Facultad de Ingeniería y Ciencias Agropecuarias

Desarrollo de sistema para el control de hospedaje de los beneficiarios en la fundación FUDIS.

Trabajo de Titulación presentado en conformidad a los requisitos
Para obtener el título de Ingeniero en Sistemas de Computación e Informática

Profesor Guía

Ing. Guillermo Ávila

Autor

John Vicente Obando Rojas

Año

2011

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando sus conocimientos para un adecuado desarrollo del tema escogido, y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación.”

_____

Guillermo Ávila

Ingeniero en Sistemas e Informática

1706769237

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes”

_____

John Vicente Obando Rojas

1104237670

AGRADECIMIENTO

A mis padres y hermanos por el apoyo, amor y confianza que me han brindado incondicionalmente durante toda mi vida.

A mis profesores, compañeros y amigos por su enriquecedor aporte tanto moral como académicamente a lo largo de mi carrera universitaria.

DEDICATORIA

Para mis hermanos Katy, Briggette y Alex.

RESUMEN

El siguiente proyecto tiene como objetivo principal la realización de un software para el control de hospedaje y generación de reportes estadísticos de los beneficiarios en la fundación FUDIS, institución sin fines de lucro que brinda alojamiento y alimentación a personas enfermas principalmente de cáncer. Para lo cual se ha investigado y usado la metodología de desarrollo *Scrum* y el *framework JBoss Seam* junto con su herramienta *Seamgen* y la librería de componentes gráficos *RichFaces*. Obteniendo satisfactoriamente en un plazo acordado de 5 meses, el producto funcional conforme a los requerimientos planteados por el cliente; observando además la agilidad y adaptabilidad a cambios que ofrece la metodología usada, así como la velocidad de desarrollo al utilizar la herramienta *Seamgen* que proporciona el *framework JBoss Seam*.

ABSTRACT

The main objective of the following project is the software development of an application to control the lodging and the creation of statistical reports of the beneficiaries of the FUDIS foundation, a non-profit organization that provides shelter and food for sick people, mainly the ones suffering from cancer. For this purpose it has been used the Scrum development methodology and the JBoss Seam framework with Seamgen application and the Rich Faces graphics components library. The functional project was successfully delivered in an agreed period of 5 months according to the requirements set by the client; and also taking into consideration the agility and adaptability to changes that offers the used methodology, as well as the development speed by using the Seamgen tool provided by the JBoss Seam framework.

ÍNDICE

Capítulo I: Introducción

1.1 Situación actual

El siguiente trabajo se desarrolla para la fundación FUDIS, entidad sin fines de lucro que contribuye principalmente al desarrollo integral y solidario de quienes padecen condiciones difíciles en términos de discapacidad y/o diagnóstico de cáncer, su trabajo va orientado a la población de escasos recursos proveniente de las distintas provincias del país, brindándoles servicio de hospedaje y alimentación a muy bajos costos.

En la actualidad la fundación FUDIS no tiene un sistema especializado y construido a la medida de sus necesidades. Alternativas como la compra de un software que facilite el registro de hospedaje y demás datos de los beneficiarios y sus acompañantes no está dentro de sus prioridades y como solución ante este escenario, han decidido llevar los datos en hojas de cálculo y formularios escritos a mano, lo que dificulta la obtención de reportes o estadísticas de dichos datos. En ocasiones la fundación debe facilitar el historial de cada una de las personas que ingresan al albergue para distintos proyectos y la falta de una base de datos impide otorgar información objetiva para quienes la requieren.

Hipótesis**:**

¿Es factible desarrollar un software que permita llevar de manera eficiente el control del hospedaje así como también generar reportes gráficos acerca de las personas que se encuentran hospedadas o que se han hospedado a lo largo de algún período determinado?

Además para colaborar con la comunidad estudiantil aportando mayores fuentes de información se ha decidido utilizar tecnologías relativamente nuevas y poco exploradas y de las cuales se tiene poca información en la biblioteca de nuestra universidad; se ha determinado utilizar la metodología de desarrollo *SCRUM*, el *framework* de *Java JBoss SEAM*, utilizando además componentes *RichFaces* con la finalidad de aportar una interfaz rica orientada a web.

En el proceso de desarrollo se utilizará herramientas de software libre que se especifica a continuación:

Sistema Operativo: *Ubuntu Linux*

Servidor de aplicaciones: *JBoss AS 5.1*

IDE de desarrollo para lenguaje *Java*: *Eclipse*

Base de datos: *Mysql*

Diseño gráfico: *Gimp* (*GNU Image Manipulation Program*)

1.2 Alcance

En primer lugar se obtendrán los requerimientos del sistema para poder realizar el modelo de dominio del mismo.

El sistema se manejará en una intranet y será una aplicación vía web, que permitirá manejar usuarios para controlar la seguridad e integridad de la información.

Se detalla a continuación los módulos que brindará la aplicación.

- Administración (creación, modificación y eliminación) de pacientes, acompañantes, habitaciones.
- Registro de las fichas de ingreso de cada paciente y acompañantes.
- Registro de salida del paciente y acompañantes.
- Generación de informes de: Camas y cuartos ocupados; historial de pacientes según el tipo de cáncer, género, edad y ciudad de residencia; estadísticas históricas de un número de pacientes acogidos en un período determinado; además se presentaran informe diario de personas hospedadas.

También se migrarán los datos ya existentes desde las hojas de cálculo que se encuentra usando la fundación hacia la nueva base de datos.

El sistema no contempla la facturación, contabilidad y administración de las donaciones que recibe la fundación.

1.3 Justificación

El desarrollo del sistema para manejo de pacientes en la fundación FUDIS agilitará el registro de ingreso y salida de los beneficiarios y sus acompañantes, permitirá conocer con exactitud la realidad de la fundación para la toma de decisiones, ayudará a generar información a la hora de realizar peticiones de donaciones para seguir con el proyecto de ayuda a la comunidad.

Además permitirá generar reportes que ayudaran a investigaciones acerca del cáncer en proyectos de tesis que se realizan en la fundación.

Un punto muy importante para la fundación es la parte económica ya que no existe presupuesto para la compra de un sistema como el que se va a desarrollar, es por eso de gran ayuda obtener este sistema sin costo alguno y como ayuda por parte de la Universidad de las Américas.

Por otra parte el desarrollo en nuevas tecnologías utilizando metodología novedosa servirá como un aporte para la biblioteca de nuestra universidad y permitirá a los estudiantes obtener una guía sobre los temas menos explorados.

1.4 Objetivos

1.4.1 General

Desarrollar un sistema para el control de hospedaje de los beneficiarios en la fundación FUDIS

1.4.2 Específicos

- Realizar el análisis y diseño del sistema
- Desarrollar la aplicación utilizando la metodología *Scrum*
- Aplicar las pruebas del sistema
- Instalar el sistema en la fundación FUDIS

Capítulo II: Marco Teórico

2.1 Metodologías de desarrollo de Software

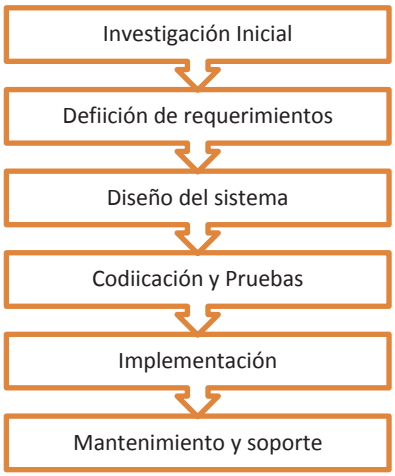
"Metodología de desarrollo de software es un marco de trabajo que permite y facilita la planificación y el desarrollo de un sistema de información proponiendo los métodos, las herramientas, los modelos y el soporte documental que determinan un camino en la elaboración de un software de calidad, flexible, *mantenible*, escalable y que satisfaga las necesidades del cliente."[1]

Existen varias metodologías que a lo largo de los años se han propuesto, de acuerdo a su enfoque se los puede resumir en las siguientes categorías.

2.1.1 Modelo en cascada.

Es una secuencia de pasos dispuestos como una cascada de agua, estas fases están superpuestas entre ellas y utilizan una documentación muy extensa que ayuda a mantener el control del proyecto así como también son determinantes los comentarios y la aprobación del cliente a la hora de iniciar una nueva fase, este modelo prioriza la planificación, cronogramas y presupuesto. Las fases que componen este modelo son: la investigación inicial, definición de requerimientos, diseño del sistema, codificación y pruebas, implementación y mantenimiento y soporte.

Gráfico 2.1 Modelo en cascada



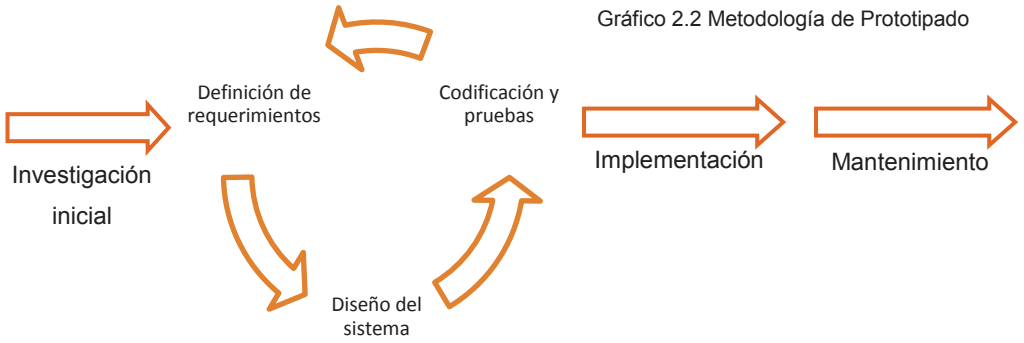
Fuente: SELECTING A DEVELOPMENT APPROACH

Es muy eficaz con proyectos rígidos ya que el modelo mantiene mucha organización en el proceso además de especificar claramente las herramientas y los requerimientos del proyecto.

[1] CENTERS for MEDICARE & MEDICAID SERVICES, Selecting a Development Approach, http://www.cms.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf,

2.1.2 Prototipado

Gráfico 2.2 Metodología de Prototipado



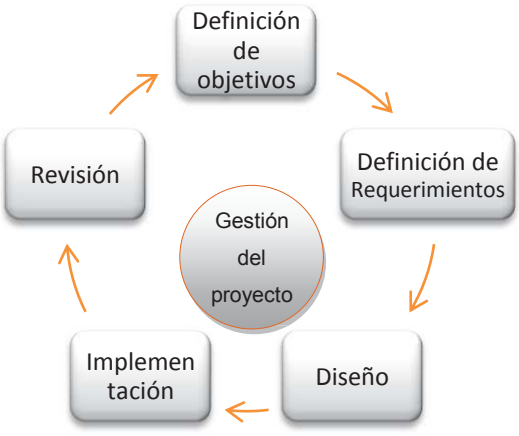
Fuente: SELECTING A DEVELOPMENT APPROACH

Es un modelo iterativo que tiene como finalidad principal el de ir creando prototipos del software hasta llegar a uno que se convierta en el producto final debido a que ha evolucionado de tal forma que los requerimientos del cliente han sido satisfechos; este modelo permite una continua interacción con el cliente, por lo tanto es más fácil el cambio de requerimientos en cualquiera de las etapas del desarrollo.

Este estilo de metodología brinda mayor utilidad si el cliente sabe los objetivos generales pero no puede identificar los requisitos detallados del software.

2.1.3 Incremental

Gráfico 2.3 Metodología Incremental



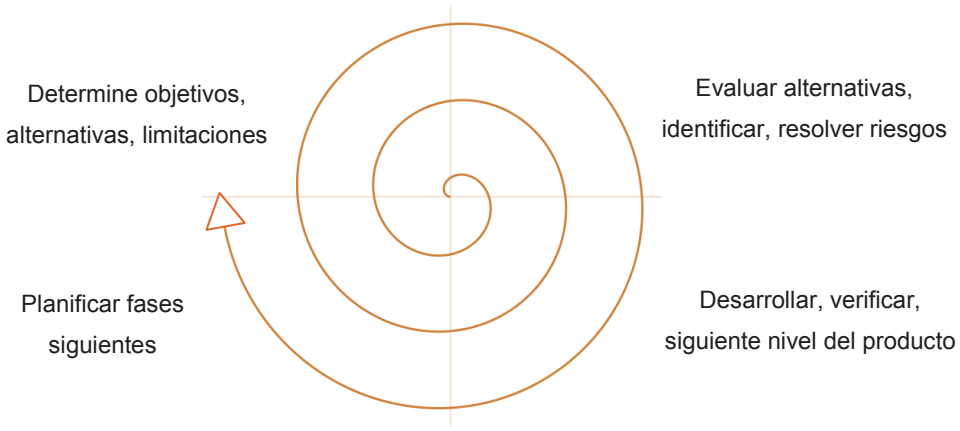
Fuente: http://mariaavilarobles.blogspot.com

Es una unión entre la metodología en cascada y la de prototipos Principalmente se trata de crear versiones que a medida de cada iteración van mejorando; tiene como objetivo reducir los riesgos que se producen al dividir el proyecto en segmentos, y facilitar los cambios durante el desarrollo. Los siguientes métodos de utilización de esta metodología son los más comunes:

- Antes de proceder al incremento siguiente se efectúa una serie de mini-Cascadas que son ejecutadas para cada segmento pequeño del sistema.
- Se determinan todos los requisitos antes de comenzar y se desarrolla la mini cascada para cada uno de los incrementos de forma individual.
- En primer lugar se utiliza el método de cascada para definir el concepto inicial del software, para determinar la arquitectura y el núcleo del sistema, y luego se continua por el iterativo de los prototipos hasta que para concluir se realiza la instalación del prototipo definitivo.

2.1.4 Espiral

Gráfico 2.4 Metodología de Espiral



Fuente: SELECTING A DEVELOPMENT APPROACH

Genera un enfoque en el control y reducción de los riesgos, aplica una división del proyecto en fragmentos para apoyar a los manejos de cambios durante el desarrollo. Cada uno de los ciclos debe atravesar Por cada parte del producto y por cada uno de los niveles de elaboración el ciclo atraviesa por la misma secuencia de pasos: determinar objetivos, alternativas, y desencadenantes de la iteración; Evaluar alternativas; identificando y resolviendo los riesgos; desarrollar y verificar los resultados de la iteración, y planear la próxima iteración. Este método es recomendable cuando se trabaja en proyectos que den mucha importancia a los riesgos y los que se necesite una alta seguridad.

2.1.5 Rapid Application Development (RAD)

Este modelo está diseñado para brindar velocidad y gran calidad al proceso de desarrollo de un procurando disminuir los costes de inversión. Brinda mayor facilidad del cambio durante el proceso del desarrollo y reduce los riesgos mediante la segmentación del proyecto en iteraciones por prototipos. Fundamentalmente se enfoca en el cumplimiento de las necesidades comerciales, dándole una menor importancia a la excelencia tecnológica o ingenieril. Cuando un proyecto empieza a retrasarse es preferible la reducción de requisitos versus un retraso en la entrega del proyecto. Los usuarios tienen una participación imprescindible. No se genera prototipo sino que más bien se genera software de producción de manera iterativa con la documentación que facilita el desarrollo y mantenimiento.

2.2 Metodología *Scrum* para el desarrollo de software

"*Scrum* es una metodología ágil, iterativa e incremental que permite la gestión y el desarrollo de software, se basa en una filosofía del desarrollo ágil"[2] planteada en el año de 1986 por los japoneses Hirotaka Takeuchi e Ikujiro Nonaka. *Scrum* proviene del vocabulario del rugby y se traduce como melé, que es una formación donde todos los compañeros se unen para empujar juntos hacia una sola dirección y conseguir adueñarse del balón, lo relevante es que si uno de ellos cae toda la formación se desmorona; es así como la metodología *Scrum* trabaja en equipos de desarrollo pequeños, con comunicación abierta entre los miembros, dando una mayor importancia a los individuos que a los procesos o herramientas, prefiriendo la entrega de soluciones antes que reportes de seguimiento y dando repuesta a los cambios antes que ceñirse a un plan.

2.2.1 Herramientas y Prácticas

2.2.1.1 Historias de Usuario

Estas son utilizadas en las metodologías ágiles para simbolizar los requerimientos funcionales de un software, al igual que lo hacen los casos de

[2] Basado en procesos ágiles, enfocándose en la gente y en los resultados y promueve iteraciones a lo largo del desarrollo.

uso con las metodologías convencionales. Conjuntamente con las pruebas de validación y las discusiones con el usuario permiten especificar los requerimientos de un sistema.

Las historias de usuario se pueden obtener mediante entrevistas con el usuario, reuniones, lluvias de ideas, etc. Están escritas en lenguaje común entendible para el usuario y expresadas en pocas líneas ya que no es necesario redactar en detalle las peticiones del cliente sino más bien llegar a discutirlas; usualmente están escritas en tarjetas de papel y colocadas sobre un pizarrón.

Algunas de las ventajas de usar historias de usuario ante el uso de casos de uso son:

- Son comprensibles para los usuarios y desarrolladores.
- Luego facilita la planeación ya que tienen el tamaño justo.
- Da más relevancia a la comunicación verbal que a la escrita.
- Diseñadas para el desarrollo iterativo.

2.2.2 La pila del Producto (*Product Backlog List*)

Es una lista de las funcionalidades que se espera tenga el sistema según los usuarios, clientes y *stakeholders*, estos deben estar priorizados y serán tomadas para desarrollarlos en las iteraciones, esta lista puede ser modificada al terminar cada iteración. El equipo dividirá cada requisito en tareas que no tomen más de 16 horas.

Tabla 2.1: Ejemplo Pila del producto

ID	PRIORIDAD	DESCRIPCIÓN	ESTIMACIÓN VALOR	ESTIMACIÓN ESFUERZO

Fuente: Autor

2.2.2.1 Sprints

Se denomina *sprint* a cada una de las iteraciones que se dan en el ciclo del desarrollo de un proyecto con *Scrum*, estos tienen una duración entre 15 y 30 días y para cada uno de ellos es necesario realizar las etapas de análisis, diseño, implementación y pruebas. El propósito del *sprint* es el de ofrecer un

producto funcional para el cliente, obteniendo además sus sugerencias que aporten a cambios dentro del mismo.

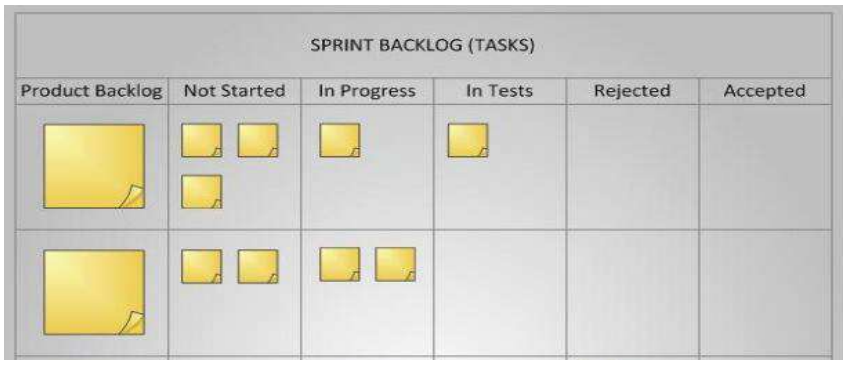
2.2.2.2 Burn down Chart

Es una gráfica que muestra si existen tareas pendientes del proyecto al comienzo de cada *Sprint* y demuestra el avance en función del tiempo para así poder manejar el proyecto, si se une los puntos de los gráficos normalmente se debería tener una línea descendente si es que todo va bien. Este modelo se lo usa en lugar de un diagrama de *PERT*[3] ya que en el desarrollo ágil el camino crítico cambia a diario y este diagrama se realizaría en vano.

2.2.2.3 Pila del Sprint *(Sprint Backlog)*

Es una lista detallada que desglosa en tareas individuales a los ítems de la *Product Backlog List,* para que el equipo o *Scrum Team* implemente durante el siguiente *Sprint*, se presentan como tareas que no superen las 16 horas. El equipo organiza la forma de tomar estas tareas y cumplirlas con el fin de alcanzar el objetivo final, así mismo pueden agregar o eliminar tareas que crean convenientes. Las tareas se agrupan de acuerdo al ítem del *Product Backlog,* cada tarea tiene su fecha de inicio, tiempo de trabajo que queda para completarla, responsable y estado (terminada, pendiente o en curso). Esta pila se la puede llevar en hojas de cálculo, pizarras físicas o en herramientas colaborativas o de gestión de proyectos.

Gráfico 2.5: Ejemplo de pila de Sprint



Fuente: http://blog.valkir.net/2011/02/24/core-ingredients-of-scrum/

[3] Representación gráfica de las relaciones entre las tareas del proyecto que permite calcular los tiempos del proyecto de forma sencilla.

2.2.2.4 Roles

Los roles se dividen en dos grupos los cerdos y las gallinas; estos nombres provienen de un chiste en el que una gallina y un cerdo acuerdan abrir un restaurante, la gallina propone llamarle “huevos con jamón”; pero el cerdo se niega porque él estaría realmente comprometido mientras la gallina solo estaría involucrada. De este modo los participantes se dividen en estos dos grupos según su participación en el proyecto.

- **Roles Cerdo:** llevan este nombre ya que ponen su “jamón en el plato”. Es decir su compromiso con el proyecto es total. A continuación quienes conforman este grupo:

  Facilitador (*Scrum Master)*

  Es el encargado de que el restante del grupo se una a las prácticas y reglas de *Scrum*, y es importante hacer hincapié en que no maneja al equipo ya que *Scrum* utiliza autogestión, el *Scrum Master* se encarga de eliminar los impedimentos, reducir los riesgos del producto así como ayuda a que el equipo se adapte a *Scrum*. Generalmente este rol lo ocupa un Gerente de proyecto o líder de equipo.

  Dueño del producto *(Product Owner)*

  El *product owner* representa la voz del cliente y es el responsable del proyecto, aportando la visión del negocio, este es el encargado de redactar las historias de usuario [4] y priorizarlas para colocarlas en la pila del producto o *Product Backlog List.*

  Debe ser una sola persona, no una comisión aunque pueden existir comisiones que lo asesoren e influencien Generalmente esta persona puede ser el *Product Manager*, *Marketing*, *Internal Customer*, etc.

  El equipo (*Scrum Team*)

  Es el equipo del proyecto que tiene como objetivo entregar el producto, estos tienen la autoridad para decidir su organización para

[4] Representaciones escritas brevemente de los requerimientos de un software usando el lenguaje común del usuario

llegar a cumplir los objetivos de las iteraciones, es decir que se auto-organizan. Normalmente es un grupo de entre 5 y 10 personas de distintas disciplinas como desarrolladores, diseñadores, etc. La composición del equipo puede ser cambiada al final de cada iteración.

- **Roles Gallina**

  Aunque *Scrum* no los hace parte del proceso, es importante el apoyo de los usuarios, expertos del negocio y a otros involucrados o *stakeholders.* Estos roles solo aportan al proyecto como la gallina colabora poniendo huevos, aunque no esté comprometida. Lo conforman las siguientes personas.

  Usuarios

  Son los que van a usar el producto final y a quienes va dirigido como último destino.

  ***Stakeholders* (Clientes, Proveedores, Inversores)**

  Son todas las personas interesadas en el proyecto y que intervienen en la realización del mismo, estos participan directamente durante las revisiones de las iteraciones.

  Managers

  Son las personas que definen el entorno y las características necesarias para el desarrollo del producto. Ayudan a los procesos que definen al *Scrum Master*, los objetivos, y los requerimientos. Y ayuda al *Scrum Master* con la reducción de la *Product Backlog List*.

2.2.2.5 Reuniones

En *Scrum* se tiene cuatro reuniones principales que se realizan a lo largo del desarrollo del proyecto.

- **Reunión de Planificación del *Sprint* (*Sprint Planning Meeting*)**

  Al iniciar el *Sprint* el *Product Owner* realiza una reunión conjuntamente con el *Scrum Team* alrededor de 4 horas para especificar cuáles de los ítems enunciados en el *Product Backlog* se van a realizar durante el próximo *Sprint*, se va tomando una a una las tareas del *Product Backlog* para preguntar a los

programadores hasta donde se podrá desarrollar. Una vez que se ha tomado la decisión de las tareas a realizar, estas pasan al *Sprint Backlog* para el inicio del desarrollo.

- **Reunión Diaria (*Daily Scrum*)**

  Como su nombre lo indica se refiere a una reunión realizada diariamente, con 15 minutos de duración cuyo horario habitual es en la mañana y preferiblemente debe realizarse de pie para obtener toda la atención del grupo, pueden asistir cualquier de los roles pero solo pueden participar los pertenecientes al grupo de los “cerdos”; además el *Scrum Master* se encarga de cuestionar a cada uno de los integrantes del equipo:

  ¿Qué hiciste el día de ayer?

  ¿Qué tienes planeado para realizar el día de hoy?

  ¿Tuviste alguna dificultad, en que necesitas ayuda?

Cualquier duda será comentada y se buscará solución, si es necesario se debe aumentar el tiempo para una tarea por problemas no previstos, simplemente se lo apunta y se prosigue con el trabajo.

- **Reunión de revisión del *sprint* (*Sprint Review Meeting*)**

  Esta reunión se realiza al finalizar cada *Sprint*, en esta participan el *Scrum Team*, el *Scrum Master*, el *Product Owner* y cualquier interesado en el producto (gallinas). Es aquí cuando el *Scrum Master* muestra la versión al resto, y se permite la participación a los presentes para hablar acerca de mejoras o cambios en el sistema. No debe durar más de cuatro horas, y se recomienda no dedicar más de 30 minutos en preparar la demostración, ya que la final de dicha reunión es una conversación con el cliente para analizar los avances del proyecto.

- **Retrospectiva del *Sprint* (*Sprint Retrospective*)**

  Esta debe durar aproximadamente 4 horas y la realizan *Scrum Master*, el *Scrum Team* y el *Product Owner* para analizar las falencias durante el último *Sprint* y aplicar mejoras en el desarrollo de los siguientes, por ejemplo fallas en la comunicación y colaboración. También se analiza lo que se hizo bien

para seguir con estas prácticas. Así se puede lograr una mejora progresiva y constante en el proceso de desarrollo.

2.2.2.6 Resumen del proceso

El desarrollo de un proyecto utilizando Scrum se determina en tres reuniones:

Planificación del *sprint*.

- Se toma y analizan cada uno de los ítems que se presentan en la *product Backlog list* y que van a desarrollarse en ese *sprint.*
- Cada uno de los ítems se subdivide en tareas específicas, para generar el *sprint Backlog.*
- Se estima el esfuerzo inicial de cada tarea y se asignan voluntarios para cada una.

Seguimiento del *sprint*.

Diariamente debe ser evaluada la evolución del *sprint* por todo el equipo.

Todos los miembros del equipo muestran la tarea en la que se encuentran trabajando en la que va a comenzar a trabajar y pide ayuda si necesita algo para su realización.

Luego cada uno debe actualizar su avance y estima la cantidad de trabajo que necesita para culminar con sus tareas actuales.

Revisión del *Sprint*.

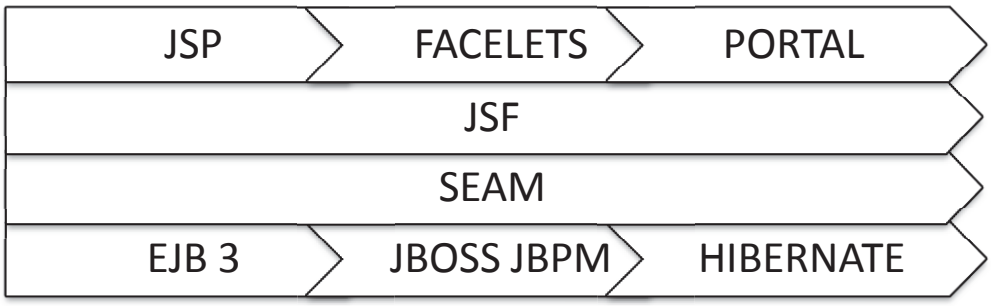
Se realiza con la finalidad de exponer el progreso del proyecto a todas las personas involucradas. Tiene una duración máxima de 4 horas. Y no se debe caer en confusión ya que estas tienen su enfoque en el producto a diferencia de las reuniones de retrospectiva que intentan mejorar las prácticas ágiles.

2.3 *Framework JBoss Seam*

JBoss Seam es una plataforma de integración de tecnologías para facilitar el desarrollo de aplicaciones *JEE*, desarrollado por la familia de *JBoss* una división de *RedHat*, el cual constituye una integración de *EJB* y *JSF*; la principal característica es que ayuda por medio de las denominadas anotaciones a que los elementos de *EJB3* en la capa de negocio respalden directamente a las

formas de *JSF* ahorrando código al momento de generar aplicaciones, tanto en *Java* como *XML*.

Grafico 2.6 Esquema Jboss Seam



Fuente: capítulo 1 de la documentación JBoss Seam

2.3.1 Seamgen

Seamgen es una aplicación sencilla en línea de comandos proporcionada en el *framework* para la generación de proyectos con sus esqueletos incluyendo configuración y librerías; además ayuda con la creación de aplicaciones *CRUD* (*Create Read Update Delete*) a partir de entidades *JPA/EJB3* o mediante ingeniería inversa, partiendo de una base de datos relacional.

Algunos de los comandos más comunes para realizar un *CRUD Application* son los siguientes.

seam setup Este tiene como fin la configuración del medio de desarrollo: directorio del servidor, información acerca de la base de datos, *workspace*, etc.

seam new-project Crear un nuevo proyecto que luego puede ser abierto en el *IDE Eclipse*.

***seam generate-enti*ties** Crea una aplicación a partir de una base de datos existente.

seam generate-ui Crea una aplicación a partir de entidades *JPA/EJB3*.

2.3.2 Contextos

Para comprender el funcionamiento de *seam* se necesita conocer dos conceptos Contexto y Componente.

Componentes son los objetos con estado por lo general *EJBs*, y una instancia de un componente se asocia a un contexto asignándole además un nombre en dicho contexto. Cada contexto es una especie de compartimento en donde se almacenan los componentes de seam, cada uno de los contextos define que objetos pueden permanecer en él y durante cuánto tiempo.

Los contextos básicos y ordenados de menor a mayor duración son:

- *Stateless context*
- *Event (ie, request) context*
- *Page context*
- *Conversation context*
- *Session context*
- *Application context*
- *Business process context*
- *Unspecified*

2.3.2.1 Stateless Context

En este contexto no se guardan las variables, todos los componentes que están dentro de este contexto se instancian cada vez que se nombre es resuelto.

2.3.2.2 Event Context

Este es el contexto con estado (*stateful*) más angosto de seam. Los componentes en este contexto mantienen sus estados en todo el proceso de una solicitud *JSF*.

2.3.2.3 Page Context

Los componentes en este contexto están ligados a una página específica y puede tener acceso a los componentes de todos los eventos emitidos por esta página.

2.3.2.4 Conversation Context

Seam define a una conversación como una sucesión de peticiones web para llevar a cabo alguna tarea. Los componentes en estos contextos mantienen su

estado a través de la conversación. Este contexto es uno de los más importantes en seam.

2.3.2.5 Session Context

Los componentes en este contexto son manejados en un objeto de sesión *HTTP*. Estos mantienen sus estados hasta que la sesión caduca, se puede tener múltiples conversaciones en una sesión.

2.3.2.6 Application Context

Es un contexto global que contiene información estática como datos de configuración, datos de referencia o meta modelos y en el que no existe el concepto de usuarios de Internet.

2.3.2.7 Business Process Context

En este contexto permanecen los componentes con estado asociados con un largo proceso de negocio manejado por el motor de *JBoss jBPM* (*Business Process Manager*). El proceso de negocio abarca múltiples interacciones con varios usuarios.

2.3.2.8 UNSPECIFIED

Es una directiva que indica que el contexto debería ser implícito.

2.3.3 Componentes de *Seam*

Los componentes en seam son los *POJOs* (*Plain Old Java Objects*), en especial pueden ser *JavaBeans* o *EJBs* 3.0 *Enterprise Beans*. Seam soporta los siguientes tipos de componentes.

- *EJB* 3.0 *stateless session beans*
- *EJB* 3.0 *stateful session beans*
- *EJB* 3.0 *entity beans* (JPA *entity classes*)
- *JavaBeans*
- *EJB* 3.0 *message-driven beans*
- *Spring beans*

2.3.3.1 Stateless session beans

Como su nombre lo indica son *beans* sin estado, estos siempre viven en el *stateless context* y no poseen variables de instancia en las cuales se guarden los datos que los métodos comparten. Estos *beans* poseen métodos que realizaran tareas determinadas dando resultados independientes del estado de la conversación que mantiene con el cliente. Es requerimiento implementar una interfaz que puede ser *local* o *remote.*

2.3.3.2 *Stateful session beans*

A diferencia de los anteriores estos si tienen estado, es decir que dentro de la sesión del usuario estos beans van a almacenar en variables de instancia datos que tienen significado en la conversación mantenida con entre el cliente y el *bean*; igualmente se necesita implementar una interfaz ya sea *local* o *remote.*

2.3.3.3 Entity beans

Un *entity bean* a diferencia de un *session bean* trabaja con un depósito de información que generalmente es una base de datos, cada unos de los *entity beans* son un *POJO* que representa a una tabla de la base de datos y cada instancia de esta clase es un registro de esta tabla. Todos deben tener una clave primaria que lo identifique, estos *beans* ayudan con el mapeo de las tablas y sus relaciones.

2.3.3.4 Java beans

Pueden ser utilizados como un *session bean* con o sin estado, pero estos no proporcionan las funcionalidad de un *session bean* (seguridad declarativa, la persistencia de *EJB* 3.0, los métodos de tiempo de espera, etc.). Por defecto se encuentran en el *event context.*

2.3.3.5 Message-driven beans

Estos *beans* ayudan a que las aplicaciones procesen mensajes asincrónicamente por medio del servicio *JMS* (*Java Messaging Service).* No pueden almacenar ningún estado ni pertenecer a ningún contexto de seam; estos no son instanciados por la aplicación sino por el contenedor *EJB* cuando un mensaje es recibido.

2.3.3.6 Spring beans

Estos *beans* surgen a partir de la integración con el *Framework Spring* y es un tema que esta fuera del alcance de esta tesis, para mayor información se puede consultar en la documentación de seam *framework*.

2.3.4 Bijection

La *bijection* es una interacción de dos vías entre la *injection* y *Outjection*; su meta es la de realizar una asociación del nombre de un componente seam que se encuentre en un contexto con su propia instancia.

2.3.4.1 Outjection

Coloca la instancia de un componente dentro de un contexto. Por ejemplo permite que un componente A este disponible en un determinado contexto.

2.3.4.2 Injection

Realiza la asociación de una variable de un componente con la instancia de otro componente que este en el contexto. Siguiendo el ejemplo anterior la *injection* permite a un componente B obtener de un contexto una referencia a una instancia del componente A, logrando que el contenedor de aplicaciones "inyecte" al componente A en una variable del componente B.

La *Bijection* tiene algunas ventajas sobre el mecanismo de *Injection.*

Contextual: Es usada para asociar componentes con estado desde diferentes contextos.

Bidireccional: Los componentes pueden ser depositados o tomados del contexto.

Dinámico: Ya que los valores de las variables de los contextos varían con el tiempo y los componentes tienen un estado definido, la *bijection* se realiza cada vez que el componente es invocado.

2.3.5 Anotaciones

Seam provee de anotaciones a sus componentes para indicar ciertas funciones, estados, relaciones; ayudando además a minimizar código *xml*. Estas son pequeñas sentencias ubicadas en lugares estratégicos de los *beans*,

una anotación puede identificarse por el signo @ antes de cada una. Estas son las anotaciones más comunes.

***@Name*(“nombreComponente”)** Nombre del componente seam para una clase, esta notación es necesario para todos los componentes de *seam*.

***@Scope*(*SESSION*)** Contexto o ámbito del componente, estos pueden ser *APPLICATION*, *BUSINESS_PROCESS*, *CONVERSATION*, *EVENT*, *PAGE*, *SESSION*, *STATELESS*, *UNSPECIFIED*. Si este no es especificado el componente tendrá el ámbito por defecto según el tipo de componente.

***@Role*(*name*="nombreDelRol", *scope*=*ScopeType.SESSION*)** Enlazar el componente a múltiples contextos, las anotaciones @*Name*/@*Scope* definen el rol por defecto y cada uno de los @*Role* agregan un rol.

@In Especificar que un atributo de un componente va a ser inyectado desde una variable de contexto al inicio de cada invocación del componente.

@Out Definir que un atributo de componente de *Seam* sea *outjected* a una variable de contexto al final de la invocación.

@Factory Específica que el método del componente es usado para inicializar el valor de la variable de contexto con nombre, cuando la variable no tiene ningún valor.

- Anotaciones en las etiquetas de persistencia

@Entity Define la clase como un bean de sesión *EJB3* para que sea mapeado contra un tabla en la base de datos.

@Table Define el nombre de la tabla en la base de datos

@Id Define el atributo de componente que será tomado como clave principal de la tabla.

@Column sirve para identificar al campo que va a ser persistente, aunque si no se lo coloca por el hecho de estar en una entidad persistente éste persiste.

@Trasient se utiliza para definir un campo que no es persistente

***@ManyToMany*(*mappedBy*="nombre clases relacionar"),** se usa para implementar las relación mucho a muchos.

***@OneToMany*(*mappedBy*="nombre clases relacionar"),** utilizada de las relaciones uno a mucho.

***@OrderBy*("nombre del campo a ordenar").** Ordenar según el campo indicado.

2.3.6 Controles *Seam JSF*

La especificación *JSF* es utilizada en la capa de presentación y nació a partir de lecciones aprendidas en otros frameworks como *Struts*, *Tapestr* y *WebWork*. *Seam* añade algunos controles de *Java Server Faces (JSF)* para complementar los controles integrados y los de terceros como de *RichFaces* y *MyFaces*.

Para poder usar estas etiquetas se necesita definir el *namespace* s al principio de la página de la siguiente forma:

<html xmlns="http://www.w3.org/1999/xhtml"

xmlns:s="http://jboss.com/products/seam/taglib">

Algunos de las etiquetas más usadas son las siguientes.

<s:button> Botón que soporta la invocación a una acción sobre la propagación de la conversación.

<s:conversationId> Agregar el id de conversación a un link de *JSF* o un botón.

<s:link> Un enlace que soporta la invocación de una acción sobre la conversación de propagación.

<s:convertDateTime> Realizar la conversión de fecha o de la hora en la zona horaria de *Seam*.

<s:convertEntity> Asignar una convertidor de entidad al componente actual. Usado en botones y listas desplegables.

<s:convertEnum> Asignar una convertidor de enumeración al componente actual. Usado en *radio buttons* y listas desplegables.

<s:decorate> Decora un campo de entrada cuando la validación falla o cuando el campo es requerido.

<s:div> Crea una etiqueta en la generación *html*.

<s:span> Crea una etiqueta en la generación *html*

<s:selectItems> Crear una lista, desde un modelo de datos o un arreglo.

<s:graphicImage> Permite crear una imagen en un componente de *seam*.

Nota: para mayor información consultar la documentación oficial de jboss seam.

2.3.7 RichFaces

Es una librería de componentes visuales que posee un *framework* para la integración sencilla de las funciones *AJAX*, es soportada a partir del *JDK* 1.5, es un proyecto de código abierto creado en principio por *Exadel* pero luego adquirido por *JBOSS*. Entre las principales características se destacan.

- Se integra completamente al ciclo de vida de *JSF* permitiendo acceder al *action* y al valor del *listener*, así como invocar a validadores y convertidores durante el ciclo de petición-respuesta de *Ajax*.
- Brinda las librerías de *Core Ajax* y la de interfaz de usuario que permiten añadir funcionalidad *ajax* a las páginas (sin escribir código *Java Script*). Además permite manipular eventos en la propia página para sincronizar componentes *JSF* después de recibir la respuesta del servidor.
- Brinda la facilidad de crear recursos binarios en tiempo real. Los recursos del *framework* pueden crear archivos como por ejemplo imágenes, sonidos, hojas de cálculo, etc.
- Dar una interfaz de usuario basadas en tecnologías de *skins*.

  A continuación se enumera algunas de las etiquetas más usadas.

- ***<aj4:support>*** Permite agregarle funcionalidad *Ajax* a cualquier etiqueta *JSF*. Ayuda al componente a generar peticiones asíncronas mediante eventos (*onclick*, *onblur*, *onchange*,...) y actualizar campos de un formulario de forma independiente.

- ***<aj4:commandButton>*** Botón de envío de formulario similar al de *JSF*. Difieren principalmente en que este puede actualizar únicamente los componentes deseados y no todo el formulario.
- ***<aj4:commandLink>*** Comportamiento similar a <*aj4*:*commandButton*> pero en un *link*.
- ***<rich:calendar>*** Utilizado para crear un calendario cuando se necesita ingresar fechas.
- ***<rich:comboBox>*** Permite crear *combo Box* editable.
- ***<rich:datascroller>*** Se utiliza para dotar de una paginación a una tabla utilizando *Ajax*
- ***<rich:dataTable>*** Sirve para crear una tabla.
- ***<rich:columns>*** : Permite crear una columna dinámica.

2.3.8 Autenticación y Seguridad

Seam maneja la autenticación gracias a *JAAS* (*Java Authentication and Authorization Service*) el cual es *framework* de seguridad que permite el control de los servicios de autenticación. Es por esto que brinda un api robusto para los requisitos más complejos. Aún así también provee métodos más sencillos para la administración de la autenticación simplificando la complejidad de *JAAS*. Para realizar una configuración sencilla de autenticación y seguridad con seam hace falta pocos pasos:

- Instalar el componente de identidad.
- Crear la clase para la autenticación.
- Crear el formulario para el *login* de los usuarios.
- Manejo de las excepciones de Seguridad
- Seguridad a las páginas y a partes de las páginas

2.3.8.1 Instalar el componente de identidad.

En el archivo *components.xml* se debe agregar la siguiente instrucción.

```
<security:identity authenticate-method="#{authenticator.authenticate}"/>
```

En donde *authenticator* es el nombre del componente seam que se va a crear y *authenticate* es el método que se utiliza para la autenticación.

2.3.8.2 Crear la clase para la autenticación

Es fundamental que se identifique al componente seam con el nombre que se colocó anteriormente; además se puede utilizar los métodos *credentials.getUsername()* y *credentials.getPassword()* para recibir el nombre y *password* del usuario.

Cuadro 2.1: Ejemplo de clase para la autenticación

```
@Name("authenticator")
public class Authenticator {
  @In EntityManager entityManager;
  @In Credentials credentials;
  @In Identity identity;
  public boolean authenticate() {
    try {
      User user = (User) entityManager.createQuery(
        "from User where username = :username and password" + " = :password")
        .setParameter("username", credentials.getUsername())
        .setParameter("password", credentials.getPassword())
        .getSingleResult();
      if (user.getRoles() != null) {
        for (UserRole mr : user.getRoles())
                identity.addRole(mr.getName());
      }
          return true;
    }
     catch (NoResultException ex) {
          return false;
    }
   }
}
```

Fuente: Capítulo 15 de la Documentación de JbossSeam

2.3.8.3 Crear el formulario para el *login* de los usuarios

El formulario utiliza las variables *credentials.username para almacenar el usuario* y *credentials.password* para la contraseña además se utiliza el *identity.login* para guardar la información introducida en dichas variables.

Cuadro 2.2: Ejemplo de formulario para el login

```
<div>
   <h:outputLabel for="name" value="Username"/>
   <h:inputText id="name" value="#{credentials.username}"/>
</div>
<div>
   <h:outputLabel for="password" value="Password"/>
   <h:inputSecret id="password" value="#{credentials.password}"/>
</div>
<div>
   <h:commandButton value="Login" action="#{identity.login}"/>
</div>
```

Fuente: Capítulo 15 de la Documentación de JbossSeam

2.3.8.4 Manejo de las excepciones de Seguridad

En el archivo *pages.xml* se puede configurar las excepciones *NotLoggedInException* y *AuthorizationException* para evitar que el usuario del sistema vea la página de error que *seam* trae por defecto.

NotLoggedInException se lanza cuando un usuario no se ha *logueado* e intenta acceder a un recurso del sistema.

AuthorizationException se lanza cuando el rol del usuario no tiene los permisos suficientes para acceder a este recurso.

En ambas situaciones se puede redireccionar a otra página y enviar el mensaje de error apropiado para cada caso.

Cuadro 2.3: Ejemplo de archivo para manejo de excepciones

```
<pages>
   ...
   <exception class="org.jboss.seam.security.NotLoggedInException">
      <redirect view-id="/login.xhtml">
         <message>Usted necesita estar logueado</message>
      </redirect>
   </exception>
   <exception class="org.jboss.seam.security.AuthorizationException">
      <end-conversation/>
      <redirect view-id="/security_error.xhtml">
         <message>Usted no tiene los permisos suficientes.</message>
      </redirect>
   </exception>
</pages>
```

Fuente: Capítulo 15 de la Documentación de JbossSeam

2.3.8.5 Seguridad a las páginas y a partes de las páginas

Para a aplicar seguridad a una página completa se puede utilizar en el archivo *.page* de la página que se desea y colocar *login-required="true"* y para administrar que roles pueden ver la página se puede agregar la etiqueta *restrict* así:

```
<page login-requied="true">
        <restrict>#{s:hasRole('nombre_del_rol')}</restrict>
</page>
```

Así mismo se puede aplicar la restricción en la propiedad *rendered* de los componentes dentro de una página así:

```
<h:outputLink action="#{xx.yy}"  redered="#{s:hasRole('admin')}"
```

Capítulo III: Desarrollo del Sistema

Para documentar el desarrollo de tal manera que se aprecie la evolución y el trabajo con la metodología *Scrum* se ha divido el presente capítulo en los *sprints* que conforman el proyecto, teniendo en cada uno las diferentes fases de análisis, diseño, desarrollo y pruebas.

3.1 *Sprint* 1

3.1.1 Análisis

En esta etapa se realiza el estudio preliminar, se define el problema y se toma los requerimientos del cliente. A partir de esto se puede hacer un análisis de costos del proyecto y plazos de entrega. Los entregables de esta etapa son:

- Las historias de usuario: Que son los requerimientos del cliente junto con sus pruebas.
- Las pantallas propuestas como idea inicial llamadas prototipos.
- El costo del proyecto y la estimación del tiempo de entrega.

3.1.1.1 Especificación de requisitos

Para la toma de requisitos se realizó una encuesta abierta al personal involucrado con la lógica de operación de la fundación. El formato de la encuesta se encuentra en los anexos. Por medio de preguntas e indagación de la lógica de operación se pudo concluir que la principal petición por parte de la fundación FUDIS es que el sistema permita el registro de pacientes y personas que los acompañan, con información relevante como sus datos personales, camas que van a ocupar y para el caso del paciente se debe almacenar además sus enfermedades, tratamientos, número de historia cínica, observaciones, fechas de ingreso y salida, etc.

Es indispensable que el sistema pueda registrar la salida y el reingreso de los hospedados.

El sistema debe también ayudar con la impresión de las entradas y salidas de los hospedados.

El software permite generar informes estadísticos únicamente de los pacientes que han sido registrados.

Y por último se pide la generación de varios usuarios para controlar el ingreso al sistema.

3.1.1.2 Historias de Usuario

Scrum indica que los requisitos se pueden recolectar en historias de usuario para luego estimar el valor y el esfuerzo que son necesario por cada una de las tareas que se obtengan.

Luego de la primera reunión se registraron los requerimientos obtenidos en historias de usuario.

Tabla 3.1: Historias de Usuario del sistema

Id: R1	Identificación al ingresar
Usuario: Administrador y Generador de Informes Para ingresar al sistema debe ingresar un usuario y contraseña, además se debe solo obtener los permisos según el usuario	
Prioridad: 1	

Id: Cp1	Pruebas de confirmación para <<Identificación al ingresar >>
-Cp1.1: Comprobar que solo se puede ingresar al sistema si se tiene un usuario y contraseña válido	
- Cp1.2: Verificar que tenga solo los permisos que le corresponden	

Id: R2	Administración de país
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar países	
Prioridad: 2	

Id: Cp2	Pruebas de confirmación para <<Administración de país >>
-Cp2.1: Comprobar que el administrador puede crear un país	
-Cp2.2: Comprobar que el administrador puede modificar un país	
-Cp2.3: Comprobar que el administrador puede eliminar un país	
-Cp2.4: Comprobar que el administrador puede buscar un país	

Id: R3	Administración de provincia
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar provincia.	
Prioridad:2	

Id: Cp3	Pruebas de confirmación para <<Administración de provincia >>
-Cp3.1: Comprobar que el administrador puede crear una provincia	
-Cp3.2: Comprobar que el administrador puede modificar una provincia	
-Cp3.3: Comprobar que el administrador puede eliminar una provincia	
-Cp3.4: Comprobar que el administrador puede buscar una provincia	

Id: R4	Administración de cantón
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar cantón.	
Prioridad: 2	

Id: Cp4	Pruebas de confirmación para <<Administración de cantón >>
-Cp4.1: Comprobar que el administrador puede crear un cantón	
-Cp4.2: Comprobar que el administrador puede modificar un cantón	
-Cp4.3: Comprobar que el administrador puede eliminar un cantón	
-Cp4.4: Comprobar que el administrador puede buscar un cantón	

Id: R5	Administración de bono
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar bonos.	
Prioridad: 2	

Id: Cp5	Pruebas de confirmación para <<Administración de bono >>
-Cp5.1: Comprobar que el administrador puede crear un bono	
-Cp5.2: Comprobar que el administrador puede modificar un bono	
-Cp5.3: Comprobar que el administrador puede eliminar un bono	
- Cp5.4: Comprobar que el administrador puede buscar un bono	

Id: R6	Administración de hospital
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar hospitales.	
Prioridad: 2	

Id: Cp6	Pruebas de confirmación para <<Administración de hospital >>
-Cp6.1: Comprobar que el administrador puede crear un hospital	
-Cp6.2: Comprobar que el administrador puede modificar un hospital	
-Cp6.3: Comprobar que el administrador puede eliminar un hospital	
-Cp6.4: Comprobar que el administrador puede buscar un hospital	

Id: R7	Administración de tratamiento
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar tratamientos.	
Prioridad: 5	

Id: Cp7	Pruebas de confirmación para <<Administración de tratamiento >>
-Cp7.1: Comprobar que el administrador puede crear un tratamiento	
-Cp7.2: Comprobar que el administrador puede modificar un tratamiento	
-Cp7.3: Comprobar que el administrador puede eliminar un tratamiento	
-Cp7.4: Comprobar que el administrador puede buscar un tratamiento	

Id: R8	Administración de tipo de enfermedad
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar tipos de enfermedades.	
Prioridad: 2	

Id: Cp8	Pruebas de confirmación <<Administración tipo de enfermedad >>
-Cp8.1: Comprobar que el administrador puede crear un tipo de enfermedad	
-Cp8.2: Comprobar que el administrador puede modificar un tipo de enfermedad	
-Cp8.3: Comprobar que el administrador puede eliminar un tipo de enfermedad	
-Cp8.4: Comprobar que el administrador puede buscar un tipo de enfermedad	

Id: R9	Administración de enfermedad
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar enfermedades.	
Prioridad: 2	

Id:Cp9	Pruebas de confirmación para <<Administración de enfermedad >>
-Cp9.1: Comprobar que el administrador puede crear una enfermedad	
-Cp9.2: Comprobar que el administrador puede modificar una enfermedad	
-Cp9.3: Comprobar que el administrador puede eliminar una enfermedad	
-Cp9.4: Comprobar que el administrador puede buscar una enfermedad	

Id: R10	Administración de ocupación
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar ocupaciones.	
Prioridad: 2	

Id: Cp10	Pruebas de confirmación para <<Administración de ocupación >>
-Cp10.1: Comprobar que el administrador puede crear una ocupación	
-Cp10.2: Comprobar que el administrador puede modificar una ocupación	
-Cp10.3: Comprobar que el administrador puede eliminar una ocupación	
-Cp10.4: Comprobar que el administrador puede buscar una ocupación	

Id: R11	Administración de habitación
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar habitaciones.	
Prioridad: 2	

Id: Cp11	Pruebas de confirmación para <<Administración de habitación >>
-Cp11.1: Comprobar que el administrador puede crear una habitación	
-Cp11.2: Comprobar que el administrador puede modificar una habitación	
-Cp11.3: Comprobar que el administrador puede eliminar una habitación	
-Cp11.4: Comprobar que el administrador puede buscar una habitación	

Id: R12	Administración de cama
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar camas.	
Prioridad: 2	

Id: Cp12	Pruebas de confirmación para <<Administración de cama >>
-Cp12.1: Comprobar que el administrador puede crear una cama	
-Cp12.2: Comprobar que el administrador puede modificar una cama	
-Cp12.3: Comprobar que el administrador puede eliminar una cama	
-Cp12.4: Comprobar que el administrador puede buscar una cama	

Id: R13	Administración de parentesco
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar parentescos	
Prioridad: 2	

Id: Cp13	Pruebas de confirmación para <<Administración de parentesco >>
-Cp13.1: Comprobar que el administrador puede crear un parentesco	
-Cp13.2: Comprobar que el administrador puede modificar un parentesco	
-Cp13.3: Comprobar que el administrador puede eliminar un parentesco	
-Cp13.4: Comprobar que el administrador puede buscar un parentesco	

Id: R14	Administración de familiar
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar familiares	
Prioridad: 5	

Id: Cp14	Pruebas de confirmación para <<Administración de familiar >>
-Cp14.1: Comprobar que el administrador puede crear un familiar	
-Cp14.2: Comprobar que el administrador puede modificar un familiar	
-Cp14.3: Comprobar que el administrador puede eliminar un familiar	
-Cp14.4: Comprobar que el administrador puede buscar un familiar	

Id: R15	Administración de paciente
Usuario: Administrador El administrador puede crear, modificar, eliminar y buscar pacientes	
Prioridad: 2	

Id: Cp15	Pruebas de confirmación para <<Administración de paciente >>
-Cp15.1: Comprobar que el administrador puede crear un paciente	
-Cp15.2: Comprobar que el administrador puede modificar un paciente	
-Cp15.3: Comprobar que el administrador puede eliminar un paciente	
-Cp15.4: Comprobar que el administrador puede buscar un paciente	

Id: R16	Ingreso de paciente
Usuario: Administrador Registrar el ingreso de un paciente con los familiares que lo acompañan	
Prioridad: 3	

Id: Cp16	Pruebas de confirmación para <<Ingreso de paciente>>
-Cp16.1: Comprobar que el administrador puede registrar el ingreso.	
-Cp16.2: Verificar que se pueda asignar las camas que van a ocupar el paciente y los familiares que lo acompañan.	
-Cp16.3: Comprobar que se puede asignar las enfermedades del paciente, así como sus tratamientos e instituciones e donde se tratan	
-Cp16.4: Ingresar los datos correctamente y verificar que el ingreso es creado y que las camas ya no están disponibles.	

Id: R17	Salida de paciente
Usuario: Administrador Registrar la salida de un paciente con los familiares que lo acompañan	
Prioridad: 3	

Id: Cp17	Pruebas de confirmación para <<Salida de paciente>>
-Cp17.1: Comprobar que el administrador puede registrar la salida.	
-Cp17.2: Verificar que se pueda asignar las condiciones en las cuales el paciente abandona la fundación	
-Cp17.3: Ingresar los datos correctamente y verificar que la salida es registradas que las camas vuelven a estar disponibles	

Id: R18	Imprimir registros de ingreso y de salida
Usuario: Administrador El administrador del sistema puede imprimir el reporte del ingreso y salida del paciente.	
Prioridad: 4	

Id: Cp18	Pruebas de confirmación para <<Imprimir registros de ingreso y de salida>>
-Cp18.1: Comprobar que el administrador puede imprimir los reportes de ingreso y salida.	

Id: 19	Generar informes
Usuario: Administrador y Generador de Informes Tanto el administrador del sistema como el usuario generador de informes pueden generar e imprimir los reportes estadísticos.	
Prioridad: 5	

Id: Cp19	Pruebas de confirmación para <<Generar informes >>
-Cp19.1: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por sexo	
-Cp19.2: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por tratamiento	
-Cp19.3: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte registros por fecha.	
-Cp19.4: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por ocupación	
-Cp19.5: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por cantón	

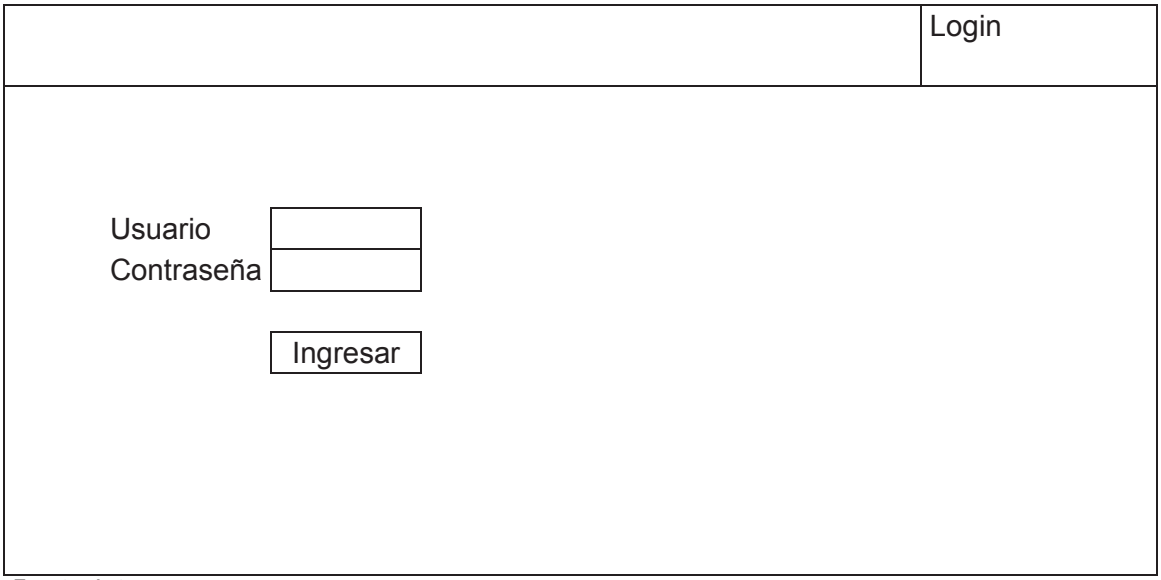
Fuente: Autor

3.1.1.3 Prototipos

Son una muestra inicial de las pantallas/páginas del sistema con la finalidad de que el usuario tenga una referencia de cómo se verá el sistema finalizado y que el equipo tenga una idea macro de la aplicación. Estas son utilizadas como guía para generar las pantallas finales, pudiendo haber cambios debido a

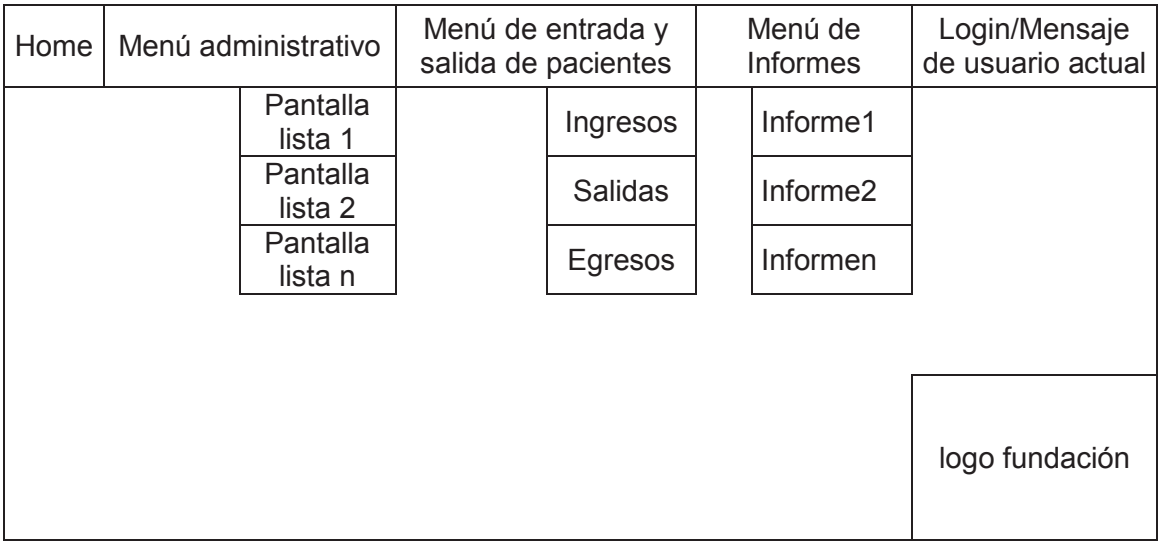
nuevos requerimientos que se presenten durante el transcurso del proyecto. A continuación se muestran los prototipos de las pantallas que se ha diseñado para los principales procesos del sistema.

Gráfico 3.1: Pantalla de Ingreso



Fuente: Autor

Gráfico 3.2: Pantalla Principal



Fuente: Autor

Pantallas de listado. En la que se puede encontrar todos los componentes de la administración como país, bonos, enfermedades, hospitales, etc.

Gráfico 3.3: Pantalla de listado

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Título
campo1
campo2
campo n
Criterio de búsqueda ○ Todo ● Cualquiera

Limpiar Buscar

Resultado de la búsqueda

campo1	campo2	campo3	campo n	Acción
				Ver Editar
				Ver Editar
				Ver Editar

<<Principio <Anterior Siguiente> Final>>

Crear nuevo

Fuente: Autor

Pantallas de vista. Usadas para ver los detalles de los componentes como provincia, ciudad, paciente, familiar, bono, etc.

Gráfico 3.4: Pantalla de vista

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Título
campo1 dato1
campo2 dato2
campo n dato n

Editar Aceptar

Fuente: Autor

Pantallas de crear/ editar. Para crear nuevos componentes de la administración por ejemplo cantones, roles, país, enfermedades, etc.

Gráfico 3.5: Pantalla para creación y modificación

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual
Título campo1 [ ] campo2 [ ] campo n [ ] Grabar/Actualizar Eliminar Cancelar				

Fuente: Autor

3.1.1.4 CRC (Class Responsibility Collaboration)

Con la finalidad de documentar el proceso de análisis de los requerimientos para la obtención de las clases del sistema se ha generado las tarjetas *CRC* a partir de las historias de usuario y los prototipos del sistema.

Las tarjetas *CRC*[5] son un modelo creado por Kent Beck y Ward Cunningham y consiste en una colección de fichas utilizadas para el diseño de software orientado a objetos y cada una de ellas se encuentran divididas en tres secciones con la finalidad de representar las clases del sistema. Las responsabilidades se pueden hallar contestando a las preguntas ¿qué información se necesita que almacene la clase? y ¿qué hace la clase?, lo que será utilizado luego como atributos y métodos de las clases.

[5] Beck Kent. *A Laboratory For Teaching Object - Oriented Thinking.* Recuperado de http://c2.com/doc/oopsla89/paper.html

Tabla 3.2: Tarjetas CRC del sistema

Usuario	
id nombre password login	Rol

Rol	
id nombre descripción	Usuario

País	
id nombre	Paciente Familiar

Provincia	
Id nombre	Paciente Familiar

Cantón	
id nombre	Paciente Familiar

Tipo de Enfermedad	
id nombre descripción	Enfermedad

Enfermedad	
id nombre descripción	Registro

Tratamiento	
id nombre	Enfermedad

Hospital	
Id nombre	Tratamiento

Bono	
id nombre descripción	Paciente

Ocupación	
Id nombre descripción	Paciente

Cama	
id número tipoCama	Registro

Observación	
id descripción	Paciente

Habitación	
id número piso	Cama

Tipo Observación	
Id nombre descripción	Observación

Paciente	
Id cedula nombre apellido genero teléfono teléfonos 2 sueldo dirección nivel de escolaridad condición económica fecha de nacimiento lugar de nacimiento lugar de residencia registrarse	Registro

Familiar	
Id cedula nombre apellido genero teléfono parentesco registrarse	Registro

Fuente: Autor

3.1.1.5 Estimación del plazo de entrega y costo del software

Scrum permite determinar un estimado del costo y el tiempo que tomara el desarrollo del software, el mismo que podrá variar conforme avancen los Sprints, este punto es muy importante ya que generalmente el cliente necesita saber con exactitud el plazo de entrega y cuánto le costará el software. Para esto se necesita determinar el esfuerzo en horas para cada una de los principales objetivos del alcance del proyecto.

Tabla 3.3: Estimación del costo del software.

Actividades	Horas estimadas
Análisis inicial	80
Investigación *JBoss Seam*	70
Investigación *RichFaces*	30
Investigación Generación de Informes	40
Administración general	60
Login de usuarios	50
Registro pacientes y familiares	60
Registro de Reingreso de paciente	40
Registro de salida de pacientes y familiares	40
Impresión de formularios	60
Generación de informes	75
Varios usuarios para el sistema	60
Implementación	70
Total	**735**

Fuente: Autor

Esta estimación se realiza tomando en cuenta que en cada *sprint* se debe realizar análisis, desarrollo, implementación y pruebas; y que además tomará un tiempo considerable el desarrollo ya que no se tiene mayor experiencia con el *framework* investigado. Con esto ya se puede estimar un tiempo de entrega dividendo el número de horas para 6 horas diarias que se dispone libres para el proyecto y esto dividirlo para 25 días que se estima se trabajaran cada mes; teniendo como resultado 4.9 meses para entregar el proyecto finalizado.

A continuación se debe calcular cuando es el costo de cada hora de trabajo. Tomando en cuenta que el desarrollo se realizo en un domicilio particular no se tiene gastos de transporte, alquiler, etc.

Los valores tomados en cuenta son los siguientes:

- Hora de programador: $ 3.12
- Desgaste del computador: $ 0.15
- Internet por hora: $ 0,13
- Energía Eléctrica: $ 0,02

Por lo tanto se determina que la hora de trabajo tiene un valor de $ 3.42 que multiplicadas por las 735 horas da como resultado $ 2513.7

A esto se debe añadir el costo del transporte a la fundación para las reuniones al comienzo de los 5 *Sprints* y 6 veces más para la implementación final y capacitación de los usuarios, los mismos que tienen un costo aproximado de $5.00 tomando en cuenta ida y vuelta, para hacer un total de $55.00

Además se debe tomar en cuenta el costo del software y hardware que se incluirá en el proyecto; que para este caso es $0, ya que se utilizará hardware que dispone la empresa y se desarrolla el proyecto utilizando software libre.

Como resultado de la suma de todos los costos, se tiene un valor del estimado del proyecto de: $2568.7

3.1.2 Diseño

En esta etapa se parte de los requerimientos y se conforma el grupo de trabajo, además de organizar las tareas a realizarse a lo largo del proyecto. Es en esta fase en donde se definen los siguientes productos:

- *Product Backlog*
- *Sprint Backlog*
- Diagrama de Clases[6]

3.1.2.1 Definición de Roles:

Es fundamental que una vez definidos los requerimientos se proceda a establecer los roles que ocupa cada persona en el proyecto, según la metodología *Scrum*. Debido a que este proyecto se realiza de forma individual, el "grupo" de trabajo solo estará conformado por una persona.

Tabla 3.4: Roles del proyecto

Scrum Master	John Obando
Product Owner	Inés Peña
Scrum Team	John Obando
Stakeholders	Gabriela Páez
	Amparo de Páez
	Universidad de las Américas

Fuente: Autor

3.1.2.2 Creación del *Product Backlog*

A continuación se presenta el *Product Backlog* que se genera a partir de los requerimientos mostrados en las historias de usuario y los diagramas *CRC*. Para más detalle ver en Anexos.

[6] DCC CHILE,*Modelo de clases*, Recuperado de http://www.dcc.uchile.cl/~psalinas/uml/modelo.html

Tabla 3.5: Product Backlog

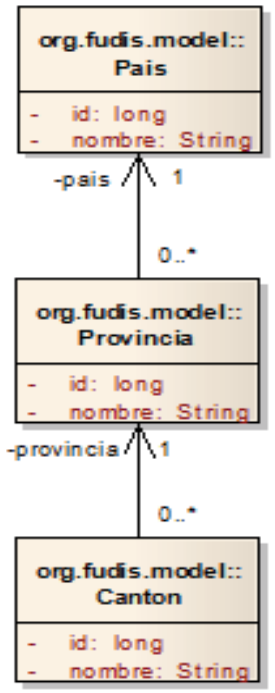
Id	Prioridad	Descripción	Sprint	Estimación de Valor	Estimación de Esfuerzo Inicial
1	1	Crear, modificar y eliminar pacientes, familiares, enfermedades, ocupaciones, observaciones, provincias, cantones, etc.	1	9	80
2	1	Controlar el ingreso al sistema por medio de usuarios.	1	9	70
3	2	Registro de Ingreso de pacientes	2	8	70
4	2	Registro de Ingreso de familiares	2	8	20
5	2	Asignación de camas a ingresados	2	7	20
6	2	Registro de Reingreso de paciente	2	8	10
7	2	Registro de salida de pacientes	2	8	15
8	2	Registro de salida de familiares	2	7	15
9	3	Impresión de formularios de Ingresos	3	6	150
10	4	Generación de informes estadísticos en formato *.xls*	4	5	150
11	5	Crear varios usuarios para el sistema y otorgar permisos	5	6	150

Fuente: Autor

3.1.2.3 Diagrama de Clases

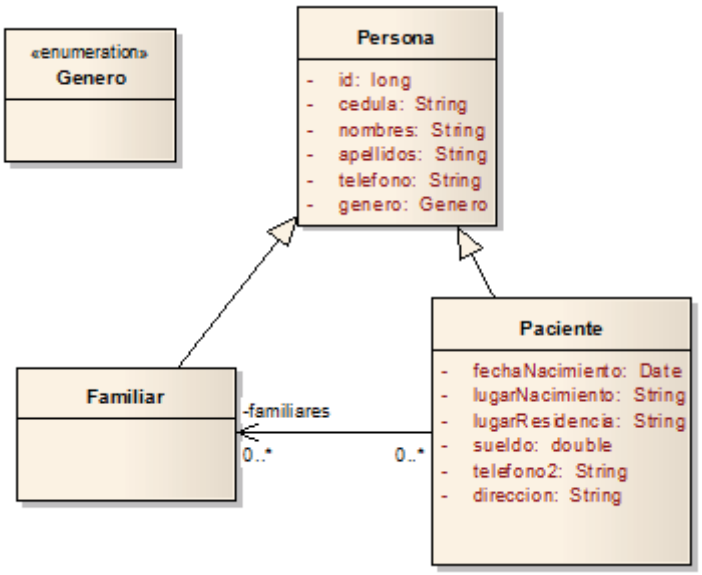
Luego de obtener la pila del producto o *Product Backlog* se ha realizado el diagrama de clases inicial usando como base las historias de usuario, los diagramas *CRC* y todo lo hablado con el cliente. El mismo que puede tener cambios conforme avance el proyecto, aunque lo ideal sería conseguir el detalle y claridad de todos los requerimientos del sistema para no tener cambios en el futuro.

Gráfico 3.6: Diagrama de clases Ubicación de personas



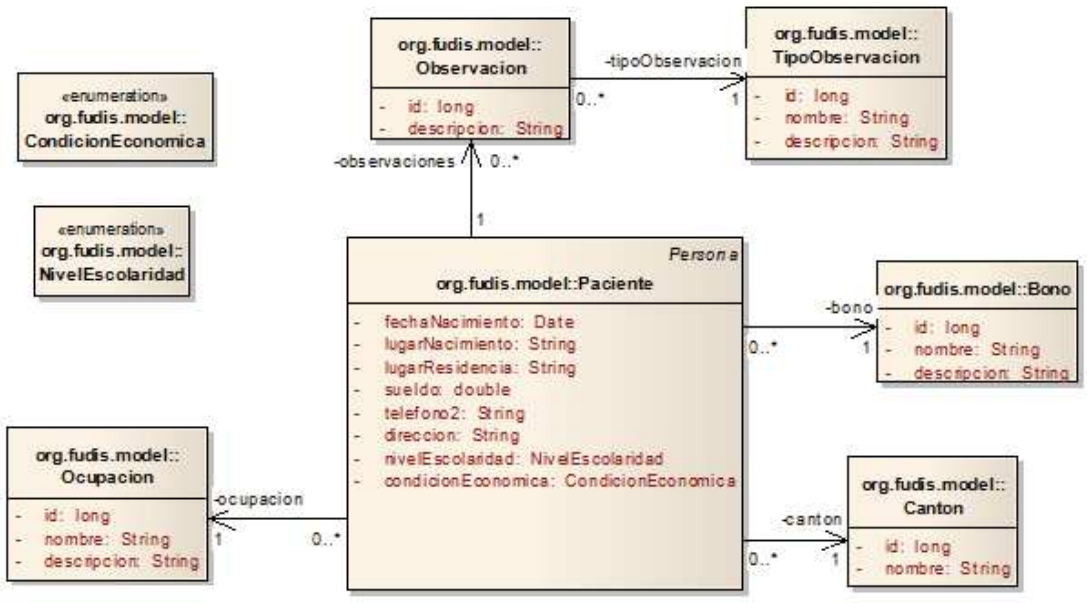
Fuente: Autor

Gráfico 3.7: Diagrama de clases Persona



Fuente: Autor

Gráfico 3.8: Diagrama de clases Paciente

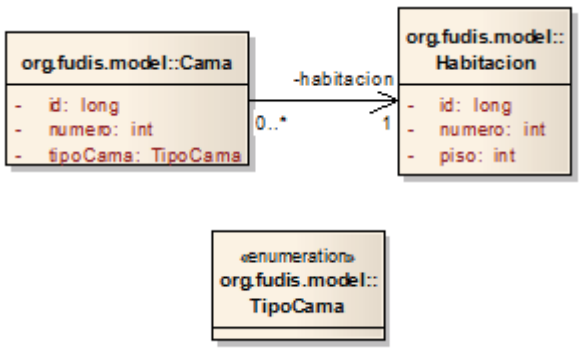


Fuente: Autor

Gráfico 3.9: Diagrama de clases Usuario

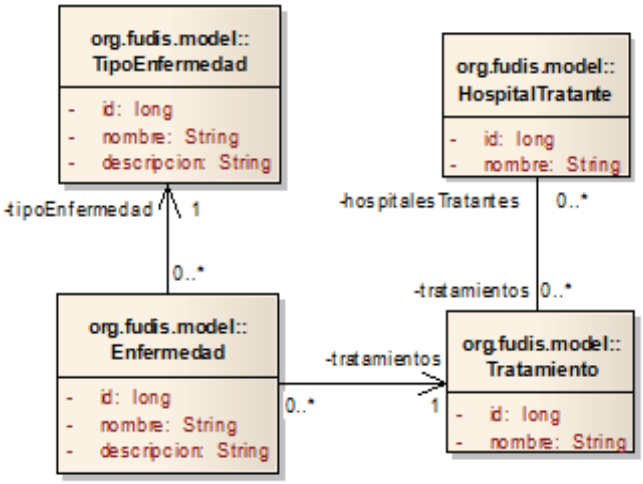
Fuente: Autor

Gráfico 3.10: Diagrama de clases Cama



Fuente: Autor

Gráfico 3.11: Diagrama de clases Enfermedad



Fuente: Autor

3.1.2.4 Desarrollo del *Sprint* 1

La creación del *SprintBacklog* es necesaria para la organización del grupo de trabajo o *Scrum Team* y así dar comienzo con el desarrollo de la aplicación dividiendo el *sprint* por tareas específicas. A continuación se muestra el *sprintbacklog* para el *sprint* 1. El avance diario se detalla en Anexos.

Tabla 3.6: Duración del Sprint1

***Sprint* 1**			
Fecha de inicio		14/09/2010	
Fecha de final		14/10/2010	
Longitud del *Sprint*		4 semanas	
Días laborables durante el *Sprint*		30	
Miembro del equipo	**Días disponibles durante el *Sprint***	**Horas Disponibles por día**	**Total horas disponibles**
John	25	6	150

Fuente: Autor

Tabla 3.7: Tareas del Sprint1

Elemento del *Product Backlog*	Tare del *Sprint*	Voluntario	Estimación de Esfuerzo Inicial
Crear, modificar y eliminar entidades en la base de datos	Análisis y Diseño del modelo	John	60
	Configuración de ambiente		6
	Creación de *crude* utilizando *seamgen*		12
	Traducción de vistas generadas		6
	Modificación de textos por combos		6
	Corrección de campos de ingreso		12
	Corrección de campos de vista		12
	Corrección de campos de listas		12
	Creación de listas en relaciones		12
Ingreso al sistema por medio de usuarios.	Manejo de *login* mediante md5		10
Pruebas	Ejecución de pruebas		2

Fuente: Autor

3.1.3 Implementación

En esta fase se debe realizar todo lo referente a la producción del software es decir la definición del ambiente y la codificación propiamente dicha. Teniendo como resultados los siguientes productos:

- La arquitectura de software.
- La arquitectura de hardware.

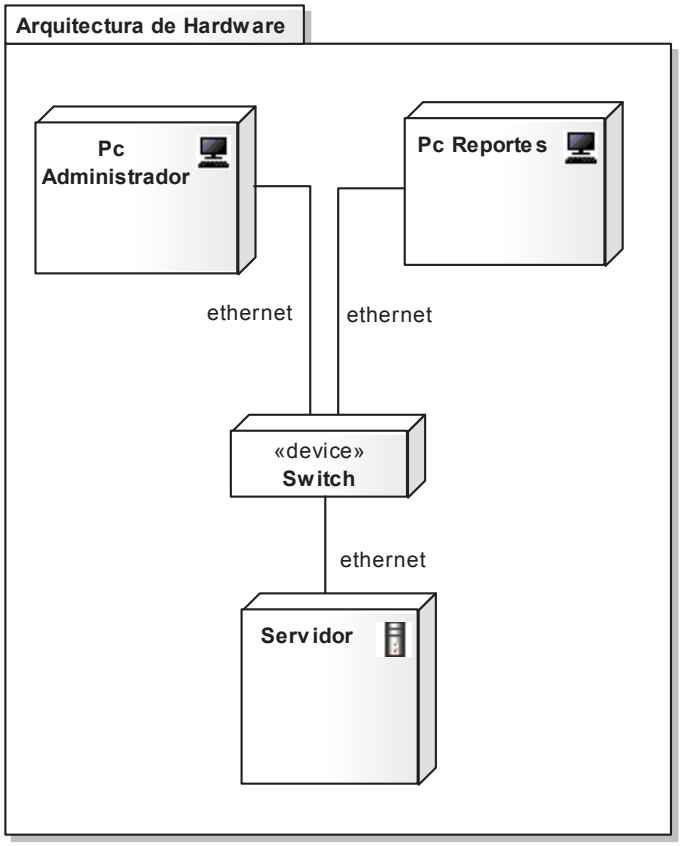
3.1.3.1 Arquitectura de la solución

- Arquitectura de Hardware

Se utilizará el hardware disponible en la empresa para no incurrir en gastos adicionales debido al ajustado presupuesto que tiene la fundación, es así que se utilizará un *Pc* para Servidor y dos más para *Pc*

Administrador y *Pc* de reportes, comunicadas a través de un *switch* como se muestra en la gráfica.

Gráfico 3.12: Arquitectura de Hardware



Fuente: Autor

A continuación se detallan las características de cada uno de los componentes.

Pc Administrador:

- Disco Duro 100 *Gb*
- Memoria *RAM* 1 *Gb*
- Procesador 1.8 *Ghz*

Pc de Reportes:

- Disco Duro 100 *Gb*
- Memoria RAM 1Gb
- Procesador 1.8 *Ghz*

Switch

- 10/100 *Mbps Fast Ethernet*
- 8 puertos

Servidor:

- Disco Duro 500 *Gb*
- Memoria *RAM* 2 *Gb*
- Procesador 2.3 *Ghz*

- Arquitectura de Software

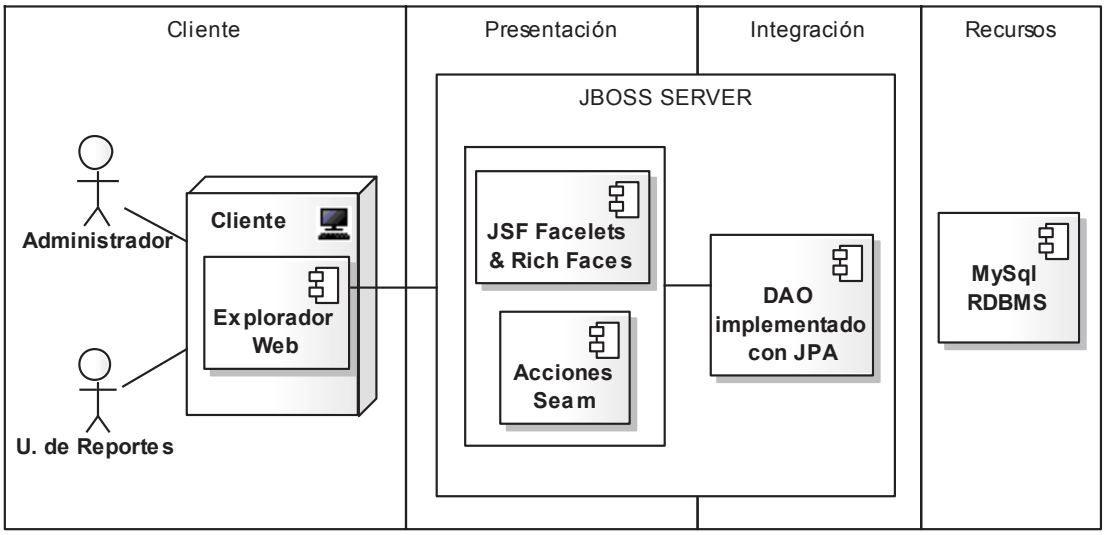
El servidor trabaja sobre el sistema operativo *Windows 7* en el cual se encuentra instalada la base de datos *Mysql 5.5* y el servidor de aplicaciones *JBoss-5.1*, elegidos por el tamaño del sistema y por la facilidad de adquirirlo debido a la gratuidad de los mismos.

Al ser una aplicación con interfaz web, las máquinas clientes solo necesitan un explorador web, se recomienda *Mozilla Firefox* 3 o 4.

La aplicación al estar elaborada bajo el *framework JBoss Seam* mantiene el patrón *DAO* (*Data Access Object*) [7] con lo que ayuda a mantener una programación independiente de la base de datos a usar, haciendo fácil el mantenimiento y si se quiere el cambio de base de datos. La plataforma de software se demuestra en el siguiente esquema:

[7] Sun Microsystems, 2001-2002, Core J2EE Patterns - Data Access Object, tomado de http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

Gráfico 3.13: Plataforma de software



Fuente: Autor

3.1.3.2 Construcción

Una vez definidos las herramientas, arquitectura, requisitos y haber configurado el ambiente de trabajo para el desarrollo se comenzó con la elaboración de la aplicación y para ello se siguió los siguientes pasos.

- Se creó una base de datos con un usuario con permisos para que acceda a la misma, mediante los comandos *create database nombreBD* y *grant all privileges on nombreBD.* to 'usuario' identified by 'clave;*
- Utilizando la aplicación *Seamgen* desde la consola se generó el proyecto mediante los comandos *seam setup* y *seam new-project.*
- En el *IDE Eclipse* se creó el modelo de clases mediante *POJOS* con las anotaciones respectivas en el proyecto generado con *seamgen*.
- Nuevamente usando *seamgen* se generó el *CRUD* de la aplicación mediante el comando *generate-ui*
- Usando el *IDE Eclipse* se procedió a modificar las clases, archivos y parámetros necesarios para ajustar el *CRUD* generado a los requerimientos.

  Se puede encontrar una parte del código fuente como ejemplo en Anexos.

3.1.4 Pruebas y evaluación

Esta etapa tiene como objetivo comprobar el correcto funcionamiento del sistema hasta donde se tenía planeado el *sprint,* para esto se toma los casos de prueba que se indican en las historias de usuario correspondientes y se verifica cada uno de ellos.

Tabla 3.8: Pruebas del sprint 1

Historia de Usuario	Id caso de prueba	Id Prueba	Prerrequisitos	Datos de Prueba	Resultados esperados	Resultados dados
R1	Cp1.1: Comprobar que solo se puede ingresar al sistema si se tiene un usuario y contraseña válido	P1	El usuario debe estar registrado	usuario=fudis, clave= no válida	El sistema deniega el ingreso	Correcto
		P2		usuario = no válido, clave = 123456	El sistema deniega el ingreso	Correcto
		P3		usuario = fudis, clave = 123456	El sistema permite el ingreso	Correcto
R2	Cp2.1: Comprobar que el administrador puede crear un país	P4	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P5		nombre= Ecuado	EL país es creado	Correcto
	Cp2.2: Comprobar que el administrador puede modificar un país	P6		nombre= vacío	El sistema muestra error	Correcto
		P7		nombre= Ecuador	EL país es modificado	Correcto
	Cp2.3: Comprobar que el administrador puede eliminar un país	P8		sin datos	EL país es eliminado	Correcto
	Cp2.4: Comprobar que el administrador puede buscar un país	P9		nombre=Perú	Se muestra el mensaje de 0 resultados	Correcto
		P10	El usuario debe estar *logueado* y el país debe existir	nombre=Ecuador	Se muestra el país buscado	Correcto
R3	Cp3.1: Comprobar que el administrador puede crear una provincia	P11	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P12		nombre= Pichincha	La provincia es creado	Correcto
	Cp3.2: Comprobar que el administrador puede modificar una provincia	P13		nombre= vacío	El sistema muestra error	Correcto
		P14		nombre= texto	La provincia es modificada	Correcto
	Cp3.3: Comprobar que el administrador puede eliminar una provincia	P15		sin datos	La provincia es eliminado	Correcto
	Cp3.4: Comprobar que el	P16	El usuario debe estar *logueado* y	nombre = Loja	Se muestra el mensaje de 0	Correcto

	administrador puede buscar una provincia por su nombre		la provincia debe existir		resultados	
		P17		nombre = Pichincha	Se muestra La provincia buscada	Correcto
R4	Cp4.1: Comprobar que el administrador puede crear un cantón	P18	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P19		nombre= Quit	El cantón es creado	Correcto
	Cp4.2: Comprobar que el administrador puede modificar un cantón	P20		nombre= vacío	El sistema muestra error	Correcto
		P21		nombre= Quito	El cantón es modificado	Correcto
	Cp4.3: Comprobar que el administrador puede eliminar un cantón	P22		sin datos	El cantón es eliminado	Correcto
	Cp4.4: Comprobar que el administrador puede buscar un cantón por su nombre	P23		nombre=Loja	Se muestra el mensaje de 0 resultados	Correcto
		P24	El usuario debe estar *logueado* y el cantón debe existir	nombre= Quito	Se muestra el cantón buscado	Correcto
R5	Cp5.1: Comprobar que el administrador puede crear un bono	P25	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P26		nombre= Bono edad	El bono es creado	Correcto
	Cp5.2: Comprobar que el administrador puede modificar un bono	P27		nombre= vacío	El sistema muestra error	Correcto
		P28		nombre= Bono tercera edad, descripción= mayores a 60 años	El bono es modificado	Correcto
	Cp5.3: Comprobar que el administrador puede eliminar un bono	P29		sin datos	El bono es eliminado	Correcto
	Cp5.4: Comprobar que el administrador puede buscar un bono por su nombre	P30		nombre = bono maternidad	Se muestra el mensaje de 0 resultados	Correcto
		P31		descripción = bono para madres	Se muestra el mensaje de 0 resultados	Correcto
		P32	El usuario debe estar *logueado* y el bono debe existir	nombre= Bono tercera edad	Se muestra el bono buscado	Correcto
		P33		descripción= mayores a 60 años	Se muestra el bono buscado	Correcto
R6	Cp6.1: Comprobar que el administrador puede crear un hospital	P34	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P35		nombre= Solca	El hospital es creado	Correcto
	Cp6.2: Comprobar que el administrador puede modificar un hospital	P36		nombre= vacío	El sistema muestra error	Correcto
		P37		nombre= SOLCA, descripción=hospital oncológico	El hospital es modificado	Correcto

	Cp6.3: Comprobar que el administrador puede eliminar un hospital	P38		sin datos	El hospital es eliminado	Correcto
	Cp6.4: Comprobar que el administrador puede buscar un hospital por su nombre	P39		nombre = militar	Se muestra el mensaje de 0 resultados	Correcto
		P40		descripción = hospital oncológico	Se muestra el mensaje de 0 resultados	Correcto
		P41	El usuario debe estar logueado y el hospital debe existir	nombre= SOLCA	Se muestra el hospital buscado	Correcto
R7	Cp7.1: Comprobar que el administrador puede crear un tratamiento	P42	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P43		nombre= Quimio terapia	El tratamiento es creado	Correcto
	Cp7.2: Comprobar que el administrador puede modificar un tratamiento	P44		nombre= vacío	El sistema muestra error	Correcto
		P45		nombre= Quimioterapia	El tratamiento es modificado	Correcto
	Cp7.3: Comprobar que el administrador puede eliminar un tratamiento	P46		sin datos	El tratamiento es eliminado	Correcto
	Cp7.4: Comprobar que el administrador puede buscar un tratamiento.	P47		nombre = radioterapia	Se muestra el mensaje de 0 resultados	Correcto
		P48	El usuario debe estar *logueado* y el tratamiento debe existir	nombre= Quimioterapia	Se muestra el tratamiento buscado	Correcto
R8	Cp8.1: Comprobar que el administrador puede crear un tipo de enfermedad	P49	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P50		nombre= cánceres	El tipo de enfermedad es creado	Correcto
	Cp8.2: Comprobar que el administrador puede modificar un tipo de enfermedad	P51		nombre= vacío	El sistema muestra error	Correcto
		P52		nombre= cánceres, descripción=todos los cánceres	El tipo de enfermedad es modificado	Correcto
	Cp8.3: Comprobar que el administrador puede eliminar un tipo de enfermedad	P53		sin datos	El tipo de enfermedad es eliminado	Correcto
	Cp8.4: Comprobar que el administrador puede buscar un tipo de enfermedad.	P54		nombre = psicológicas	Se muestra el mensaje de 0 resultados	Correcto
		P55		descripción = mentales	Se muestra el mensaje de 0 resultados	Correcto
		P56	El usuario debe estar logueado y el tipo de	nombre= cánceres	Se muestra el tipo de enfermedad	Correcto

			enfermedad debe existir		buscado	
		P57		descripción= todos los cánceres	Se muestra el tipo de enfermedad buscado	Correcto
R9	Cp9.1: Comprobar que el administrador puede crear una enfermedad	P58	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P59		nombre= cáncer a los pulmones	La enfermedad es creada	Correcto
	Cp9.2: Comprobar que el administrador puede modificar una enfermedad	P60		nombre= vacío	El sistema muestra error	Correcto
		P61		nombre= cáncer a los pulmones, descripción= pulmones	La enfermedad es modificada	Correcto
	Cp9.3: Comprobar que el administrador puede eliminar una enfermedad	P62		sin datos	La enfermedad es eliminada	Correcto
	Cp9.4: Comprobar que el administrador puede buscar una enfermedad.	P63		nombre = cáncer al útero	Se muestra el mensaje de 0 resultados	Correcto
		P64		descripción = respiratorias	Se muestra el mensaje de 0 resultados	Correcto
		P65	El usuario debe estar *logueado* y la enfermedad debe existir	nombre= cáncer a los pulmones	Se muestra la enfermedad buscada	Correcto
		P66		descripción= pulmones	Se muestra la enfermedad buscada	Correcto
R10	Cp10.1: Comprobar que el administrador puede crear una ocupación	P67	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P68		nombre= agricultura	La ocupación es creada	Correcto
	Cp10.2: Comprobar que el administrador puede modificar una ocupación	P69		nombre= vacío	El sistema muestra error	Correcto
		P70		nombre= agricultura, descripción= sin químicos	La ocupación es modificada	Correcto
	Cp10.3: Comprobar que el administrador puede eliminar una ocupación	P71		sin datos	La ocupación es eliminada	Correcto
	Cp10.4: Comprobar que el administrador puede buscar una ocupación.	P72		nombre = enfermería	Se muestra el mensaje de 0 resultados	Correcto
		P73		descripción = dentro de hospital	Se muestra el mensaje de 0 resultados	Correcto
		P74	El usuario debe estar *logueado* y la ocupación debe existir	nombre= agricultura	Se muestra la ocupación buscada	Correcto
		P75		descripción= sin químicos	Se muestra la ocupación buscada	Correcto
R11	Cp11.1: Comprobar que el administrador puede crear una	P76	El usuario debe estar *logueado*	número = vacío	El sistema muestra error	Correcto
		P77		número= 1	La habitación es creada	Correcto

	habitación					
	Cp11.2: Comprobar que el administrador puede modificar una habitación	P78		número = vacío	El sistema muestra error	Correcto
		P79		número= 1, piso=1	La habitación es modificada	Correcto
	Cp11.3: Comprobar que el administrador puede eliminar una habitación	P80		sin datos	La habitación es eliminada	Correcto
	Cp11.4: Comprobar que el administrador puede buscar una habitación.	P81	El usuario debe estar *logueado* y la habitación debe existir	sin datos	Se puede encontrar la habitación	Correcto
R12	Cp12.1: Comprobar que el administrador puede crear una cama	P82	El usuario debe estar *logueado*	número = vacío	El sistema muestra error	Correcto
		P83		número= 10	La cama es creada	Correcto
	Cp12.2: Comprobar que el administrador puede modificar una cama	P84		número = vacío	El sistema muestra error	Correcto
		P85		número= 1	La cama es modificada	Correcto
	Cp12.3: Comprobar que el administrador puede eliminar una cama	P86		sin datos	La cama es eliminada	Correcto
	Cp12.4: Comprobar que el administrador puede buscar una cama.	P87	El usuario debe estar *logueado* y la cama debe existir	sin datos	Se puede encontrar la cama	Correcto
R13	Cp13.1: Comprobar que el administrador puede crear un parentesco	P88	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P89		nombre= padre	El parentesco es creado	Correcto
	Cp13.2: Comprobar que el administrador puede modificar un parentesco	P90		nombre= vacío	El sistema muestra error	Correcto
		P91		nombre= padre, descripción=padre o padrastro	El parentesco es modificado	Correcto
	Cp13.3: Comprobar que el administrador puede eliminar un parentesco	P92		sin datos	El parentesco es eliminado	Correcto
	Cp13.4: Comprobar que el administrador puede buscar un parentesco	P93		nombre = hermano	Se muestra el mensaje de 0 resultados	Correcto
		P94		descripción = parentesco que no existente	Se muestra el mensaje de 0 resultados	Correcto
		P95	El usuario debe estar *logueado* y el parentesco debe existir	nombre= padre	Se muestra el parentesco buscado	Correcto
		P96		descripción= padre o padrastro	Se muestra el parentesco buscado	Correcto

R14	Cp14.1: Comprobar que el administrador puede crear un familiar	P97	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P98		nombre= Juan, apellido=Perez, cédula= 1104237670, sexo=masculino, teléfono=2543543	El familiar es creado	Correcto
	Cp14.2: Comprobar que el administrador puede modificar un familiar	P99		nombre= vacío	El sistema muestra error	Correcto
		P100		nombre= Diana, sexo=femenino	El familiar es modificado	Correcto
	Cp14.3: Comprobar que el administrador puede eliminar un familiar	P101		sin datos	El familiar es eliminado	Correcto
	Cp14.4: Comprobar que el administrador puede buscar un familiar	P102		nombre = Carmen	Se muestra el mensaje de 0 resultados	Correcto
		P103		apellido = Rosales	Se muestra el mensaje de 0 resultados	Correcto
		P104		cedula = 1101082829	Se muestra el mensaje de 0 resultados	Correcto
		P105	El usuario debe estar *logueado* y el familiar debe existir	nombre= Juan	Se muestra el familiar buscado	Correcto
		P106		apellido = Perez	Se muestra el familiar buscado	Correcto
		P107		cédula = 1104237670	Se muestra el familiar buscado	Correcto
R15	Cp15.1: Comprobar que el administrador puede crear un paciente	P108	El usuario debe estar *logueado*	nombre= vacío	El sistema muestra error	Correcto
		P109		nombre= María, apellido=Castro, cédula= 1104237670, sexo=femenino, teléfono=092553967	El paciente es creado	Correcto
	Cp15.2: Comprobar que el administrador puede modificar un paciente	P110		nombre= vacío	El sistema muestra error	Correcto
		P111		apellido= Torres, teléfono=042678765	El paciente es modificado	Correcto
	Cp15.3: Comprobar que el administrador puede eliminar un paciente	P112		sin datos	El paciente es eliminado	Correcto
	Cp15.4: Comprobar que el administrador puede buscar un paciente	P113		nombre = Carmen	Se muestra el mensaje de 0 resultados	Correcto
		P114		apellido = Rosales	Se muestra el mensaje de 0 resultados	Correcto
		P115		cedula = 1101082829	Se muestra el mensaje de 0 resultados	Correcto
		P116	El usuario debe estar *logueado* y el paciente debe	nombre=María	Se muestra el paciente buscado	Correcto

		P117	existir	apellido = Torres	Se muestra el paciente buscado	Correcto
		P118		cédula = 1104237670	Se muestra el paciente buscado	Correcto

Fuente: Autor

3.2 *Sprint* 2

3.2.1 Análisis

3.2.1.1 Nuevos Requisitos

- Cambiar el texto “genero” por “sexo”
- Guardar edad en el registro
- El texto “Hospital” cambiar por "Hospital o institución de referencia”
- Agregar observaciones en el egreso

3.2.1.2 Prototipos

Pantallas para de registrar entrada. Consta de dos partes una en la cual se escoge el paciente y la segunda para los datos del registro.

Gráfico 3.14: Pantalla para búsqueda de paciente

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Usuario actual

Buscador de Pacientes

campo1 [ ]
campo2 [ ]
campo n [ ]

Criterio de búsqueda ○ Todo ● Cualquiera

[Limpiar] [Buscar]

Resultado de la búsqueda

campo1	campo2	campo3	campo n	Acción
				Ver Registrar
				Ver Registrar
				Ver Registrar

<<Principio <Anterior Siguiente> Final>>

[Crear nuevo Paciente]

Fuente: Autor

Gráfico 3.15: Pantalla para registro de entrada

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Paciente *paciente seleccionado*

Fecha Ingreso

Enfermedades

Agregar		
Enfermedad	Tratamiento	Acción
		Quitar
		Quitar
		Quitar

Paciente

Agregar		
Nombre	Parentesco	Acción
		Quitar
		Quitar
		Quitar

Cama

Agregar				
Número	Habitación	Tipo	Piso	Acción
				Quitar
				Quitar
				Quitar

Observación

Agregar		
Nombre	Tipo	Acción
		Quitar
		Quitar
		Quitar

Grabar/Actualizar Eliminar Cancelar

Fuente: Autor

Pantallas para registrar salida. Dos pantallas: la primera permite elegir el registro y la segunda editar el mismo y grabar la salida.

Gráfico 3.16 Pantalla para búsqueda de registros

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Buscador de Registros

campo1	
campo2	
campo n	

Criterio de búsqueda ○ Todo ● Cualquiera

Limpiar Buscar

Resultado de la búsqueda

campo1	campo2	campo3	campo n	Acción
				Ver Registrar Salida
				Ver Registrar Salida
				Ver Registrar Salida

<<Principio <Anterior Siguiente> Final>>

Fuente: Autor

Gráfico 3.17 Pantalla para registro de salida

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Paciente *paciente seleccionado*
Fecha Ingreso
Enfermedades

Enfermedad	Tratamiento	Acción
		Quitar
		Quitar
		Quitar

Paciente

Nombre	Parentesco	Acción
		Quitar
		Quitar
		Quitar

Cama

Número	Habitación	Tipo	Piso	Acción
				Quitar
				Quitar
				Quitar

Observación

Nombre	Tipo	Acción
		Quitar
		Quitar
		Quitar

Fecha Salida

Grabar/Actualizar Eliminar Cancelar

Fuente: Autor

Gráfico 3.18: Pantalla para generar informes

Home	Menú administrativo	Menú de registros de entrada y salida de pacientes	Menú de Informes	Mensaje de usuario actual

Informe de ……

Fecha Inicial | Fecha Final

Edad Inicial | Edad Final

Resultado de la búsqueda

campo1	campo2	campo3	campo n

Importar a Excel

Fuente: Autor

3.2.1.3 CRC

Se encontraron algunas correcciones del modelo de *CRC* original con respecto a la relación que existe entre un familiar y un paciente, además la forma de modelar el registro. A continuación se muestran las modificaciones.

Tabla 3.9: Correcciones en las tarjetas CRC

Observación	
id descripción	Registro

Parentesco	
id nombre descripción	Relación

Relación	
Id Registrar parentesco	Paciente Parentesco Familiar

Enfermedad Registrada	
id registrar enfermedad	Enfermedad Paciente Registro

Observación Registrada	
Id registrar observación	Observación Paciente Registro

Registro	
Id historia clínica fecha de ingreso fecha de salida registrarse	Enfermedad Registrada Observación Registrada Cama Relación

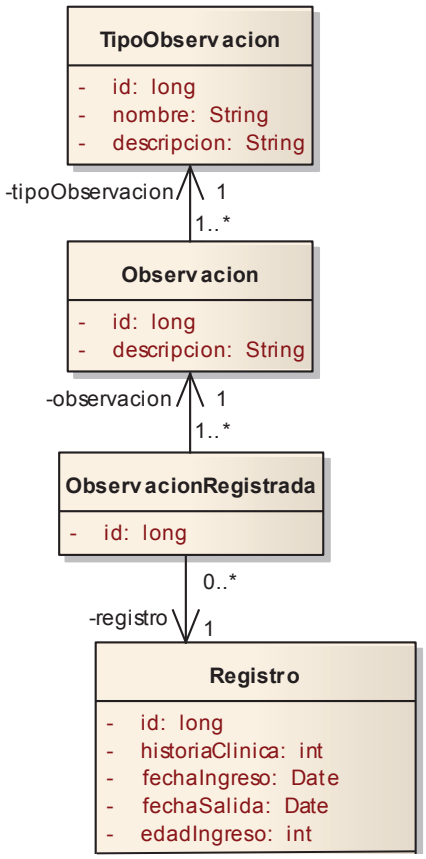
Fuente: Autor

3.2.2 Diseño

3.2.2.1 Corrección de Diagrama de Clases

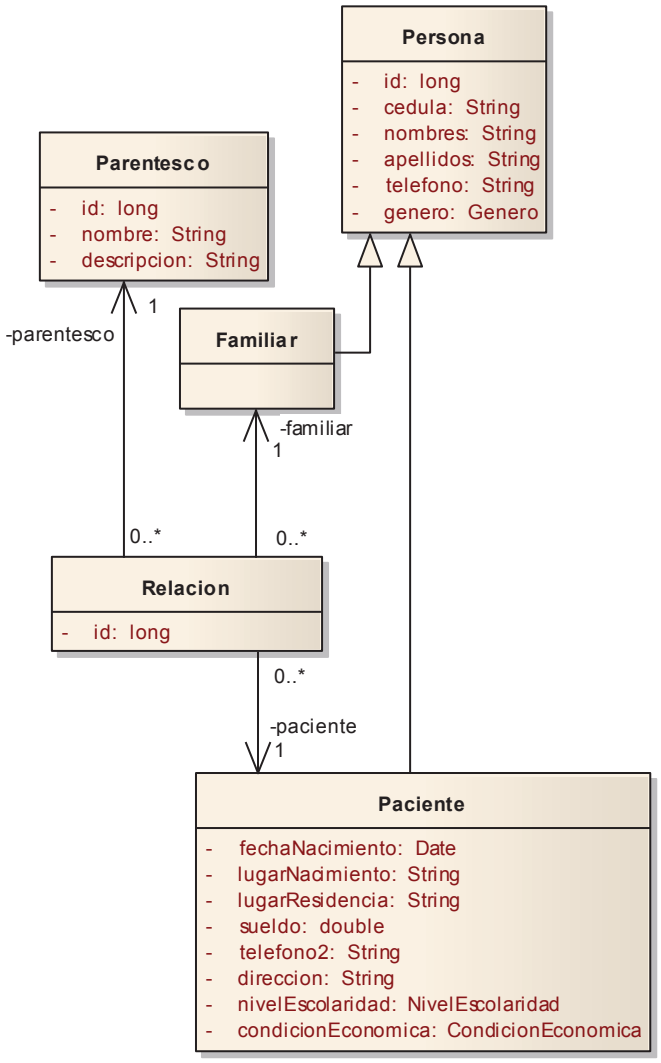
Así mismo en base a las modificaciones de los diagramas *CRC*, se corrige el diagrama de clases.

Gráfico 3.19: Diagrama de clases de Registro



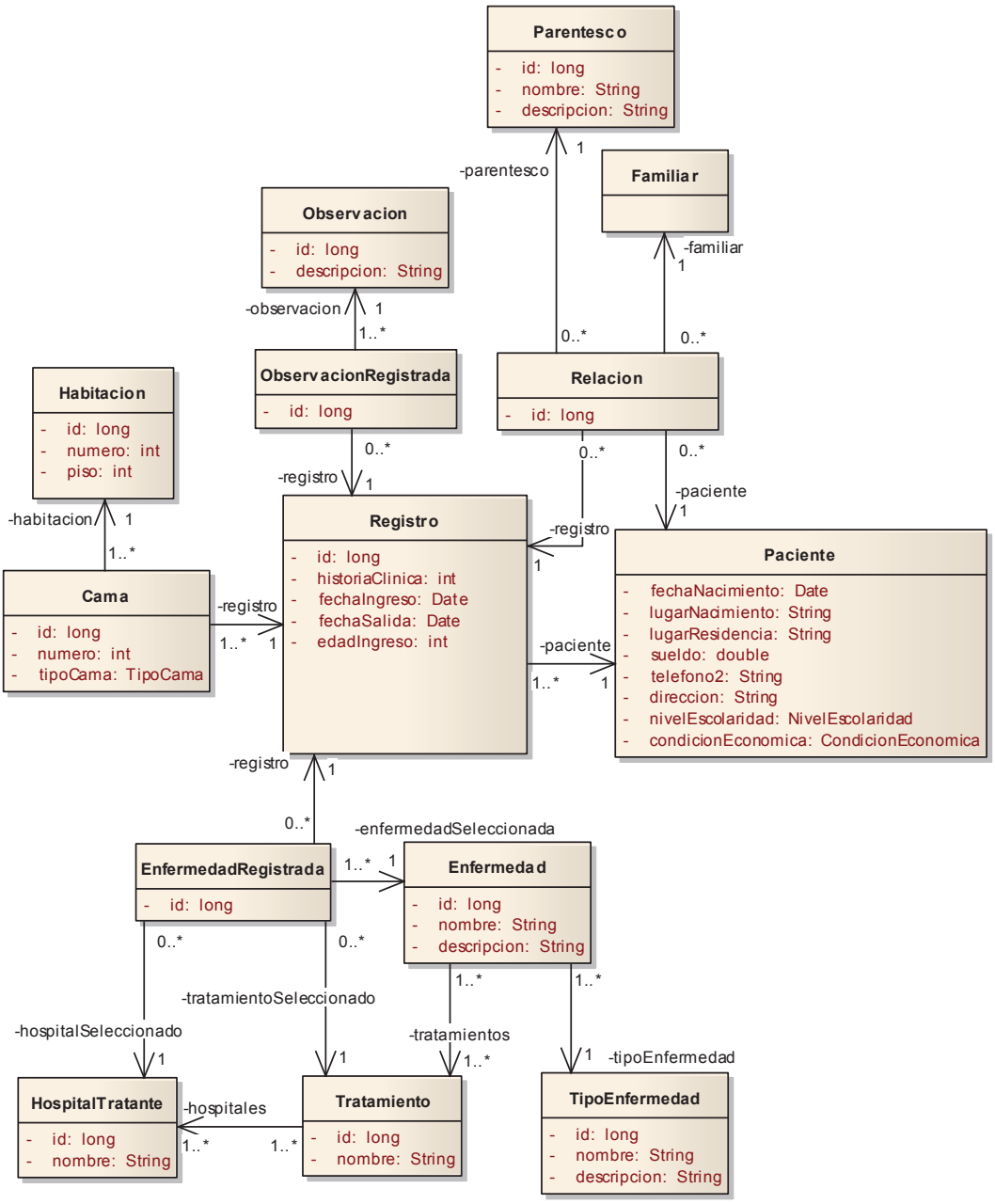
Fuente: Autor

Gráfico 3.20: Diagrama de clases de Paciente



Fuente: Autor

Gráfico 3.21: Diagrama de clases



Fuente: Autor

3.2.2.2 Desarrollo *Sprint* 2

El avance diario se detalla en los Anexos.

Tabla 3.10: Duración del Sprint 2

Sprint 2			
Fecha de inicio		16/10/2010	
Fecha de final		16/11/2010	
Longitud del *Sprint*		4 semanas	
Días laborables durante el *Sprint*		31	
Miembro del equipo	**Días disponibles durante el *Sprint***	**Horas Disponibles por día**	**Total horas disponibles**
John	25	6	150

Fuente: Autor

Tabla 3.11: Tareas del Sprint 2

Elemento del *Product Backlog*	**Tarea del *Sprint***	**Voluntario**	**Estimación de Esfuerzo Inicial**
Corrección	Análisis y corrección de diseño de clases	John	20
	Modificación de clases (programación)		18
Registro Ingreso de pacientes, Registro Ingreso de familiares, Registro Reingreso paciente, Asignación de camas	Creación de pantalla de búsqueda de pacientes no registrados		18
	Creación de pantalla de registro		20
Registro de Reingreso de paciente, Registro de salida de pacientes, Registro de salida de familiares	Creación de pantalla de búsqueda de registros ingresados		20
	Creación de pantalla de salida		20
	Creación de pantalla de búsqueda de todos los registros		18
Pruebas	Ejecución de pruebas		6

Fuente: Autor

3.2.3 Implementación

3.2.3.1 Construcción

En este sprint únicamente se hizo uso del *IDE Eclipse* para generar los *Pojos*, controladores y vistas necesarios para el registro de entrada y salida. Se puede encontrar una parte del código fuente como ejemplo en los Anexos

3.2.4 Pruebas y evaluación

Tabla 3.12: Pruebas del Sprint 2

Historia de Usuario	Id caso de prueba	Id Prueba	Prerrequisitos	Datos de Prueba	Resultados esperados	Resultados dados
R16	Cp16.1: Comprobar que el administrador puede registrar el ingreso.	P119	El usuario debe estar *logueado*, debe existir el paciente	paciente=María Torres, historia clínica= 1234565, fecha de ingreso= 03/02/2011	Se registra las camas	Correcto
	Cp16.2: Verificar que se pueda asignar las camas que van a ocupar el paciente y los familiares que lo acompañan.	P120	El usuario debe estar *logueado*, debe existir las camas 1 y 2	camas 1 y 2	Se registra las enfermedades	Correcto
	Cp16.3: Comprobar que se puede asignar las enfermedades del paciente, así como sus tratamientos e instituciones e donde se tratan	P121	El usuario debe estar *logueado*, debe existir cáncer al estomago y a los pulmones, quimioterapia y solca	enfermedades= cáncer al estómago y a los pulmones, tratamiento = quimioterapia, hospital = SOLCA	Registro creado, camas ocupadas	Correcto
	Cp16.4: Ingresar los datos correctamente y verificar que el ingreso es creado y que las camas ya no están disponibles.	P122	Registro creado		La salida debe registrarse	Correcto
R17	Cp17.1: Comprobar que el administrador puede registrar la salida.	P123	El usuario debe estar *logueado*, debe existir el registro.	fecha de salida= 10/02/2011	La salida debe registrarse con sus observaciones	Correcto
	Cp17.2: Verificar que se pueda asignar las condiciones de	P124		observación= en tratamiento, tipo de observación= salida	Las camas vuelven a estar disponibles.	Correcto

	salida del paciente.					
	Cp17.3: Ingresar los datos correctamente y verificar que la salida es registradas que las camas vuelven a estar disponibles	P125			Debe imprimirse los reportes	Correcto

Fuente: Autor

3.3 *Sprint* 3

3.3.1 Análisis

En este *sprint* no existieron nuevos requerimientos.

3.3.2 Diseño

3.3.2.1 Desarrollo *Sprint* 3

El avance diario se detalla en los Anexos.

Tabla 3.13: Duración del Sprint 3

***Sprint* 3**			
Fecha de inicio		17/11/2010	
Fecha de final		18/12/2010	
Longitud del *Sprint*		4 semanas	
Días laborables durante el *Sprint*		31	
Miembro del equipo	**Días disponibles durante el *Sprint***	**Horas Disponibles por día**	**Total horas disponibles**
John	25	6	150

Fuente: Autor

Tabla 3.14: Tareas del Sprint 3

Elemento del *Product Backlog*	**Tarea del *Sprint***	**Voluntario**	**Estimación de Esfuerzo Inicial**
Análisis	Pruebas de impresión	John	20
Impresión de formularios de Ingresos	Generación de pantallas de impresión		20
	Formato de pantalla		20
	Impresión con Java Script		20

	Inserción de botones para la impresión en las pantallas correspondientes		20
Pruebas	Ejecución de pruebas		2

Fuente: Autor

3.3.3 Implementación

3.3.3.1 Construcción

Este sprint se realizó las pantallas correspondientes para la impresión, además de las modificaciones en otra creadas anteriormente. Usando el *IDE Eclipse*. Se puede encontrar una parte del código fuente como ejemplo en los Anexos.

3.3.4 Pruebas y evaluación

Tabla 3.15: Pruebas del Sprint 3

Historia de Usuario	Id caso de prueba	Id Prueba	Prerrequisitos	Datos de Prueba	Resultados esperados	Resultados dados
R18	Cp18.1: Comprobar que el administrador pueden imprimir los reportes de ingreso y salida.	P126	El usuario debe estar *logueado*, debe existir el registro.		Debe generarse el informe	Correcto

Fuente: Autor

3.4 *Sprint* 4

3.4.1 Análisis

3.4.1.1 Nuevos Requisitos

Se necesita informes que respondan a las siguientes preguntas.

- ¿Qué cáncer son incidentes en hombres y mujeres; niños, jóvenes, adultos?
- ¿Qué enfermedades se tiene en cada cantón?
- ¿Qué ocupaciones pueden ser generadoras de cáncer?
- ¿Qué tratamientos medios son mas aplicados según a cada cáncer?
- ¿Qué número de pacientes se alojó en un período de tiempo determinado?

3.4.2 Diseño

3.4.2.1 Desarrollo *Sprint* 4

El avance diario se detalla en los Anexos.

Tabla 3.16: Duración del Sprint 4

Sprint 4			
Fecha de inicio		19/12/2010	
Fecha de final		21/01/2011	
Longitud del *Sprint*		4 semanas	
Días laborables durante el *Sprint*		33	
Miembro del equipo	**Días disponibles durante el *Sprint***	**Horas Disponibles por día**	**Total horas disponibles**
John	25	6	150

Fuente: Autor

Tabla 3.17: Tareas del Sprint 4

Elemento del *Product Backlog*	**Tarea del Sprint**	**Voluntario**	**Estimación Esfuerzo Inicial**
Generación de informes estadísticos en formato *.xls*	Creación de consultas para informes sexo y tratamiento	John	16
	Creación de pantalla filtro para sexo y tratamiento		12
	Creación de pantalla sexo y tratamiento *(xls)*		18
	Creación de consultas para informes cantón y ocupación		16
	Creación de pantalla filtro para cantón y ocupación		12
	Creación de pantalla cantón y ocupación *(xls)*		18
	Creación de consultas para informes alojamiento		16
	Creación de pantalla filtro para alojamiento		12
	Creación de pantalla alojamiento *(xls)*		18
Pruebas	Ejecución de pruebas		12

Fuente: Autor

3.4.3 Implementación

3.4.3.1 Construcción

Al igual que en el sprint anterior en este se hizo uso de *Eclipse* para la generación de los reportes. Se puede encontrar una parte del código fuente como ejemplo en los Anexos.

3.4.4 Pruebas y evaluación

Tabla 3.18: Pruebas del Sprint 4

Historia de Usuario	Id caso de prueba	Id Prueba	Prerrequisitos	Datos de Prueba	Resultados esperados	Resultados dados
R19	Cp19.1: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por sexo	P127	Usuario Administrador	fecha inicio=01/11/2010 fechafin:01/02/2011 edadInicio=5 edadFin=60	Debe generarse el informe	Correcto
		P128	Usuario de reportes		Debe generarse el informe	Correcto
	Cp19.2: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por tratamiento	P129	Usuario Administrador	fecha inicio=01/11/2010 fechafin:01/02/2011 edadInicio=5 edadFin=60	Debe generarse el informe	Correcto
		P130	Usuario de reportes		Debe generarse el informe	Correcto
	Cp19.3: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte registros por fecha.	P131	Usuario Administrador	fecha inicio=01/11/2010 fechafin:01/02/2011 edadInicio=5 edadFin=60	Debe generarse el informe	Correcto
		P132	Usuario de reportes		Debe generarse el informe	Correcto
	Cp19.5: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por ocupación	P133	Usuario Administrador	fecha inicio=01/11/2010 fechafin:01/02/2011 edadInicio=5 edadFin=60	Debe generarse el informe	Correcto
		P134	Usuario de reportes		Debe generarse el informe	Correcto
	Cp19.5: Comprobar que el administrador y el usuario generador de reportes pueden generar e imprimir el reporte enfermedad por cantón	P135	Usuario Administrador	fecha inicio=01/11/2010 fechafin:01/02/2011 edadInicio=5 edadFin=60	Debe generarse el informe	Correcto
		P136	Usuario de reportes		Debe generarse el informe	Correcto

Fuente: Autor

3.5 *Sprint* 5

3.5.1 Análisis

- Inesperadamente se presenta el requerimiento de eliminar el atributo historia clínica del Registro y por lo tanto el sistema debe modificarse desde el *Pojo*

hasta las vistas, este cambio se produjo debido a que las instituciones que colaboran con la fundación ya no les facilitaran este dato del paciente.

3.5.2 Diseño

3.5.2.1 CRC

Existe un cambio en el registro, ahora ya no existe historia clínica

Tabla 3.19: Tarjeta CRC del registro

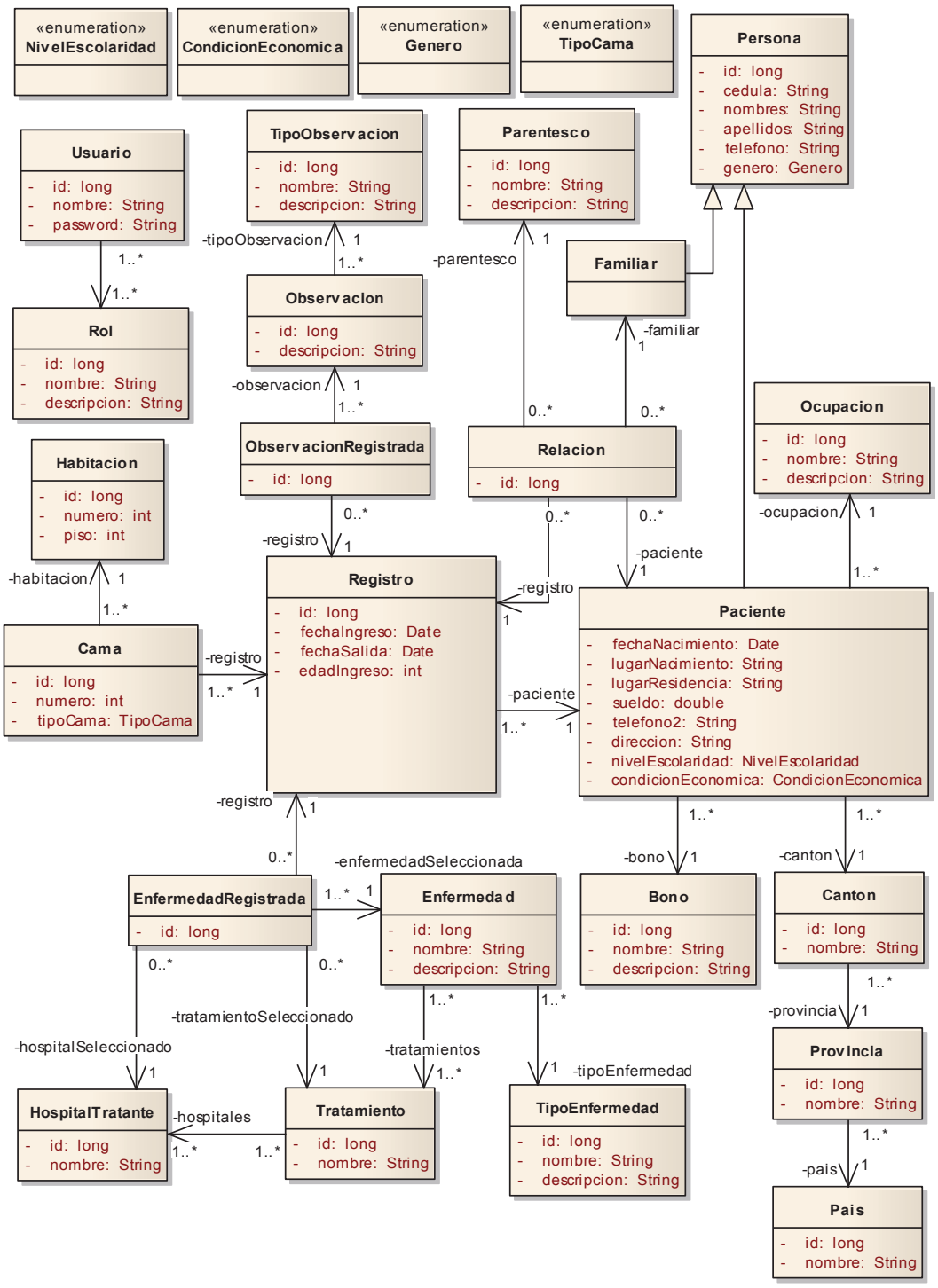
Registro	
Id fecha de ingreso fecha de salida registrarse	Enfermedad Registrada Observación Registrada Cama Relación

Fuente: Autor

3.5.2.2 Diagrama de clases Final

Finalmente el diagrama de clases se ha consolidado y especificado de acuerdo a todos los detalles que la fundación necesita, durante el desarrollo de cada iteración se ha encontrado correcciones que se han añadido o actualizado en el diagrama inicial.

Grafico: 3.22: Diagrama de clases final



Fuente: Autor

3.5.2.3 Desarrollo *Sprint* 5

El avance diario se detalla en los Anexo.

Tabla 3.20: Duración del Sprint 5

Sprint 5			
Fecha de inicio		22/01/2011	
Fecha de final		22/02/2011	
Longitud del *Sprint*		4 semanas	
Días laborables durante el *Sprint*		31	
Miembro del equipo	**Días disponibles durante el *Sprint***	**Horas Disponibles por día**	**Total horas disponibles**
John	25	6	150

Fuente: Autor

Tabla 3.21: Tareas del Sprint 5

Elemento del *Product Backlog*	**Tare del *Sprint***	**Voluntario**	**Estimación de Esfuerzo Inicial**
Cambios	Quitar historia clínica	John	12
Crear varios usuarios para el sistema y otorgar permisos	Crear los usuarios y roles		6
	Definir las acciones para cada rol		12
	Definir los roles de cada usuarios		6
	Crear seguridad para administrador		20
	Crear seguridad para reportes		20
	Correcciones finales debido a los roles		20
Pruebas	Ejecución de pruebas		6

Fuente: Autor

3.5.3 Implementación

3.5.3.1 Construcción

Usando *Eclipse* se modifico las pantallas y archivos necesarios para controlar el acceso a la aplicación según el rol del usuario, además de esto se creó en la base de datos un usuario con el rol de administrador para que pueda acceder al sistema. Se puede encontrar una parte del código fuente como ejemplo en los Anexos.

3.5.4 Pruebas y Evaluación

Tabla 3.22: Pruebas del Sprint 5

Historia de Usuario	Id caso de prueba	Id Prueba	Prerrequisitos	Datos de Prueba	Resultados esperados	Resultados dados
R1	Cp1.2: Verificar que tenga solo los permisos que le corresponden	P137	El usuario debe estar registrado	usuario = fudis, clave = 123456	El sistema muestra todo el menú	Correcto
		P138		usuario = reportes, clave = 123456	El sistema muestra solo el menú para generar reportes	Correcto

Fuente: Autor

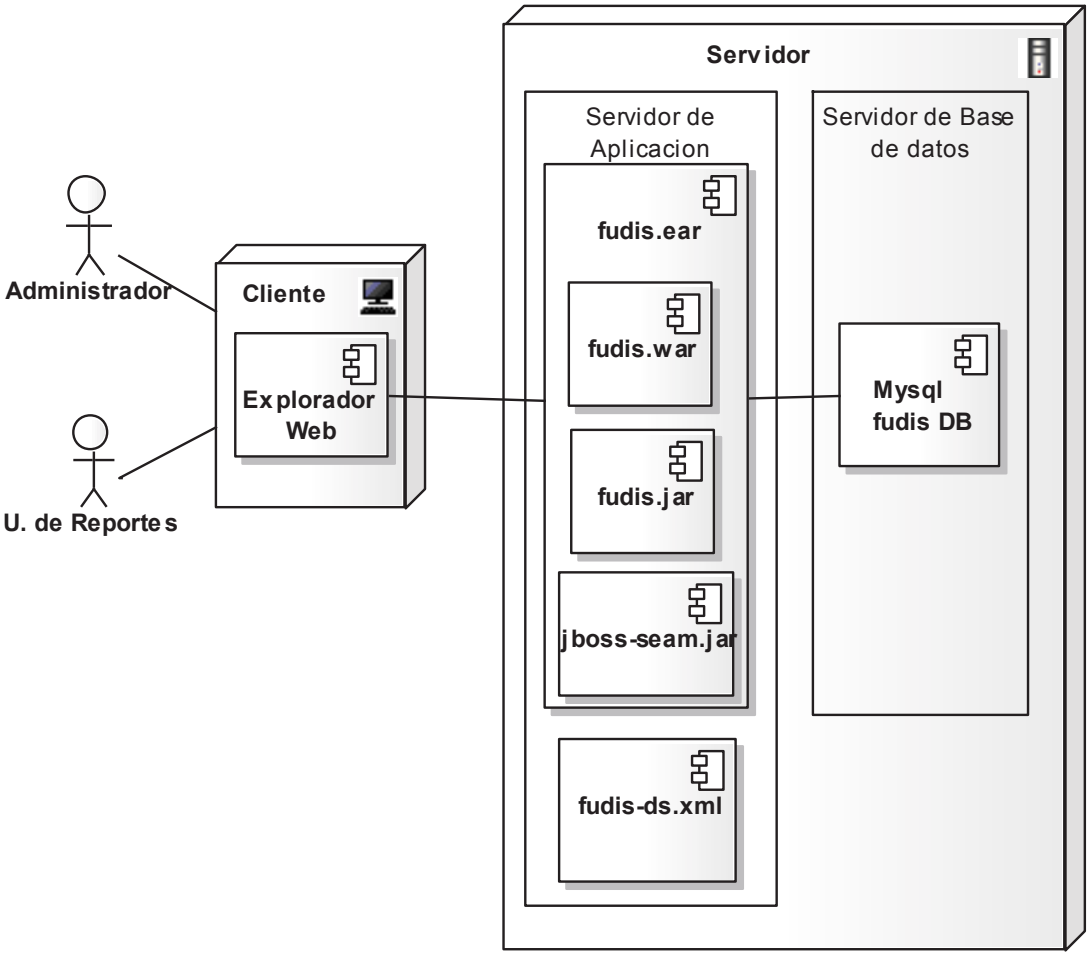
3.5.5 Implantación

Al finalizar el sistema y corregir los errores encontrados en el sistema, se procedió con la implantación de la aplicación en la fundación FUDIS, el siguiente fue el procedimiento:

- Instalar en el servidor la máquina virtual de *Java*, base de datos *Mysql*, servidor de aplicaciones *JBoss*.
- Crear la base de datos y un usuario con permisos solo para la base de datos de la aplicación.
- Crear accesos directos (marcadores) de la aplicación en los exploradores para la rápida ubicación de la aplicación.
- Indicar al personal el proceso para ingresar a la aplicación.

NOTA: para mayor detalle, se puede consultar en los Anexos el manual de implantación.

Gráfico: 3.23: Diagrama de Implantación



Fuente: Autor

Capítulo IV: Conclusiones y Recomendaciones

4.1 Conclusiones

- El sistema desarrollado funciona correctamente y cumple con las necesidades del beneficiario de este trabajo. Aportando rapidez en el proceso de ingreso y salida de pacientes así como también en la generación de datos estadísticos acerca de los hospedados.

- El uso de *JBoss Seam* como *framework* de desarrollo junto con su herramienta *Seamgen* agilita la creación de los archivos de configuración y el *CRUD* de un sistema; además brinda más tiempo para el análisis y la implementación de los procesos más complejos de la lógica de negocio.

- La metodología de desarrollo *Scrum* permite: un desarrollo ágil, lo que evita la excesiva documentación; apoya la comunicación entre todos los involucrados en el proyecto; permite una gran respuesta a cambios en cualquier etapa del desarrollo y, disminuye la posibilidad del fracaso de un proyecto.

- Gracias a la metodología *Scrum* se pudo mostrar los avances al cliente al final de cada *sprint*, completar con éxito el proyecto y presentarlo en el tiempo estimado.

- No es necesario de una herramienta de gestión de proyectos cuando se necesita desarrollar un sistema pequeño, pero es fundamental hacer el debido seguimiento de una manera ordenada y sistematizada según la indicación de la metodología.

- *Scrum* permite el análisis del avance del sistema de una forma eficiente y por iteraciones mediante los *Burndown Chart* permitiendo tomar medidas reactivas en caso de atrasos en los plazos de cumplimiento.

4.2 Recomendaciones

- Dividir la documentación del desarrollo en iteraciones ayuda a visualizar y comprender el ciclo de elaboración de un proyecto realizado con una metodología iterativa.

- Es aconsejable usar la herramienta *Seamgen* de *JBoss Seam* cuando se trate de aplicaciones pequeñas ya que al tener una lógica de negocio compleja se debería utilizar una capa más en el desarrollo.

- Para facilitar la definición de las clases del sistema se recomienda apoyarse en herramientas como las historias de usuario y las tarjetas *CRC*; que aunque no están especificadas por *Scrum* son de gran ayuda para un análisis correcto de los requisitos.

- Es imprescindible la comunicación con el cliente para la toma de cualquier decisión del proyecto, es por eso recomendable la constante notificación al *Product Owner* antes de realizar un cambio.

- Para evitar demoras en el desarrollo y posteriormente en la implantación se aconseja utilizar las versiones estables de todas las herramientas, aplicaciones y librerías que se ha de utilizar en el proyecto.

- Al momento de realizar la implantación se tuvo problemas con el servidor de aplicaciones, es por esto que se recomienda, en la medida de lo posible, desarrollar la aplicación en un entorno igual al que va a ser puesto en producción

Bibliografía

- CENTERS for MEDICARE & MEDICAID SERVICES, Selecting a Development Approach, http://www.cms.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf, fecha original de emisión: 17/02/2005, fecha de consulta: 10/07/2010.
- SÁNCHEZ, María A., Metodologías De Desarrollo De Software, http://www.informatizate.net, 2004, fecha de consulta: 10/06/2010.
- UVA, Marcelo, Metodologías de Desarrollo de Software, http://dc.exa.unrc.edu.ar, 2010, fecha de consulta: 11/07/2010.
- WIRWIN, Metodología SCRUM para la dirección de proyectos informáticos, http://ejecucion.wordpress.com/2009/06/10/metodologia-scrum-para-la-direccion-de-proyectos-informaticos/, 2009, fecha de consulta: 12/08/2010.
- PYMECRUNCH, SCRUM: metodología “ágil” para tus proyectos, http://pymecrunch.com/scrum-metodologia-agil-para-tus-proyectos, 2008, fecha de consulta: 12/08/2010.
- RODRÍGUEZ, L., Scrum, una metodología Ágil (II), http://es.debugmodeon.com/articulo/scrum-una-metodologia-agil-ii, 2009, fecha de consulta: 13/08/2010.
- RISING, L., JANOFF, N. S., The Scrum Software Development Process for Small Teams, IEEE Software, http://members.cox.net/risingl1/Articles/IEEEScrum.pdf, 2000, fecha de consulta: 15/08/2010, p. 2-8.
- SCHWABER, K., SUTHERLAND, J., Advanced Development Methods. SCRUM Development Process, http://jeffsutherland.com/scrumpapers.pdf, 2010, fecha de consulta: 17/08/2010.
- SCOTT W. A., Class Responsibility Collaborator (CRC) Models, http://www.agilemodeling.com/artifacts/crcModel.htm, 2003, fecha de consulta: 18/08/2010.

- SCOTT W. A., CRC Modeling: Bridging the Communication Gap Between Developers and Users, http://www.uml.org.cn/UMLApplication/pdf/crcModeling.pdf,1998, fecha de consulta: 01/09/2010.
- SCHWABER, K., SUTHERLAND, J., Nut, Bolts, and Origins of an Agile Framework, http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software, 2010, fecha de consulta: 05/09/2010.
- PERALTA, A., Metodología Scrum, http://athenea.ort.edu.uy/publicaciones/ingsoft/investigacion/ayudantias/SCRUM.pdf, 2003, fecha de consulta: 05/09/2010.
- SCHWABER, K., Guía sobre Scrum, http://www.scrum.org/storage/scrumguides/Gua%20sobre%20Scrum.pdf, Mayo, 2009, fecha de consulta: 06/09/2010.
- PALACIO, J., Flexibilidad con Scrum Principios de diseño e implantación de campos de Scrum, http://www.navegapolis.net/content/view/694/61/, 2008, fecha de consulta: 07/09/2010.
- DEEMER, P., BENEFIELD, B., LARMAN, C., VODDE, B., THE SCRUM PRIMER**,** http://scrumtraininginstitute.com/home/stream_download/scrumprimer-es , 2009, fecha de consulta: 08/09/2010
- SÁNCHES, J., Introducción a RichFaces, http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=richFacesJsfIntro, 2010, fecha de consulta: 05/10/2010.
- MADEJA, RichFaces, http://www.juntadeandalucia.es/xwiki/bin/view/MADEJA/RichFaces, 2010, fecha de consulta: 06/10/2010.
- ALCAZAR, Israel, Introducción JSF: RichFaces e IceFaces, http://www.slideshare.net/israelalcazar/introduccin-jsf-richfaces-e-icefaces-2267358, 2009, fecha de consulta: 01/11/2010.

- KING, G., MUIR, P., RICHARDS, N., otros, http://docs.jboss.org/seam/2.1.1.GA/reference/en-US/html/security.html#d0e10796, 2009, 03-01-2011.

Anexos

Product Backlog

A continuación se muestran el avance que se tuvo en el *product backlog*, se puede observar las celdas marcadas que definen la fecha límite para acabar con el ítem, lo que demuestra que algunos de ellos se terminaron luego del plazo fijado.

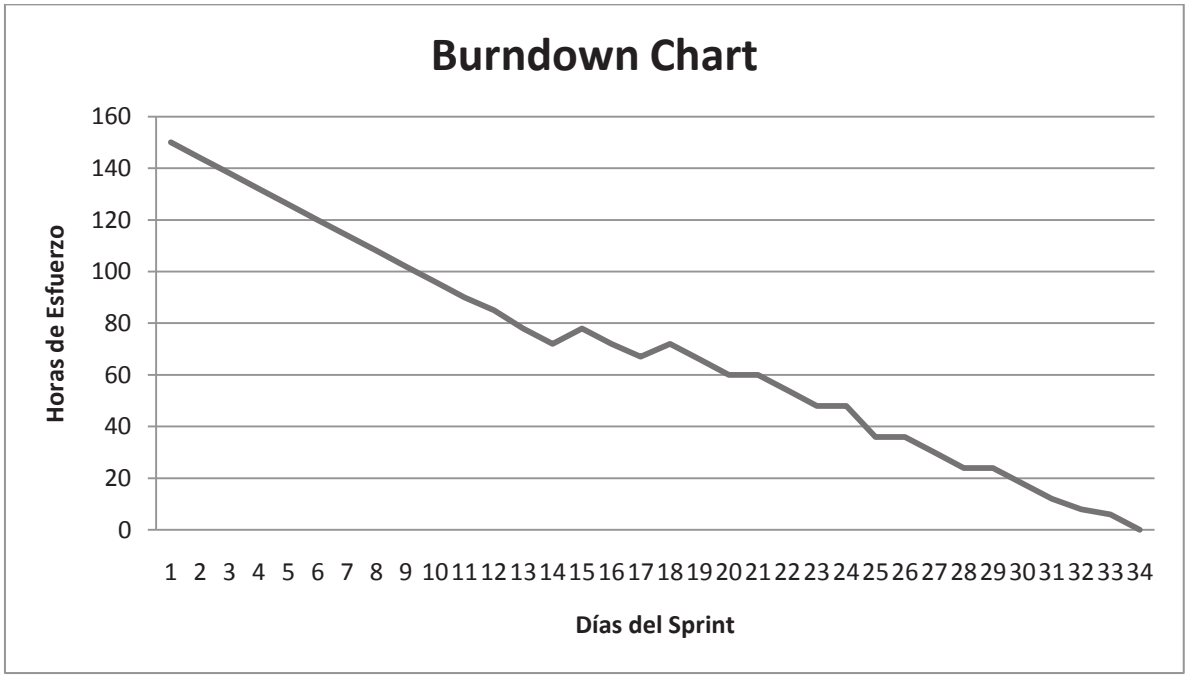
Id	Prioridad	Descripción	Sprint	Estimación de Valor	Estimación de Esfuerzo Inicial	Nuevo Esfuerzo Estimado (final del *sprint*)				
1	1	Crear, modificar y eliminar pacientes, familiares, enfermedades, ocupaciones, observaciones, provincias, cantones, etc.	1	9	80	0				
2	1	Controlar el ingreso al sistema por medio de usuarios.	1	9	70	0				
3	2	Registro de Ingreso de pacientes	2	8	70	70	0			
4	2	Registro de Ingreso de familiares	2	8	20	20	0			
5	2	Asignación de camas a ingresados	2	7	20	20	0			
6	2	Registro de Reingreso de paciente	2	8	10	10	0			
7	2	Registro de salida de pacientes	2	8	15	15	0			
8	2	Registro de salida de familiares	2	7	15	15	0			
9	3	Impresión de formularios de Ingresos	3	6	150	150	150	0		
10	4	Generación de informes estadísticos en formato *.xls*	4	5	150	150	150	150	0	
11	5	Crear varios usuarios para el sistema y otorgar permisos	5	6	150	150	150	150	150	0

***Sprint Backlog* Diario**

Sprint1

Elemento del *Product Backlog*	Tare del *Sprint*	Voluntario	Estimación Esfuerzo Inicial	Nuevo Esfuerzo Estimado (al final del día)																																
Crear, modificar y eliminar entidades en la base de datos	Análisis y Diseño del modelo	John	60	54	48	42	36	30	24	18	12	6	0																							
	Configuración de ambiente		6	6	6	6	6	6	6	6	6	6	6	1	0																					
	Creación de *crude* utilizando *seamgen*		12	12	12	12	12	12	12	12	12	12	12	12	6	0																				
	Traducción vistas generadas		6	6	6	6	6	6	6	6	6	6	6	6	6	6	12	6	1	0																
	Modificación de textos por combos		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	12	6	0														
	Corrección de campos de ingreso		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	6	0											
	Corrección de campos de vista		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	0									
	Corrección de campos de listas		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	6	0						
	Creación de listas en relaciones		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	6	0			
Ingreso al sistema por medio de usuarios.	Manejo de *login* mediante md5		10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	6	0	
Pruebas	Ejecución de pruebas		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	6	0
Total			150	144	138	132	126	120	114	108	102	96	90	85	78	72	78	72	67	72	66	60	60	54	48	48	36	36	30	24	24	18	12	8	6	0

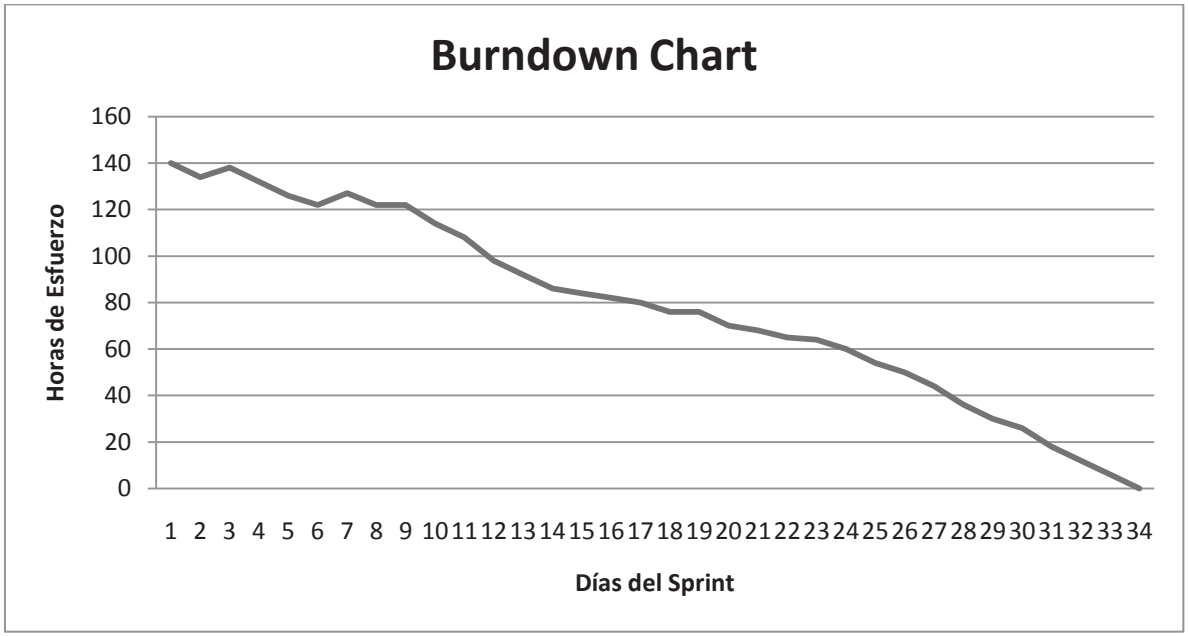
Burndown chart sprint1



Sprint 2

Elemento del *Product Backlog*	Tarea del *Sprint*	Voluntario	Estimación de Esfuerzo Inicial	Nuevo Esfuerzo Estimado (al final del día)																																
Corrección	Análisis y corrección de diseño de clases	John	20	14	18	12	6	2	0																											
	Modificación de clases (programación)		18	18	18	18	18	18	25	20	20	12	6	0																						
Registro de Ingreso de pacientes, Registro de Ingreso de familiares, Registro de Reingreso de paciente, Asignación de camas	Creación de pantalla de búsqueda de pacientes no registrados		18	18	18	18	18	18	18	18	18	18	18	14	8	2	0																			
	Creación de pantalla de registro		20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	18	16	12	12	6	4	1	0											
Registro de Reingreso de paciente, Registro de salida de pacientes, Registro de salida de familiares	Creación de pantalla de búsqueda de registros ingresados		20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	16	10	6	0							
	Creación de pantalla de salida		20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	12	6	2	0			
	Creación de pantalla de búsqueda de todos los registros		18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	12	6	0	
Pruebas	Ejecución de pruebas		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	0
Total			140	134	138	132	126	122	127	122	122	114	108	98	92	86	84	82	80	76	76	70	68	65	64	60	54	50	44	36	30	26	18	12	6	0

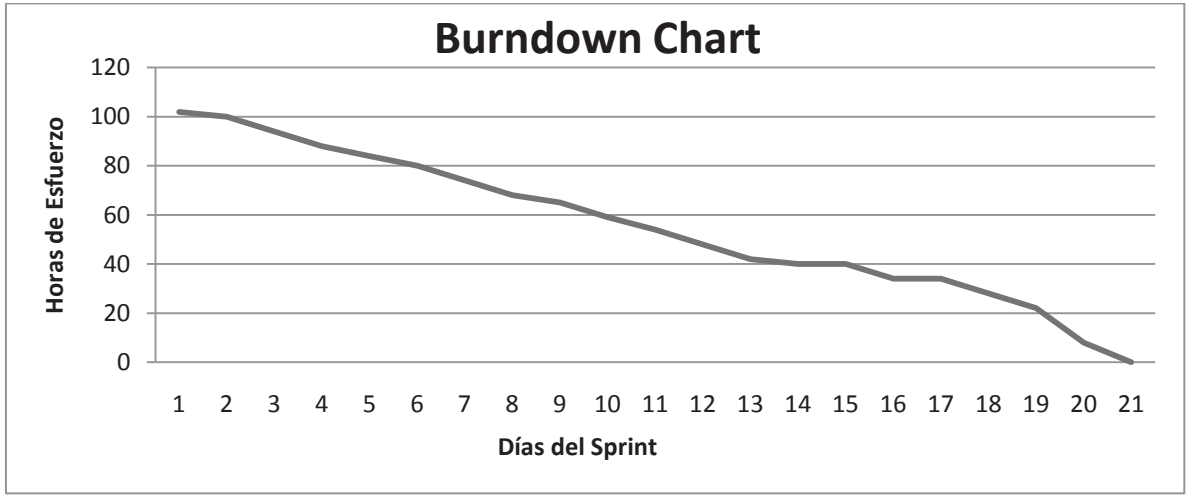
Burndown chart sprint 2



Sprint 3

Elemento del *Product Backlog*	Tarea del *Sprint*	Voluntario	Estimación de Esfuerzo Inicial	Nuevo Esfuerzo Estimado (al final del día)																					
Análisis	Pruebas de impresion	John	20	18	12	6	2	0																	
Impresión de formularios de Ingresos	Generación de pantallas de impresión		20	20	20	20	20	18	12	6	3	0													
	Formato de pantalla		20	20	20	20	20	20	20	20	20	17	12	6	0										
	Impresión con Java Script		20	20	20	20	20	20	20	20	20	20	20	20	20	18	18	12	12	6	0				
	Inserción de botones para la impresión en las pantallas correspondientes		20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	6	0		
Pruebas	Ejecución de pruebas		2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	0		
Total			102	100	94	88	84	80	74	68	65	59	54	48	42	40	40	34	34	28	22	8	0		

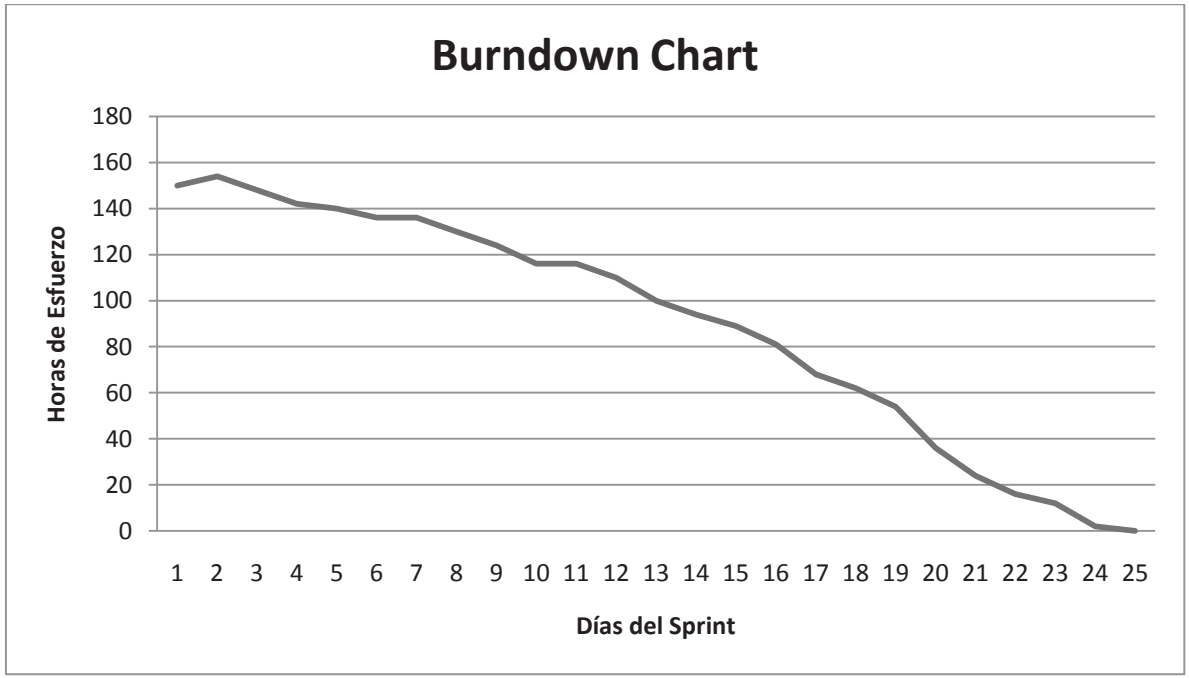
Burndown chart sprint 3



Sprint 4

Elemento del *Product Backlog*	Tarea del Sprint	Voluntario	Estimación Esfuerzo Inicial	Nuevo Esfuerzo Estimado (al final del día)																												
Generación de informes estadísticos en formato *.xls*	Creación de consultas para informes sexo y tratamiento	John	16	20	14	8	6	2	0																							
	Creación de pantalla filtro para sexo y tratamiento		12	12	12	12	12	12	14	8	2	0																				
	Creación de pantalla sexo y tratamiento *(xls)*		18	18	18	18	18	18	18	18	18	12	12	6	0																	
	Creación de consultas para informes cantón y ocupación		16	16	16	16	16	16	16	16	16	16	16	16	12	6	1	0														
	Creación de pantalla filtro para cantón y ocupación		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	5	0													
	Creación de pantalla cantón y ocupación *(xls)*		18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	10	4	0											
	Creación de consultas para informes alojamiento		16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	12	0										
	Creación de pantalla filtro para alojamiento		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	6	0									
	Creación de pantalla alojamiento *(xls)*		18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	18	12	4	0							
Pruebas	Ejecución de pruebas		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	2	0					
Total			150	154	148	142	140	136	136	130	124	116	116	110	100	94	89	81	68	62	54	36	24	16	12	2	0					

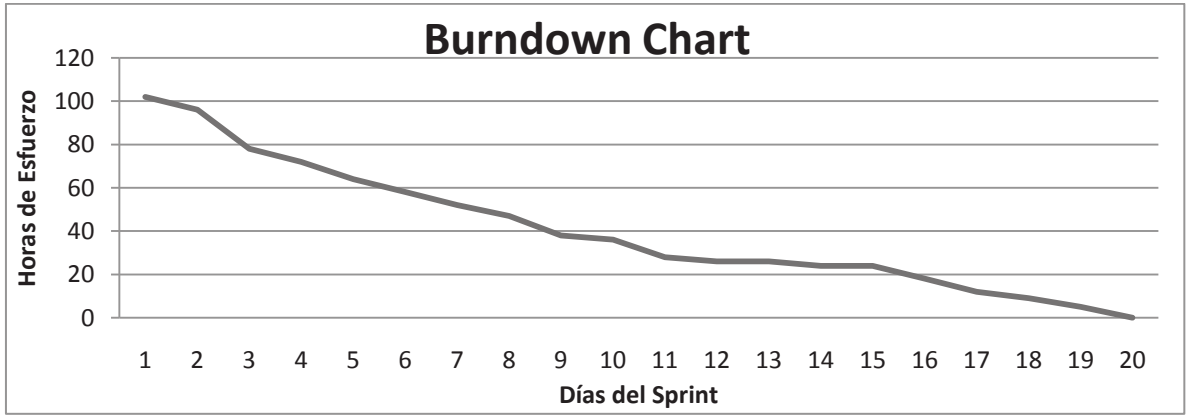
Burndown chart sprint 4



Sprint 5

Elemento del *Product Backlog*	Tare del *Sprint*	Voluntario	Estimación de Esfuerzo Inicial	Nuevo Esfuerzo Estimado (al final del día)																		
Cambios	Quitar historia clinica	John	12	6	0																	
Crear varios usuarios para el sistema y otorgar permisos	Crear los usuarios y roles		6	6	0																	
	Definir las acciones para cada rol		12	12	6	0																
	Definir los roles de cada usuarios		6	6	6	6	0															
	Crear seguridad para administrador		20	20	20	20	18	12	6	1	0											
	Crear seguridad para reportes		20	20	20	20	20	20	20	20	12	10	2	0								
	Correcciones finales debido a los roles		20	20	20	20	20	20	20	20	20	20	20	20	20	18	18	12	6	3	0	
Pruebas	Ejecución de pruebas		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	0
Total			102	96	78	72	64	58	52	47	38	36	28	26	26	24	24	18	12	9	5	0

Burndown chart sprint 5



Formato de encuesta.

Se realizó una encuesta con preguntas abiertas y tomando en cuenta la decisión de todos los involucrados, además se aclaró dudas con respecto a cómo el sistema puede ayudar a satisfacer sus necesidades. A continuación las preguntas más relevantes:

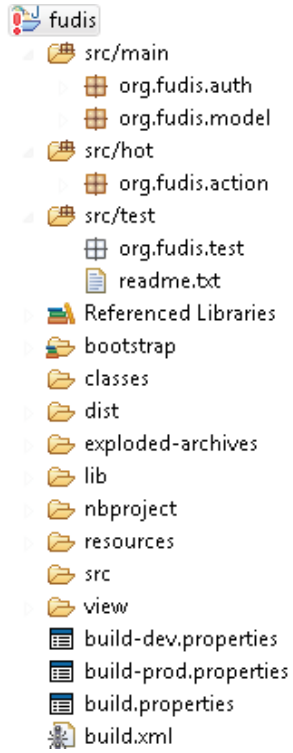
- ¿Cuál es la situación actual?
- ¿Cuáles son las necesidades que tiene la fundación y que esperan solucionar con un sistema informático?
- ¿Cómo se realizan los procesos que se involucran en el sistema?
- ¿Qué información es necesario almacenar en el sistema?
- ¿Quiénes manejaran el sistema?
- ¿Cuántos usuarios manejaran el sistema?
- ¿Cuánto tiempo se puede dar como plazo máximo para la entrega?
- Se necesitó además indagar en las características de cada uno de los procesos para ir obteniendo la mayor cantidad de detalles y así poder comenzar con el proyecto.

Código Fuente

Debido a la extensión del código fuente no es posible colocarlo en su totalidad, por eso a continuación se encuentra la explicación de la organización del código del proyecto y el flujo completo para la creación de la entidad Provincia para que sirva como ejemplo, cabe recordar que el proyecto ha sido generado con la herramienta *Seamgen* por lo que se ha conservado la estructura del creada por el mismo además código respeta las convenciones de *Sun*[8].

La captura de pantalla del proyecto abierto en el *IDE Eclipse* muestra la estructura que incluye las siguientes carpetas y *packages*:

- *src/main*: *Pojos* utilizados para el mapeo con ayuda de las anotaciones.
- *src/hot*: Clases *action* para la integración entre las vistas y la persistencia de la base de datos.
- *src/text*: Clases para las pruebas de *seam*.
- *resources*: Archivos de configuración
- *view*: Archivos para las Vistas de la aplicación.



[8]Sun Microsystems, Code Conventions for the Java Programming Language tomado de http://www.oracle.com/technetwork/java/codeconv-138413.html

A continuación se muestra las clases y archivos que componen el *CRUD* de la entidad provincia.

- *Pojo* de provincia. Provincia.java

```
package org.fudis.model;

import java.util.LinkedList; ..

/**
 * @author John
 * @version 1.0
 * @created 27-sep-2010 18:27:02
 *  Clase para manejar la residencia de los pacientes, junto con las clases país y
 *  cantón.
 */
@Entity
@Table(name = "provincia")
public class Provincia {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(nullable = false)
	private long id;

	 @Length(min = 4, max = 30)
	@Column(nullable = false, length = 30)
	private String nombre;

	@ManyToOne
	private Pais pais;

	 @OneToMany(mappedBy = "provincia")
	private List<Canton> cantones;

	public Provincia() {
		cantones = new LinkedList<Canton>();
	}

	public long getId() {
		return id;
	}

	public void setId(long id) {
		this.id = id;
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public Pais getPais() {
		return pais;
```

```
	}

	public void setPais(Pais pais) {
		this.pais = pais;
	}

	public List<Canton> getCantones() {
		return cantones;
	}

	public void setCantones(List<Canton> cantones) {
		this.cantones = cantones;
	}
}
```

- *Action* llamado provinciaHome.

```
package org.fudis.action;

import org.fudis.model.*; …

@Name("provinciaHome")
public class ProvinciaHome extends EntityHome<Provincia> {

	@In(create = true)
	PaisHome paisHome;

	public void setProvinciaId(Long id) {
		setId(id);
	}

	public Long getProvinciaId() {
		return (Long) getId();
	}

	@Override
	protected Provincia createInstance() {
		Provincia provincia = new Provincia();
		return provincia;
	}

	public void load() {
		if (isIdDefined()) {
			wire();
		}
	}

	public void wire() {
		getInstance();
		Pais pais = paisHome.getDefinedInstance();
		if (pais != null) {
			getInstance().setPais(pais);
		}
```

```
        }

        public boolean isWired() {
                return true;
        }

        public Provincia getDefinedInstance() {
                return isIdDefined() ? getInstance() : null;
        }

        public List<Canton> getCantones() {
                return getInstance() == null ? null : new ArrayList<Canton>(
                                getInstance().getCantones());
        }
}
```

- Vista de la pantalla de detalles de una provincia. Provincia.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:s="http://jboss.com/products/seam/taglib"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:rich="http://richfaces.org/rich"
   template="layout/template.xhtml">

<ui:define name="body">

   <rich:panel>
      <f:facet name="header">Detalles de la Provincia</f:facet>

      <s:decorate id="nombre" template="layout/display.xhtml">
         <ui:define name="label">Nombre</ui:define>
         <h:outputText value="#{provinciaHome.instance.nombre}"/>
      </s:decorate>

      <s:decorate id="pais" template="layout/display.xhtml">
         <ui:define name="label">País</ui:define>
         <h:outputText value="#{provinciaHome.instance.pais.nombre}"/>
      </s:decorate>
      <div style="clear:both"/>
   </rich:panel>
   <div class="actionButtons">

      <s:button view="/ProvinciaEdit.xhtml"  id="edit"  value="Editar"/>

         <s:button view="/#{empty provinciaFrom ? 'ProvinciaList' :
         provinciaFrom}.xhtml"  id="done"  value="Aceptar"/>
    </div>
```

```
<rich:tabPanel switchType="ajax">
<rich:tab>
   <f:facet name="label">
      <h:panelGroup>
               <h:graphicImage value="/img/onetomany.gif"
                style="vertical-align:    middle; padding-right:4px;"/> Cantones
      </h:panelGroup>
   </f:facet>
<h:form styleClass="association" id="cantonesChildren">

<h:outputText value="No existen cantones asociados con esta provincia."
            rendered="#{empty provinciaHome.cantones}"/>

<rich:dataTable value="#{provinciaHome.cantones}"  var="_canton"
           rendered="#{not empty provinciaHome.cantones}"
           rowClasses="rvgRowOne,rvgRowTwo" id="cantonesTable">
      <rich:column sortBy="#{_canton.id}">
         <f:facet name="header">Id</f:facet>
         <h:outputText value="#{_canton.id}"/>
      </rich:column>
      <rich:column sortBy="#{_canton.nombre}">
         <f:facet name="header">Nombre</f:facet>
         <h:outputText value="#{_canton.nombre}"/>
      </rich:column>
      <h:column>
         <f:facet name="header">Acción</f:facet>
         <s:link id="selectcanton"
             value="Seleccionar" view="/Canton.xhtml">
            <f:param name="cantonId"
                  value="#{_canton.id}"/>
            <f:param name="cantonFrom" value="Provincia"/>
         </s:link>
      </h:column>
   </rich:dataTable>
</h:form>

<div class="actionButtons">
   <s:button
        value="Agregar cantón"   view="/CantonEdit.xhtml">
      <f:param name="provinciaId"
            value="#{provinciaHome.instance.id}"/>
      <f:param name="cantonFrom" value="Provincia"/>
   </s:button>
</div>
</rich:tab>
</rich:tabPanel>
</ui:define>
</ui:composition>
```

- Página de configuración de la vista anterior. Provincia.page.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<page xmlns="http://jboss.com/products/seam/pages"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://jboss.com/products/seam/pages
      http://jboss.com/products/seam/pages-2.2.xsd"  login-required="true">
   <restrict>#{s:hasRole('Administrador')}</restrict>
   <param name="provinciaFrom"/>
   <param name="provinciaId" value="#{provinciaHome.provinciaId}"/>
   <param name="paisFrom"/>
   <param name="paisId" value="#{paisHome.paisId}"/>
</page>
```

- Vista para la edición de de una provincia. ProvinciaEdit.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:s="http://jboss.com/products/seam/taglib"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:a="http://richfaces.org/a4j"
   xmlns:rich="http://richfaces.org/rich"
   template="layout/template.xhtml">

<ui:define name="body">
   <h:form id="provincia" styleClass="edit">
      <rich:panel>
<f:facet name="header">#{provinciaHome.managed ? 'Editar' : 'Agregar'} Provincia </f:facet>

         <s:decorate id="nombreField" template="layout/edit.xhtml">
            <ui:define name="label">Nombre</ui:define>
            <h:inputText id="nombre" required="true"  value="#{provinciaHome.instance.nombre}">
                     <s:validate/>
            </h:inputText>
         </s:decorate>

          <s:decorate id="paisField" template="layout/edit.xhtml">
              <ui:define name="label">País</ui:define>
              <h:selectOneMenu required="true" value="#{provinciaHome.instance.pais}">
                     <s:selectItems var="_pais" value="#{paisList.resultList}"  label="#{_pais.nombre}"
                     noSelectionLabel="-- Seleccione --"/>
                      <s:convertEntity/>
               </h:selectOneMenu>
           </s:decorate>

         <div style="clear:both">
            <span class="required">*</span>
            Información requerida
         </div>
      </rich:panel>

      <div class="actionButtons">

         <h:commandButton id="save"
                   value="Grabar"
                  action="#{provinciaHome.persist}"
                disabled="#{!provinciaHome.wired}"
                rendered="#{!provinciaHome.managed}"/>

         <h:commandButton id="update"
```

```
                    value="Grabar"
                   action="#{provinciaHome.update}"
                  rendered="#{provinciaHome.managed}"/>

        <h:commandButton id="delete"
                    value="Eliminar"
                   action="#{provinciaHome.remove}"
                immediate="true"
                  rendered="#{provinciaHome.managed}"/>

        <s:button id="cancelEdit"
              value="Cancelar"
              propagation="end"
               view="/Provincia.xhtml"
               rendered="#{provinciaHome.managed}"/>

        <s:button id="cancelAdd"
              value="Cancelar"
              propagation="end"
               view="/#{empty provinciaFrom ? 'ProvinciaList' : provinciaFrom}.xhtml"
               rendered="#{!provinciaHome.managed}"/>

      </div>
    </h:form>

<rich:tabPanel switchType="ajax">
    <rich:tab label="Cantones">
      <h:form styleClass="association" id="cantonesChildren">
        <h:outputText value="No existen cantones asociadas a esta provincia."
                  rendered="#{empty provinciaHome.cantones}"/>

        <rich:dataTable value="#{provinciaHome.cantones}"
                    var="_canton"    rendered="#{not empty provinciaHome.cantones}"
                     rowClasses="rvgRowOne,rvgRowTwo"   id="cantonesTable">
          <rich:column sortBy="#{_canton.id}">
            <f:facet name="header">Id</f:facet>
            <h:outputText value="#{_canton.id}"/>
          </rich:column>
          <rich:column sortBy="#{_canton.nombre}">
            <f:facet name="header">Nombre</f:facet>
            <h:outputText value="#{_canton.nombre}"/>
          </rich:column>
        </rich:dataTable>
      </h:form>

      <f:subview rendered="#{provinciaHome.managed}" id="cantones">
      <div class="actionButtons">
        <s:button id="addcanton"  value="Agregar cantón"  view="/CantonEdit.xhtml"
              propagation="none">
           <f:param name="provinciaId"
                value="#{provinciaHome.instance.id}"/>
          <f:param name="cantonFrom" value="Provincia"/>
        </s:button>
      </div>
      </f:subview>
    </rich:tab>
</rich:tabPanel>
</ui:define>
</ui:composition>
```

- Archivo de configuración de la vista de edición. ProvinciaEdit.page.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns="http://jboss.com/products/seam/pages"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.com/products/seam/pages
http://jboss.com/products/seam/pages-2.2.xsd"
    no-conversation-view-id="/ProvinciaList.xhtml"
    login-required="true">

  <restrict>#{s:hasRole('Administrador')}</restrict>
  <begin-conversation join="true" flush-mode="MANUAL"/>
  <action execute="#{provinciaHome.wire}"/>

  <param name="provinciaFrom"/>
  <param name="provinciaId" value="#{provinciaHome.provinciaId}"/>
  <param name="paisFrom"/>
  <param name="paisId" value="#{paisHome.paisId}"/>

  <navigation from-action="#{provinciaHome.persist}">
    <rule if-outcome="persisted">
      <end-conversation/>
      <redirect view-id="/Provincia.xhtml"/>
    </rule>
  </navigation>

  <navigation from-action="#{provinciaHome.update}">
    <rule if-outcome="updated">
      <end-conversation/>
      <redirect view-id="/Provincia.xhtml"/>
    </rule>
  </navigation>

  <navigation from-action="#{provinciaHome.remove}">
    <rule if-outcome="removed">
      <end-conversation/>
      <redirect view-id="/ProvinciaList.xhtml"/>
    </rule>
  </navigation>
</page>
```

- Clase que devuelve la lista de las provincias almacenadas en la base de datos. ProvinciaList.java

```
package org.fudis.action;

import java.util.Arrays; …

@Name("provinciaList")
public class ProvinciaList extends EntityQuery<Provincia> {

        private static final String EJBQL = "select provincia from Provincia provincia";

        private static final String[] RESTRICTIONS = {
                "lower(provincia.nombre) like lower(concat(#{provinciaList.provincia.nombre},'%'))", };

        private Provincia provincia = new Provincia();

        public ProvinciaList() {
                setEjbql(EJBQL);
                setRestrictionExpressionStrings(Arrays.asList(RESTRICTIONS));
                setMaxResults(25);
                setOrderColumn("nombre");
```

```
        }

        public Provincia getProvincia() {
                return provincia;
        }
}
```

- Vista para la búsqueda y listado de las provincias. ProvinciaList.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:s="http://jboss.com/products/seam/taglib"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:rich="http://richfaces.org/rich"
   template="layout/template.xhtml">

<ui:define name="body">
   <h:form id="provinciaSearch" styleClass="edit">
      <rich:simpleTogglePanel label="Buscador de Provincias" switchType="ajax">
         <s:decorate template="layout/display.xhtml">
            <ui:define name="label">Nombre</ui:define>
            <h:inputText id="nombre" value="#{provinciaList.provincia.nombre}"/>
         </s:decorate>
      </rich:simpleTogglePanel>

      <div class="actionButtons">
         <h:commandButton id="search" value="Buscar" action="/ProvinciaList.xhtml"/>
         <s:button id="reset" value="Limpiar" includePageParams="false"/>
      </div>
   </h:form>

   <rich:panel>
      <f:facet name="header">Resultado de búsqueda (#{empty provinciaList.resultList ? 0 :
(provinciaList.paginated ? provinciaList.resultCount : provinciaList.resultList.size)})</f:facet>
   <div class="results" id="provinciaList">

   <h:outputText value="No existen resultados."
            rendered="#{empty provinciaList.resultList}"/>

   <rich:dataTable id="provinciaList"    var="_provincia"   value="#{provinciaList.resultList}"
         rendered="#{not empty provinciaList.resultList}">
      <h:column>
         <f:facet name="header">
            <ui:include src="layout/sort.xhtml">
               <ui:param name="entityList" value="#{provinciaList}"/>
               <ui:param name="propertyLabel" value="Nombre"/>
               <ui:param name="propertyPath" value="provincia.nombre"/>
            </ui:include>
         </f:facet>
         <h:outputText value="#{_provincia.nombre}"/>
      </h:column>
      <h:column>
         <f:facet name="header">
            <ui:include src="layout/sort.xhtml">
```

```
                <ui:param name="entityList" value="#{provinciaList}"/>
                <ui:param name="propertyLabel" value="País"/>
                <ui:param name="propertyPath" value="provincia.pais.nombre"/>
              </ui:include>
            </f:facet>
            <h:outputText value="#{_provincia.pais.nombre}"/>
          </h:column>
          <rich:column styleClass="action">
            <f:facet name="header">Action</f:facet>
            <s:link view="/#{empty from ? 'Provincia' : from}.xhtml"
                  value="#{empty from ? 'Ver' : 'Seleccionar'}"
                  propagation="#{empty from ? 'none' : 'default'}"
                  id="provinciaViewId">
              <f:param name="provinciaId"   value="#{_provincia.id}"/>
            </s:link>
            #{' '}
            <s:link view="/ProvinciaEdit.xhtml"   value="Editar"
                    propagation="none"     id="provinciaEdit"    rendered="#{empty from}">
                    <f:param name="provinciaId"    value="#{_provincia.id}"/>
            </s:link>
          </rich:column>
        </rich:dataTable>

        </div>
        </rich:panel>

        <div class="tableControl">

          <s:link view="/ProvinciaList.xhtml"     rendered="#{provinciaList.previousExists}"
                value="#{messages.left}#{messages.left} Principio"   id="firstPage">
               <f:param name="firstResult" value="0"/>
          </s:link>

          <s:link view="/ProvinciaList.xhtml"    rendered="#{provinciaList.previousExists}"
                value="#{messages.left} Anterior"   id="previousPage">
                <f:param name="firstResult"  value="#{provinciaList.previousFirstResult}"/>
          </s:link>

          <s:link view="/ProvinciaList.xhtml"    rendered="#{provinciaList.nextExists}"
                value="Siguiente #{messages.right}"  id="nextPage">
                <f:param name="firstResult"    value="#{provinciaList.nextFirstResult}"/>
          </s:link>

          <s:link view="/ProvinciaList.xhtml"     rendered="#{provinciaList.nextExists}"
                value="Final #{messages.right}#{messages.right}"  id="lastPage">
              <f:param name="firstResult"  value="#{provinciaList.lastFirstResult}"/>
          </s:link>

        </div>

        <s:div styleClass="actionButtons" rendered="#{empty from}">
          <s:button view="/ProvinciaEdit.xhtml"      id="create"
            propagation="none"    value="Crear provincia">
            <f:param name="provinciaId"/>
          </s:button>
        </s:div>

    </ui:define>
```

```
</ui:composition>
```

- Archivo de configuración para la vista de listado de provincias. ProvinciaList.page.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns="http://jboss.com/products/seam/pages"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://jboss.com/products/seam/pages
      http://jboss.com/products/seam/pages-2.2.xsd"    login-required="true">

   <restrict>#{s:hasRole('Administrador')}</restrict>

   <param name="firstResult" value="#{provinciaList.firstResult}"/>
   <param name="sort" value="#{provinciaList.orderColumn}"/>
   <param name="dir" value="#{provinciaList.orderDirection}"/>

   <param name="from"/>
   <param name="nombre" value="#{provinciaList.provincia.nombre}" validateModel="false"/>

</page>
```

A continuación se presenta las clases y archivos para generar el informe de enfermedades según el sexo del paciente.

- Clase para crear el listado de las enfermedades

  ReporteEnfermedadSexo.java

```
package org.fudis.action;

import java.util.Arrays ; …

@Name("reporteEnfermedadSexoList")
public class ReporteEnfermedadSexoList extends EntityQuery<EnfermedadRegistrada> {

        private static final String EJBQL = "select count(DISTINCT paciente.id),
        enfermedad.nombre, paciente.genero from EnfermedadRegistrada
        enfermedadRegistrada join enfermedadRegistrada.registro registro join
        enfermedadRegistrada.enfermedad enfermedad  join registro.paciente paciente";

        private static final String[] RESTRICTIONS = {
        "registro.fechaIngreso >= #{reporteEnfermedadSexoList.registro.fechaIngresoDesde}",
        "registro.fechaIngreso <= #{reporteEnfermedadSexoList.registro.fechaIngresoHasta}",
        "registro.edadIngreso >= #{reporteEnfermedadSexoList.registro.edadIngresoDesde}",
        "registro.edadIngreso <= #{reporteEnfermedadSexoList.registro.edadIngresoHasta}",
        };

        private Registro registro = new Registro();
        private Paciente paciente = new Paciente();
        private Enfermedad enfermedad = new Enfermedad();
        private EnfermedadRegistrada enfermedadRegistrada = new EnfermedadRegistrada();

        public ReporteEnfermedadSexoList() {
                setEjbql(EJBQL);
```

```
                setRestrictionExpressionStrings(Arrays.asList(RESTRICTIONS));
                setOrderColumn("paciente.genero");
                setOrderDirection("desc");
                setGroupBy("enfermedad.id, paciente.genero");
        }

        @Transient
        public Registro getRegistro() {
                return registro;
        }

        @Transient
        public Paciente getPaciente() {
                return paciente;
        }

        public EnfermedadRegistrada getEnfermedadRegistrada() {
                return enfermedadRegistrada;
        }

        @Transient
        public Enfermedad getEnfermedad() {
                return enfermedad;
        }
}
```

- Pantalla para generar el reporte según la fecha y edad indicada. ReporteEnfermedadSexo.xhtml

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
   xmlns:s="http://jboss.com/products/seam/taglib"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:a="http://richfaces.org/a4j"
   xmlns:rich="http://richfaces.org/rich"
   template="/layout/template.xhtml">

<ui:define name="body">

   <h:form id="reporteEnfermedadSexo" styleClass="edit">

                <h2>Reporte de las enfermedades según el sexo del paciente </h2>
      <rich:panel>
         <s:decorate id="fechaIngresoField" template="/layout/edit.xhtml">
            <ui:define name="label">Fecha ingreso</ui:define>
            <h:outputText value="Desde: " />
            <rich:calendar id="fechaIngresoDesde"
                  value="#{reporteEnfermedadSexoList.registro.fechaIngresoDesde}"
                   datePattern="dd/MM/yyyy" />
            <h:outputText value="Hasta: " />
            <rich:calendar  id="fechaIngresoHasta"
                   value="#{reporteEnfermedadSexoList.registro.fechaIngresoHasta}"
                   datePattern="dd/MM/yyyy" />
```

```
        </s:decorate>

        <s:decorate id="sueldoField" template="/layout/edit.xhtml">
            <ui:define name="label">Edad paciente</ui:define>
            <h:outputText value="Desde: " />
                    <h:inputText id="edadDesde"
                    value="#{reporteEnfermedadSexoList.registro.edadIngresoDesde}" />
                     <h:outputText value="Hasta: " />
                     <h:inputText id="edadHasta"
                    value="#{reporteEnfermedadSexoList.registro.edadIngresoHasta}" />
        </s:decorate>

        <div style="clear:both"/>
    </rich:panel>

    <div class="actionButtons">
         <h:commandButton id="search" value="Buscar"
         action="/reporte/ReporteEnfermedadSexo.xhtml"/>
        <s:button id="reset" value="Limpiar" includePageParams="false"/>
    </div>
</h:form>

         <h:form id="reporte">
         <h:outputText value="No existen resultados."
          rendered="#{empty reporteEnfermedadSexoList.resultList}"/>

          <rich:dataTable id="reporteEnfermedadSexoList"
           var="_enfermedadRegistrada"
           value="#{reporteEnfermedadSexoList.resultList}"
           rendered="#{not empty reporteEnfermedadSexoList.resultList}">


    <h:column>
        <f:facet name="header">
                <ui:include src="/layout/sort.xhtml">
                    <ui:param name="propertyLabel" value="Cantidad"/>
                </ui:include>
        </f:facet>
        <h:outputText value="#{_enfermedadRegistrada[0]}"/>
    </h:column>

    <h:column>
        <f:facet name="header">
            <ui:include src="/layout/sort.xhtml">
                <ui:param name="entityList" value="#{reporteList}"/>
                <ui:param name="propertyLabel" value="Enfermedad"/>
                <ui:param name="propertyPath"
                 value="enfermedadRegistrada.enfermedad.nombre"/>
            </ui:include>
        </f:facet>
        <h:outputText value="#{_enfermedadRegistrada[1]}"/>
    </h:column>

       <h:column>
        <f:facet name="header">
            <ui:include src="/layout/sort.xhtml">
                <ui:param name="entityList" value="#{reporteList}"/>
                <ui:param name="propertyLabel" value="Sexo"/>
```

```
                <ui:param name="propertyPath"
                    value="enfermedadRegistrada.registro.paciente.genero.label"/>
              </ui:include>
            </f:facet>
            <h:outputText value="#{_enfermedadRegistrada[2]}"/>
          </h:column>
      </rich:dataTable>

       <s:button value="Exportar a Excel" action="/reporte/ExcelEnfermedadSexo.xhtml"
            rendered="#{not empty reporteEnfermedadSexoList.resultList}"/>
      </h:form>
</ui:define>
</ui:composition>
```

- Archivo de configuración de la pantalla del reporte. ReporteEnfermedadSexo.page.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns="http://jboss.com/products/seam/pages"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://jboss.com/products/seam/pages
      http://jboss.com/products/seam/pages-2.2.xsd" login-required="true">

  <param name="firstResult" value="#{reporteEnfermedadSexoList.firstResult}"/>
  <param name="sort" value="#{reporteEnfermedadSexoList.orderColumn}"/>
  <param name="dir" value="#{reporteEnfermedadSexoList.orderDirection}"/>
  <param name="logic" value="#{reporteEnfermedadSexoList.restrictionLogicOperator}"/>

  <param name="from"/>
  <param name="fechaSalida" value="#{reporteEnfermedadSexoList.registro.fechaSalida}"/>
  <param name="fechaIngresoDesde"
          value="#{reporteEnfermedadSexoList.registro.fechaIngresoDesde}"
          converterId="org.jboss.seam.ui.DateTimeConverter"/>
  <param name="fechaIngresoHasta"
          value="#{reporteEnfermedadSexoList.registro.fechaIngresoHasta}"
          converterId="org.jboss.seam.ui.DateTimeConverter"/>
  <param name="edadDesde"
          value="#{reporteEnfermedadSexoList.registro.edadIngresoDesde}"/>
 <param name="edadHasta"
          value="#{reporteEnfermedadSexoList.registro.edadIngresoHasta}"/>
</page>
```

- Archivo de generación del reporte en formato .xls. ExcelEnfermedadSexo.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
          xmlns:ui="http://java.sun.com/jsf/facelets"
          xmlns:e="http://jboss.com/products/seam/excel"
          xmlns:f="http://java.sun.com/jsf/core">
<e:workbook type="#{exportType}">
          <e:worksheet name="Registros" var="_enfermedadRegistrada
          value="#{reporteEnfermedadSexoList.resultList}" ">
          <e:header>
```

```
                <f:facet name="center">
                    <e:cell style="xls-wrap:true">
                      FUNDACION FUDIS.
                      Fecha: #date#
                      Reporte de enfermedades por el sexo del paciente.
                   </e:cell>
                </f:facet>
        </e:header>
        <e:column style="xls-column-autosize: true">
                <f:facet name="header">
                        <e:cell value="Cantidad" style="xls-font: blue bold 12 Verdana;
                        xls-border:thin black; xls-alignment: centre; xls-background-color:
                        GREY_25_PERCENT"/>
                </f:facet>
                        <e:cell  value="#{_enfermedadRegistrada[0]}" forceType="text"
                                style="xls-border:thin black; xls-alignment: centre"/>
                </e:column>

        <e:column style="xls-column-autosize: true">
                <f:facet name="header">
                        <e:cell value="Enfermedad"  style="xls-font: blue bold 12
                        Verdana; xls-border:thin black; xls-alignment: centre; xls-
                        background-color: GREY_25_PERCENT"/>
                </f:facet>
                        <e:cell  value="#{_enfermedadRegistrada[1]}" forceType="text"
                                style="xls-border:thin black; xls-alignment: centre"/>
                </e:column>

        <e:column style="xls-column-autosize: true">
                <f:facet name="header">
                        <e:cell value="Sexo"  style="xls-font: blue bold 12 Verdana; xls-
                        border:thin black; xls-alignment: centre; xls-background-color:
                        GREY_25_PERCENT"/>
                </f:facet>
                        <e:cell value="#{_enfermedadRegistrada[2]}" forceType="text"
                                style="xls-border:thin black; xls-alignment: centre"/>
                </e:column>
        </e:worksheet>
</e:workbook>
</html>
```

- Archivo de configuración para el informe generado de Excel. ExcelEnfermedadSexo.page.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns="http://jboss.com/products/seam/pages"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://jboss.com/products/seam/pages
    http://jboss.com/products/seam/pages-2.2.xsd" login-required="true">

  <param name="from"/>
  <param name="fechaSalida"
       v alue="#{reporteEnfermedadSexoList.registro.fechaSalida}"/>
  <param name="fechaIngresoDesde"
       value="#{reporteEnfermedadSexoList.registro.fechaIngresoDesde}"
       converterId="org.jboss.seam.ui.DateTimeConverter"/>
  <param name="fechaIngresoHasta"
        value="#{reporteEnfermedadSexoList.registro.fechaIngresoHasta}"
       converterId="org.jboss.seam.ui.DateTimeConverter"/>
  <param name="edadDesde"
       value="#{reporteEnfermedadSexoList.registro.edadIngresoDesde}"/>
  <param name="edadHasta"
       value="#{reporteEnfermedadSexoList.registro.edadIngresoHasta}"/>
 </page>
```

Manual de Usuario

Sistema de Registro de Pacientes, Fundación FUDIS

Manual de Usuario.

El sistema de registro necesita un explorador web para su utilización, puede utilizar el que sea de su preferencia aunque se recomienda el uso de *Mozilla Firefox* 3.6.

Para ingresar abra el navegador web y acceda a la dirección de la aplicación (esta puede variar debido a las configuraciones de red de la fundación), Usted podrá ver la página de inicio del sistema.

- **Ingresar al Sistema:** Haga clic en Ingresar y podrá observar el formulario para donde debe escribir su usuario y contraseña, a continuación debe hacer clic en Aceptar.

Mientras que este registrado en el sistema Usted podrá observar el panel superior que consta de los siguientes elementos: El título FUDIS; el enlace de Inicio para regresar a la página principal en cualquier momento; el nombre del usuario que ingresó al sistema; el botón salir para abandonar el sistema y según el usuario con el que se ha ingresado se mostraran las opciones: Administración, Registro y Reportes para los usuarios con permisos de administrador, y solo Reportes para los usuarios con permisos de Generador de reportes.

- **Cambiar contraseña**: Haga clic en el nombre del usuario que se encuentra en el panel superior y se desplegará el menú que permite acceder a la página para cambiar la contraseña. Se debe ingresar la clave anterior y la nueva clave con confirmación. Luego clic en guardar.

Administración: El primer menú del panel superior permite la administración de todos los elementos del sistema, es decir la creación, edición, consulta y borrado, usted puede acceder solo si tiene permisos de administrador. La administración de los elementos tiene un proceso similar para todos, y por eso se demostrara tres ejemplos: país, usuario y paciente.

- **Administración de País**: En el menú Administración ingrese a País y se mostrará el cuadro para realizar la búsqueda de países por su nombre. Además se tiene el botón para crear un nuevo país y los enlaces para ver y editar sus detalles.

En la pantalla de crear únicamente deberá llenar los datos que se piden y hacer clic en guardar, luego de esto se mostrará la pantalla de detalles del país

En la página de detalles se muestran los datos del país que ha sido guardado con éxito, también desde aquí se puede editar y agregar provincias.

La pantalla de editar muestra los datos que pueden ser cambiados además de permitir eliminar el país.

- **Administración de Usuarios**: En el menú Administración debe hacer clic en Usuarios. Esta pantalla muestra la lista de usuarios y permite realizar la búsqueda por sus nombres. Muestra además el botón para crear uno nuevo; ver y editar sus detalles y cambiar la contraseña.

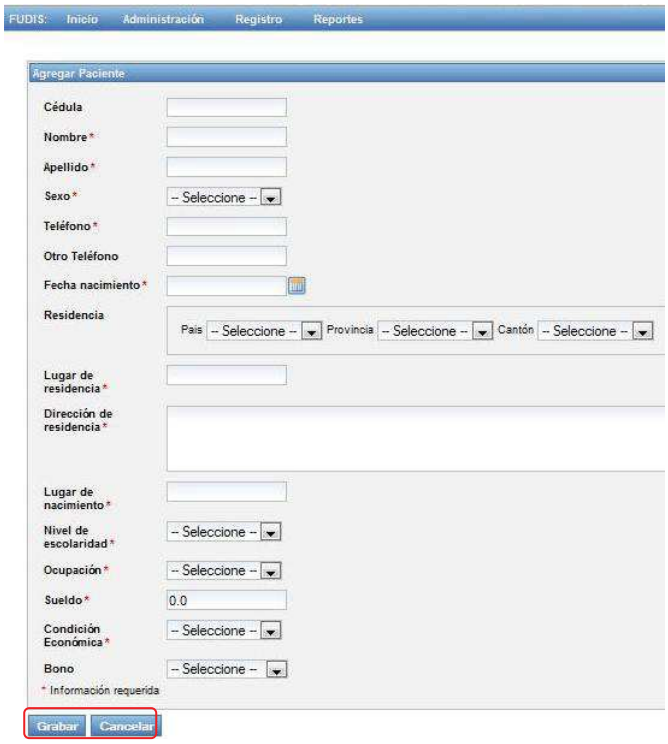
Al crear un nuevo usuario se necesita nombre y contraseña pero además se debe asignar el rol, para haga clic en Asignar Roles y aparecerá la lista para asignarlos al usuario.

La pantalla de detalle muestra el usuario y sus datos. Desde aquí también puede editar el usuario o aceptar y regresar al buscador de usuarios.

En la página de edición se puede cambiar los datos del usuario y también eliminarlo por completo.

- **Administración de paciente:**

Ya que es igual que la administración de usuario y país en este manual solo se muestra las pantallas de creación y de detalles. Para crear un nuevo paciente se debe llenar todo el formulario, incluyendo la residencia (país, provincia y cantón).



Luego de grabar se puede observar la página de detalles. En donde además se pueden ver los registros que ha tenido el paciente.

- **Registros**

El menú de registro contiene las opciones para listar los registros existentes y registrar los ingresos y salidas de los pacientes

- **Listar Registros:** El enlace de Todos los registros permite la búsqueda de los mismos de acuerdo a un rango de fechas según su ingreso.

- **Registrar entrada de paciente:** Para crear un ingreso haga clic en Registrar Ingreso ubicado en el menú Registro. Aquí se muestra la lista de pacientes para realizar el ingreso mediante el enlace registrar. Si es un nuevo paciente debe primero crearlo con el botón de crear paciente.

Al momento de hacer clic en registrar se muestra la pantalla del registro en donde se debe seleccionar la fecha de ingreso con el calendario de ayuda y además las enfermedades, camas y observaciones con los enlaces en rojo.

Haciendo clic en cada uno de los enlaces para “asignar” se muestran las siguientes ventanas para añadir los diferentes ítems al registro.

Asignar enfermedades: se muestra la lista en donde se puede elegir la enfermedad, el tratamiento y el hospital.

Estos atributos pueden dejarlos en blanco para ser llenados posteriormente.

Asignar familiar: la ventana muestra dos opciones: en la parte superior se puede buscar un familiar existente y luego de escoger el parentesco se puede agregar al registro; la segunda opción es crear un nuevo familiar llenando el formulario de la parte inferior.

Asignar camas: en esta vista se muestran las camas que no se encuentran ocupadas por otros pacientes, simplemente tiene que escoger las camas que va a utilizar el paciente y sus familiares.

Asignar Observación: la ventana muestra dos opciones: en la parte superior se puede buscar una observación existente para agregarla o puede crear una nueva observación llenando el formulario de la parte inferior.

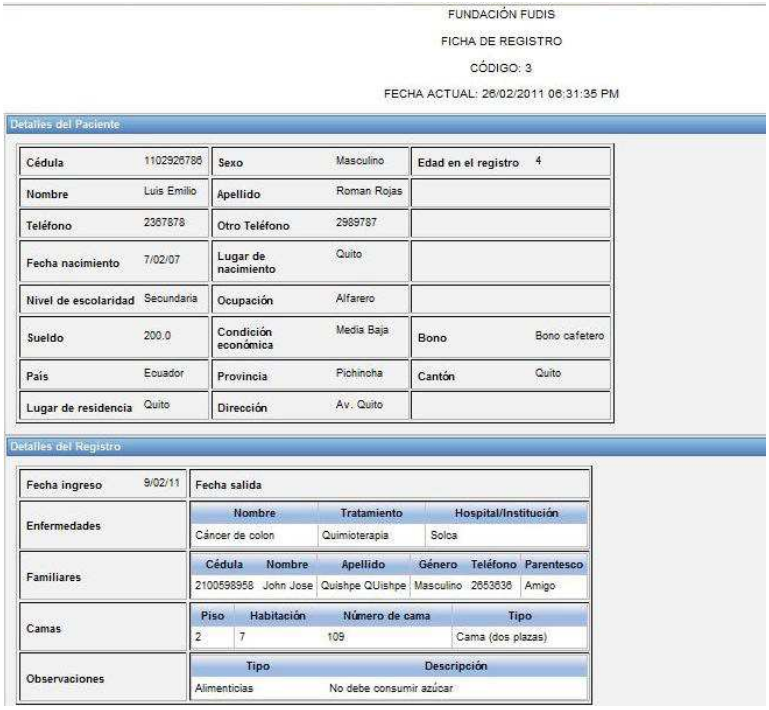
La pantalla de detalle del registro presenta los datos del mismo y en las pestañas inferiores se muestra las enfermedades, familiares, camas y observaciones.

Registrar Salida: En el menú Registro ubicado en el panel superior seleccione Registrar salida, luego busque el registro por cualquiera de los campos indicados y haga clic en Salida.

Una vez haga clic en Salida se mostrará el registro almacenado en donde debe ingresar la fecha de salida y las observaciones en caso de ser necesario.

Imprimir Registro: Al momento de realizar el ingreso o la salida de un paciente se muestra la pantalla de detalles del registro, en el que existe el botón para imprimir.

Deberá escoger la impresora y enviar a imprimir. Además se muestra la vista previa de la siguiente forma:



FUNDACIÓN FUDIS

FICHA DE REGISTRO

CÓDIGO: 3

FECHA ACTUAL: 26/02/2011 06:31:35 PM

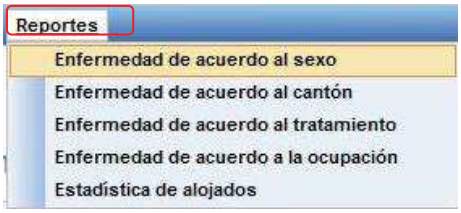
Detalles del Paciente

Cédula	1102926786	Sexo	Masculino	Edad en el registro	4
Nombre	Luis Emilio	Apellido	Roman Rojas		
Teléfono	2367878	Otro Teléfono	2989787		
Fecha nacimiento	7/02/07	Lugar de nacimiento	Quito		
Nivel de escolaridad	Secundaria	Ocupación	Alfarero		
Sueldo	200.0	Condición económica	Media Baja	Bono	Bono cafetero
País	Ecuador	Provincia	Pichincha	Cantón	Quito
Lugar de residencia	Quito	Dirección	Av. Quito		

Detalles del Registro

Fecha ingreso	9/02/11	Fecha salida				
Enfermedades	Nombre	Tratamiento	Hospital/Institución			
	Cáncer de colon	Quimioterapia	Solca			
Familiares	Cédula	Nombre	Apellido	Género	Teléfono	Parentesco
	2100598958	John Jose	Quishpe QUishpe	Masculino	2653636	Amigo
Camas	Piso	Habitación	Número de cama	Tipo		
	2	7	109	Cama (dos plazas)		
Observaciones	Tipo	Descripción				
	Alimenticias	No debe consumir azúcar				

- **Reportes:** El menú de reportes del panel superior muestra las opciones para generar todos los reportes de los registros según diferentes aspectos.



- **Reportes de acuerdo al sexo del paciente:** Los reportes de acuerdo al sexo, al cantón, al tratamiento y a la ocupación son similares, ya que necesitan que se seleccione el rango de la fecha y el rango de edad que se desea, luego se muestra el botón de exportar a Excel.

- **Reporte de alojamiento:** A diferencia de los otros reportes en este solo es necesario el rango de la fecha. Para generar se usa el botón de Exportar a Excel

Manual de Implantación

El manual de implantación se detalla para la configuración que se tiene en el servidor de la fundación FUDIS; se indican los enlaces para descargar los instaladores desde el internet pero en el CD que se adjunta se encuentra el servidor listo y los instaladores de la base de datos y de la máquina virtual de java:

- Instalar la máquina virtual de *Java*.

  Desde la página http://www.java.com/es/download/ se debe descargar el instalador de *Java 6.0*, una vez descargado ejecútelo y siga los pasos del ayudante de instalación.

  Luego necesario definir la variable de entorno *JAVA_HOME*. Para esto debe hacer clic derecho en Equipo, clic en Propiedades. Escoja Opciones avanzadas del sistema y elija la pestaña Avanzado; después haga clic en Variables de entorno y en el panel de variables del sistema haga clic en Nueva. Como nombre escriba *JAVA_HOME* y como valor escriba la ruta en donde se encuentra instalado java.

- Instalar la base de datos *Mysql 5.5*

  Descargue el instalador de *Mysql* desde la siguiente página http://www.mysql.com/downloads/mysql, una vez descargado siga el ayudante de instalación y luego ejecute el ayudante de configuración de la instancia. Escoja las siguientes características:

  - *Server Machine*
  - *Transactional Database Only*
  - Elija la ubicación que desee
  - *Manual Setting* (10 usuarios)
  - *Character set utf8*
  - *Include bin directory in Windows path*
  - Elija la contraseña del usuario *root* y ejecute la instalación.

- Instalar servidor de aplicaciones *JBoss*.

  Ingrese a http://www.jboss.org/jbossas/downloads.html y descargue la versión 5.1 del servidor *JBoos*, luego descomprima la carpeta del servidor en la ubicación que prefiera. (Desde ahora se llamará a esta ubicación *JBOSS_HOME*)

  Descargue el *Mysql connector para java* desde: http://dev.mysql.com/downloads/connector/j/ y copie el archivo mysql-connector-java-5.0.8-bin.jar en: JBOSS_HOME/server/default/lib/ y renómbrelo a *mysql.jar*

  Luego copie los archivos fudis.ear y fudis-ds.xml que se encuentran en el CD a: JBOSS_HOME/server/default/deploy

- Creación de la base de datos

  Abra la consola pulsando las teclas de Windows + R y luego escriba *cmd* y pulse Aceptar. Dentro de la consola escriba: Mysql –uroot –p*clave_de_root* y pulse *Enter*. A continuación escriba los siguientes comandos:

  - *create database fudis;*
  - *grant all privileges on fudis.* to 'fudis' identified by '123456';*
  - *use fudis;*
  - *source /fudis.sql;* (fudis.sql es la ruta completa al archivo que se encuentra en el cd)
  - *insert into rol (nombre,descripcion) values ('Administrador','Permisos para administrar, Generar reportes y Registrar entradas y salidas.');*
  - *insert into rol (nombre,descripcion) values ('Generador de Reportes', 'Permisos para generar reportes.');*
  - *insert into usuario (nombre,password) values ('prueba', md5('prueba'));*
  - *insert into usuario_rol (usuario_id,rol_id) values ('1','1');*

- Ejecución del servidor

  Abra la consola pulsando las teclas de Windows + R y luego escriba *cmd* y pulse Aceptar.

  Ubíquese en JBOSS_HOME/bin usando el comando CD, luego escriba run.bat –b 0.0.0.0 y presione enter.

- Ingresar al sistema.

  Abra el explorador y en la barra de direcciones escriba http://localhost:8080/fudis Ingrese a la aplicación utilizando como usuario prueba y contraseña prueba