  Los atributos o características de una Clase pueden ser de tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

  Publico **(+)**, el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

  Privado **(-)**, el atributo sólo será accesible desde dentro de la clase.

  Protegido **(#)**, el atributo no será accesible desde fuera de la clase, pero si podrá ser accedido por métodos de la clase además de las subclases que se deriven.

- Relaciones de generalización, asociación y agregación.

  Generalización, se usa para indicar herencia, en donde una subclase hereda los métodos y atributos especificados por una Súper Clase.

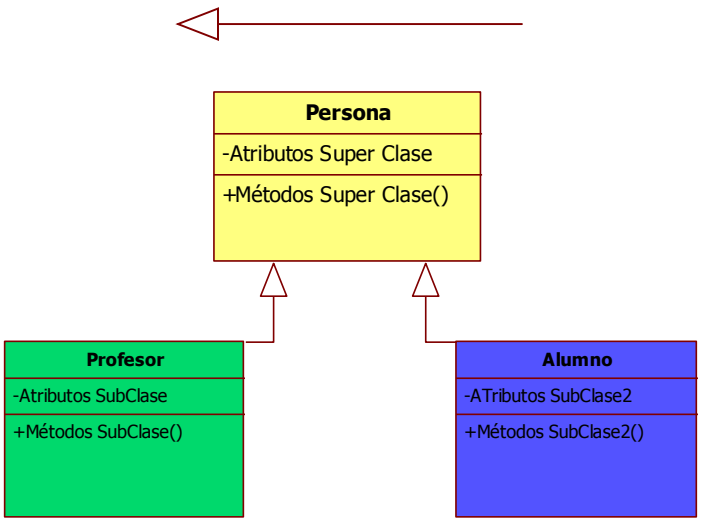


Figura 2.14.2 Generalización
Fuente: Autor

- Asociación, permite asociar objetos que colaboran entre sí. El tiempo de vida de un objeto no depende del otro.

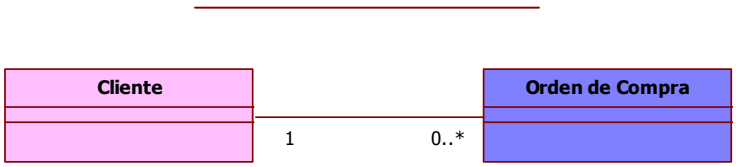


Figura 2.14.3 Asociación
Fuente: Autor

- Agregación, se usan para describir elementos que están compuestos de componentes más pequeños.

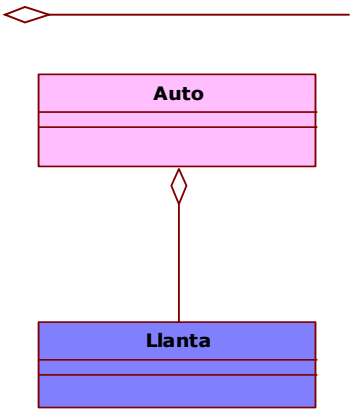


Figura 2.14.4 Agregación
Fuente: Autor

2.14.2. Diagrama de Secuencia

El diagrama de secuencia representa el núcleo del modelo dinámico. Representa la secuencia de mensajes entre las instancias de clases, componentes, subsistemas o actores. En el diagrama, el tiempo fluye hacia abajo y muestra el flujo de control de un participante a otro.

Espacio en Blanco puesto para completar la página.

2.14.2.1. Elementos del Diagrama de Secuencia

En la Figura 2.14.1 se muestra cómo interactúan los elementos de un diagrama de secuencia.

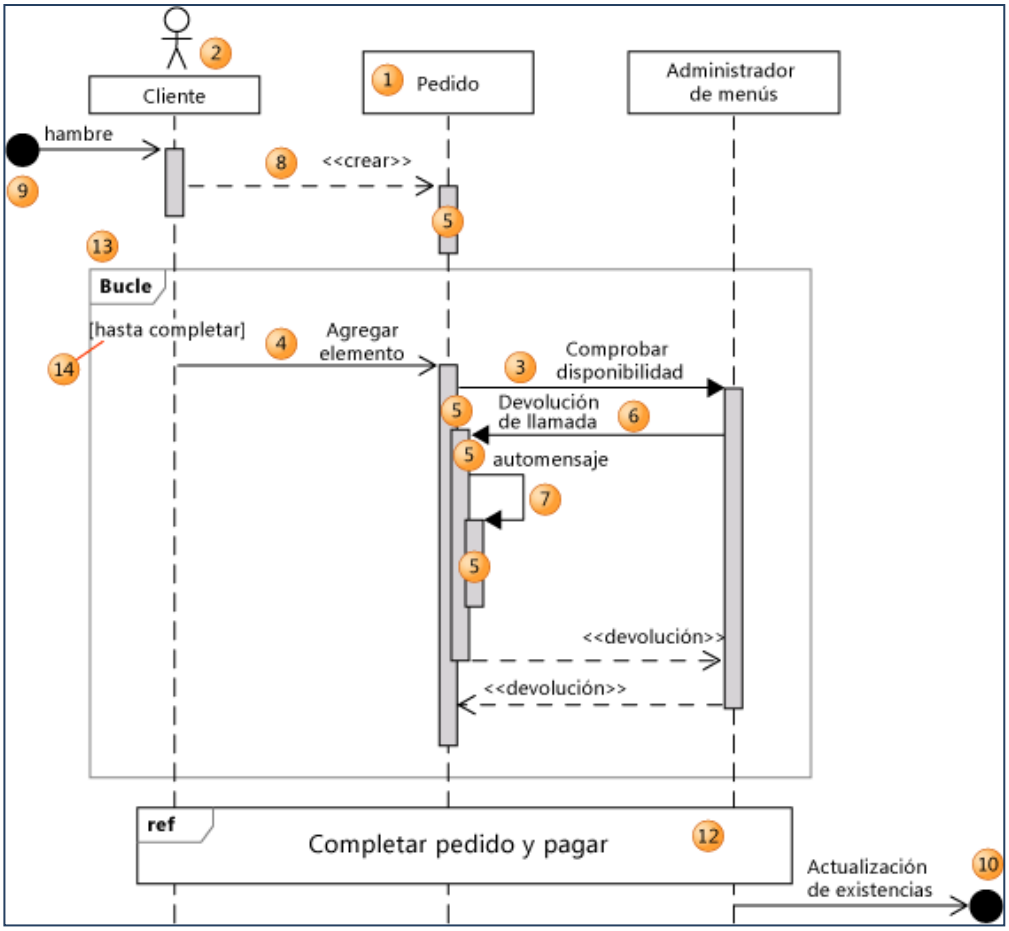


Figura 2.14.5 Elementos del Diagrama de Secuencia
Fuente: http://msdn.microsoft.com

En la Tabla 2.14.1 se indica la descripción de cada uno de los elementos que componen un diagrama de secuencia.

Forma	Elemento	Descripción
1	Línea de vida	Una línea vertical que representa la secuencia de eventos que se producen en un participante durante una interacción, mientras el tiempo avanza. Este participante puede ser una instancia de una clase,

		componente o actor.
2	Actor	Un participante que es externo al sistema que está desarrollando.
		Puede hacer que aparezca un símbolo de actor en la parte superior de una línea de vida estableciendo su propiedad Actor.
3	Mensaje sincrónico	El remitente espera una respuesta a un mensaje sincrónico antes de continuar. El diagrama muestra la llamada y el retorno. Los mensajes sincrónicos se utilizan para representar llamadas de función ordinarias dentro de un programa, así como otros tipos de mensaje que se comportan de la misma manera.
4	Mensaje asincrónico	Un mensaje que no requiere una respuesta antes de que el remitente continúe. Un mensaje asincrónico muestra sólo una llamada del remitente. Se utiliza para representar la comunicación entre subprocesos diferentes o la creación de un nuevo subproceso.
5	Incidencia de ejecución	Un rectángulo sombreado vertical que aparece en la línea de la vida de un participante y representa el período durante el que el participante está ejecutando una operación.
		La ejecución comienza donde el participante recibe un mensaje. Si el mensaje inicial es un mensaje

		sincrónico, la ejecución finalizará con una flecha de «devolución» al remitente.
6	Mensaje de devolución de llamada	Un mensaje que vuelve a un participante que está esperando la devolución de una llamada anterior. La aparición de ejecución resultante aparece encima de la existente.
7	Automensaje	Un mensaje de un participante a sí mismo. La aparición de ejecución resultante aparece encima de la ejecución de envío.
8	Crear mensajes	Un mensaje que crea un participante. Si un participante recibe un mensaje de creación, este debe ser el primer mensaje que recibe.
9	Mensaje encontrado	Un mensaje asincrónico de un participante desconocido o no especificado.
10	Mensaje perdido	Un mensaje asincrónico a un participante desconocido o no especificado.
11	Comentarios	Un comentario se puede adjuntar a cualquier punto de una línea de vida.
12	Uso de interacción	Agrega una secuencia de mensajes que se definen en otro diagrama. Para crear un Uso de interacción, haga clic en la herramienta y arrastre por las líneas de vida que desee incluir.
13	Fragmento combinado	Una colección de fragmentos. Cada fragmento puede agregar uno o más mensajes. Existen distintos tipos

		de fragmentos combinados. Para obtener más información, vea Describir el flujo de control con fragmentos de diagramas de secuencia de UML.
		Para crear un fragmento, haga clic con el botón secundario en un mensaje, elija Rodear con y, a continuación, haga clic en un tipo de fragmento.
14	Protección de fragmentos	Se puede utilizar para enunciar una condición relativa a si el fragmento se producirá.
		Para establecer la protección, seleccione un fragmento, seleccione después la protección y escriba un valor.
	Interacción	La colección de mensajes y líneas de vida que se muestra en el diagrama de secuencia. Para ver las propiedades de una interacción, debe seleccionarla en el Explorador de modelos UML.

Tabla 2.14.1 Descripción de los Elementos del Diagrama de Secuencia
Fuente: http://msdn.microsoft.com

2.15. Herramientas de Desarrollo

Para el desarrollo del presente proyecto se ha considerado la utilización de las siguientes herramientas de software libre:

2.15.1. Java Enterprise Edition

Java Platform, Enterprise Edition o JEE, es una plataforma de desarrollo diseñada para construir aplicaciones empresariales multicapas basadas en

componentes estandarizados, modulares y reutilizables, permitiendo así, manejar muchos aspectos de la programación de forma automática, logrando aplicaciones robustas que trabajen a través de internet.[15]

2.15.2. Framework JUnit

Framework de código abierto diseñado para realizar pruebas de unidad repetitivas en lenguaje de programación Java, de manera rápida y fácil. Permite a los desarrolladores construir gradualmente bancos de pruebas para medir el progreso y detectar efectos secundarios no deseados, obteniendo resultados inmediatos.

Utilizar un software de pruebas obliga al desarrollador a verificar el resultado esperado de una porción de código, actualizarlo y comprobar el impacto generado de forma inmediata. JUnit promueve la idea de: Testear – Codificar – Testear, de tal manera que aumenta la productividad del programador, la estabilidad del código del programa, reduce el estrés y el tiempo de depuración.

2.15.3. Apache Tomcat

Servidor web y de aplicaciones, de código abierto, es mantenido y desarrollado por miembros de la Apache Software Foundation y voluntarios independientes. Gestiona solicitudes y es servidor de aplicaciones o contenedor de Servlets/JSP (Java Server Pages).

[15] JEE, Recuperado 27 de noviembre, 2011, de http://www.oracle.com

2.15.4. Eclipse

Proyecto de desarrollo de software de código abierto, cuyo objetivo es la construcción de herramientas integradas para el desarrollo de aplicaciones. [16] Brinda un entorno de desarrollo integrado multiplataforma que facilita la implementación de diferentes lenguajes de programación, y, proporciona apertura para el desarrollo y extensión de los sistemas.

El SDK *(Software Development Kit)* de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java.

2.15.5. LifeRay

Plataforma web corporativa de código abierto, diseñada para desarrollar soluciones empresariales con resultados inmediatos y valor a largo plazo.[17] Incluye una amplia gama de funcionalidades, tales como:

- Diseño de interfaces de usuario de manera fácil y ágil.
- Framework de integración de aplicaciones.
- Construcción rápida de sitios.
- Creación de páginas con solo un clic.
- Publicación de contenidos basada en roles.
- Personalización de usuarios.
- Calendario compartido.
- Anuncios y alarmas, entre otros.

[16] Eclipse, Recuperado 19 de marzo, 2012, de http://www.eclipse.org/
[17] LifeRay, Recuperado 19 de marzo, 2012, de http://www.liferay.com

Liferay ayuda a las empresas a desarrollar soluciones sólidas que ofrecen los resultados deseados en múltiples ámbitos:

- Portales de autoservicio
- Espacios de trabajo para el intercambio de conocimiento
- Sitios Web 2.0 dinámicos
- Redes sociales que generan ingresos
- Integración con aplicaciones corporativas

2.15.6. Hibernate

Herramienta de Mapeo objeto-relacional para la plataforma Java, distribuida bajo los términos de la licencia GNU LGPL , que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos XML o anotaciones en los beans de las entidades.

2.16. Adobe Flex

Herramienta de código abierto, muy productiva para la creación y el mantenimiento de aplicaciones web RIA *(Rich Interface Aplication)* que se implantan coherentemente en los principales exploradores, equipos de escritorio y sistemas operativos.

2.17. My SQL

Sistema de gestión de bases de datos relacional, multihilo que permite soportar alta carga de forma muy eficiente.

Las principales características de este gestor de bases de datos son las siguientes:

1. Aprovecha la potencia de sistemas multiprocesador, gracias a su implementación multi-hilo.
2. Soporta gran cantidad de tipos de datos para las columnas.
3. Dispone de API's en diversos lenguajes (C, C++, Java, PHP, etc).
4. Gran portabilidad entre sistemas.
5. Soporta hasta 32 índices por tabla.
6. Gestión de usuarios y contraseñas, manteniendo un muy buen nivel de seguridad en los datos.

MySQL brinda rendimiento, alta fiabilidad y facilidad de uso, permite ahorrar tiempo y dinero en grandes volúmenes de sitios Web, sistemas críticos de negocio y software empaquetado. Se ejecuta en más de 20 plataformas, incluyendo Linux, Windows, Mac OS, Solaris, AIX de IBM, que le da el tipo de flexibilidad con alto grado de control[18]

2.18. Star UML

Herramienta case de código abierto, utilizada para el modelado de datos con UML, arquitectura, metadatos, etc., para el diseño y análisis de software de manera fácil y ágil.

[18] MySQL, Recuperado 20 de marzo, 2012, de http://www.mysql.com

CAPÍTULO 3: DESARROLLO DEL SISTEMA

3.1. Modelo Global

FDD manda a que se diseñe un modelo global para conocer completamente el contexto general y los requerimientos del sistema a desarrollar, definiendo los límites del software y la implicación de los aspectos sociales y organizacionales.

Con este modelo se debe lograr describir lo que requiere el cliente en lenguaje natural, de manera que se pueda establecer una base para la creación del diseño de software, y definir un conjunto de requisitos que se pueda validar una vez que se construye el software.

3.1.1. Descripción del Negocio

La granja avícola “Marco Antonio Vivanco Álvarez” está situada en la parroquia Rocafuerte del cantón Chone en la Provincia de Manabí. Desde hace 20 años su actividad puntual dentro del área avícola es la cría de pollos de engorde, para su posterior venta en pie.

La granja ha contribuido al crecimiento del área avícola en la zona norte de Manabí, con un promedio de ventas mensuales de 11500 pollos. Su actividad diaria se centra en el control de todo el proceso de producción hasta que los lotes estén listos para salir a la venta, tiempo que varía entre siete y ocho semanas.

El proceso de producción inicia con el ingreso de lotes de pollitos de un día de nacidos; cada lote se distribuye en galpones; y a partir de ahí empiezan las actividades del proceso de cría:

- Control de Alimentación.
- Control de Medicación.
- Control de Mortalidad.
- Control de Peso.
- Control de Vacunación.
- Control de Gastos

Las actividades durante la cría son ejecutadas por los obreros, supervisadas por los administradores de producción y se realizan hasta que el lote esté listo para salir a la venta. A más de ello los administradores de producción deben llevar el control de gastos, provisiones y asignación de obreros para trabajar en cada lote. Agregado al área de producción, también está la de contabilidad y gerencia, mismas que se encuentran ubicadas en la ciudad de Quito.

En la Figura 3.1.1 se muestra el proceso de producción de la granja.

El problema que presenta la granja es que a pesar de tener sus procesos de control en la producción ya definidos, la mayor parte se llevan registrados en cuadernos y unos pocos en Excel; no existe una fuente de consulta centralizada y puntual que facilite la toma de decisiones, y, no hay

acceso a datos históricos ni en los formatos necesarios para las áreas administrativas.

Se necesita contar con un software que facilite a los administradores de producción vigilar las actividades del proceso de cría; que permita controlar la distribución de insumos en los galpones, que provea acceso rápido a la información hacia las otras áreas administrativas y genere los reportes necesarios para su seguimiento.

En la Figura 3.1.2, se muestra una lluvia de problemas a causa de la necesidad tecnológica en la granja.

Además la investigación realizada indica que otras granjas de similar nivel que se dedican al mismo negocio, han mejorado el seguimiento del proceso de producción, gracias al uso de herramientas de software puntuales para su trabajo; por lo tanto, este desarrollo responde a ese problema con la propuesta de un sistema web administrable que sirva de soporte a los empleados y automatice actividades claves dentro de la granja.

Espacio en Blanco puesto para completar la página.

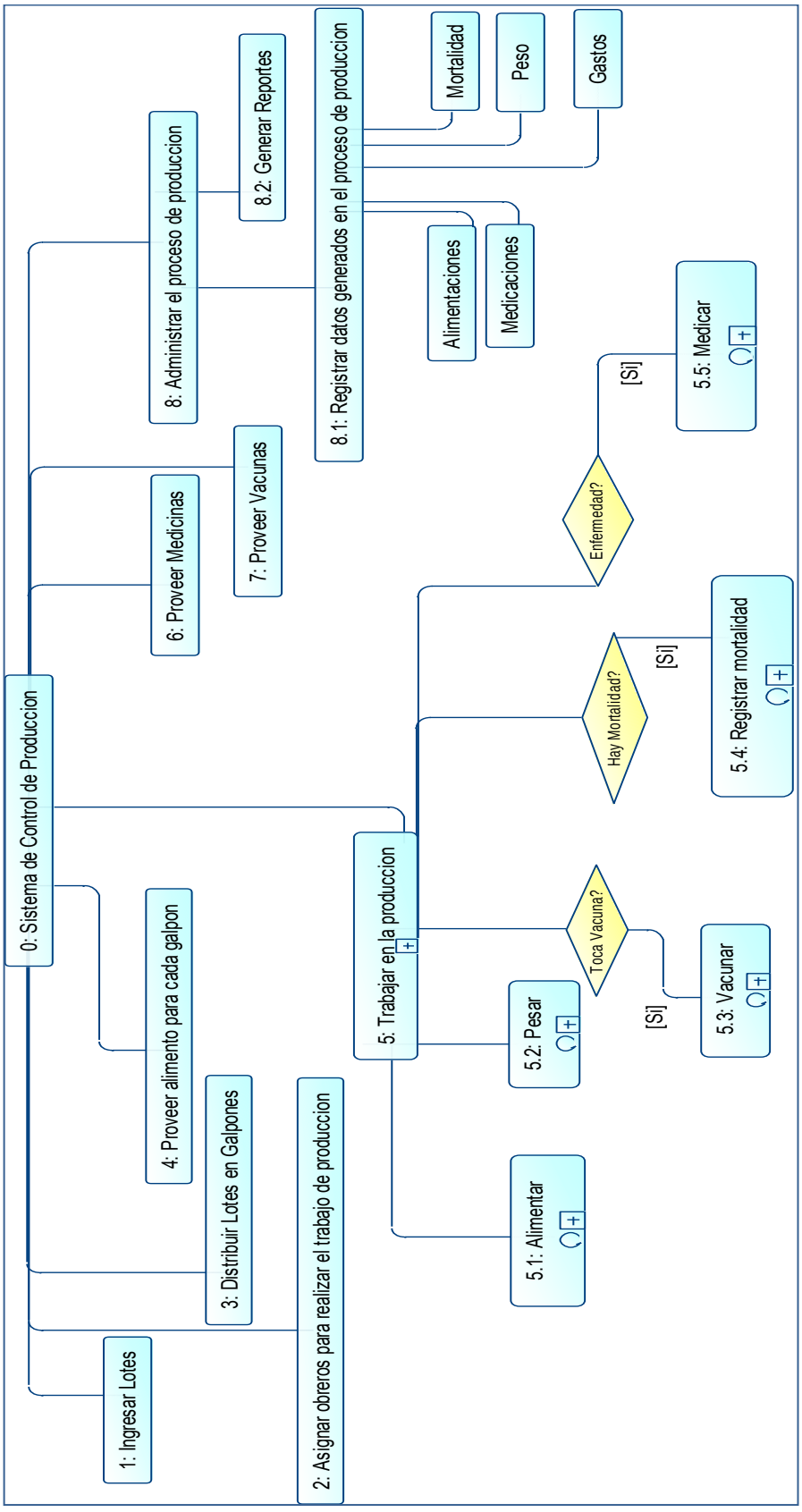


Figura 3.1.1 Proceso de producción de la granja
Fuente: Autor

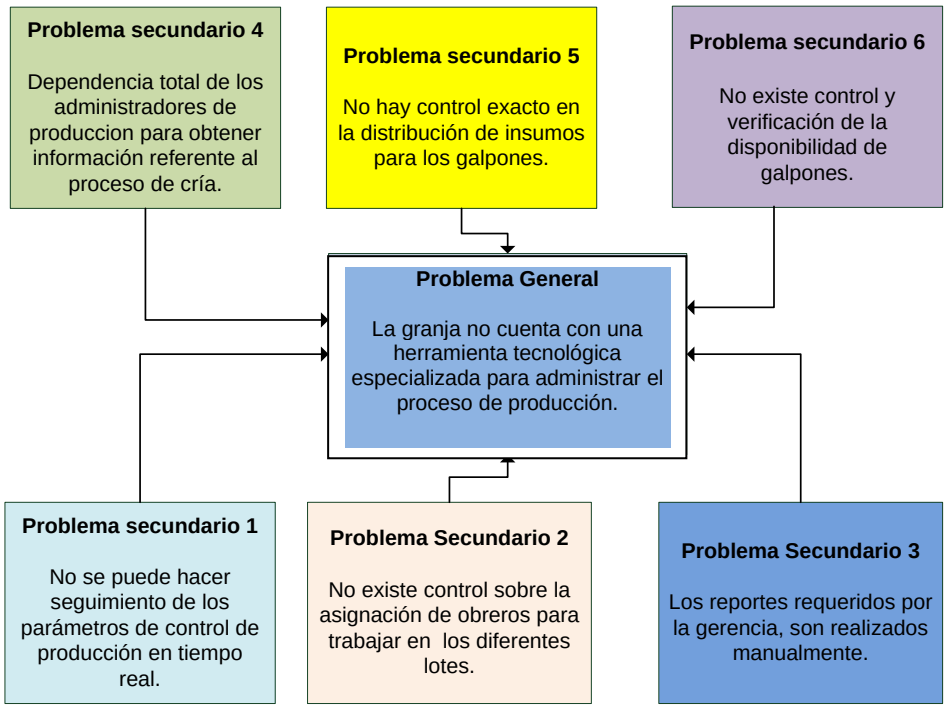


Figura 3.1.2 Árbol de problemas en el flujo del proceso de producción
Fuente: Autor

3.1.2. Requerimientos

El levantamiento de los requisitos con los que debe contar el sistema se los realizó aplicando la técnica de entrevistas al nivel gerencial y administrativo de la granja. Además se verificó con los funcionarios de administración de producción las actividades que se cumplen en el proceso diario de la cría de pollos y se determinó cuales son susceptibles a ser automatizadas en el sistema.

La metodología FDD no utiliza casos de uso para la especificación de requerimientos, al ser una metodología ágil restringe la lista de características del sistema por aquellas de verdadero valor para el cliente. FDD propone que se realice un listado de características del sistema denominado “Características

Evaluadas por el Cliente", mismo que debe estar escrito en lenguaje natural y entendible para el cliente. Las características deben estar escritas en el modelo Acción – Resultado – Objeto, de tal manera que se provea fuertes pistas de las operaciones requeridas en el sistema y de las clases a las que se deben aplicar.

La Especificación de Características Evaluadas por el Cliente se encuentra en el Anexo A.

Después de haber recopilado y analizado toda la información sobre las características del sistema, se efectuó la redacción del documento final de acuerdo entre las partes, es decir, el documento de Especificación de Requerimientos de Software (*ERS*), el cual se basó en el estándar internacional IEEE 830 propuesto por el Instituto de Ingenieros Eléctricos y Electrónicos, como una descripción completa del comportamiento del sistema que se pretende desarrollar.

El documento ERS se encuentra en el Anexo B.

3.2. Modelo de Análisis

En el modelo de análisis, los requisitos capturados son refinados y estructurados, de modo que se comprendan mejor y permitan armar el sistema y su arquitectura.

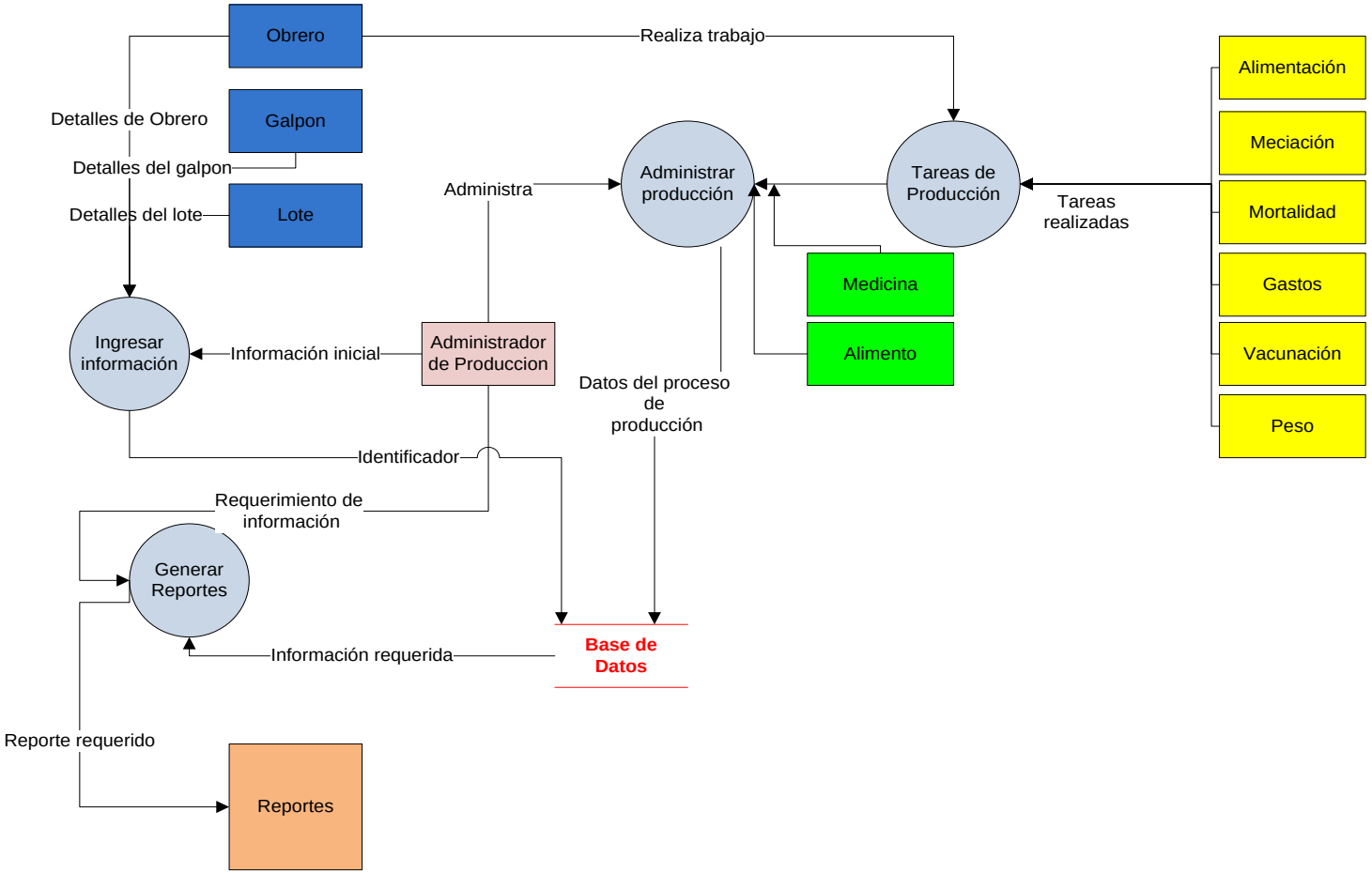


Figura 3.2.1 Modelo de Flujo de Datos Nivel 0
Fuente: Autor

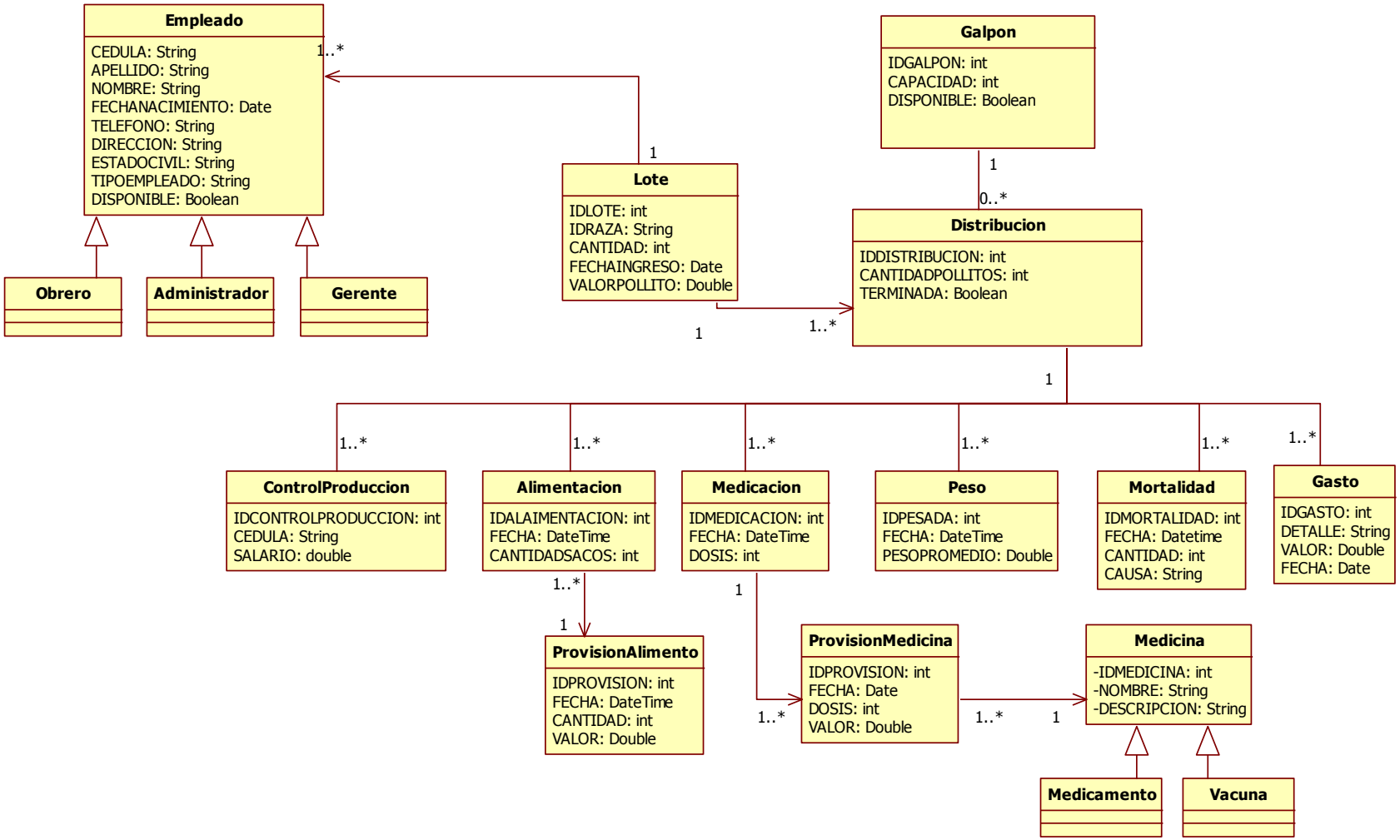


Figura 3.2.2 Diagrama de Clases del Sistema – Parte I
Fuente: Autor

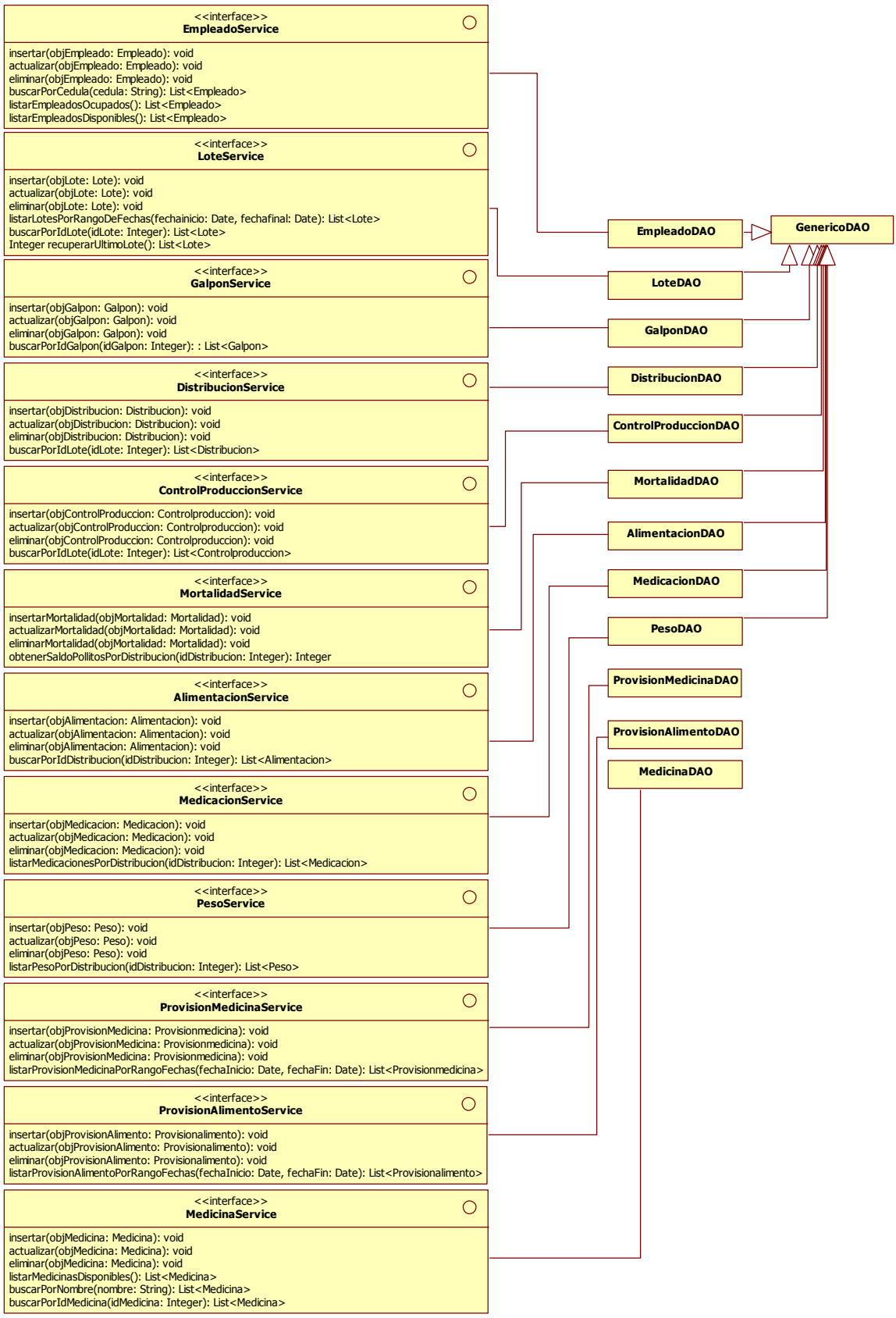


Figura 3.2.3 Diagrama de Clases del Sistema – Parte II
Fuente: Autor

3.2.1. Planeación

La construcción de cada grupo de funcionalidades dura, máximo 1 semana, y al final de este período se realizará una exposición del avance del sistema al cliente.

En total, la construcción del sistema dura 11 semanas, incluyendo las pruebas finales y aceptación por parte del cliente.

En la Tabla 3.2.1 se muestra la lista de características evaluadas por el cliente en la granja.

No.	Característica	Clases
1	Registrar la información de los lotes de pollitos que ingresan a la granja: Fecha de Ingreso, Raza, Cantidad de pollitos, Valor por Pollito.	Lote.
2	Distribuir cada lote en galpones.	Lote, Distribución.
3	Verificar la disponibilidad de los obreros (empleados que no están a cargo de ningún lote).	Lote, Distribución, Galpón.
4	Asignar uno o varios obreros para trabajar, por cada lote	Lote, Distribución, Empleado, ControlProducción.
5	Registrar el salario para cada obrero por el trabajo de control de un determinado lote.	Lote, Distribución, Empleado, ControlProducción.
6	Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser: Administrador de producción,	Empleado.

	Administrador Contable u Obrero.	
7	Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.	ProvisionAlimento.
8	Gestionar la información de las provisiones de medicina, indicando el tipo de medicina: vacuna o medicamento.	ProvisionMedicina.
9	Ingresar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se usó.	Alimentación, Distribución, Lote, ProvisiónAlimento
10	Ingresar los datos de las medicaciones que se realiza a cada galpón.	Medicación, Distribución, Medicina, ProvisionMedicina.
11	Registrar los datos de la toma de peso; el peso promedio de la pesada total del galpón.	Peso, Distribución, Lote.
12	Registrar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.	Mortalidad, Distribución.
13	Registrar los gastos que genera cada lote.	Gastos, Distribución
14	Listado de lotes ingresados en un rango de fecha determinado.	Lote, Distribución, ControlProduccion
15	Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.	Empleado, ControlProduccion

16	Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.	Galpón
17	Listado de galpones disponibles.	Galpón.
18	Provisiones de medicinas y vacunas en un rango de fechas determinado.	ProvisionMedicina, Medicina, Medicacion
19	Provisiones de alimento en un rango de fechas determinado.	ProvisionAlimento, Alimentación
20	Reporte de medicaciones por galpón desde su fecha de ingreso hasta la actualidad.	Galpón, Medicación, Distribución.
21	Reporte de los gastos generados por lote desde su fecha de ingreso hasta la actualidad.	Gasto, Distribución, Lote, Galpón
22	Reporte de mortalidad por cada galpón desde su fecha de ingreso hasta la actualidad	Mortalidad, Distribución, Galpón
23	Reporte de Alimentaciones por lote en un rango de fecha determinado.	Alimentación, Distribución, Galpón.

Tabla 3.2.1 Lista de Características evaluadas por el Cliente
Fuente: Autor

3.2.2. Agrupación por Iteraciones

Iteración 1

1. Registrar la información de los lotes de pollitos que ingresan a la granja.
2. Distribuir cada lote en galpones.
3. Verificar la disponibilidad de los obreros (empleados que no están a cargo de ningún lote).
4. Asignar uno o varios obreros para trabajar, por cada lote.
5. Registrar el salario para cada obrero por el trabajo de control de un determinado lote.

Iteración 2

6. Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser: Administrador de producción, Administrador Contable u Obrero.

Iteración 3

7. Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.
8. Gestionar la información de las provisiones de medicina, indicando el tipo de medicina.

Iteración 4

9. Ingresar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se usó.
10. Ingresar los datos de las medicaciones que se realiza a cada galpón.

Iteración 5

11. Registrar los datos de la toma de peso; el peso promedio de la pesada total del galpón.
12. Registrar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.
13. Registrar los gastos que genera cada lote.

Iteración 6

14. Listado de lotes ingresados en un rango de fecha determinado.
15. Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.

Iteración 7

16. Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.
17. Listado de galpones disponibles.

Iteración 8

18. Provisiones de medicinas y vacunas en un rango de fechas determinado.
19. Provisiones de alimento en un rango de fechas determinado.

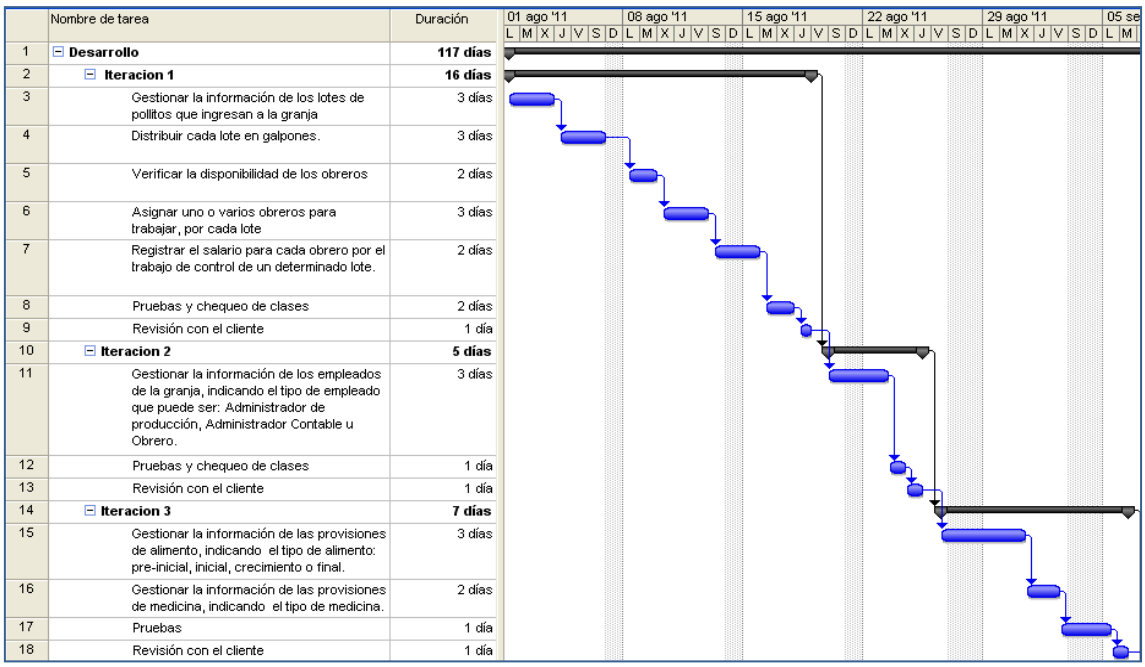
Iteración 9

20. Reporte de medicaciones por galpón desde su fecha de ingreso hasta la actualidad.
21. Reporte de los gastos generados por lote desde su fecha de ingreso hasta la actualidad.
22. Reporte de mortalidad por cada galpón desde su fecha de ingreso hasta la actualidad.

23. Reporte de Alimentaciones por lote en un rango de fecha determinado.

3.2.3. Plan de Entregas

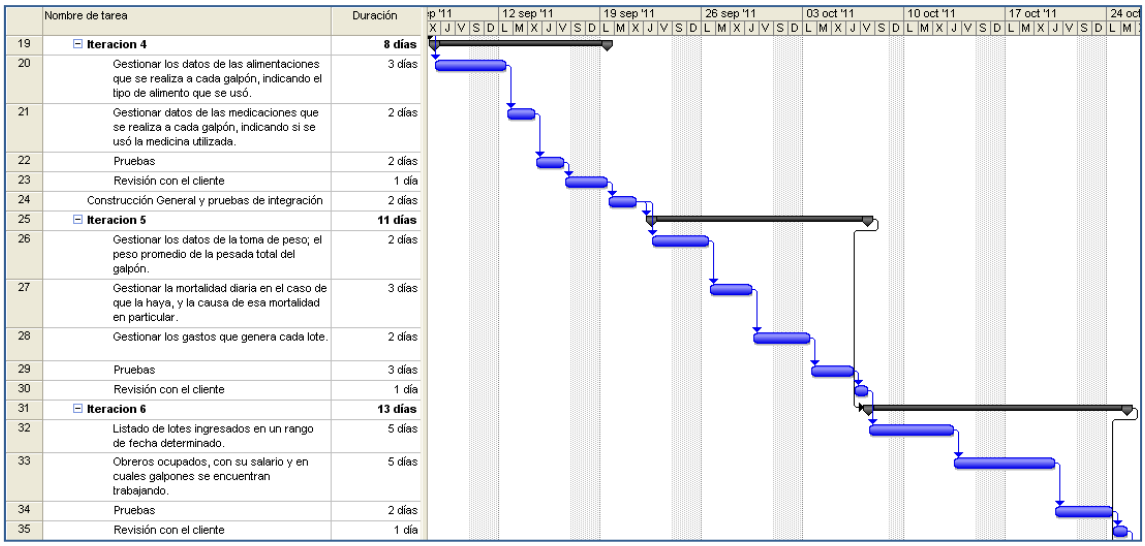
El desarrollo de cada iteración se realizará en un tiempo máximo de dos semanas. Cada iteración involucra su diseño, desarrollo, pruebas y aceptación de parte del cliente. Se ha ordenado cada iteración de acuerdo a la prioridad indicada por el cliente.



	Nombre de tarea	Duración
1	**Desarrollo**	**117 días**
2	**Iteracion 1**	**16 días**
3	Gestionar la información de los lotes de pollitos que ingresan a la granja	3 días
4	Distribuir cada lote en galpones.	3 días
5	Verificar la disponibilidad de los obreros	2 días
6	Asignar uno o varios obreros para trabajar, por cada lote	3 días
7	Registrar el salario para cada obrero por el trabajo de control de un determinado lote.	2 días
8	Pruebas y chequeo de clases	2 días
9	Revisión con el cliente	1 día
10	**Iteracion 2**	**5 días**
11	Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser: Administrador de producción, Administrador Contable u Obrero.	3 días
12	Pruebas y chequeo de clases	1 día
13	Revisión con el cliente	1 día
14	**Iteracion 3**	**7 días**
15	Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.	3 días
16	Gestionar la información de las provisiones de medicina, indicando el tipo de medicina.	2 días
17	Pruebas	1 día
18	Revisión con el cliente	1 día

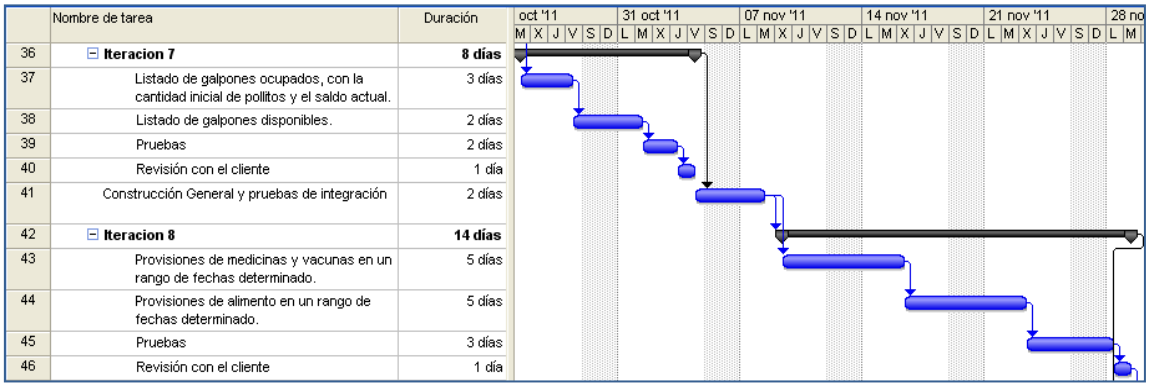
Figura 3.2.4 Plan de Entregas
Fuente: Autor

Espacio en Blanco puesto para completar la página.



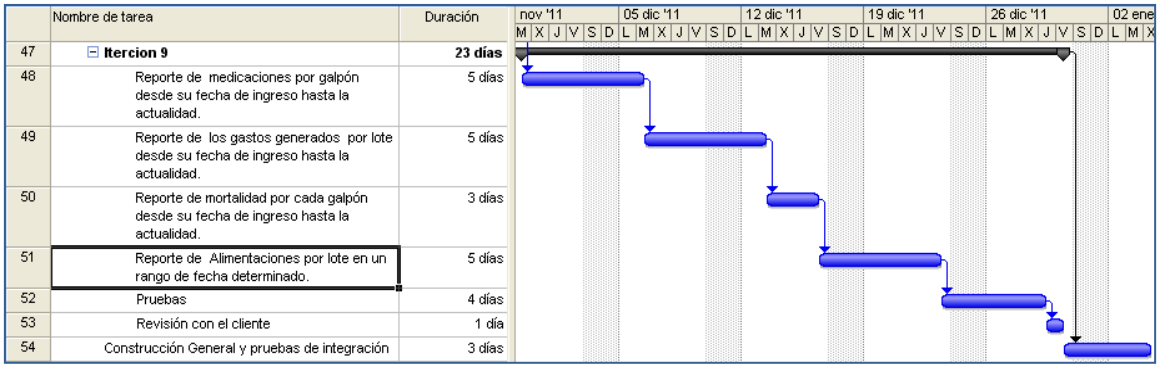
	Nombre de tarea	Duración
19	**Iteracion 4**	**8 días**
20	Gestionar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se usó.	3 días
21	Gestionar datos de las medicaciones que se realiza a cada galpón, indicando si se usó la medicina utilizada.	2 días
22	Pruebas	2 días
23	Revisión con el cliente	1 día
24	Construcción General y pruebas de integración	2 días
25	**Iteracion 5**	**11 días**
26	Gestionar los datos de la toma de peso; el peso promedio de la pesada total del galpón.	2 días
27	Gestionar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.	3 días
28	Gestionar los gastos que genera cada lote.	2 días
29	Pruebas	3 días
30	Revisión con el cliente	1 día
31	**Iteracion 6**	**13 días**
32	Listado de lotes ingresados en un rango de fecha determinado.	5 días
33	Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.	5 días
34	Pruebas	2 días
35	Revisión con el cliente	1 día

Figura 3.2.5 Plan de Entregas - Parte II
Fuente: Autor



	Nombre de tarea	Duración
36	**Iteracion 7**	**8 días**
37	Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.	3 días
38	Listado de galpones disponibles.	2 días
39	Pruebas	2 días
40	Revisión con el cliente	1 día
41	Construcción General y pruebas de integración	2 días
42	**Iteracion 8**	**14 días**
43	Provisiones de medicinas y vacunas en un rango de fechas determinado.	5 días
44	Provisiones de alimento en un rango de fechas determinado.	5 días
45	Pruebas	3 días
46	Revisión con el cliente	1 día

Figura 3.2.6 Plan de Entregas Parte III
Fuente: Autor



	Nombre de tarea	Duración
47	**Itercion 9**	**23 días**
48	Reporte de medicaciones por galpón desde su fecha de ingreso hasta la actualidad.	5 días
49	Reporte de los gastos generados por lote desde su fecha de ingreso hasta la actualidad.	5 días
50	Reporte de mortalidad por cada galpón desde su fecha de ingreso hasta la actualidad.	3 días
51	Reporte de Alimentaciones por lote en un rango de fecha determinado.	5 días
52	Pruebas	4 días
53	Revisión con el cliente	1 día
54	Construcción General y pruebas de integración	3 días

Figura 3.2.7 Plan de Entregas Parte IV
Fuente: Autor

3.3. Diseño, Desarrollo y Pruebas

En el diseño, FDD utiliza Diagramas de Secuencia para describir cada característica, una vez hecho esto se prosigue con el desarrollo y las pruebas unitarias. La iteración no termina sino hasta que el cliente queda completamente satisfecho con el resultado mostrado.

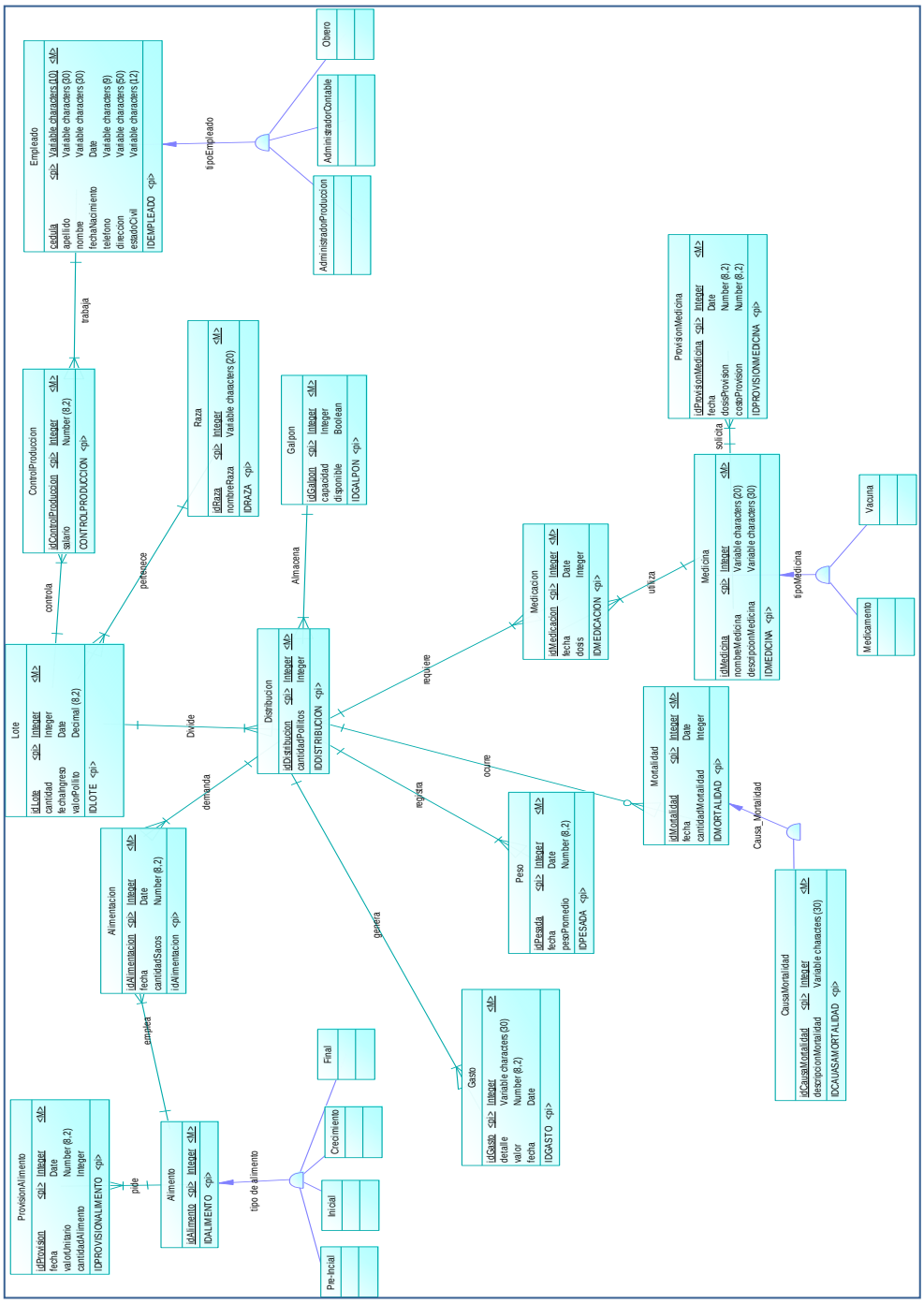
3.3.1. Control de Versiones

FDD manda a llevar el control de las versiones de todos los artefactos que se programen, en este caso como la programación está realizada bajo Java, se utilizará Apache Subversion que es un sistema de código abierto para el control de versiones en esta tecnología.

Espacio en Blanco puesto para completar la página

3.3.2. Diseño de la Base de Datos

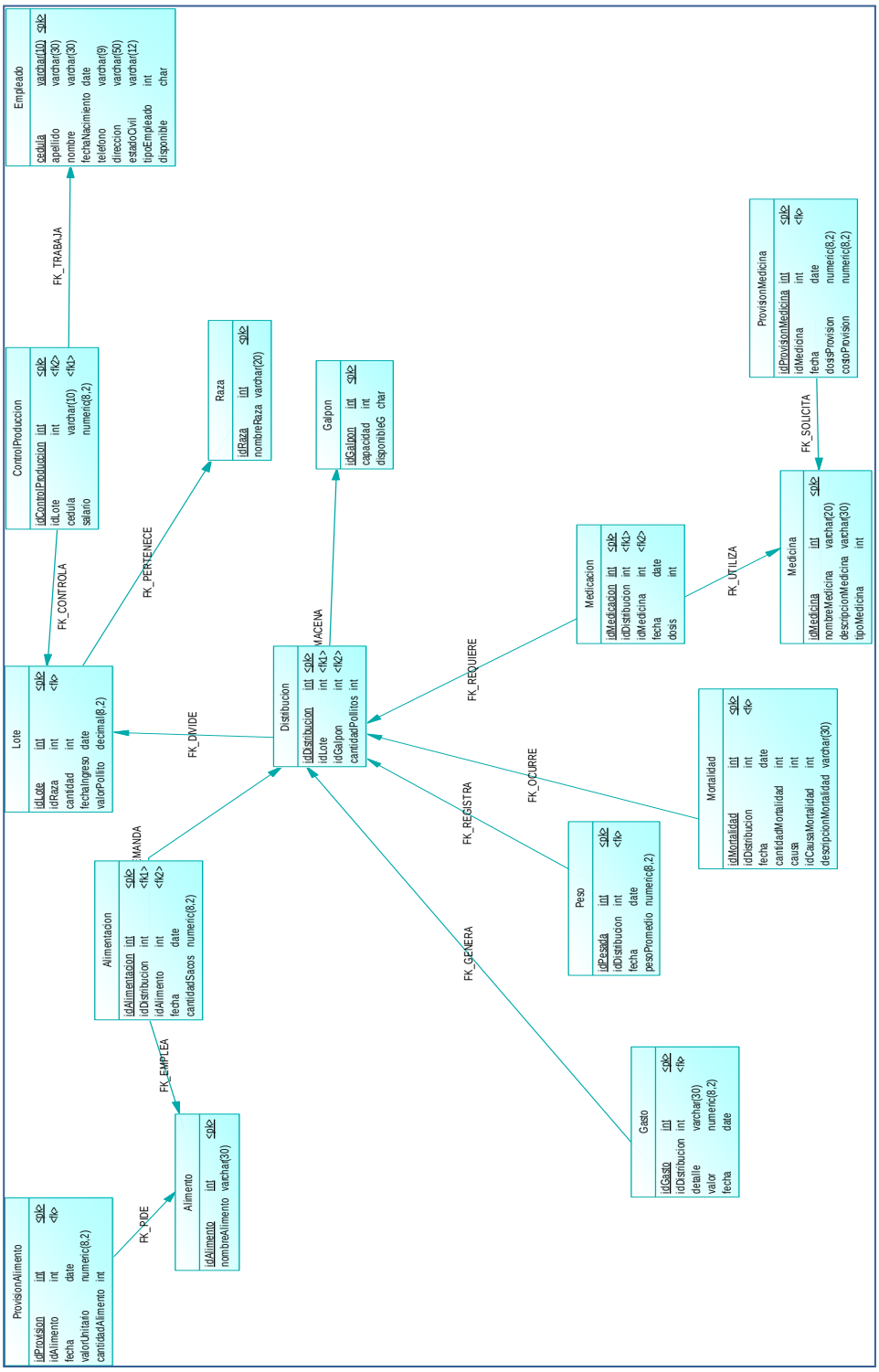
3.3.2.1. Modelo Lógico



Fuente: Autor. Fecha: 21 de noviembre de 2011

Figura 3.3.1 Modelo Lógico de la Base de Datos
Fuente: Autor

3.3.2.2. Modelo Físico



Fuente: Autor. Fecha: 25 de noviembre de 2011

Figura 3.3.2 Modelo Físico de la Base de Datos
Fuente: Autor

3.3.3. Diseño Arquitectónico

3.3.3.1. Arquitectura Lógica

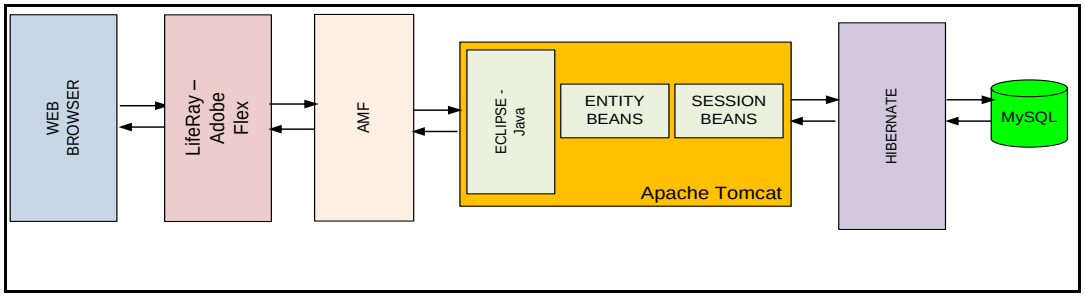


Figura 3.3.3 Arquitectura Lógica
Fuente: Autor

3.3.3.2. Arquitectura Física

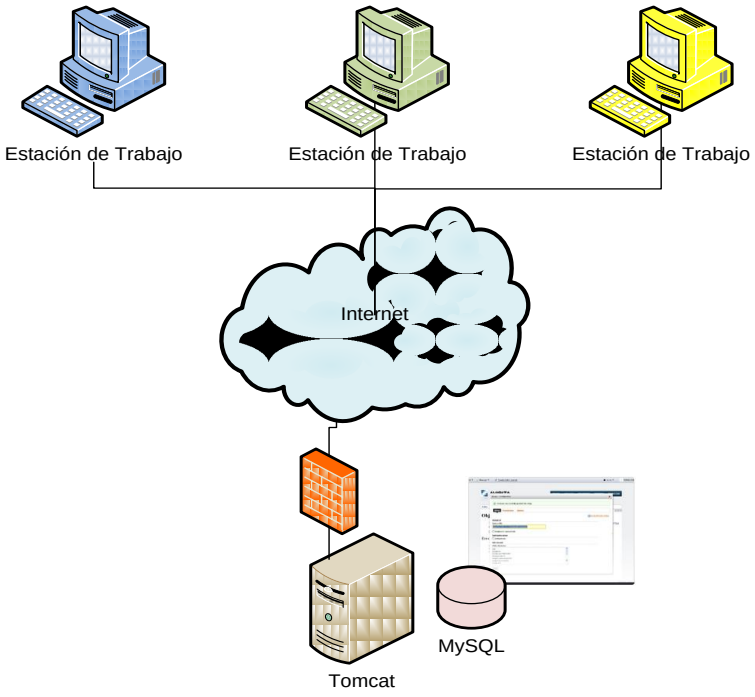


Figura 3.3.4. Arquitectura Física
Fuente: Autor

Cliente: El sistema ha sido validado de modo que sea compatible con la mayoría de los exploradores vigentes para que de esta forma no presente inconvenientes en la navegación de los usuarios.

Presentación: La presentación del sistema está conformada por: las páginas HTML que contienen el código a nivel de presentación y que acceden a la funcionalidad del sistema a través de servicios web, el sistema desarrollado cumple con los estándares de la W3C para generación de contenido dinámico específico y diseño de interfaces.

Negocio: La lógica de negocio está encapsulada en el componente EJB que guarda el comportamiento de la aplicación, gestiona y provee los servicios.

Acceso a datos: Finalmente los datos están contenidos en una base de datos en el gestor MySQL.

3.3.3.3. Prueba de Conexión hacia la Base de Datos

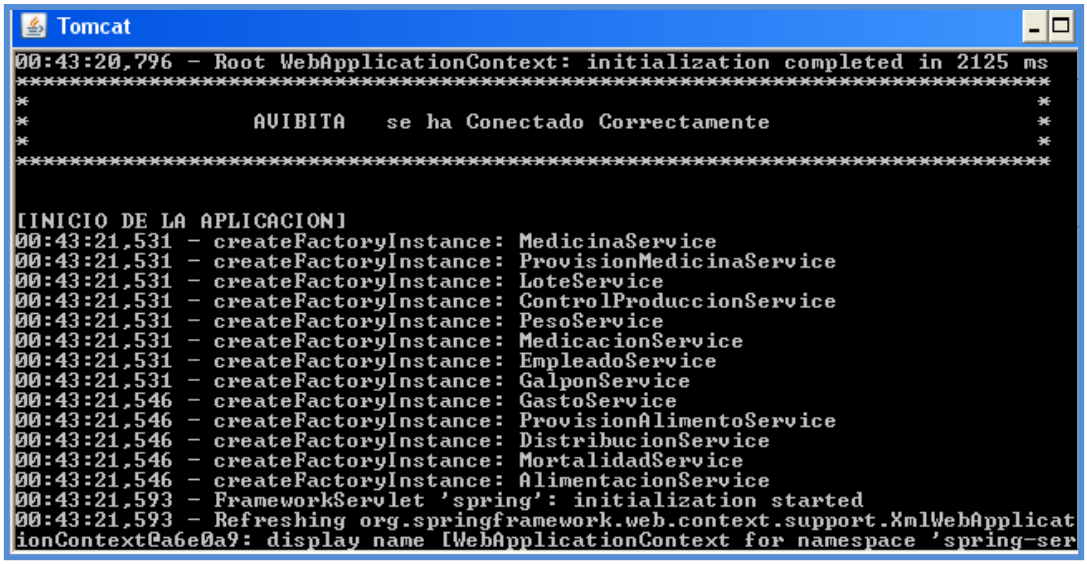


Figura 3.3.5 Prueba de Conexión a la Base de Datos
Fuente: Autor

3.3.4. Desarrollo de Iteraciones

3.3.4.1. Iteración 1

1. Registrar la información de los lotes de pollitos que ingresan a la granja.

2. Distribuir cada lote en galpones.
3. Verificar la disponibilidad de los obreros (empleados que no están a cargo de ningún lote).
4. Asignar uno o varios obreros para trabajar, por cada lote.
5. Registrar el salario para cada obrero por el trabajo de control de un determinado lote.

Diagramas de Secuencia

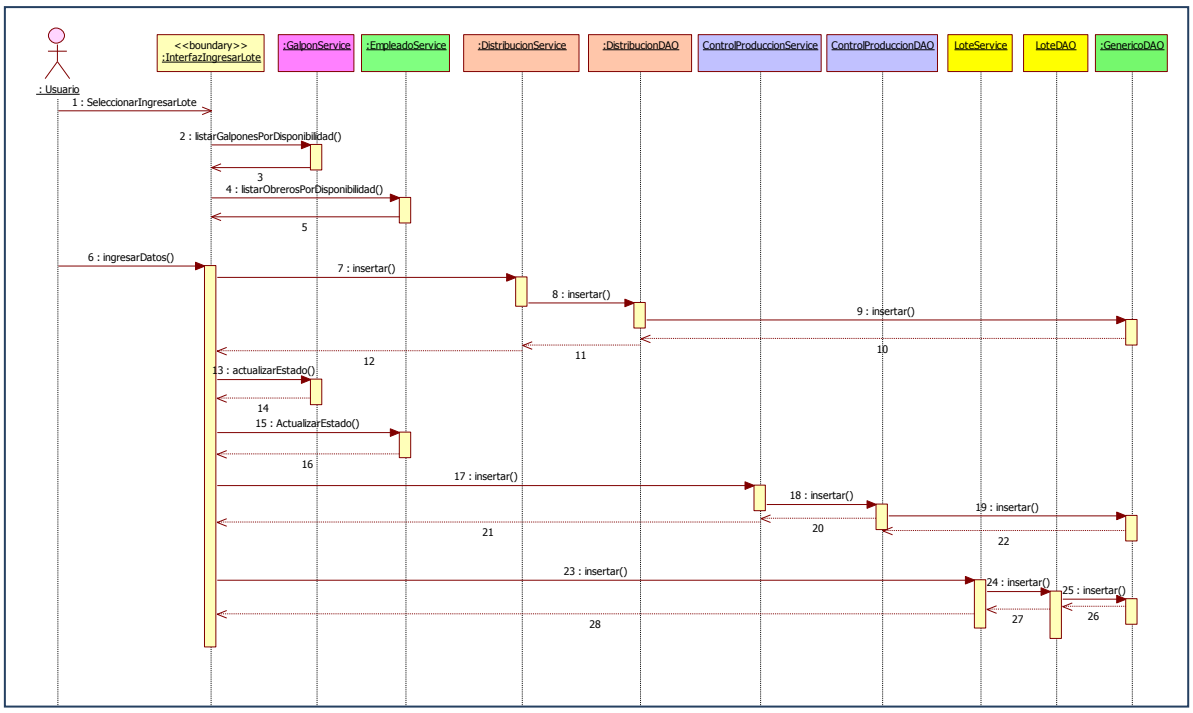


Figura 3.3.6 Diagrama de Secuencia Iteración 1
Fuente: Autor

Pruebas de Unidad

Después de programar lo que indica el detalle de las características de la iteración y lo diseñado en el diagrama de secuencia, se continúa con la ejecución de las pruebas de unidad para verificar la existencia de errores, antes de llamar a los servicios desde la capa de aplicación.

```
public void testSaveAndRemove() throws ParseException {
testEntityLote.setCantidad(1);
testEntityLote.setFechaingreso(Date.valueOf("2011-11-28"));
testEntityLote.setIdRaza("Ross");
testEntityLote.setValorpollito(0.56);
dao.save(testEntityLote);
Lote test = dao.getById(testEntityLote.getIdlote());
dao.delete(testEntityLote)}
```

```
public void testSaveAndRemove() {
testEntityGalpon.setCapacidad(4555);
testEntityGalpon.setDisponibleg("1");
dao.save(testEntityGalpon);
Galpon test = dao.getById(testEntityGalpon.getIdgalpon());
assertNotNull("testSaveAndRemove",
testEntityGalpon.getIdgalpon());
dao.delete(testEntityGalpon);
test = dao.getById(testEntityGalpon.getIdgalpon());
assertNull("testSaveAndRemove", test);
}
```

```
public void testSaveAndRemove() throws ParseException {
testEntityEmpleado.setApellido("");
testEntityEmpleado.setCedula("1304502068");
testEntityEmpleado.setDireccion("Rocafuerte");
testEntityEmpleado.setDisponible("1");
testEntityEmpleado.setEstadocivil("Soltero");
testEntityEmpleado.setNombre("Julio");
testEntityEmpleado.setTelefono("Zambrano");
testEntityEmpleado.setTipoempleado("Obrero");
dao.save(testEntityEmpleado);
Empleado test = dao.getById(testEntityEmpleado.getCedula());
assertNotNull("testSaveAndRemove", test);
dao.delete(testEntityEmpleado);
test = dao.getById(testEntityEmpleado.getCedula());
assertNull("testSaveAndRemove", test);}
```

```
public void testFindByDisponible() {
List<Empleado> list = dao.findByDisponible("1");
assertNotNull("testFindByDisponible", list);
assertFalse("testFindByDisponible", list.isEmpty());}
```

Espacio en Blanco puesto para completar la página

En la Figura 3.3.7 se muestra la ejecución exitosa de un grupo de pruebas automatizadas con JUnit.

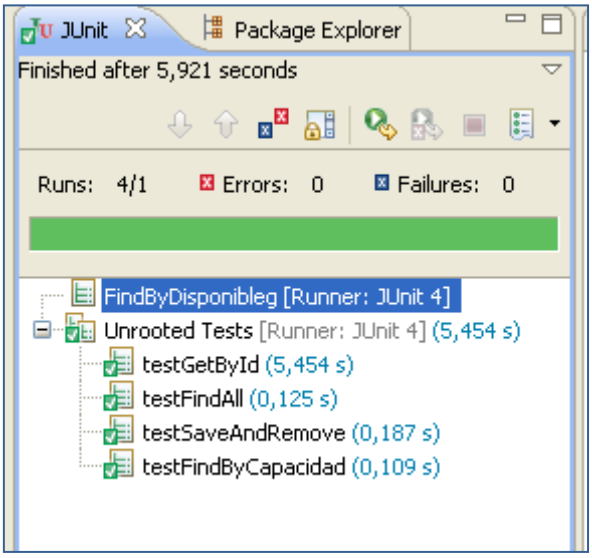


Figura 3.3.7. Prueba de Unidad Ejecutada satisfactoriamente.
Fuente: Autor

Control de Versiones

Después de analizar y corregir los errores, se confirma la versión en Subversion. *En la Figura 3.3.8 se muestra el proceso de actualizar las versiones en Java.*

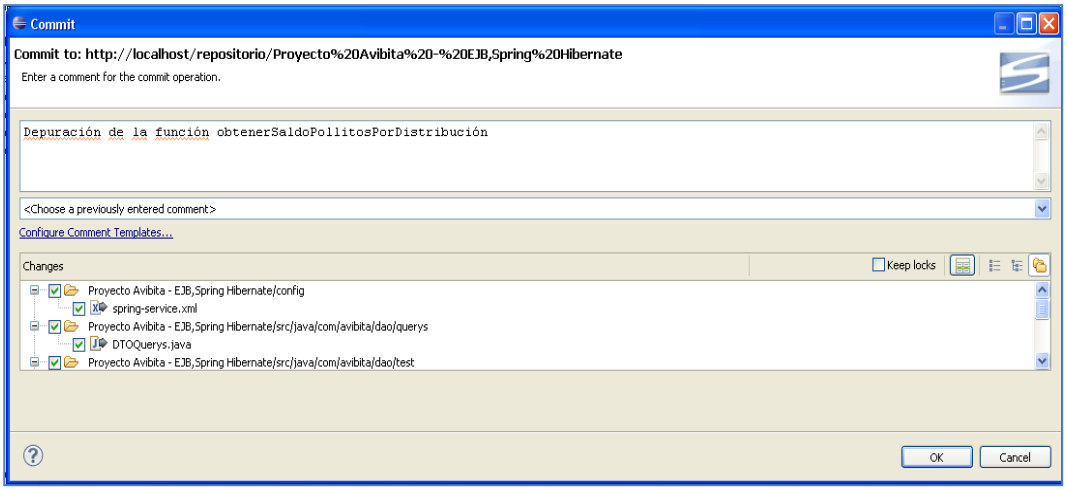


Figura 3.3.8. Control de Versiones del Código
Fuente: Autor

3.3.4.2. Iteración 2

En las siguientes iteraciones se sigue el mismo proceso: se analiza la lista de características que forman la iteración, se diseñan los diagramas de secuencia, se realiza la programación con las clases involucradas, se corren las pruebas de unidad, se corrigen los posibles errores y se confirma la versión.

6. Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser: Administrador de producción, Administrador Contable u Obrero.

Diagramas de Secuencia

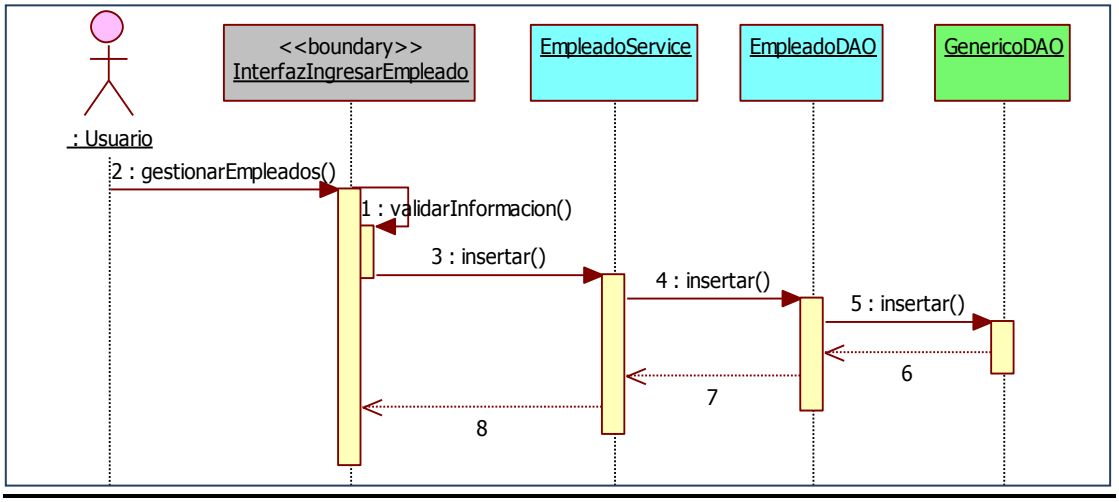


Figura 3.3.9. Diagrama de Secuencia Iteración 2
Fuente: Autor

Pruebas de Unidad

```
public void testSaveAndRemove() throws ParseException {
testEntityEmpleado.setCedula("1304502068");
testEntityEmpleado.setDisponible("1");
testEntityEmpleado.setEstadocivil("Soltero");
testEntityEmpleado.setNombre("Julio");
testEntityEmpleado.setTelefono("Zambrano");
testEntityEmpleado.setTipoempleado("Obrero");
dao.save(testEntityEmpleado);
Empleado test = dao.getById(testEntityEmpleado.getCedula());
dao.delete(testEntityEmpleado);
test = dao.getById(testEntityEmpleado.getCedula());}
```

3.3.4.3. Iteración 3

7. Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.
8. Gestionar la información de las provisiones de medicina, indicando el tipo de medicina.

Diagramas de Secuencia

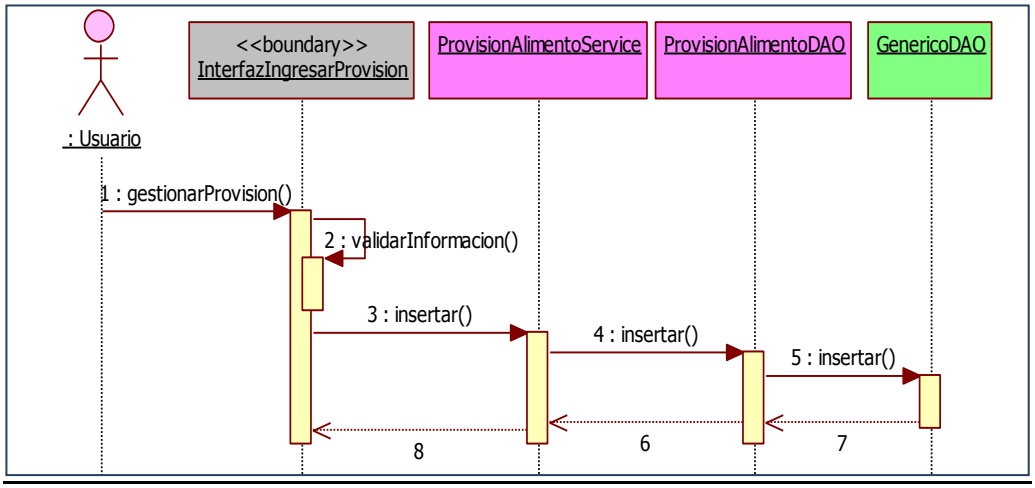


Figura 3.3.10. Diagrama de Secuencia de la Característica 7
Fuente: Autor

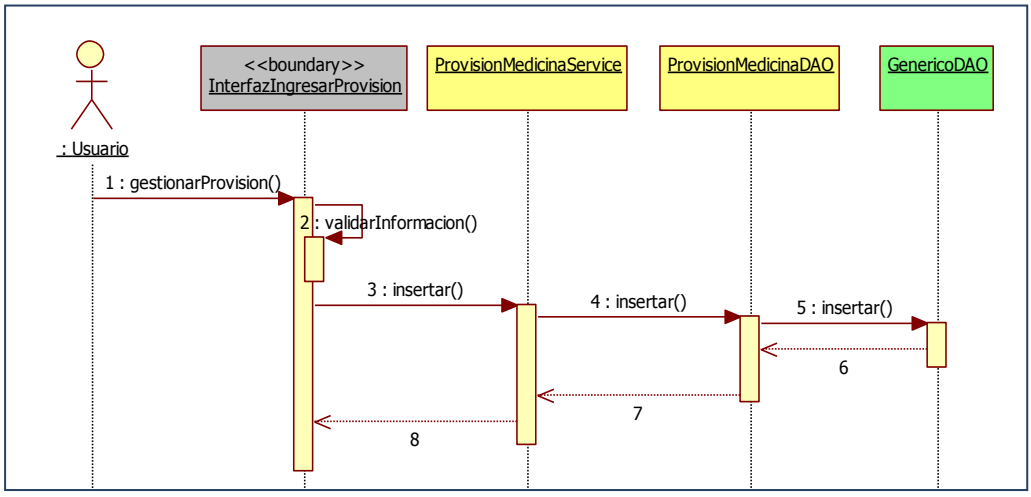


Figura 3.3.11. Diagrama de Secuencia de la Característica 8
Fuente: Autor

Pruebas de Unidad

```
public void testSaveAndRemove() throws ParseException {
 testEntityProvisionalimento.setTipoalimento("FINAL");
 testEntityProvisionalimento.setCantidadalimento(1.0);
 testEntityProvisionalimento.setFecha(java.sql.Date.valueOf("2011-11-27")););
testEntityProvisionalimento.setIdprovision(1);
String aString = "12";
double aDouble = Double.parseDouble(aString);
testEntityProvisionalimento.setValorunitario(aDouble);
dao.save(testEntityProvisionalimento);
Provisionalimentotest=
dao.getById(testEntityProvisionalimento.getIdprovision());
}
```

```
public void testSaveAndRemove() throws ParseException {
 testEntityProvisionmedicina.setCostoprovision(4.5);
 testEntityProvisionmedicina.setFecha(java.sql.Date.valueOf("2011-11-27"));
 testEntityProvisionmedicina.setIdmedicina(1);
 dao.save(testEntityProvisionmedicina);
 Provisionmedicinatest=
 dao.getById(testEntityProvisionmedicina.getIdprovision());}
```

```
public void testFindByFecha() throws ParseException {
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date = sdf.parse("2011-11-20");
java.sql.Timestamp timest = new java.sql.Timestamp(date.getTime());
List<Provisionmedicina> list = dao.findByFecha(timest);
assertNotNull("testFindByFecha", list);
assertFalse("testFindByFecha", list.isEmpty());
}
```

```
public void testFindBetweenRangoFechas() throws ParseException
{
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date = sdf.parse("2011-09-01");
Date date2=sdf.parse("2011-12-30");
java.sql.Timestamp timest = new java.sql.Timestamp(date.getTime());
java.sql.Timestamp timest2 = new java.sql.Timestamp(date2.getTime());
List<Provisionalimento> list = dao.findBetweenRangoFechas(timest, timest2);
assertEquals("7", list.get(0).getIdprovision().toString());
}
```

3.3.4.4. Iteración 4

9. Ingresar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se usó.
10. Ingresar los datos de las medicaciones que se realiza a cada galpón.

Diagramas de Secuencia

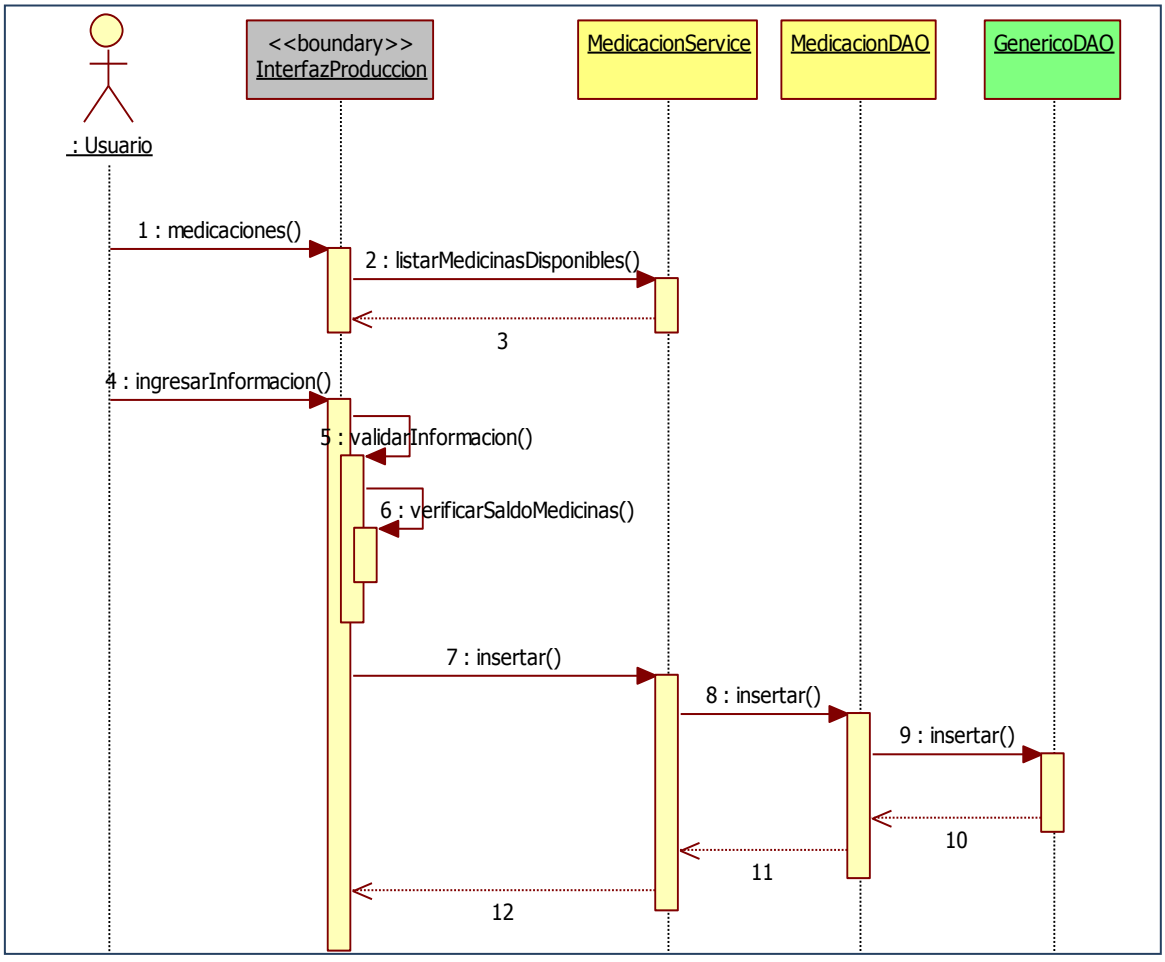


Figura 3.3.12. Diagrama de Secuencia de la Características 10
Fuente: Autor

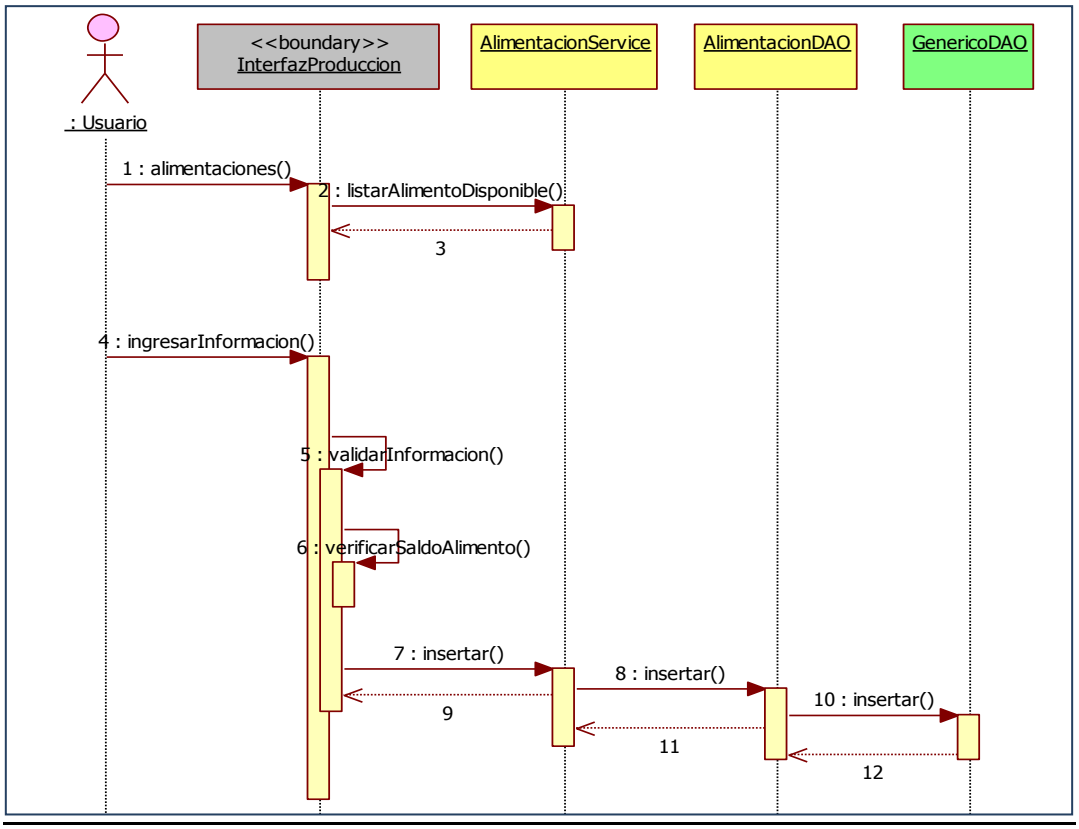


Figura 3.3.13. Diagrama de Secuencia de la Característica 9
Fuente: Autor

Pruebas de Unidad

```
public void testSaveAndRemove() throws ParseException {
double aDouble = Double.parseDouble("12");
testEntityAlimentacion.setCantidadsacos(aDouble);
testEntityAlimentacion.setIdDistribucion(1);
testEntityAlimentacion.setFecha(java.sql.Date.valueOf("2011-11-27"));
testEntityAlimentacion.setIdalimentacion(null);
dao.save(testEntityAlimentacion);
Alimentacion test = dao.getById(testEntityAlimentacion.getIdalimentacion());
dao.delete(testEntityAlimentacion);
test = dao.getById(testEntityAlimentacion.getIdalimentacion());}
```

```
public void testSaveAndRemove() throws ParseException {
testEntityMedicacion.setIdDistribucion(1);
testEntityMedicacion.setDosis(5000);
testEntityMedicacion.setFecha(Date.valueOf("2011-11-27"));
testEntityMedicacion.setIdMedicina(1);
testEntityMedicacion.setIdprovision(1);
dao.save(testEntityMedicacion);
Medicacion test = dao.getById(testEntityMedicacion.getIdmedicacion());
dao.delete(testEntityMedicacion);}
```

3.3.4.5. Iteración 5

11. Registrar los datos de la toma de peso; el peso promedio de la pesada total del galpón.
12. Registrar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.
13. Registrar los gastos que genera cada lote.

Diagramas de Secuencia

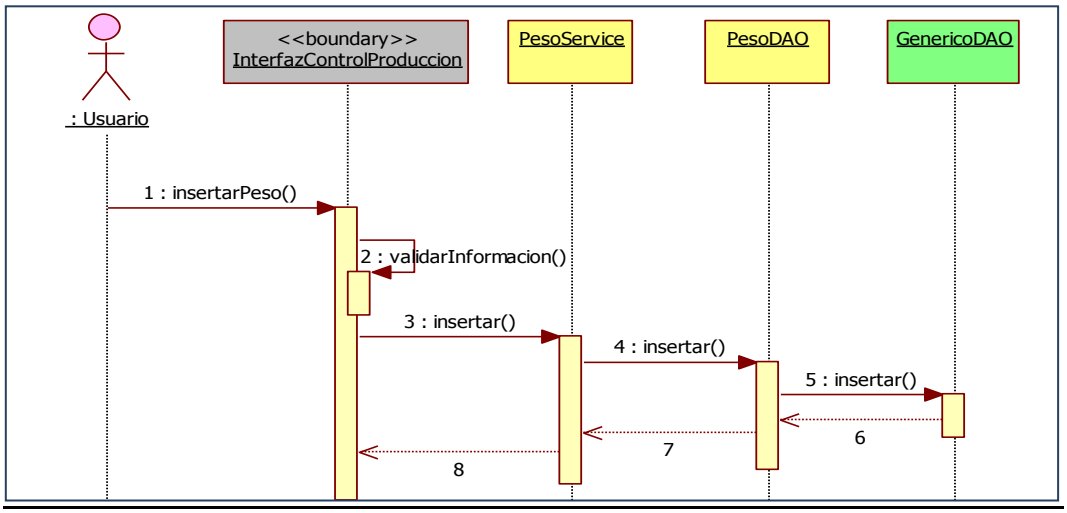


Figura 3.3.14. Diagrama de Secuencia de la Característica 11
Fuente: Autor

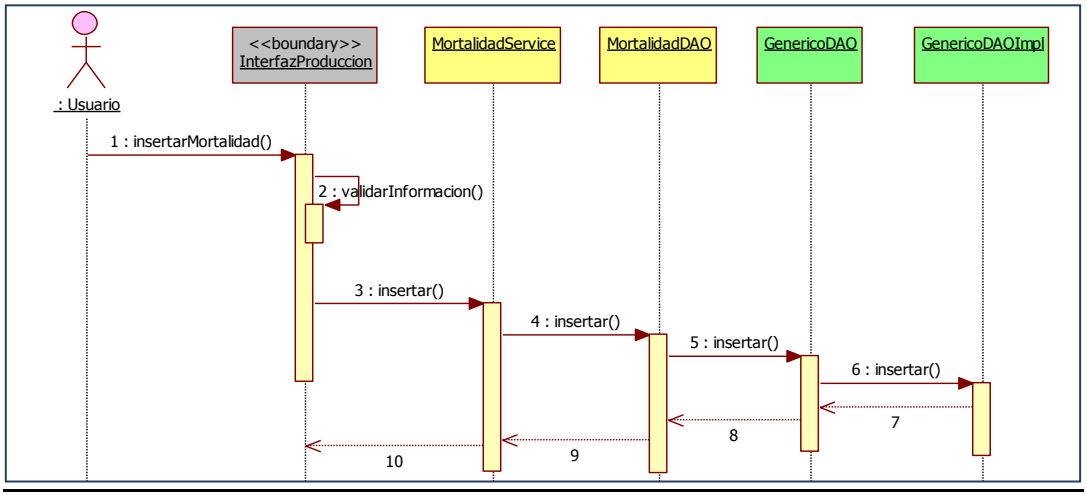


Figura 3.3.15. Diagrama de Secuencia de la Característica 12
Fuente: Autor

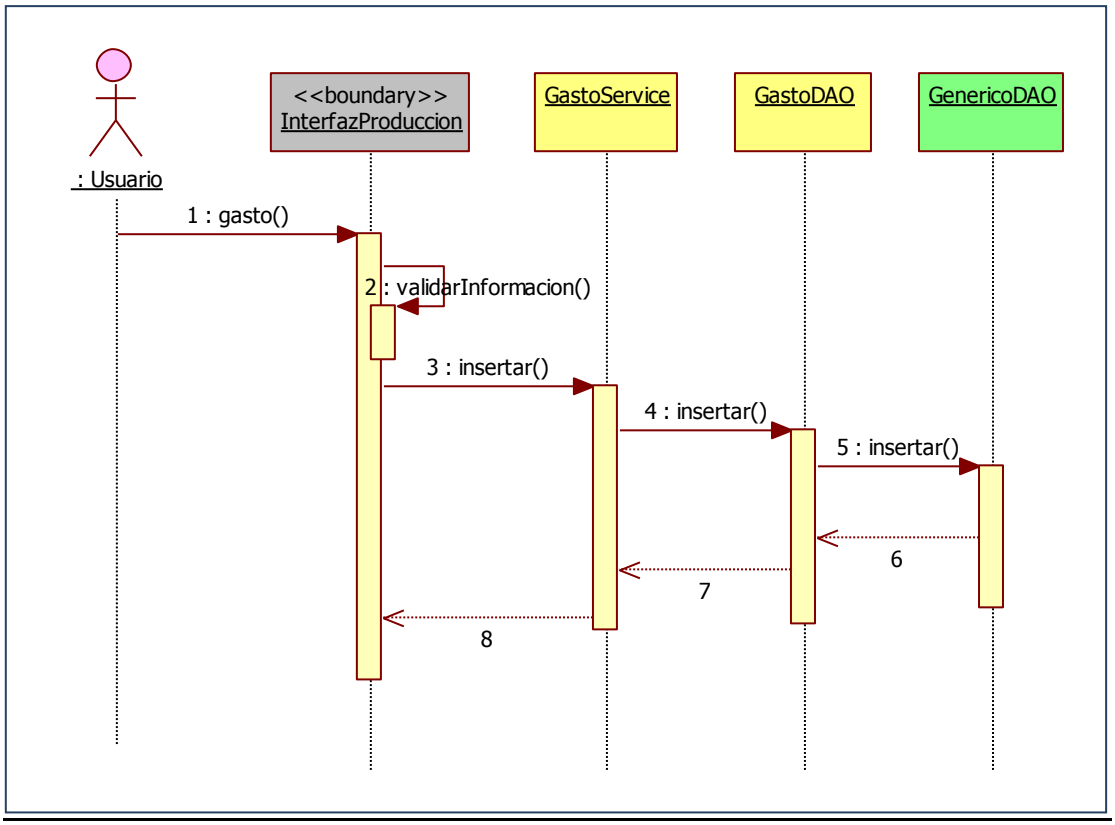


Figura 3.3.16. Diagrama de Secuencia de la Característica 13
Fuente: Autor

Pruebas de Unidad

```
public void testSaveAndRemove() throws ParseException {
testEntityPeso.setFecha(Date.valueOf("2011-12-14"));
String aString = "12";
double aDouble = Double.parseDouble(aString);
testEntityPeso.setPesopromedio(aDouble);
testEntityPeso.setIdDistribucion(1);
dao.save(testEntityPeso);
Pesotest=dao.getById(testEntityPeso.getIdpesada());
assertNotNull("testSaveAndRemove", t          estEntityPeso.getIdpesada());
 assertNotNull("testSaveAndRemove", test);
dao.delete(testEntityPeso);
test = dao.getById(testEntityPeso.getIdpesada());
assertNull("testSaveAndRemove", test);}
```

Espacio en Blanco puesto para completar la página.

```
public void testSaveAndRemove() throws ParseException {
testEntityMortalidad.setCantidadmortalidad(1);
testEntityMortalidad.setDescripcionmortalidad("Gripe");
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date = sdf.parse("2011-11-20");
testEntityMortalidad.setFecha(null);
testEntityMortalidad.setIdcausamortalidad("Natural");
testEntityMortalidad.setIdmortalidad(1);
testEntityMortalidad.setIdDistribucion(1);
dao.save(testEntityMortalidad);
Mortalidad test =        dao.getById(testEntityMortalidad.getIdmortalidad());
assertNotNull("testSaveAndRemove",              testEntityMortalidad.getIdmortalidad());
assertNotNull("testSaveAndRemove", test);
dao.delete(testEntityMortalidad);
test = dao.getById(testEntityMortalidad.getIdmortalidad());
assertNull("testSaveAndRemove", test);          }
```

```
public void testSaveAndRemove() throws ParseException {
testEntityGasto.setDetalle("Bebederos");
testEntityGasto.setIdDistribucion(1);
testEntityGasto.setValor(100.13);
testEntityGasto.setFecha(java.sql.Date.valueOf("2011-11-       27"));
dao.save(testEntityGasto);
Gasto test = dao.getById(testEntityGasto.getIdgasto());
assertNotNull("testSaveAndRemove",              testEntityGasto.getIdgasto());
 assertNotNull("testSaveAndRemove", test);
dao.delete(testEntityGasto);
test = dao.getById(testEntityGasto.getIdgasto());
assertNull("testSaveAndRemove", test);}
```

Reportes

Los reportes requieren de consultas SQL personalizadas que quedan fuera del dominio de Hibernate, no utilizan pruebas de unidad con el framework JUnit, sino pruebas de verificación de funcionalidad de entrada – salida.

3.3.4.6. Iteración 6

14. Listado de lotes ingresados en un rango de fecha determinado.
15. Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.

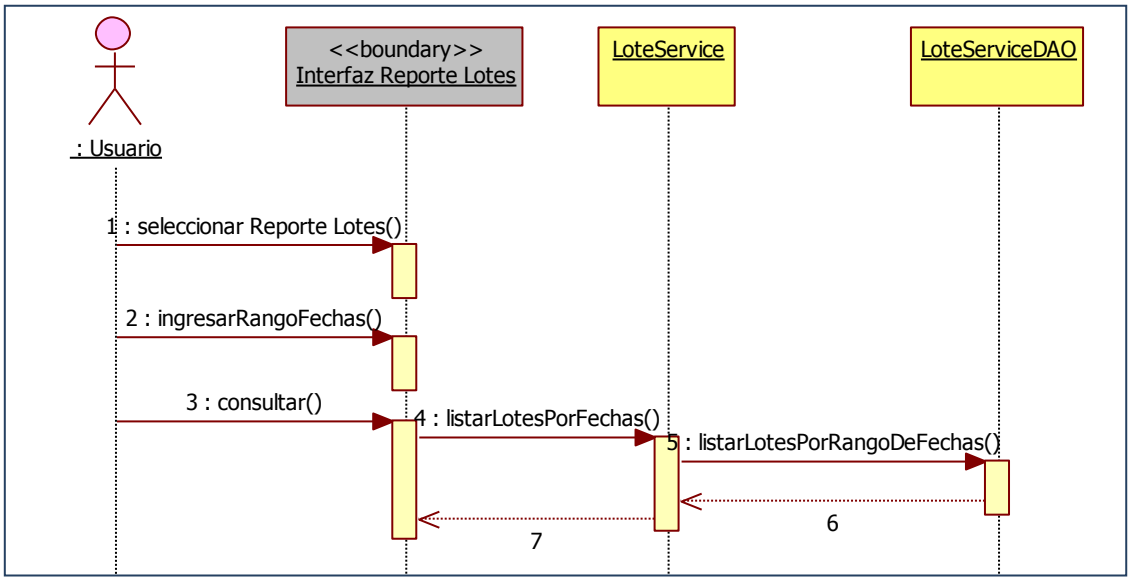


Figura 3.3.17. Diagrama de Secuencia de la Característica 14
Fuente: Autor

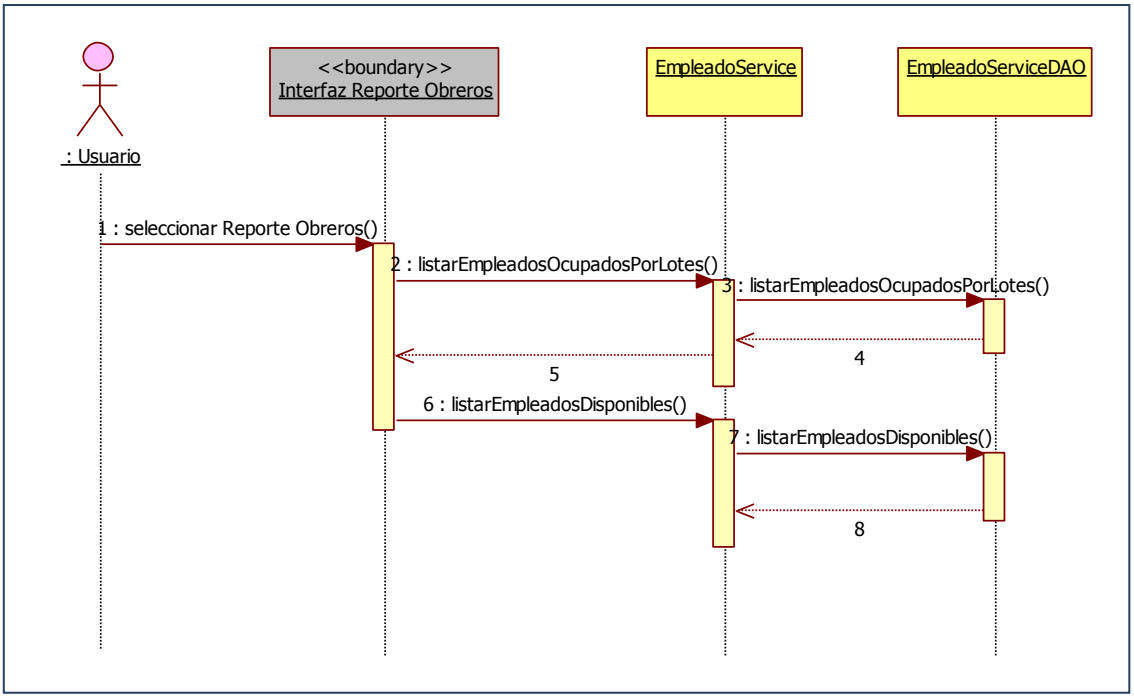


Figura 3.3.18. Diagrama de Secuencia de la Característica 15
Fuente: Autor

3.3.4.7. Iteración 7

16. Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.
17. Listado de galpones disponibles.

Diagramas de Secuencia

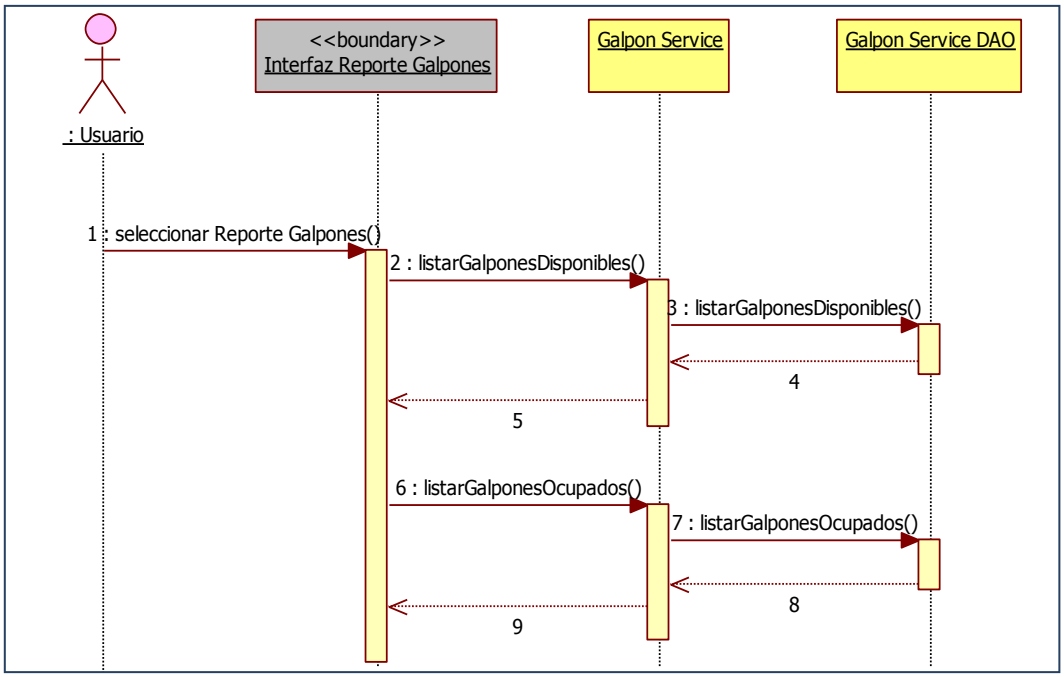


Figura 3.3.19. Diagrama de Secuencia de la Iteración 7
Fuente: Autor

3.3.4.8. Iteración 8

18. Provisiones de medicinas y vacunas en un rango de fechas determinado.
19. Provisiones de alimento en un rango de fechas determinado.

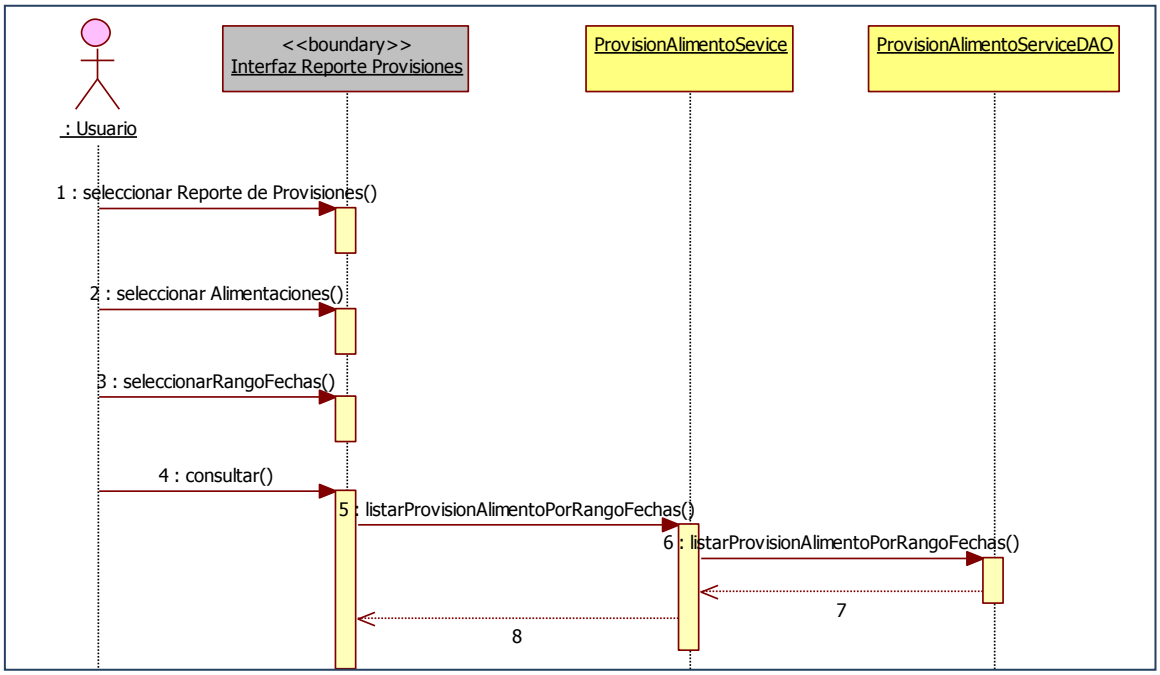


Figura 3.3.20. Diagrama de Secuencia de la Característica 18
Fuente: Autor

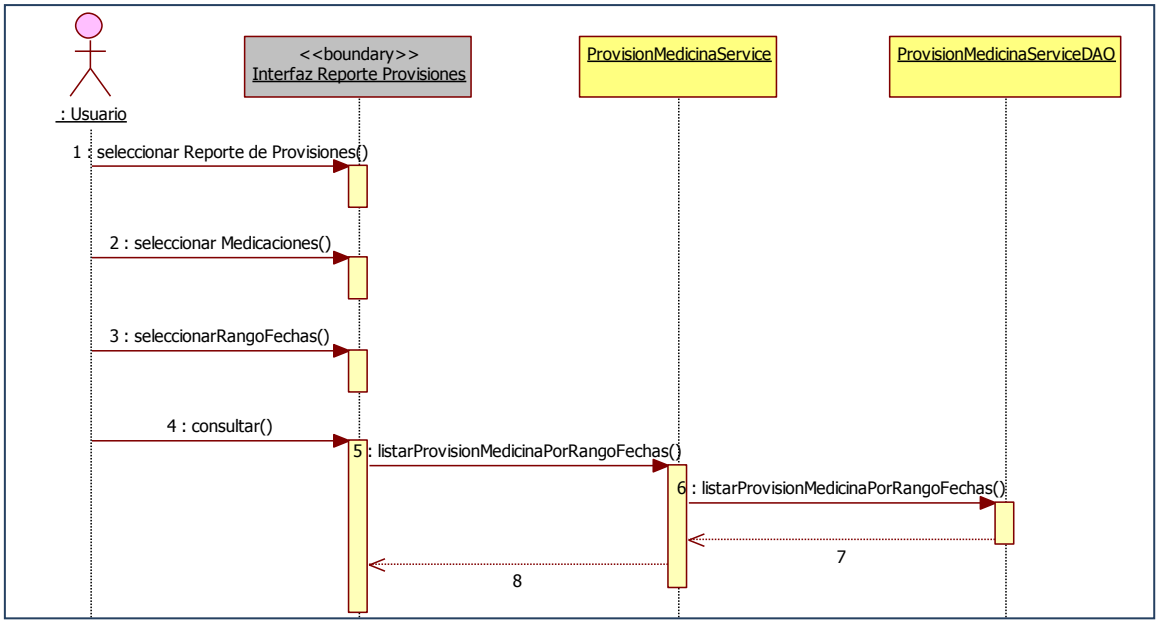


Figura 3.3.21. Diagrama de Secuencia de la Característica 19
Fuente: Autor

3.3.4.9. Iteración 9

20. Reporte de medicaciones por galpón desde su fecha de ingreso hasta la actualidad.
21. Reporte de los gastos generados por lote desde su fecha de ingreso hasta la actualidad.
22. Reporte de mortalidad por cada galpón desde su fecha de ingreso hasta la actualidad.
23. Reporte de Alimentaciones por lote en un rango de fecha determinado.

Diagramas de Secuencia

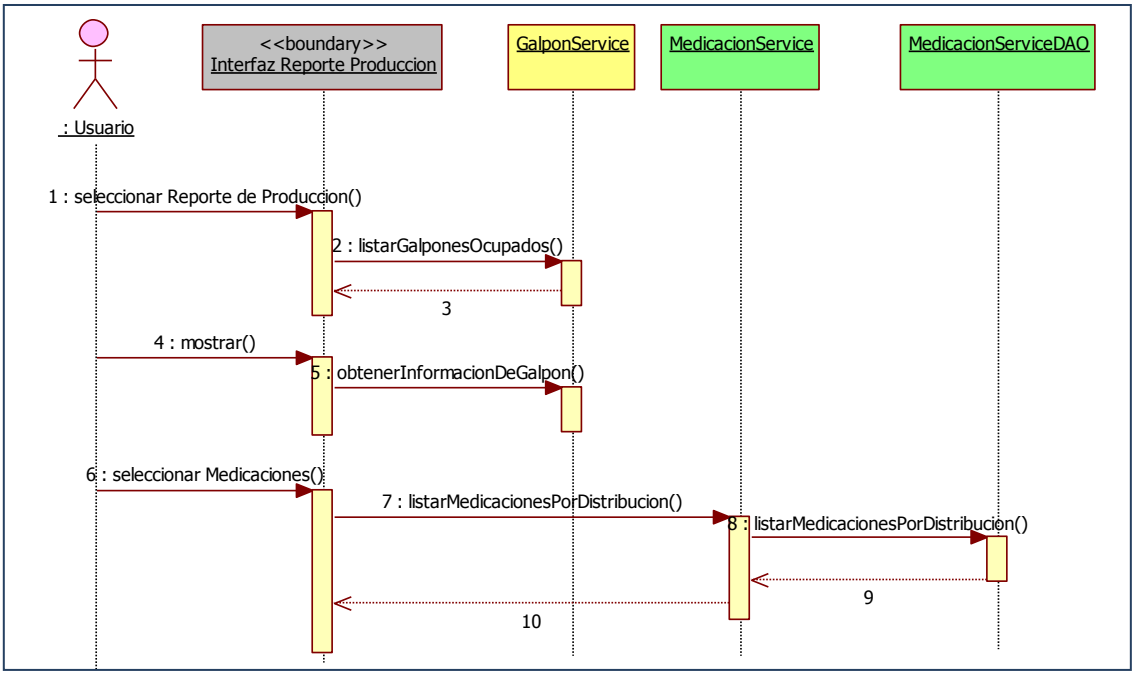


Figura 3.3.22. Diagrama de Secuencia de la Característica 20
Fuente: Autor

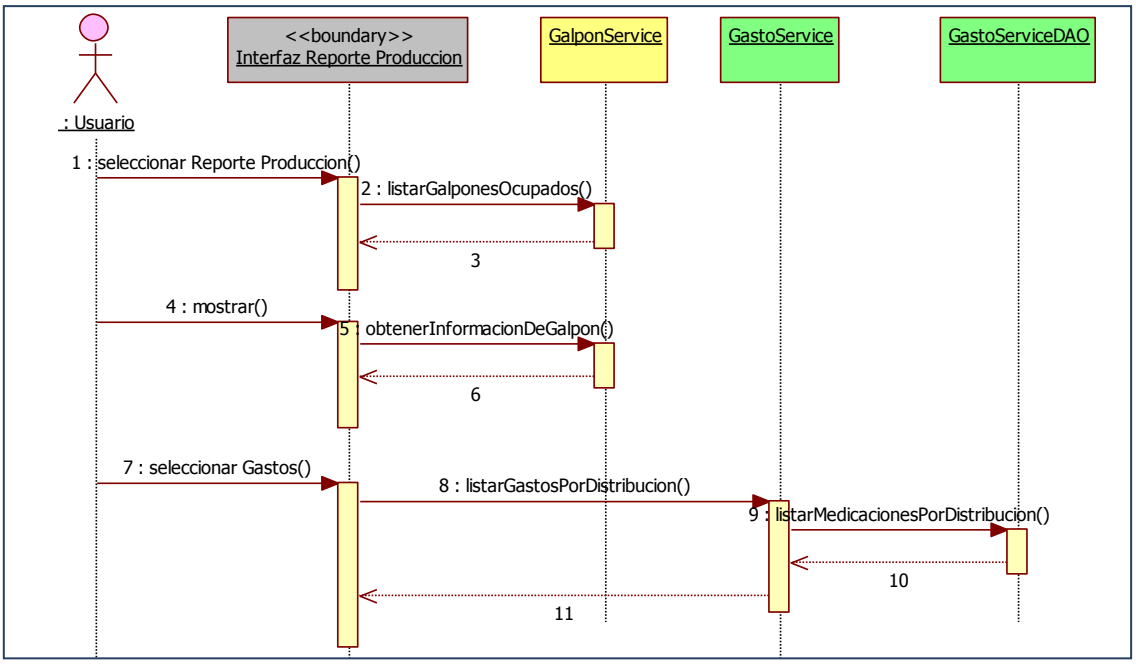


Figura 3.3.23. Diagrama de Secuencia de la Característica 21
Fuente: Autor

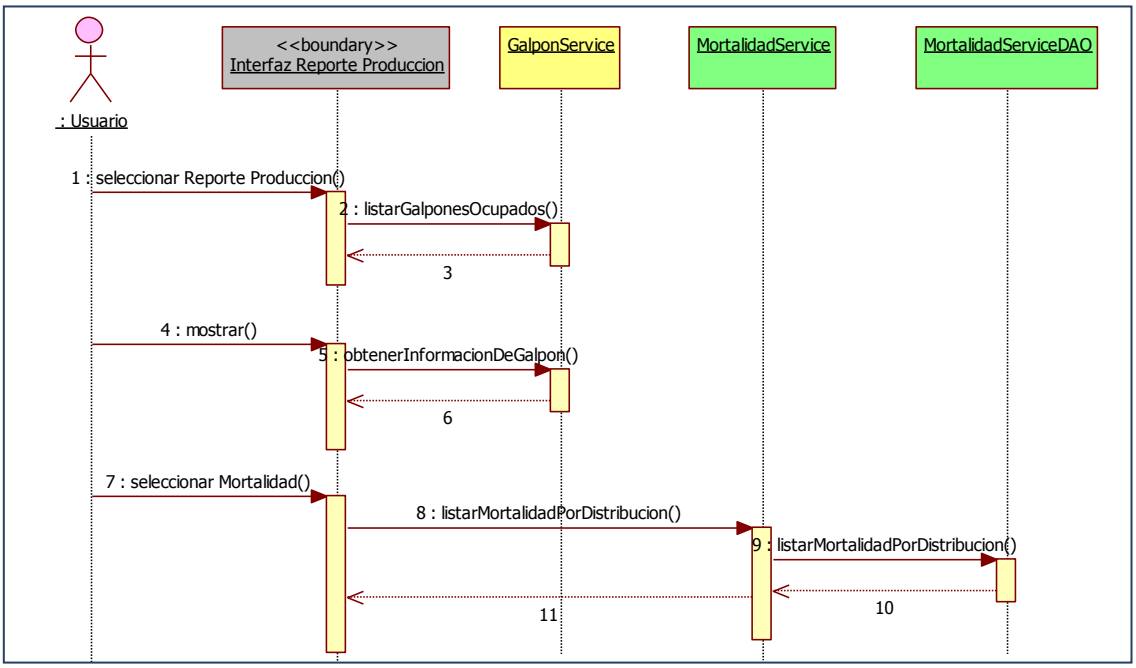


Figura 3.3.24. Diagrama de Secuencia de la Característica 22
Fuente: Autor

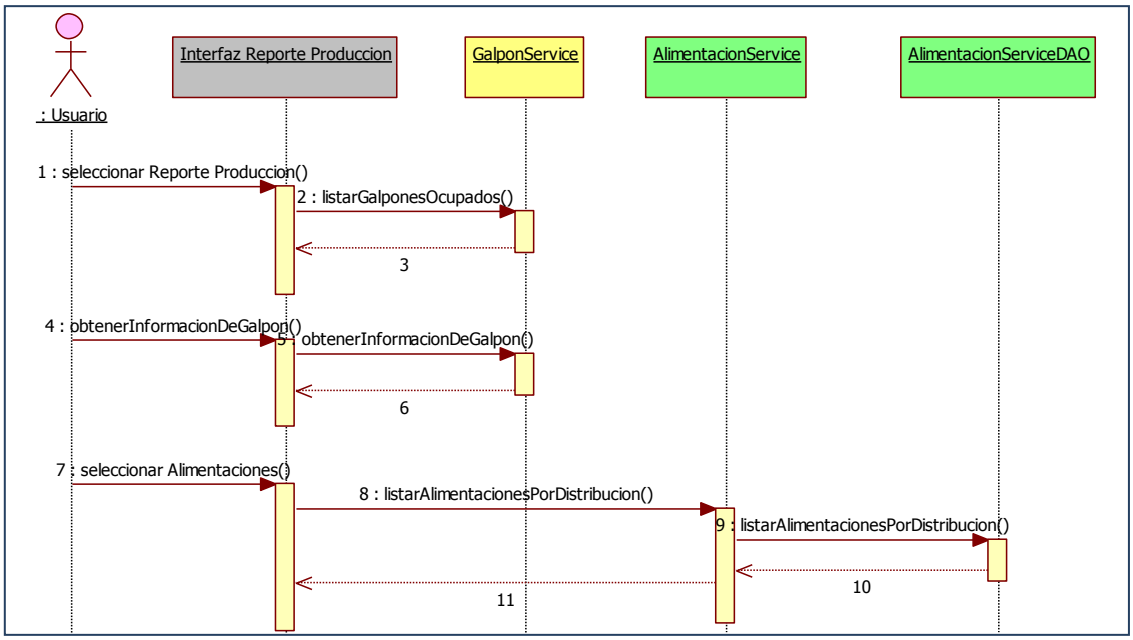


Figura 3.3.25. Diagrama de Secuencia de la Característica 23
Fuente: Autor

3.4. Pruebas

3.4.1. Pruebas Unitarias Ejecutadas

Resultado	Prueba	Proyecto	Error	Duración	Tipo de Prueba	Tiempo de Espera	Nombre de la Clase
Aprobado	testFindAll	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	AlimentacionDAOImpl
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	AlimentacionDAOImp
Aprobado	testFindByIddistribucion	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	AlimentacionDAOImp
Aprobado	testGetById	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	ControlproduccionDAOTest
Aprobado	testFindAll	com.avibita.dao.test		00:00:05	P. Unitaria	00:00:00	ControlproduccionDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	ControlproduccionDAOTest
Aprobado	testFindByIdlote	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	ControlproduccionDAOTest
Aprobado	testFindByCedula	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	ControlproduccionDAOTest
Aprobado	testGetById	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	DistribucionDAOTest
Aprobado	testFindAll	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	DistribucionDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	DistribucionDAOTest
Aprobado	testFindByIdlote	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	DistribucionDAOTest
Aprobado	testFindByIdgalpon	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	DistribucionDAOTest
Aprobado	testFindByCantidadpollitos	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	DistribucionDAOTest

Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	EmpleadoDAOTest
Aprobado	testFindByApellido	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	EmpleadoDAOTest
Aprobado	testFindByNombre	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	EmpleadoDAOTest
Aprobado	testFindByDisponible	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	EmpleadoDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	GalponDAOTest
Aprobado	testFindByDisponibleg	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	GalponDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	GastoDAOTest
Aprobado	testFindByIddistribucion	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	GastoDAOTest
Aprobado	testFindByFecha	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	GastoDAOTest
Aprobado	testFindByFechaingreso	com.avibita.dao.test		00:00:05	P. Unitaria	00:00:00	LoteDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	LoteDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	MedicacionDAOTest
Aprobado	testFindByIddistribucion	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	MedicacionDAOTest
Aprobado	testFindAll	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	MedicacionDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	MedicinaDAOTest
Aprobado	testFindByNombremedicina	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	MedicinaDAOTest
Aprobado	testFindByTipomedicina	com.avibita.dao.test		00:00:05	P. Unitaria	00:00:00	MedicinaDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	MortalidadDAOTest
Aprobado	testFindByIddistribucion	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	MortalidadDAOTest

Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	PesoDAOTest
Aprobado	testFindByIddistribucion	com.avibita.dao.test		00:00:02	P. Unitaria	00:00:00	PesoDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	ProvisionalimentoDAOTest
Aprobado	testFindBetweenRangoFechas	com.avibita.dao.test		00:00:04	P. Unitaria	00:00:00	ProvisionalimentoDAOTest
Aprobado	testSaveAndRemove	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	ProvisionmedicinaDAOTest
Aprobado	testFindByIdmedicina	com.avibita.dao.test		00:00:03	P. Unitaria	00:00:00	ProvisionmedicinaDAOTest
Aprobado	testFindByFecha	com.avibita.dao.test		00:00:01	P. Unitaria	00:00:00	ProvisionmedicinaDAOTest

Tabla 3.4.1 Listado de Pruebas de Unidad Ejecutadas
Fuente: Autor

3.4.2. Pruebas de Aceptación

3.4.2.1. Proceso de Producción

¿Qué pasaría si el usuario intenta eliminar un lote que tiene obreros y/o parámetros de producción asociados?

El sistema no permitirá la eliminación de ningún lote que tenga asociado obreros, alimentaciones, medicaciones, mortalidad, peso y gastos.

Se recomienda verificar el reporte de control de producciones por galpón antes de empezar una operación de eliminación.

¿Qué pasaría si el usuario intenta eliminar un lote que tiene obreros y/o parámetros de producción asociados?

El sistema no permitirá la eliminación de ningún lote que tenga asociado obreros, alimentaciones, medicaciones, mortalidad, peso y gastos.

Se recomienda verificar el reporte de control de producciones por galpón antes de empezar una operación de eliminación.

¿Qué pasaría si el usuario intenta ingresar alimentaciones o medicaciones con una cantidad de alimento mayor a lo disponible en bodega?

El sistema mostrará una alerta indicando que se corrija la información ingresada.

¿Se podría modificar la cantidad de pollitos de los lotes, una vez que ya han sido divididos en galpones?

El sistema no permite realizar esta actualización, por seguridad en la división del lote en galpones. Se debe eliminar el lote y volver a crear uno nuevo, pero antes de que se haya registrado cualquier parámetro de control de producción, de lo contrario se deberán eliminar primero todos los registros de control de producción asociados a ese lote.

¿Qué pasaría si el usuario intenta registrar un lote en galpones ya ocupados?

El sistema solo permite ocupar el galpón una sola vez. Cuando se ingrese el siguiente lote el galpón ocupado no se mostrará en la lista.

¿Qué pasaría si el usuario por error dejara algún dato en blanco?

El sistema no permite dejar campos en blanco, en caso de que ocurra se lanzará una alerta indicando que se realice la corrección.

CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES

4.1. Conclusiones

Después de haber realizado el estudio de la metodología FDD, se pudo constatar que la misma facilita la etapa de diseño, al no exigir casos de uso sino una lista de características evaluadas por el cliente; esto gracias a que FDD le presta más importancia a la comunicación y participación activa con el usuario que a la documentación excesiva.

El resultado de una buena planificación en FDD se verá reflejado en el tiempo de implementación de cada iteración, mismo que no debe ser mayor a dos semanas, de no ser así, las características estarían refiriéndose a aspectos demasiado generales y deben ser corregidas.

El acoplarse a FDD no es una tarea difícil, sin embargo se necesita de un alto grado de disciplina para aplicar las buenas prácticas de desarrollo ágil, que en principio podrían demandar mayor esfuerzo y tiempo, como por ejemplo el llevar el control de versiones de todos los artefactos, pero es lo único que nos asegurará que al final podamos tener un producto eficiente y de calidad.

Al trabajar desarrollando el sistema en pequeñas iteraciones, se pudo entregar al cliente resultados funcionales en poco tiempo, dándole satisfacción y motivándolo a participar activamente para alcanzar el producto deseado.

Realizar pruebas antes de la etapa de desarrollo permitió tener mayor seguridad en la implementación del código, además se pudo optimizar el tiempo de depuración de errores gracias a la utilización del framwork JUnit.

Se desarrolló el sistema web empleando herramientas de software libre de alta calidad, reduciendo costos y haciendo que el tiempo de retorno de la inversión no sea mayor a un año. Dentro del software utilizado están: el motor de base de datos MySQL, la herramienta de desarrollo Eclipse, el kit de desarrollo de java JDK, el servidor web y de aplicaciones Tomcat y la herramienta de modelado Star UML.

Durante el tiempo de desarrollo del sistema, se pudo constatar el alto desempeño de las herramientas utilizadas, puesto que ellas proveyeron confiabilidad y eficiencia en cada reproceso y ajuste de las características.

Para el manejo de contenidos se utilizó la plataforma web LifeRay, misma que permitió acoplar fácilmente la tecnología EJB de java, al consumir los beans en forma de portlets, además automatizó el formulario de ingreso al sistema.

La arquitectura lógica del sistema se la dividió en cuatro capas: Presentación, Servicios, Negocio y Datos; lo que permitió aislar la funcionalidad, del despliegue de los resultados finales al usuario, utilizando tecnologías diferentes en ambas: Flex y Java, respectivamente.

La división en capas brinda al programador una guía para hacer mejoras a la aplicación, sin que se torne en una tarea tediosa y desgastante, siguiendo el estándar establecido para tal fin y dividiendo las tareas en partes específicas para cada capa del proyecto.

Al realizar la programación del sistema en capas, el código se volvió mucho más claro y fácil de mantener, a medida que se avanzaba en el cumplimiento de las iteraciones.

4.2. Recomendaciones

Para que la metodología FDD tenga pleno éxito en su implementación, se recomienda que las características evaluadas por el cliente se refieran a aspectos específicos del negocio, de tal manera que involucren la menor cantidad de clases posibles.

Para obtener software de calidad, se recomienda utilizar conjuntamente con FDD: patrones de diseño, semántica correcta e integración continua.

Se recomienda utilizar solo el mínimo código necesario que permita realizar las pruebas de verificación de cada característica, de lo contrario se perdería la objetividad funcional de las iteraciones y se estaría incurriendo en el error de tomar en cuenta aspectos que tal vez no estén dentro de las necesidades del cliente.

Para la implementación de FDD se recomienda contar con un líder de equipo con experiencia, debido a que esta metodología inicia con un modelo global que es la base para construir la lista de iteraciones, y se necesita de alguien que sirva de guía al resto del equipo de desarrollo.

Cuando se trabaje en desarrollos de gran dimensión, se recomienda dividirlos en módulos funcionales específicos, y, trabajar aisladamente en cada uno con la lista de características y pruebas por iteración, para poder entregar el software a medida de la necesidad del cliente.

Se recomienda que se enseñe FDD a más de las metodologías de desarrollo ágiles tradicionales XP y Scrum, en las materias de ingeniería de software.

Se recomienda que el desarrollo de aplicaciones se lo realice en varias capas, al menos 3: Presentación, Negocio y de Datos, de tal manera que el producto final pueda gozar de seguridad, fiabilidad, escalabilidad y portabilidad.

Se recomienda que las pruebas unitarias no se las realice con el fin de lograr la perfección del flujo de procesos, sino para detectar posibles errores que puedan afectar al resto de componentes.

El administrador del sistema deberá contar con un perfil de formación de ingeniero o mayor, y necesitará manejar los siguientes temas: sistemas Web, bases de datos y herramientas de manejo de contenido.

BIBLIOGRAFÍA

- Stephen R. Palmer – John M. Felsing (2002). *FDD Driven Development, Practical Guide to Feature-Driven Development.*
- Colusso R. – Gabardini J. "Una introducción a las metodologías ágiles de desarrollo de software". (2011); http://agilesintro.wordpress.com/article/desarrollo-agil-de-software-3satfj6065tbv-2/
- Infante L. "Metodología Ágil – Introducción". (2009); http://www.bi-la.com/profiles/blogs/metodologia-agil-introduccion
- Ambler S. "Feature Driven Development (FDD) and Agile Modeling". (2006); http://www.agilemodeling.com/essays/fdd.htm
- Stephen R. Palmer. "Software Development and other stuff". http://www.step-10.com/
- Universidad Unión Bolivariana. "FDD (Feature Driven Development)"; http://www.ingenieriadesoftware.mex.tl/61162_FDD.html
- *Stephen R. Palmer. "Feature – Driven Development";* http://www.step10.com/SoftwareProcess/FeatureDrivenDevelopment/index.html
- Martin. "FDD & Web Development". (2003); http://www.featuredrivendevelopment.com/node/550
- Soderguit C. "Java Persistence API (JPA). Parte 1". (2008); http://www.juguy.org/index.php?option=com_content&view=article&id=157:java-persistence-api-jpa-parte-1&catid=77:capa-de-integraci&Itemid=48
- Greanier T. "Discover the secrets of the Java Serialization API". (2000); http://java.sun.com/developer/technicalArticles/Programming/serialization/
- Castaneda R. – Bartlett D. "Thinking in Java". 3rd ed. Revisión 2.0; http://www.cs.hut.fi/Docs/Eckel/TIJ3ed/TIJ321.htm
- Lacalle A. "Diseño con estándares". (2005); http://albertolacalle.com/diseno-estandares.htm

Anexo A: CARACTERÍSTICAS EVALUADAS POR EL CLIENTE

Anexo A: Características Evaluadas por el cliente para el desarrollo del sistema web para control de producción en la granja avícola "Marco Antonio Vivanco Álvarez".

Para la obtención de las características se identificó primeramente a las personas claves que servirían para la recopilación de información precisa sobre el proceso de control de producción; estas personas fueron dos funcionarios del área de Administración de Producción. A ellos se les realizó una serie de entrevistas en conjunto y por separado, con lo que se pudo saber a ciencia cierta las actividades que se desarrollan en las jornadas de trabajo diarias.

A pesar de que FDD no tiene una especificación concreta, recomienda que se lleve un registro de todos los artefactos cambiantes en el proceso de desarrollo, por tal motivo las entrevistas tienen cada una su registro, a fin de contar con información de respaldo sobre las características finales del sistema.

A continuación se presenta una revisión de las características obtenidas en cada una de las entrevistas y una especificación final de las características aprobadas por el cliente, no se toman en cuenta la información general del ámbito del negocio puesto que esa información se detalló en el modelo global, este documento tiene como objetivo principal la recopilación de características funcionales que describen puntualmente la necesidad del cliente

Debido al trabajo diario de los empleados las entrevistas tuvieron una duración promedio de 1.30 horas.

Historia de Revisiones

Fecha	Técnica aplicada	Objetivo	Duración	Participantes	Cargo
13-08-2011	Entrevista	Conocer el negocio de la granja	2 Horas	Ing. Ivanny Pallaroso	Jefe de Administración de Producción
				Ing. Marco Vivanco	Gerente.

- La granja cuenta con un total de 25 empleados entre directivos, administradores, obreros y choferes.
- Actualmente el área de producción se encuentra ubicada en la ciudad de Chone y sus áreas de gerencia y contable se encuentran ubicadas en la ciudad de Quito.
- La granja cuenta con un total de 12 galpones, mismos que se van ocupando a medida que se ingresan los lotes.
- Los lotes de pollitos con los que se trabaja, se compran de un día de nacidos, y solo se compran lotes de una sola raza a la vez (no se compran lotes con mezcla de razas).
- Los obreros no tienen un salario fijo, a ellos se les cancela por el control de lotes que realicen.
- El tiempo de producción dura entre 7 y ocho semanas.

Fecha	Técnica aplicada	Objetivo	Duración	Participantes	Cargo
15-08-2011	Entrevista	Conocer las actividades que se realizan durante el proceso de producción.	1 Hora	Ing. Ivanny Pallaroso	Jefe de Administración de Producción

El proceso de producción inicia con la compra de pollitos en pie, los pollitos son comprados por lotes y cada lote debe ser solo de una raza.

A cada lote se le asigna varios obreros para realizar todo el trabajo de campo de control de producción.

Cada lote de pollitos es distribuido en diferentes galpones en cantidades más pequeñas.

El control de producción se lo administra en cada galpón por separado e involucra las siguientes tareas de campo:

- Alimentación
- Medicación
- Mortalidad
- Peso
- Medicación
- Vacunación

En la granja se realizan provisiones de alimento y de medicina según la necesidad de cada galpón.

El alimento puede ser de cuatro tipos: pre-inicial, inicial, crecimiento y final; según la necesidad actual.

Fecha	Técnica aplicada	Objetivo	Duración	Participantes	Cargo
19-08-2011	Entrevista	Conocer los detalles de la información requerida	1 Hora	Ing. Ivanny Pallaroso	Jefe de Administración de Producción

La información de los lotes debe incluir la fecha de ingreso a la granja, la cantidad de pollitos y la raza.

La información de los galpones debe incluir la capacidad y su disponibilidad.

La información de empleados debe incluir cédula, apellido, nombre, fecha de nacimiento, estado civil, dirección, teléfono y el tipo de empleado.

La información de medicina debe incluir el tipo: medicación o vacuna.

Las provisiones tanto de alimento como de medicinas deben incluir la fecha, detalle del tipo de alimento/medicina y el valor unitario.

En el registro de alimentaciones, se debe indicar la fecha, el tipo de alimento y la cantidad de alimento en sacos empleada.

El registro de medicaciones debe indicar la fecha, la dosis de medicación, el nombre y el tipo de medicina utilizado.

El registro de peso debe incluir la fecha y el peso promedio del galpón.

El registro de mortalidad debe incluir la fecha, cantidad de pollitos muertos y algo muy importante es la causa de la mortalidad.

El registro de gastos debe incluir la fecha, descripción y el valor.

Fecha	Técnica aplicada	Objetivo	Duración	Participantes	Cargo
23-08-2011	Entrevista	Conocer las funciones de los empleados durante el proceso de producción	1 Hora	Ing. Ivanny Pallaroso	Jefe de Administración de Producción

Los obreros son los responsables de ejecutar todas las tareas de campo del proceso de producción: Alimentación, medicación, peso, revisar la mortalidad.

Los Administradores de Producción son los encargados de:

- Ingresar la información de cada lote

- Distribuir los lotes en galpones
- Asignar obreros por lote
- Registrar la información generada por las tareas realizadas por los obreros.
- Llevar el registro de los gastos generados por cada galpón.
- Generar los reportes solicitados por la gerencia y por el área contable, entre los cuales están:
    - Listado de alimentaciones, medicaciones, peso, mortalidad, vacunaciones, gastos, por cada galpón, desde su fecha de ingreso hasta la actualidad.
    - Listado de lotes ingresados en un rango de fecha determinado.
    - Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.
    - Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.
    - Provisiones de alimento en un rango de fechas determinado.
    - Provisiones de medicinas y vacunas en un rango de fechas determinado.
    - Total de alimento consumido por cada galpón, clasificado por el tipo de alimento.
    - Medicinas utilizadas por cada galpón.
    - Detalle de medicinas en bodega.
    - Total de alimento en bodega, clasificado por tipo.
- Registrar toda la información que se genera durante el proceso de cría, en el sistema.

- Los usuarios podrán acceder al sistema solo si cuentan con un código de ingreso.
- Los usuarios que quisieren ingresar al sistema, deben tener conocimientos básicos de informática, y de producción avícola para que puedan entender los reportes que genera el sistema.

Fecha	Técnica aplicada	Objetivo	Duración	Participantes	Cargo
26-08-2011	Entrevista	Conocer los requerimientos de diseño y desarrollo	1 Hora	Ing. Marco Vivanco	Gerente

Se requiere de una interfaz fácil e intuitiva en su manejo, que proporcione navegabilidad al usuario.

Los colores identificativos de la granja son azul y blanco.

El logo de la página principal será provisto por la granja.

El sistema debe mantener su diseño en cualquier navegador en el que se visualice.

Las herramientas que se utilicen para el desarrollo del software deben ser de carácter libre a fin de abaratar costos.

Características funcionales verificadas y aceptadas por el cliente

El sistema debe permitir:

- Gestionar la información de los galpones.
- Gestionar la información de los lotes de pollitos.
- Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser:
    - Administrador de producción

    - Administrador Contable
    - Obrero.

- Verificar la disponibilidad de los obreros (empleados que no están a cargo de ningún lote).
- Asignar uno o varios obreros para trabajar, por cada lote.
- Registrar el salario para cada obrero por el trabajo de control de un determinado lote.
- Verificar el estado de los galpones, ocupado o desocupado.
- Distribuir cada lote en galpones.
- Verificar que la cantidad de pollitos que se ingresa a un galpón no sea mayor a la cantidad del lote al que pertenece.
- Verificar que la cantidad de pollitos que se ingresa en un galpón no sea mayor a la capacidad del mismo.
- Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.
- Gestionar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se usó.
- Gestionar datos de las medicaciones que se realiza a cada galpón, indicando si se usó un medicamento o vacuna.
- Gestionar los datos de la toma de peso de cada galpón.
- Gestionar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.
- Gestionar los gastos que genera cada lote.

El sistema debe generar los siguientes reportes:

- Listado de lotes ingresados en un rango de fecha determinado.
- Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.
- Listado de galpones ocupados y disponibles.
- Provisiones de alimento en un rango de fechas determinado.
- Provisiones de medicinas y vacunas en un rango de fechas determinado.
- Medicaciones por cada galpón desde su fecha de ingreso hasta la actualidad.
- Alimentaciones por cada galpón.
- Gastos generados por galpón.
- Mortalidad por cada galpón.
- Medicaciones por cada galpón.
- Detalle de medicinas en bodega.

Anexo B: ESPECIFICACIÓN DE REQUERIMIENTOS

Anexo B: Especificación de Requerimientos de Software para el desarrollo del sistema web para control de producción en la granja avícola Marco Antonio Vivanco Álvarez.

Introducción

El presente documento es una especificación de requerimientos de software (ERS) para el desarrollo del sistema vía web de control de producción en la granja avícola Marco Antonio Vivanco Álvarez.

En el mismo se identifican las necesidades puntuales de la granja referente a su proceso de producción, y, se lo ha estructurado basándose en el estándar IEEE 830.

1.1 Propósito

Esta especificación tiene como propósito definir de forma precisa y clara las funcionalidades, restricciones y alcance del sistema a desarrollar.

El documento va dirigido tanto a los desarrolladores como a los directivos de la granja, después de su aprobación, este documento constituirá la base sobre la cual se establecerán las características funcionales, y requerimientos de hardware y software, siendo esta la guía en el proceso de desarrollo del sistema.

La presente especificación de requerimientos está dirigida al área gerencial de la granja para que sirva como una visión general del sistema, y para los desarrolladores, director y codirector como un resumen funcional del software que se quiere obtener.

1.2 Ámbito del Sistema

El producto de software que se desarrollará es un sistema vía web para la granja avícola “Marco Antonio Vivanco Álvarez”, que permita administrar su proceso de producción y ofrecer servicios de acceso a la información en línea, constituyéndose así como una herramienta efectiva para el área administrativa.

Los módulos que se incluyen son para administrar la información que se genera en todo el proceso de producción, controlar insumos, controlar gastos y generar reportes. La información inicial que presentará el sistema será provista por la granja.

1.3 Definiciones, acrónimos y abreviaturas

- ***ERS***: Especificación de requerimientos de software.
- ***Parámetros Zootécnicos***: Se refiere a mortalidad, alimentación, medicación, gasto y peso, de cada producción.
- ***Producción***: Ciclo que cumple el lote en el proceso de cría.
- ***Galpón***: Lugar físico en el que son almacenados los pollos.
- ***T.I.:*** Tecnologías de Información

2 Descripción General

2.1 Perspectiva del Producto

Este sistema es el primero en construirse para la granja, y será el pionero en implementarse en el área avícola dentro de la zona norte de la provincia de Manabí.

El desarrollo del sistema se lo hará en su totalidad con herramientas de software libre, abaratando costos y obteniendo un retorno de la inversión a corto plazo.

El sistema brindará una interfaz amigable e intuitiva para el usuario utilizando manejador de contenidos que facilite el acceso a la información.

2.1.1 Interfaces de Usuario

Las interfaces que mostrará el sistema son:

- Gestión de empleados.
- Gestión de galpones.
- Gestión de lotes
- Gestión de medicinas
- Control de Producción.
- Provisiones
- Reportes

2.1.2 Interfaces de Hardware

Debido a que el acceso al sistema es vía Internet, se utilizará el puerto 80 para el intercambio de información entre los computadores clientes y el Servidor. Se utilizara el protocolo AMF para el consumo de los servicios web.

2.1.3 Interfaces de Software

- Sistema operativo del servidor en plataforma Linux.

  Nombre: Cent OS

  Versión: 6

- Servidor de aplicaciones Java para plataforma Linux.

  Nombre: Apache Toncat

Versión: 2.4.1

- Motor de base de datos para plataforma Linux.

  Nombre: MySQL Database Server

  Versión: 5.0.45

- Herramientas de desarrollo Java para plataforma Linux.

  Nombre: Java Development Kit (JDK)

  Versión: 3.6

2.1.4 Operaciones

El sistema tendrá su página de presentación que será pública, en la que se pedirá el usuario y la clave para acceder al sistema, además existirá otra parte privada donde se podrán administrar funcionalidades especiales que tienen que ver con la actividad de la granja.

El sistema será multiusuario, soportará conexiones de varios usuarios al mismo tiempo.

La sesión de las páginas que tengan una inactividad mayor a 5 minutos será deshabilitada.

2.1.5 Funciones del producto

- *Módulo de Ingresos,* que permitirá ingresar, modificar o eliminar la información de empleados, galpones y lotes.
- *Módulo de Control de Producción*, permitirá gestionar la información generada en el proceso de cría, desde el ingreso de los pollitos hasta su salida para la venta.
- *Módulo de Insumos,* permitirá ingresar, actualizar o eliminar una provisión de alimento y medicina.

- *Módulo de Reportes*, que permitirá revisar reportes como: Lotes ingresados en un rango de fecha determinado, Distribución de los lotes en galpones, Repartición de insumos por lote, asignación de obreros, saldo de pollitos por galpón, actividades de producción por galpón (alimentaciones, medicaciones, registro de peso, mortalidad, gastos), estándares de cría.

2.2 Características de los Usuarios

El sistema manejará dos perfiles de usuario: Administrador del sistema y Administrador de producción.

El administrador del sistema deberá contar con un perfil de formación de ingeniero o mayor, y necesitará manejar los siguientes temas:

- Sistemas Web.
- Base de datos MySQL
- Herramientas de Manejo de Contenidos.

El Administrador de producción deberá conocer el negocio de la granja, a fin de que pueda gestionar la información del proceso de producción, además necesitaran contar con experiencia en aplicaciones web, de tal manera que puedan manejar los reportes de producción y entender rápidamente los términos utilizados.

2.3 Restricciones

Se considerarán las siguientes restricciones:

Para poder acceder al sistema, los usuarios necesitan de una conexión a internet y deben tener instalado un navegador web en sus ordenadores.

Todos los usuarios necesitan de un código de usuario y una contraseña para poder ingresar al sistema.

3. Especificación de Requerimientos

3.1 Interfaces Externas

El ancho de todas las ventanas será de 800 pixeles, y el alto variará de acuerdo al contenido de cada página web.

3.1.1 Página principal

En la Figura 3.1 se presenta la estructura de la página inicial del sistema web, su descripción se indica en la Tabla 3.1.

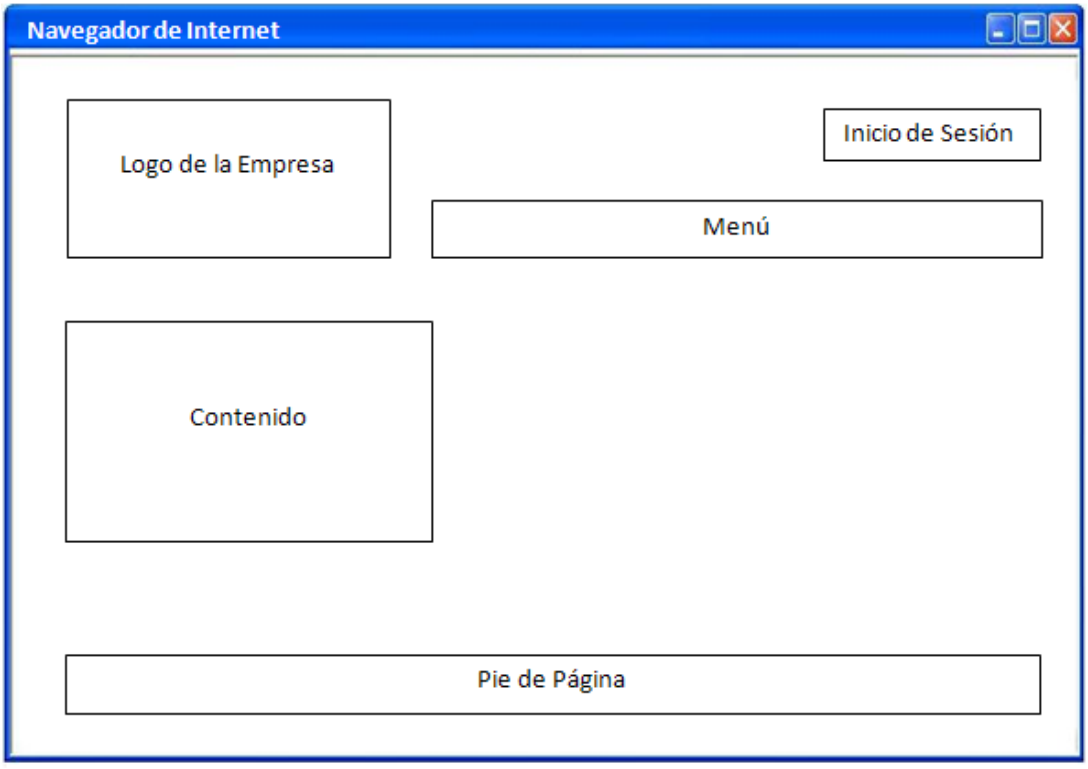


Figura 3.1 Diseño de la página principal del sistema
Fuente: Autor

Objeto	Descripción
Logo	Muestra el logo identificativo de la empresa.
Inicio de Sesión	Opción que activa la interfaz para el ingreso de nombre de usuario y contraseña.
Menú	Muestra el menú del sistema.
Contenido	Muestra la interfaz para ingresar los datos de nombre de usuario y contraseña para acceder al sistema.
Pie de Página	Muestra los datos de autor y los datos de ubicación de la empresa.

Tabla 3.1 Descripción de la página Principal del Sistema
Fuente: Autor

3.1.2 Página secundaria

En la Figura 3.2 se presenta la estructura de las páginas secundarias del sistema web, la cual presentará enlaces a las diferentes secciones, su descripción se indica en la Tabla 3.2.

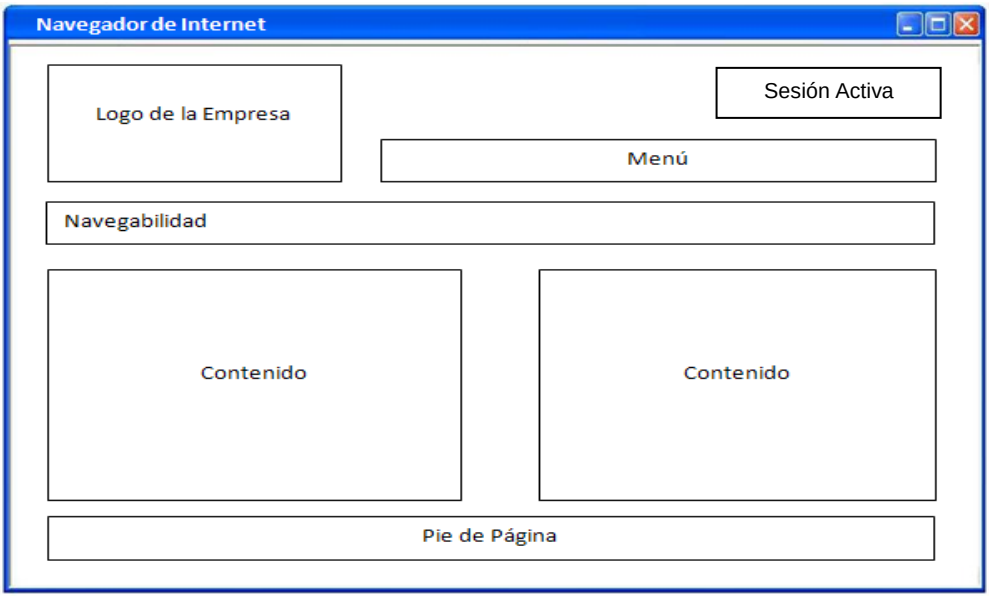


Figura 3.2 Diseño de la página secundaria del sistema
Fuente: Autor

Objeto	Descripción
Logo	Muestra el logo identificativo de la empresa.
Inicio de Sesión	Indica el nombre de usuario que ha iniciado sesión.
Menú	Muestra el menú del sistema.
Navegabilidad	Muestra la opción del menú seleccionada.
Contenido	Muestra el contenido de la opción del menú seleccionada.
Pie de Página	Muestra los datos de autor y los datos de ubicación de la empresa.

Tabla 3.2 Descripción de la página secundaria del sistema
Fuente: Autor

3.2 Especificación de requerimientos

A continuación se detalla la lista de características evaluadas y aceptadas por el cliente, misma que se obtuvo en el diseño del modelo global de la metodología FDD.

El sistema debe permitir:

- Gestionar la información de los galpones.
- Gestionar la información de los lotes de pollitos que se compran.
- Gestionar la información de los empleados de la granja, indicando el tipo de empleado que puede ser:
    - Administrador de producción
    - Administrador Contable
    - Obrero

- Verificar la disponibilidad de los obreros (empleados que no están a cargo de ningún lote).
- Asignar uno o varios obreros para trabajar, por cada lote.
- Registrar el salario para cada obrero por el trabajo de control de un determinado lote.
- Verificar el estado de los galpones, ocupado o desocupado.
- Distribuir cada lote en galpones.
- Verificar que la cantidad de pollitos que se ingresa a un galpón no sea mayor a la cantidad del lote al que pertenece.
- Verificar que la cantidad de pollitos que se ingresa en un galpón no sea mayor a la capacidad del mismo.
- Gestionar la información de las provisiones de alimento, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.
- Gestionar la información de las provisiones de medicina, indicando el tipo de alimento: pre-inicial, inicial, crecimiento o final.
- Gestionar los datos de las alimentaciones que se realiza a cada galpón, indicando el tipo de alimento que se utilizó.
- Gestionar los datos de las medicaciones que se realiza a cada galpón, indicando si se usó un medicamento o vacuna.
- Gestionar los datos de la toma de peso; el peso promedio de la pesada.
- Gestionar la mortalidad diaria en el caso de que la haya, y la causa de esa mortalidad en particular.
- Gestionar los gastos que genera cada lote.

El sistema debe generar los siguientes reportes:

- Listado de lotes ingresados en un rango de fecha determinado.
- Obreros ocupados, con su salario y en cuales galpones se encuentran trabajando.
- Listado de galpones ocupados, con la cantidad inicial de pollitos y el saldo actual.
- Provisiones de alimento en un rango de fechas determinado.
- Provisiones de medicinas en un rango de fechas determinado.
- Medicaciones por lote desde su fecha de ingreso hasta la actualidad.
- Alimentaciones por lote desde su fecha de ingreso hasta la actualidad.
- Gastos generados por lote desde su fecha de ingreso hasta la actualidad.
- Mortalidad por cada galpón desde su fecha de ingreso hasta la actualidad.
- Total de alimento consumido por cada galpón, clasificado por el tipo de alimento.
- Detalle de medicinas en bodega.
- Total de alimento en bodega, clasificado por tipo.

3.3 Requerimientos de desarrollo

Se utilizará la metodología de desarrollo ágil FDD (Desarrollo Basado en Características), misma que permite realizar iteraciones cortas y entregas rápidas al cliente para su verificación y aceptación.

Además el modelo de proceso que se empleará será evolutivo incremental, puesto que es el que más de adapta a la naturaleza rápida e incremental de FDD. El trabajo se orientará hacia el desarrollo de un sistema flexible que permita incorporar de manera sencilla cambios y nuevas funcionalidades.

3.4 Requisitos de rendimiento

El sistema permitirá mostrar la información generada a lo largo del proceso de producción.

3.5 Atributos del sistema

3.5.1 Fiabilidad

El sistema web será fiable porque será realizado bajo metodologías y procesos de ingeniería de software, además se utilizarán estándares W3C (World Wide Web Consortium).

3.5.2 Disponibilidad

El sistema estará disponible 24/7, porque estará publicado en Internet, también se mantendrá un cronograma de la base de datos y el software para respaldar el sistema web y su contenido.

3.5.3 Seguridad

El sistema web será seguro porque utilizará protocolos adecuados para lograr una transferencia segura de datos en Internet. Se utilizará usuarios y contraseñas para la validación de identidades. Las contraseñas se guardarán en la base de datos de manera encriptada. Además existirán permisos de acceso a ciertas secciones que se les asigna a los usuarios, estos permisos son establecidos únicamente por el administrador del sistema.

3.5.4 Escalabilidad

El sistema web será escalable porque estará diseñado en una arquitectura distribuida, y definida adecuadamente de acuerdo a los fundamentos de la ingeniería de software.

3.5.5 Portabilidad

El sistema será portable porque se desarrollará en lenguaje Java. Una de las principales características de este lenguaje es que es independiente de la plataforma, es decir que los programas escritos en Java pueden ejecutarse en cualquier tipo de hardware y sistema operativo.

ANEXO C: MANUAL DE INSTALACIÓN DE REQUERIMIENTOS

Anexo C: Manual de Instalación de Requerimientos del sistema vía web para control de producción en la granja avícola Marco Antonio Vivanco Álvarez.

El presente manual está dirigido a la instalación y configuración técnica del sistema web para control de producción en la granja avícola “Marco Antonio Vivanco Álvarez”, en el cual se analizará las características y requerimientos técnicos para un correcto funcionamiento.

La entrega de los archivos del sistema se la realiza en formato electrónico adjuntado en un CD.

Instalación LifeRay 6.1

Liferay es una de las aplicaciones más flexibles en el mercado hoy en día con respecto a los entornos de servidor de aplicaciones, y debido a ello es muy fácil de instalar. Si se cuenta con un servidor de aplicaciones ya configurado, se pueden utilizar las herramientas para el despliegue que vienen con el servidor de aplicaciones. Si no se tiene un servidor de aplicaciones, Liferay ofrece varios paquetes de servidores de aplicaciones entre las que elegir, estos son muy fáciles de instalar y con una pequeña cantidad de configuración se pueden tener listos para un entorno de producción.

Un paquete es una aplicación de servidor de código abierto que ya viene con LifeRay pre instalado.

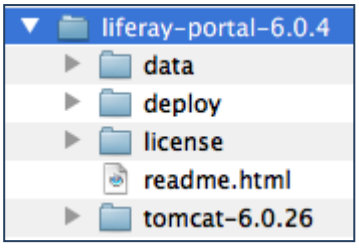
- Descargar el JDK 6, de la página http://www.oracle.com. Realizar la instalación siguiendo los siguientes pasos:

```
#./jdk-6u22-linux-x64-rpm.bin
Unpacking...
[....]
Press Enter to continue.....
# java -version
java version "1.6.0_17"
OpenJDK Runtime Environment [....]
# alternatives --install /usr/bin/java java
/usr/java/latest/bin/java
# alternatives --config java
Hay 3 programas que proporcionan 'java'.
[....]
# java -version
java version "1.6.0_22"
Java(TM) SE Runtime Environment [....]
```

- Descargar el paquete que viene configurado con el servidor Tomcat, de la página oficial de LifeRay: http://www.liferay.com.

Instalación del paquete de LifeRay con Tomcat

1. Descomprimir el paquete en el directorio /opt. El paquete contiene la misma estructura de directorios, indiferentemente del servidor de aplicaciones que se esté utilizando:



2. Descargar el driver JDBC para MySQL y copiarlo en la ruta /liferay6.1.0-ce-ga1/[Tomcat]/lib/ext
3. Los paquetes de LifeRay vienen con una aplicación de ejemplo llamada *sevencogs-hook*, en la que se demuestran algunas de las características del portal. A fin de tener una base de datos limpia se

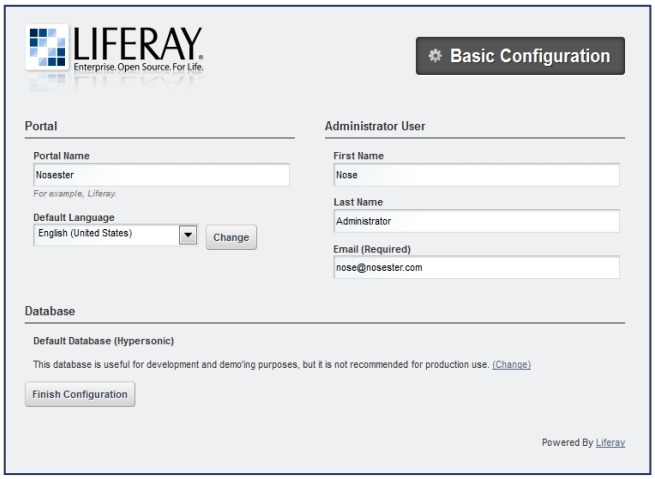
debe eliminar esta aplicación del siguiente directorio: [Tomcat Home]/webapps folder.

4. Iniciar el servicio de liferay ejecutando el siguiente comando: /liferay6.1.0-ce-ga1/[Tomcat]/bin$ ./startup.sh start.

Nuestro portal está listo para cargar la configuración de diseño y base de datos.

En el nombre del portal escribimos: avibita.com

En el lenguaje por defecto seleccionamos Español

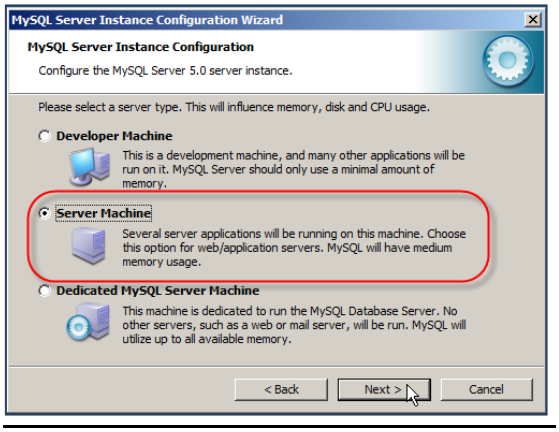


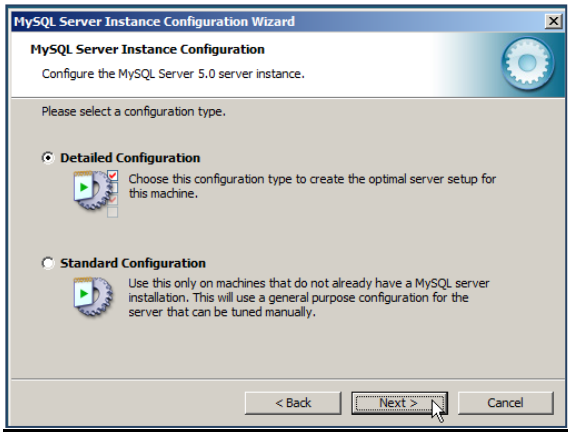
Seleccionamos la sección de Base de Datos, en la que configuraremos la conexión hacia nuestra base MySQL, con los siguientes datos:

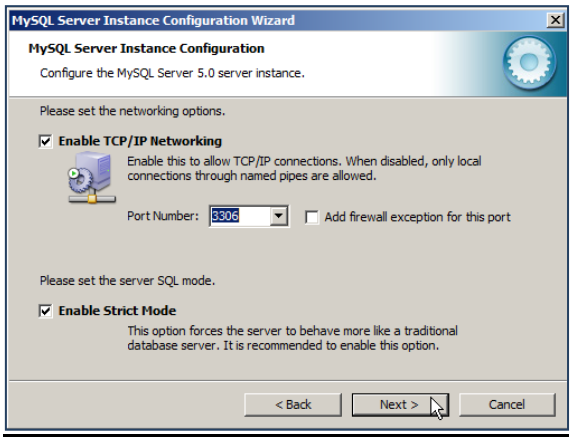
- JDBC: com.mysql.jdbc.Driver
- Base de Datos: avibita
- Nombre de Usuario: root
- Contraseña: ibmlnx

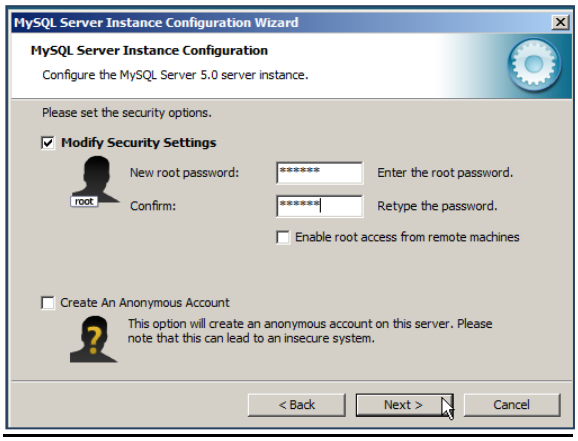
Espacio en Blanco puesto para completar la página.

Instalación del Sistema de Gestión de Base de Datos MySQL

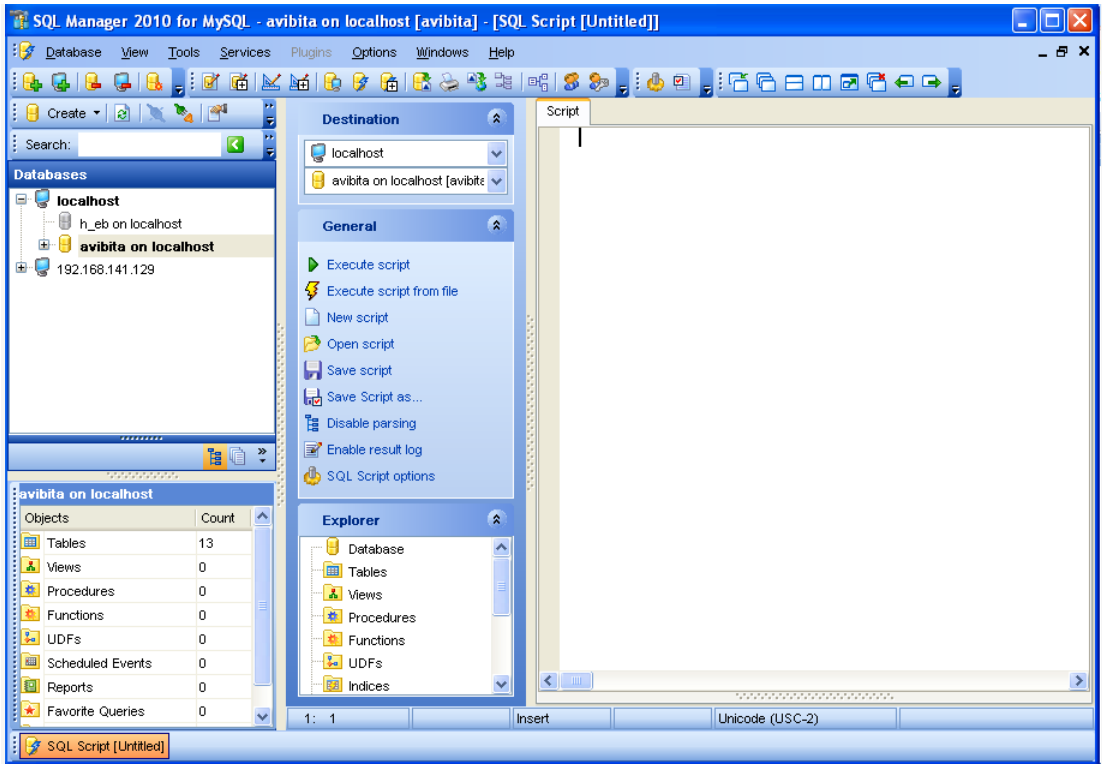








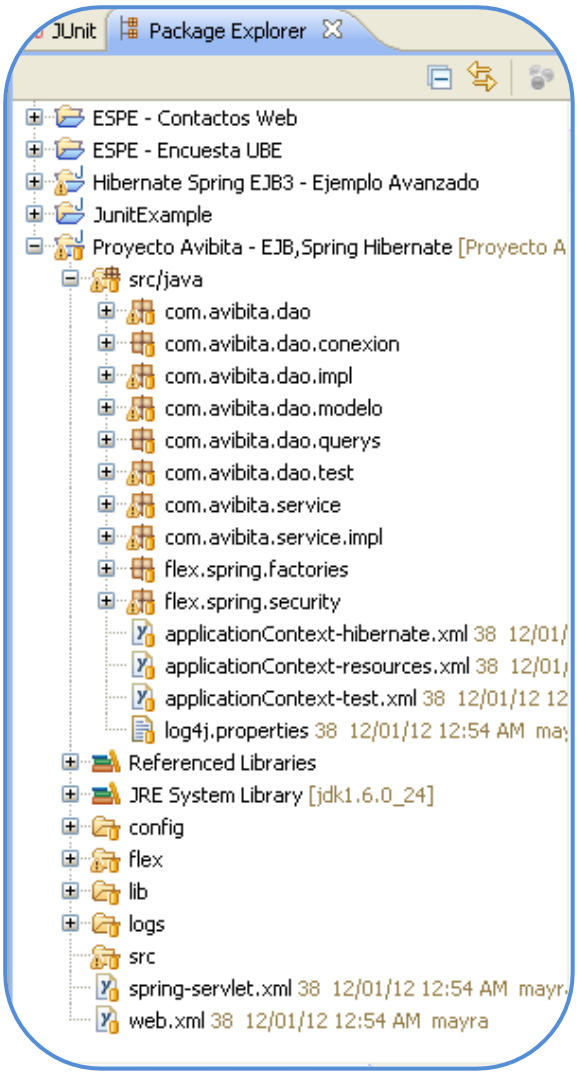
Se carga el editor de MySql para ejecutar el Script de creación de la base de datos avibita.sql, mismo que se encuentra en el CD de entrega del sistema.



Espacio en Blanco puesto para completar la página.

Configuración de Archivos

La lógica de negocio del sistema web está encapsulada en un solo directorio llamado Avibita, el mismo debe ser copiado en la siguiente dirección del servidor:/opt/lamp/avibita



Anexo D: MANUAL DE USO DEL SISTEMA

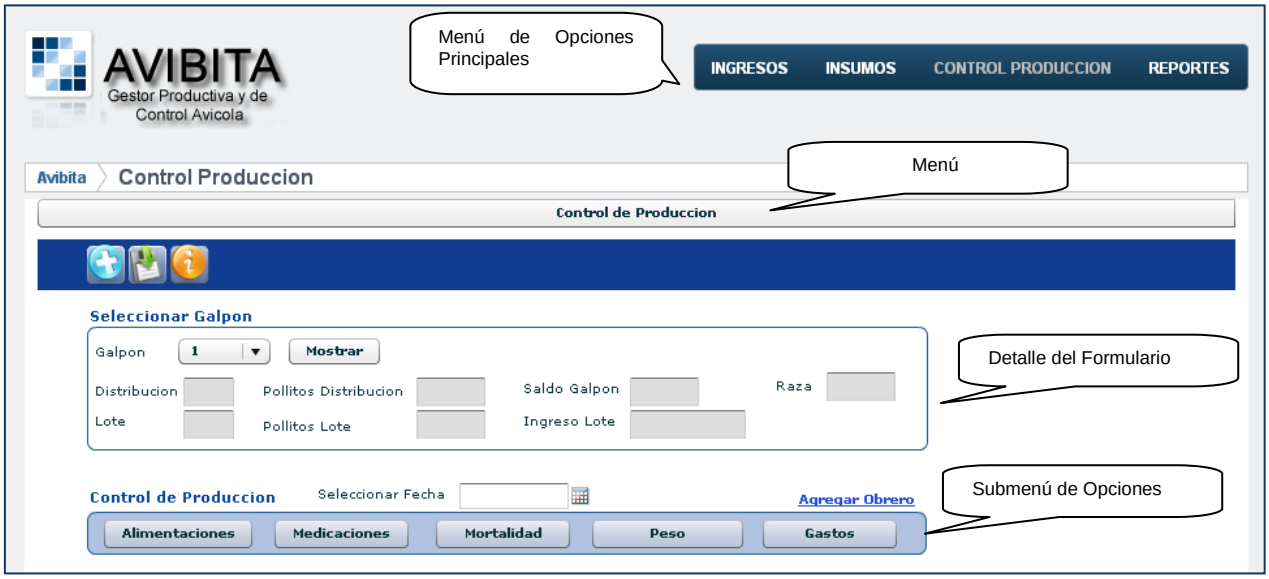
Anexo D: Manual de Uso del Sistema para Control de Producción en la Granja Avícola "Marco Antonio Vivanco Álvarez"

Para ingresar al sistema es necesario contar con un usuario y una contraseña, creados por el administrador del sistema.

En la parte izquierda de la página de principal se encuentra el formulario de inicio de sesión. Aquí se debe ingresar el *usuario*, la *contraseña* y pulsar la opción *Ingresar* para poder tener acceso al sistema.

Espacio en Blanco puesto para completar la página

Una vez iniciada la sesión, se puede acceder a los diferentes enlaces del menú de navegación del proceso de producción:



Detalle de las opciones principales del Sistema:

Ingresos: Permite registrar la información de lotes, empleados, galpones y medicinas.

Insumos: Sirve para llevar el control de las provisiones de medicinas y de alimento.

Control de Producción: Permite realizar el seguimiento de los parámetros zootécnicos de cada galpón, generados en el proceso de producción.

Consideraciones Generales

La información se debe llenar de arriba hacia abajo.

El símbolo decimal para llenar los campos numéricos es el punto (.).

Las filas de las grillas que muestran información, pueden ser ordenadas por cualquiera de los datos mostrados.

Los formularios cuentan con un botón de ayuda  para introducir al usuario sobre su uso.

Todos los formularios tienen una barra de Ejecución, que muestra las opciones que se pueden realizar a los datos:

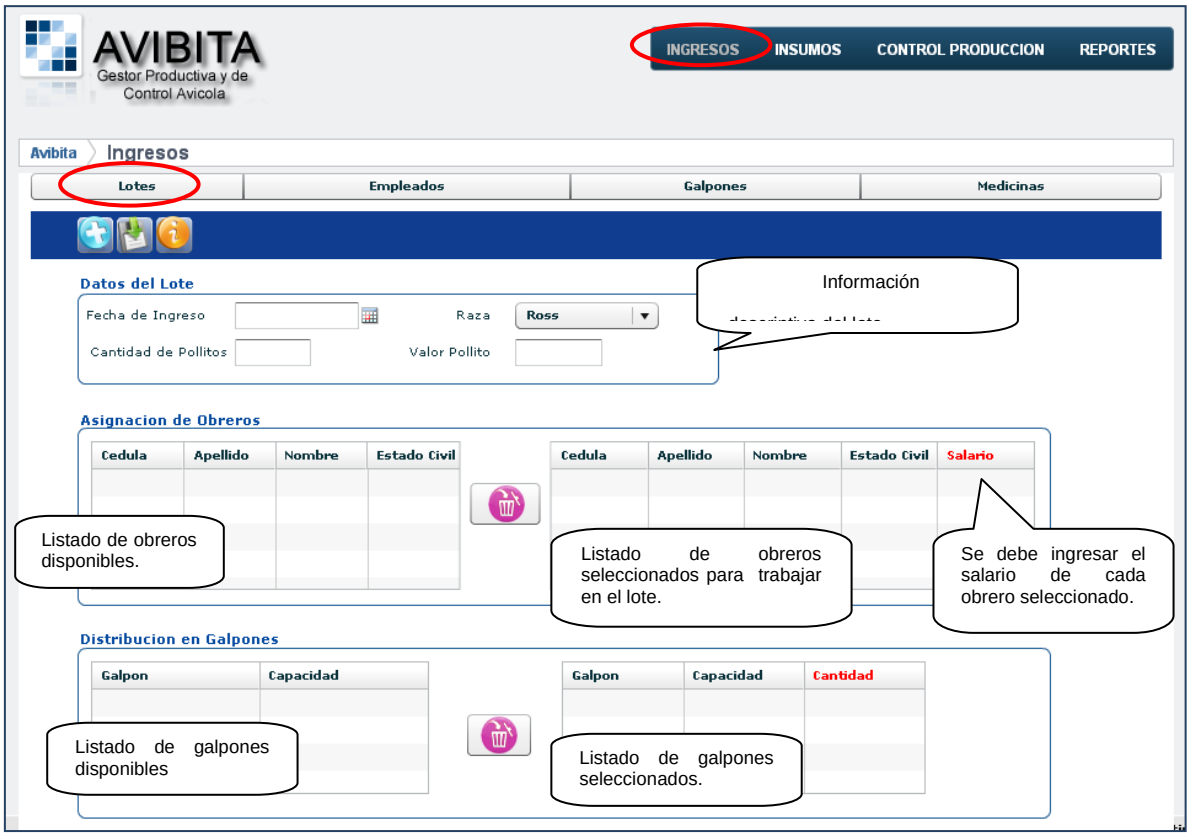
Introducción a los Formularios

Ingreso – Lotes: Este formulario permite ingresar la información de los lotes de pollitos de cero días de nacidos.

Secuencia de ingreso:

1. Se debe dar clic en el botón Nuevo  , para activar las casillas de texto en blanco.
2. Ingresar la información descriptiva del lote: Fecha de Ingreso, Cantidad de pollitos, Raza y Valor por pollito.
3. Seleccionar los obreros que trabajaran en el lote.
4. Ingresar el salario de cada obrero.
5. Seleccionar los galpones en los que se distribuirá el lote.
6. Ingresar la cantidad de pollitos distribuida en cada galpón.

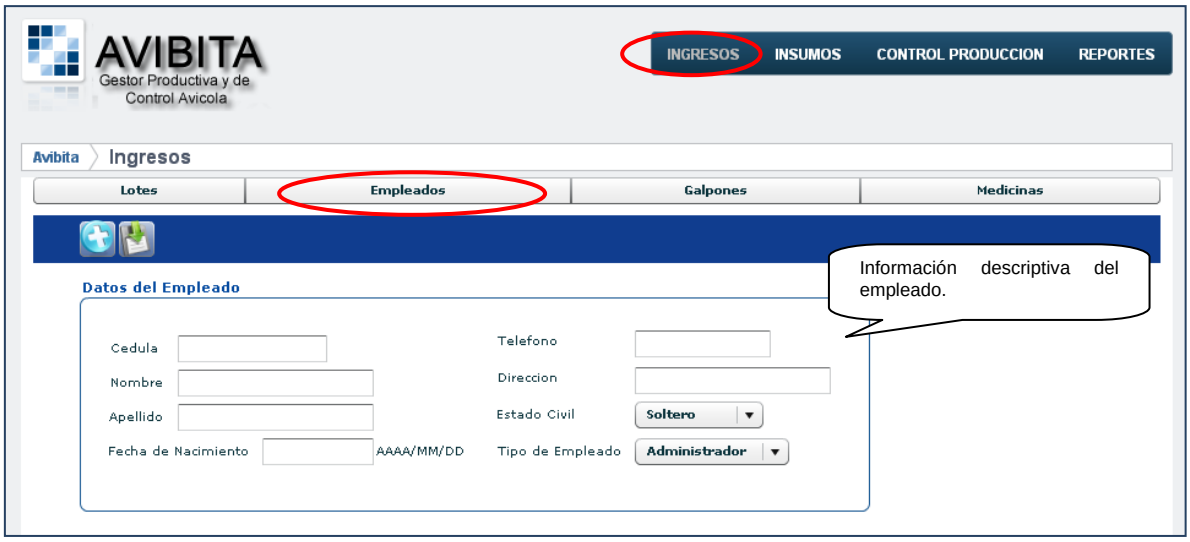
Espacio en Blanco puesto para completar la página



Ingreso – Empleados: Este formulario permite registrar la información de todos los empleados de la granja, indicando su clasificación entre: Gerente, Administrativo y Obrero.

Secuencia de ingreso:

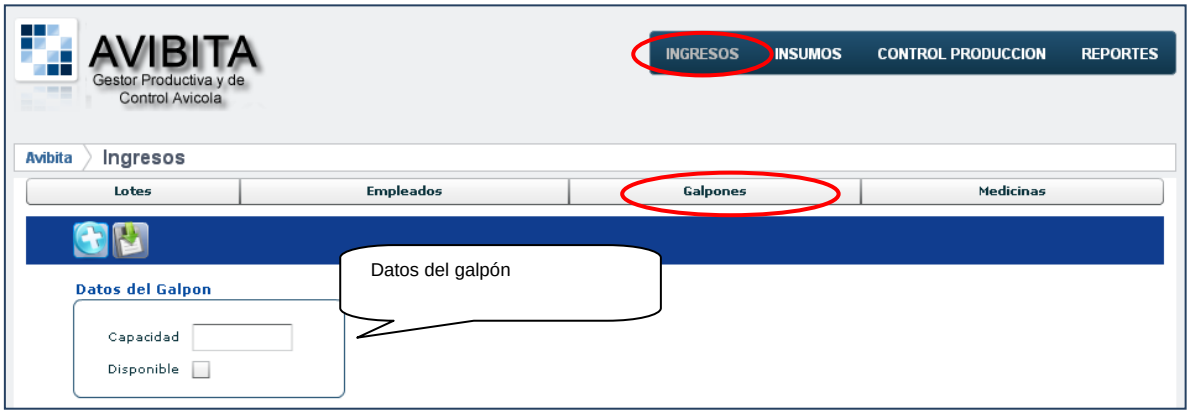
1. Se debe dar clic en el botón Nuevo , para activar las casillas de texto en blanco.
2. Ingresar la información solicitada del empleado.
3. Los campos Nombre y Apellido, tienen un máximo de 50 caracteres.
4. El teléfono debe incluir el código de provincia.
5. Confirmar la información dando clic en el botón Guardar .



Ingreso – Galpón: Este formulario sirve para registrar la información física de los galpones con los que cuenta la granja.

Secuencia de ingreso:

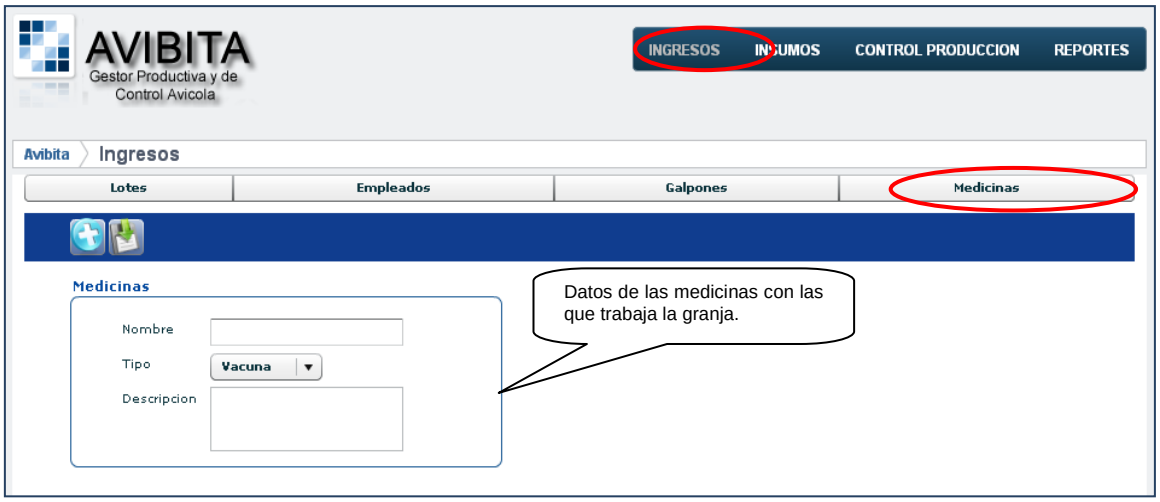
1. Se debe dar clic en el botón Nuevo  , para activar las casillas de texto en blanco.
2. La capacidad debe ser ingresada solo con valores enteros.
3. El campo Disponible debe ser obligatoriamente marcado.
4. Confirmar la información dando clic en el botón Guardar  .



Ingreso – Medicinas: Este formulario sirve para registrar la información de las medicinas con las que se trabaja en el control de producción.

Secuencia de ingreso:

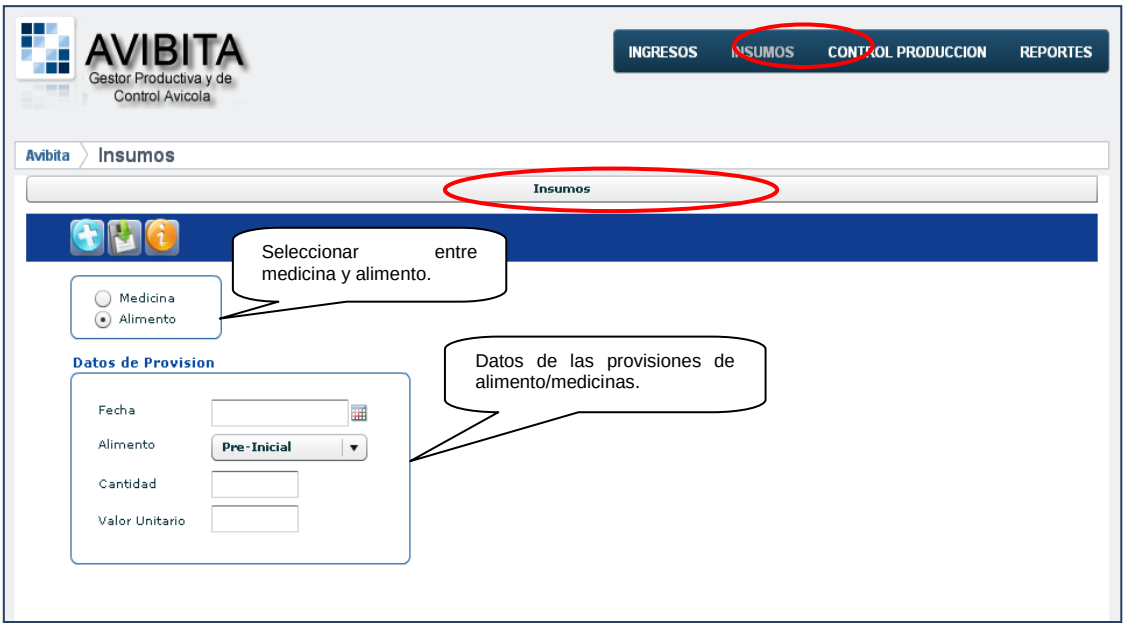
1. Se debe dar clic en el botón Nuevo , para activar las casillas de texto en blanco.
2. El campo Descripción tiene un máximo de 50 caracteres.
3. Confirmar la información dando clic en el botón Guardar .



Insumos – Medicinas/Alimento: Este formulario sirve para registrar las provisiones, tanto de medicinas como de alimento que se compran.

Secuencia de ingreso:

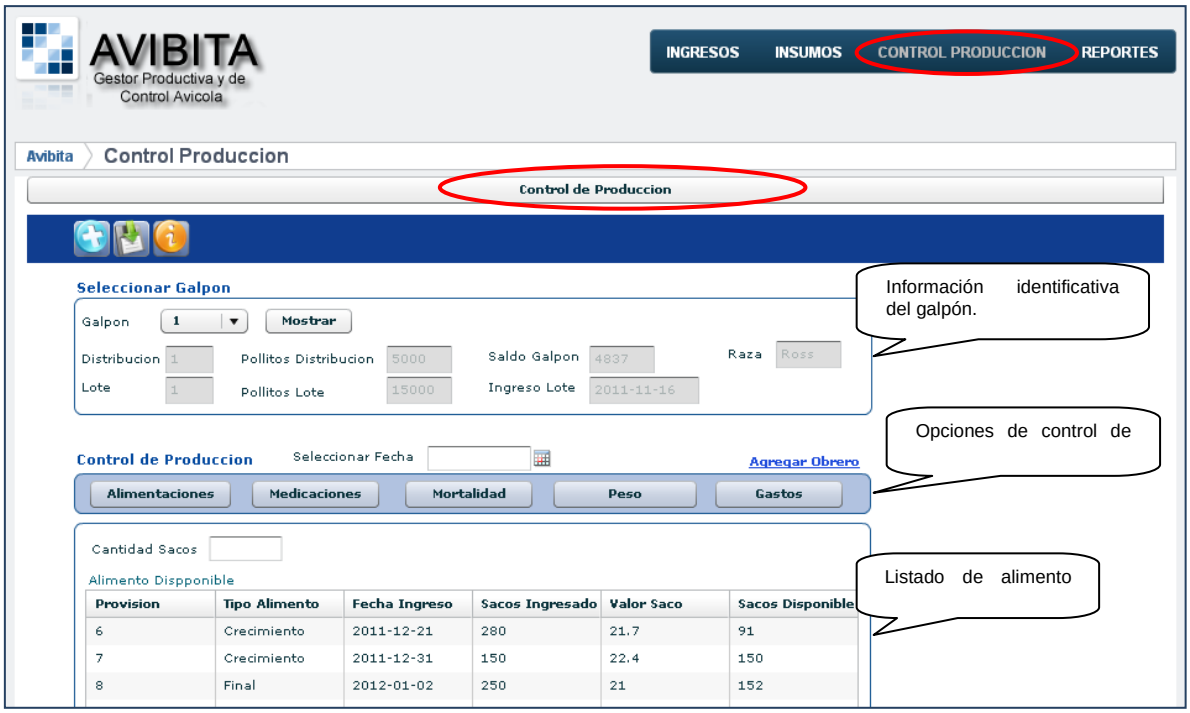
1. Se debe dar clic en el botón Nuevo , para activar las casillas de texto en blanco.
2. Seleccionar entre Medicina o Alimento.
3. Llenar la información solicitada.
4. Confirmar la información dando clic en el botón Guardar .



Control de Producción: Este formulario permite registrar la información generada durante todo el proceso de producción, por cada galpón.

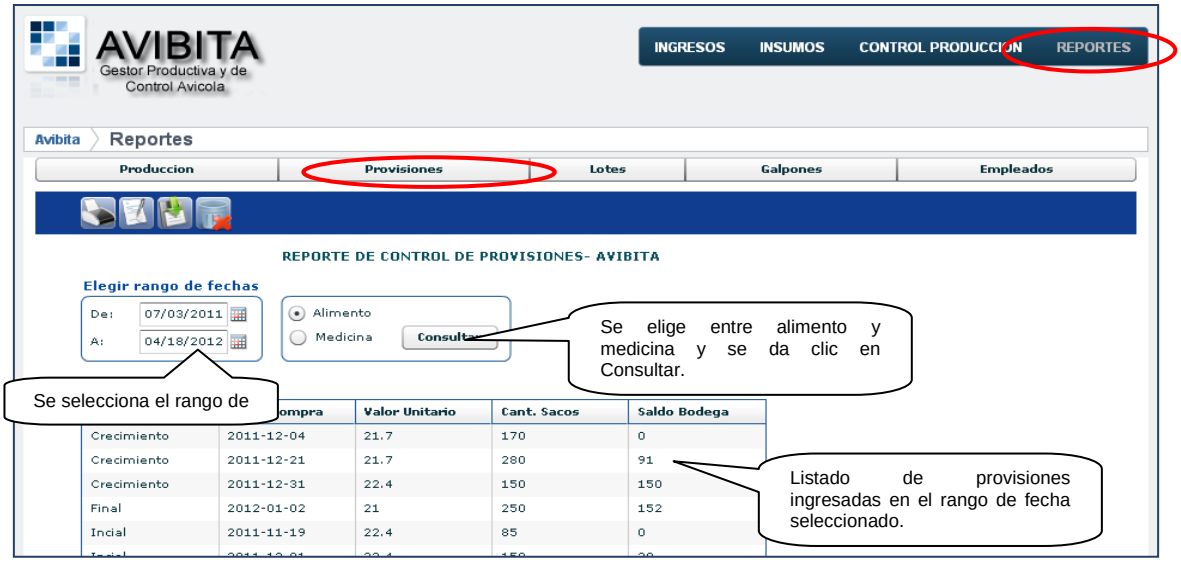
Secuencia de ingreso:

1. Se debe dar clic en el botón Nuevo  , para activar las casillas de texto en blanco.
2. Seleccionar el galpón, dar clic en el botón Mostar para que se despliegue la información de resumen.
3. Seleccionar la fecha.
4. Seleccionar el parámetro de producción a registrar: Alimentaciones, Medicaciones, Mortalidad, Peso, Gastos.
5. Registrar la información solicitada.
6. Confirmar la información dando clic en el botón Guardar  .

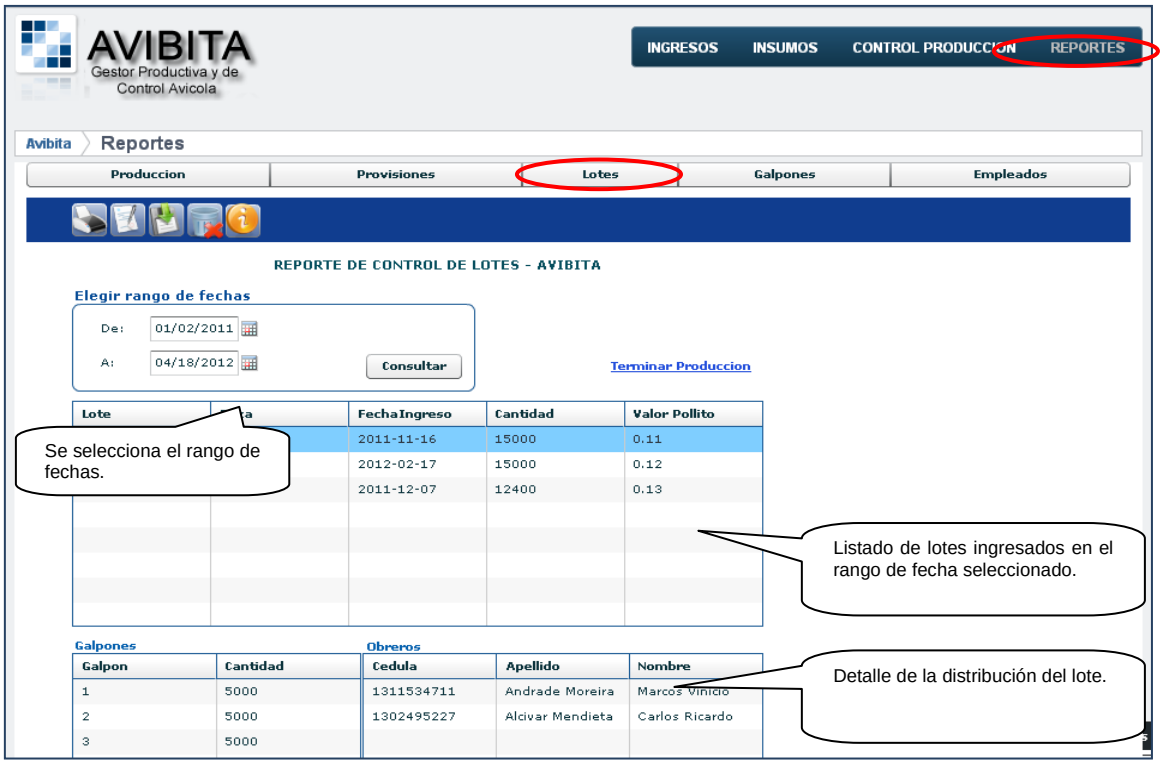


Reporte - Producción: Este reporte permite revisar la información de control de producción, generada desde el ingreso del lote a la granja. Además muestra un resumen de la información descriptiva del estado del galpón.

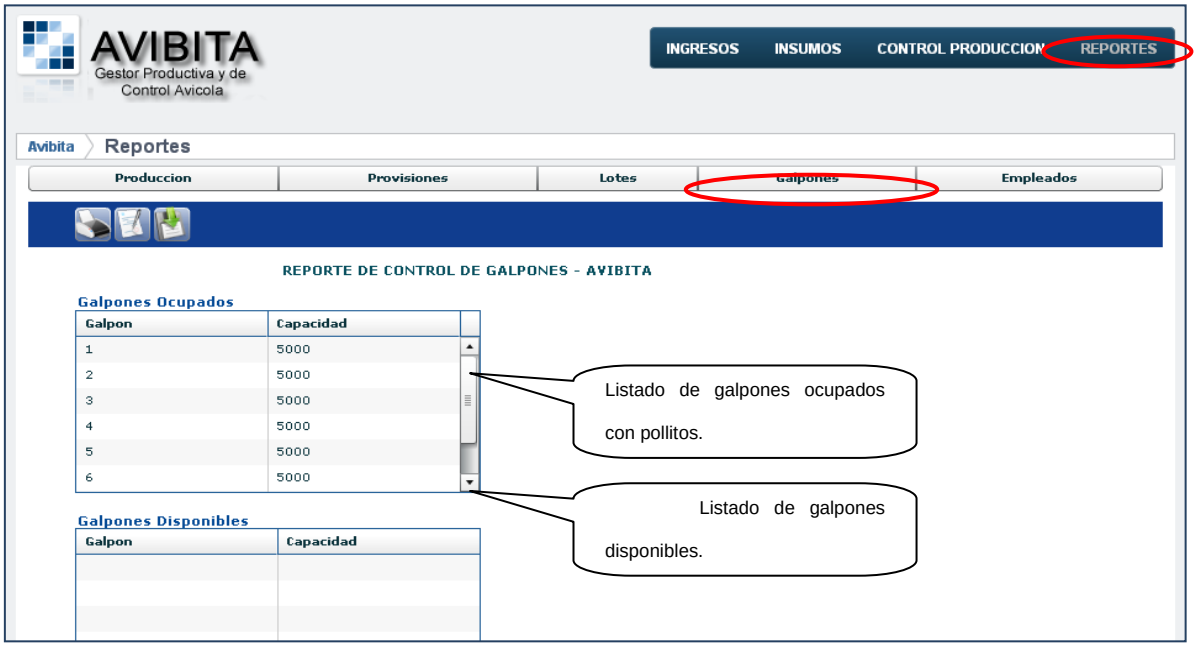
Reporte – Provisiones: Este reporte permite verificar la información de las provisiones de alimento y medicinas, ingresadas a la granja en un rango de fechas determinado.



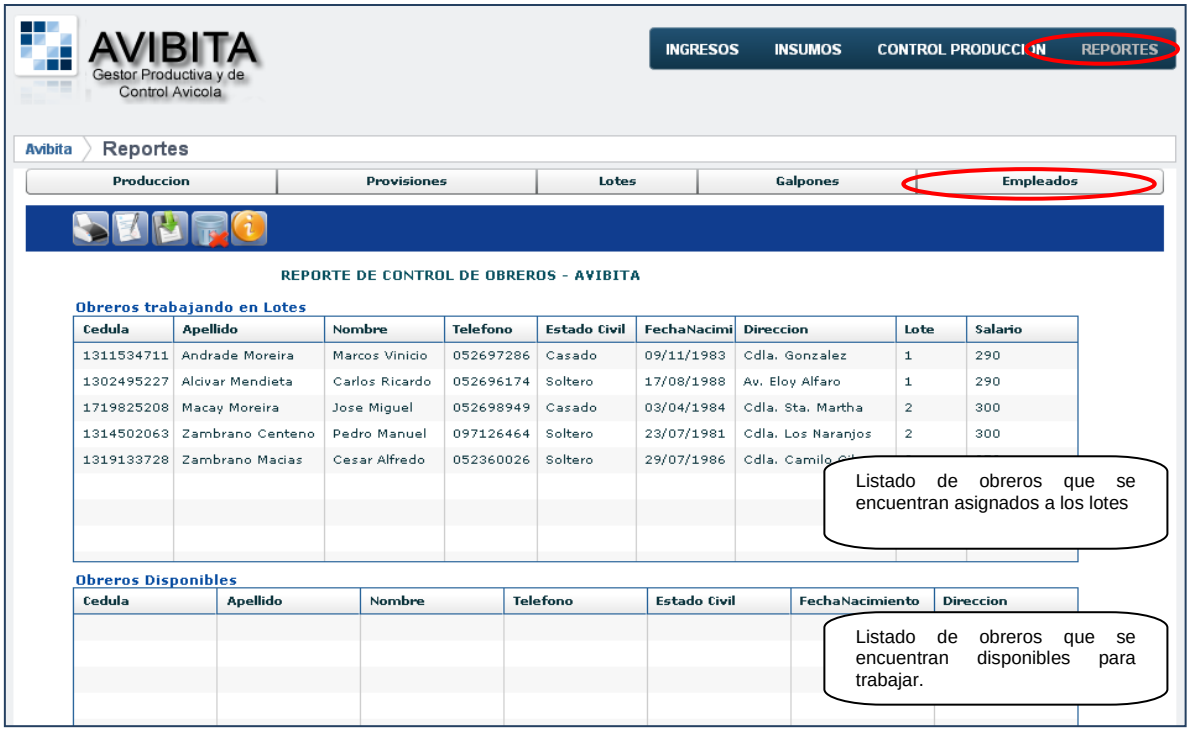
Reporte de Control de Lotes: Este reporte permite verificar los lotes que se han ingresado a la granja en un rango de fechas determinado.



Reporte – Galpones: Este reporte permite revisar los galpones que se encuentran vacios y aquellos que se encuentran en producción.



Reporte – Empleados: Este reporte permite revisar quienes son los obreros que se encuentran trabajando en algún lote y quienes están disponibles.



BIOGRAFÍA

MAYRA ALEJANDRA SAMANIEGO PALLAROSO

Lugar y fecha de nacimiento: Chone, 9 de febrero de 1988.

FORMACIÓN ACADÉMICA

Primaria: Escuela Trajano Centeno Rivadeneira, Chone.

Secundaria: Instituto Técnico Superior de Informática Chone, Chone.

Superior: Carrera de Ingeniería de Sistemas e Informática, Escuela Politécnica del Ejército, Sangolquí.

Títulos obtenidos: Bachiller en Comercio y Administración Especialidad Informática, 2005.

Honores obtenidos: Abanderada en primaria (1998) y secundaria (2004). Primer lugar en concurso Provincial de Oratoria, Manabí, 1998. Primer lugar, concurso de Poesía Romántica, Chone, 1999. Segundo Lugar, concurso de Poesía Protesta, Chone, 2002. Tercer Lugar, concurso de Escritura de Poesía Narrativa, Chone, año 2003. Primer Lugar, concurso de Oratoria – Día de la Raza, Chone, 2004.

Experiencia Laboral:

Proyecto Biblioteca Virtual ESPE – Año 2007.

Centro de Investigaciones Científicas y Tecnológicas del Ejército – Año 2007.

Caja Central Cooperativa Financoop – Año 2009 / 2010.

Distribuidora Importadora DIPOR – Año 2011.

HOJA DE LEGALIZACIÓN DE FIRMAS

ELABORADA POR

Mayra Alejandra Samaniego Pallaroso

DIRECTOR DE LA CARRERA

Ing. Mauricio Campaña
Sangolquí, 04 de mayo de 2012