PROYECTO DE GRADO

SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB, PARA SOPORTAR EL PROCESO DEL SERVICIO DE CAFETERÍA OFRECIDO POR EL BIENESTAR UNIVERSITARIO.

AUTORES

JONATHAN CARRERO BONILLA

CARLOS JAVIER PRIETO ROA

DIRECTOR

ING. ENRIQUE TORRES LÓPEZ

CODIRECTOR

CONSUELO SERRANO VEGA

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniaría de Sistemas e Informática

2011

SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB, PARA SOPORTAR EL PROCESO DEL SERVICIO DE CAFETERÍA OFRECIDO POR EL BIENESTAR UNIVERSITARIO.

AUTORES

JONATHAN CARRERO BONILLA

CARLOS JAVIER PRIETO ROA

Trabajo de grado para optar el título de Ingeniero de Sistemas

DIRECTOR

ING. ENRIQUE TORRES LÓPEZ

CODIRECTOR

CONSUELO SERRANO VEGA

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniaría de Sistemas e Informática

2011

AGRADECIMIENTOS JONATHAN CARRERO BONILLA

Cuando este sueño de ser Ing. de sistemas UIS comenzó, siempre tuve el apoyo incondicional e inigualable de mis padres Jaime Omar Carrero y Esperanza Bonilla, en el transcurso de este proceso, con buenos y malos momentos nunca dudaron en estar siempre presente con una voz de aliento o una felicitación, por esta razón quiero hacer un reconocimiento especial a ellos y agradecerles por haberme dado la oportunidad de emprender este camino hacia el éxito.

Otra persona fundamental para cumplir esta meta fue mi nona Verónica Carrero Reyes, que con su forma de quererme y educarme aunque a veces alcahueta logró demostrarme que en cualquier momento ella siempre estaría presente para ayudarme, y hoy desde el cielo se que está contenta porque su nieto preferido está logrando un objetivo más.

También quiero agradecerle a mis hermanas Shirley Yasay, Jenny Carolina y Cindy Katerine y a mi sobrino Santiago por demostrarme el afecto y apoyo en todos los momentos que tuve que pasar en mi carrera universitaria, gracias a ello pude salir fácilmente de los altibajos que la vida nos pone como prueba.

En este proceso intervinieron muchas personas como mis familiares y amigos que también fueron importantísimos para generar un entorno propicio y agradable en esta etapa de mi vida, a todos y cada uno de ellos que intervinieron de alguna manera, quiero agradecerles por el apoyo brindado que fue vital para culminar satisfactoriamente la carrera de Ingeniería de Sistemas.

AGRADECIMIENTOS CARLOS JAVIER PRIETO ROA

Un agradecimiento muy grande a Dios padre todo poderoso que me ha permitido culminar este proceso de formación y quién me brindará las oportunidades de seguirme formándome profesional y personalmente.

A mis padres por todo el apoyo, confianza, respaldo y consejos que me han brinda durante estos 23 años de vida para hacer de mí una persona con sólidos valores morales y espirituales.

A todos mis amigos, especialmente a Jefferson, Julián y Sergio con quienes construí a lo largo de la carrera un lazo de amistad fuerte, el cual permitió y nos seguirá permitiendo alcanzar muchos logros personal y profesionales.

A la División de Servicios de Información de la Universidad Industrial de Santander por la oportunidad de realizar este proyecto.

Un agradecimiento muy grande al Ingeniero Enrique Torres, quien con su dedicación, paciencia, disponibilidad y actitud de colaboración nos permitió enriquecernos con sus conocimientos y poder desarrollar un proyecto de excelente calidad.

A la División de Bienestar Universitario por brindarnos la oportunidad de impactar positivamente en los proceso de la organización desarrollando un sistema de información que mejorará la satisfacción de la comunidad universitario UIS y el cual contribuirá al proceso de mejora continua de dicha división.

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

TITULO

SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB, PARA SOPORTAR EL PROCESO DEL SERVICIO DE CAFETERÍA OFRECIDO POR EL BIENESTAR UNIVERSITARIO.*

AUTORES

PRIETO ROA CARLOS JAVIER

CARRERO BONILLA JONATHAN**

PALABRAS CLAVES

Bienestar Universitario, productos, menú, software, Sistema de Información Web, Java

DESCRIPCIÓN:

La división Bienestar Universitario (DBU) cuenta con múltiples beneficios que están conformados en diferentes tópicos como lo son atención en salud, programas educativo preventivos y atención socioeconómica para la comunidad en general de la Universidad Industrial de Santander (UIS), algunos de estos beneficios son las residencias estudiantiles ,servicio de comedores, becas a hijos o conyugues de servidores entre otros, y el beneficio que vamos a mejorar con el proyecto uno de estos beneficios es el servicio de cafetería, el cual permite la venta de productos de calidad a un buen costo para los usuarios de la misma.

Teniendo en cuenta la importancia que tiene para los usuarios de cafetería y el alto impacto de este beneficio para la comunidad universitaria y que la universidad cuenta con una división de servicios de información para el desarrollo de sistemas de información, nace la necesidad de desarrollar un software orientado a soportar y controlar los procesos de cafetería, el cual facilitará y brindará apoyo para lograr una optimización de los servicios ofrecidos por la cafetería de la DBU para beneficio de toda la comunidad UIS.

*Trabajo de Investigación

**Facultad de Ingenierías Físico-Mecánicas, Escuela de Ingeniería de Sistemas. Director: Ingeniero Enrique Torres López Codirector: Consuelo Serrano Vega

SUMMARY

TITLE

*WEB INFORMATION SYSTEM TO SUPPORT THE PROCESS OF CAFETERIA SERVICE OFFERED BY THE WELFARE UNIVERSITY**

AUTHORS

PRIETO ROA CARLOS JAVIER

CARRERO BONILLA JONATHAN**

KEYWORDS

University Welfare, Products, Menu, Software, Web information system, Java.

DESCRIPTION

Welfare Division University (WDU) has many benefits that such us health care, preventive education and care programs for the Industrial University of Santander (UIS) community, some of these benefits are the residence halls, dining services, scholarships to children, and the benefit that we will improve with this project is the cafeteria service, which allows the sale of quality products at good cost for users.

Given the importance for the users of the cafeteria and the high impact of this benefit to the university community and the information services division for the development of information systems that the university has, comes the need to develop software designed to support and control the processes of the cafeteria, which will makes easier and provides support to achieve an optimization of the services offered by the cafeteria of the DBU to benefit of whole Industrial University of Santander UIS community.

* Degree Project in the modality Research

**Physique-Mechanics Sciences Department, Computer Science Engineering, Engineer Enrique Torres Lopez – Consuelo Serrano Vega

GLOSARIO TÉCNICO

API: (Application Programming Interface). Interfaz de Programación de Aplicaciones). Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que éstos prestan. En otras palabras, una API representa un interfaz de comunicación entre componentes software.

BASE DE DATOS: Se define como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información de una empresa o negocio en particular. Una base de datos es un sistema de archivos electrónico.

COMPILACIÓN: Proceso de traducción de un código fuente (escrito en un lenguaje de programación de alto nivel) a lenguaje máquina (código objeto) para que pueda ser ejecutado por la computadora.

CONTRASEÑA: (Clave o Password) Conjunto finito de caracteres limitados que forman una palabra secreta que sirve a un usuario para acceder a un determinado recurso.

DBA: (Data Base Administrator). Administrador de la Base de Datos, quien se encarga del control general del Sistema de Base de Datos.

DBA: (Data Base Administrator). Administrador de la Base de Datos, quien se encarga del control general del Sistema de Base de Datos.

EJB: Son componentes del contexto de servidor que cubren la necesidad de intermediar entre la capa web y diversos sistemas empresariales. Son una de las

API que forman parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE 5.0) de Oracle Corporation (inicialmente desarrollado por Sun Microsystems).

FRAMEWORK: Se refiere a “ambiente de trabajo, y ejecución”. En general los framework son soluciones completas que contemplan herramientas de apoyo a la construcción (ambiente de trabajo o desarrollo) y motores de ejecución (ambiente de ejecución).

HTML: (HyperText Markup Language). Lenguaje de Marcado de Hipertexto, es el lenguaje de programación predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, además de complementar el texto con objetos tales como imágenes.

INFORMIX: Es uno de los cuatro grandes plataformas o manejadores de la bases de datos (DBMS) junto con DB2 de IBM, SQL Server de Microsoft y Oracle.

INTERFAZ: En software, parte de un programa que permite el flujo de información entre un usuario y la aplicación, o entre la aplicación y otros programas o periféricos. Esa parte de un programa está constituida por un conjunto de comandos y métodos que permiten estas intercomunicaciones.

INTERNET: Es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP/IP, garantizando que las redes físicas heterogéneas que la componen funcionen como una red lógica única, de alcance mundial.

INTRANET: Es una red de ordenadores privada basada en los estándares de Internet. Las Intranets utilizan tecnologías de Internet para enlazar los recursos

informativos de una organización, desde documentos de texto a documentos multimedia, desde bases de datos legales a sistemas de gestión de documentos.
JAVA: Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

JAVA EE: (Java Platform Enterprise Edition). Anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4, es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de N niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

JBOSS DEVELOPER STUDIO (JBDS): Es un entorno de desarrollo comercial creado y actualmente desarrollado por JBoss (una división de Red Hat) y Exadel, Inc.

JBOSS HIBERNATE: Hibernate es un potente servicio de mapeo objeto/relacional de alto desempeño para consulta y persistencia. Le permite desarrollar clases persistentes siguiendo el lenguaje orientado a objetos, incluyendo la asociación, herencia, polimorfismo, composición y colecciones.

JBOSS SEAM: Es un framework que integra y unifica los distintos estándares de la plataforma Java EE 5.0. Ha sido diseñado intentado simplificar al máximo el desarrollo de aplicaciones, basando el diseño en Plain Old Java Objects (POJOs) con anotaciones. Estos componentes se usan desde la capa de persistencia hasta la de presentación, poniendo todas las capas en comunicación directa.

JDK: (Java Development Kit). Es un grupo de herramientas para el desarrollo de software provisto por Sun Microsystems Inc. Incluye las herramientas necesarias para escribir, testear, y depurar aplicaciones y applets de Java.

JPA: (Java Persistence API). Es un lenguaje de programación Java que permite a los desarrolladores gestionar datos relacionales en aplicaciones que usan Java Platform Standard Edition y Java Platform Enterprise Edition.

JPQL: (Java Persistence Query Language). Es un lenguaje de consulta orientado a objetos, definida como parte de la especificación Java Persistence API. JPQL se utiliza para hacer consultas a las entidades almacenados en una base de datos relacional. Está fuertemente inspirado en SQL y sus preguntas se asemejan a las consultas SQL en la sintaxis, pero operan contra los objetos entidad JPA y no directamente con las tablas de base de datos.

JVM: (Java Virtual Machine). Máquina virtual Java. Es un programa nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el Java bytecode), el cual es generado por el compilador del lenguaje Java.

INTRODUCCIÓN

La división de Bienestar Universitario como dependencia administrativa de la Universidad Industrial de Santander encargada de promover y contribuir al desarrollo de las personas que conforman la comunidad universitaria UIS y al mejoramiento de su calidad de vida a través del apoyo social y económico. Uno de los servicios que contribuye al cumplimiento de la función misional es el servicio de cafetería el cual ofrece productos preparados como desayunos, almuerzos, onces y productos no preparados de muy buena calidad a un bueno costo.

Debido al aumento del número de beneficiarios se hace necesario aumentar la calidad y cobertura de los servicios que ofrecen. Por tanto la universidad ha visto la necesidad de implementar un sistema de información para las cafeterías de Bienestar Universitario que permitan agilizar y optimizar tiempos y recursos en los procesos que desarrolla la misma.

Este proyecto pretender diseñar, implementar y implantar un sistema de información para las cafeterías de Bienestar Universitario que permita a los administradores gestionar eficientemente los inventarios de los puntos de venta, y a la comunidad UIS en general consultar los productos y menús disponibles en los diferentes puntos de venta.

Para la realización de este proyecto se contó con el apoyo de la División de Servicios de Información (DSI) de la Universidad Industrial de Santander, entidad encargada de la administración y desarrollo de la tecnología de información, recibiendo todo el soporte necesarios para ello, utilizando los estándares definidos la DSI para el desarrollo de software y con herramientas como Java EE 5, informix entre otros software utilizados.

En el desarrollo del proyecto se realizaron diversas reuniones con la división beneficiada para la construcción de los prototipos y producto final, proceso que se describe en el presente documento.

CAPITULO 1

1 PRESENTACIÓN DEL PROYECTO

1.1 Especificaciones Del Proyecto

1.1.1 Título

SISTEMA DE INFORMACIÓN ORIENTADO A LA WEB, PARA SOPORTAR EL PROCESO DEL SERVICIO DE CAFETERÍA OFRECIDO POR EL BIENESTAR UNIVERSITARIO.

Director del proyecto:

Nombre: Ing. Enrique Torres López

Institución: Universidad Industrial de Santander

Cargo: Profesional adscrito a la división de Servicios de Información.

Codirector del proyecto:

Nombre: Consuelo Serrano Vega

Institución: Universidad Industrial de Santander

Cargo: Profesional adscrito a Bienestar Universitario

Estudiantes:

Nombre: Jonathan Carrero Bonilla

Código: 2020560

Carrera: Ingeniería de sistemas e informática

Nombre: Carlos Javier Prieto Roa

Código: 2051694

Carrera: Ingeniería de sistemas e informática

Entidades interesadas en el proyecto:

Bienestar universitario

División de Servicios de Información

Universidad Industrial de Santander

1.1.2 Objetivos

1.1.2.1 Objetivos generales

Diseñar, implementar y poner en funcionamiento el sistema de información orientado a web para la cafetería de Bienestar Universitario que permita el manejo de los puntos de venta instalados, manejo de inventarios, control financiero de los servicios que se prestan, consultas, informes y estadísticas.

1.1.2.1 Objetivos específicos

- Recopilar la información referente a los procesos llevados a cabo en la cafetería de Bienestar Universitario de la Universidad Industrial de Santander.

- Analizar los requerimientos recopilados y traducirlas en datos, arquitectura y diseño procedimental.

- Diseñar el sistema bajo el estándar UML (Lenguaje de Modelado Unificado) que incluirá los diagramas de casos de uso, de clases y de secuencia.

- Desarrollar e implementar la aplicación software orientada a la Web que permita:
  - Atención de los puntos de ventas de los servicios que ofrecen.
  - El control de inventarios de los productos que se ofrecen.
  - Control financiero de las ventas realizadas.
  - Integración con los sistemas de información institucionales.

- Realizar pruebas necesarias y ajustes al software que garantice el correcto funcionamiento del sistema.

- Documentar y capacitar al usuario final y el personal técnico encargado del mantenimiento del sistema.

1.1.3 Justificación y definición del problema

La división de Bienestar Universitario de la Universidad Industrial de Santander cuenta con una cafetería a través de la cual se ofrece servicios de desayunos, almuerzos, ventas de varios productos comestibles, bebidas, la mayoría de ellos preparados a través de recetas previamente definidas, sin contar con un sistema de información que: a) permita la atención de los puntos de ventas de los servicios que se ofrecen, b) administrar los inventarios de los puntos de venta c) administrar los turnos del personal de los puntos de venta, d) control financiero de las ventas realizadas y e) integración con los sistemas de información institucionales. Actualmente el seguimiento de estas actividades se hace de manera manual, lo cual puede causar perdida de información, mayores costos en tiempo y recursos, no contar con información actualizada para su consulta y para la generación de informes y estadísticas.

Otro servicio que ofrece Bienestar Universitario es la reserva de menús programados o especiales por parte de las unidades académico administrativas para los diferentes eventos que realizan las mismas. Actualmente dicha solicitud se realiza de forma telefónica o diligenciando un formato lo cual permite incurrir en errores que afectarían la calidad del servicio prestado.

Por lo anterior, la División de Servicios de Información de la Universidad Industrial de Santander en su afán de generar soluciones informáticas de la más alta calidad técnica que permitan el proceso de modernización institucional y que faciliten a la comunidad universitaria el acceso a la información ve de vital importancia , contar

con un sistema de información orientado a la web que permita el soporte de los procesos que se realizan en la cafetería de Bienestar Universitario, integrado con Los sistemas de información de la institución y realizado con tecnologías avanzadas cumpliendo los estándares de desarrollo establecidos por la universidad para tal fin.

El nuevo sistema será elaborado con un diseño orientado a objetos con UML y JAVA 5, estándares utilizados por la División de Servicios de Información para el desarrollo de las nuevas versiones de los sistemas de información institucionales.

1.1.4 Impacto

Este sistema en primer lugar , le permitirá al personal encargado de administrar los procesos desarrollados en la cafetería de Bienestar Universitario brindar soluciones oportunas a los inconvenientes que se puedan presentar y analizar y estudiar oportunidades de mejora que contribuyan a prestar un servicio de mejor calidad a toda la comunidad universitaria.

En segundo lugar, aumentará de forma considerable la satisfacción de la comunidad universitaria respecto al servicio ofrecido por Bienestar Universitario debido a que se podrá consultar a través de la web los productos y menús disponibles a la venta en los diferentes puntos de venta; por su parte los jefes de las unidades académico administrativas podrán gestionar las reservas realizadas desde cualquier lugar.

1.1.5 Viabilidad

La viabilidad del proyecto es total, debido a que este es realizado con herramientas de desarrollo de software de libre distribución, o adquiridos por la

división de Servicios de Información, y se ha desarrollado en los servidores pertenecientes a la universidad.

En cuanto a hardware se cuenta con instalaciones adecuadas, con los equipos requeridos y el soporte tecnológico necesario para el desarrollo del mismo. Además, se tiene la supervisión por parte del director del proyecto y la colaboración del equipo de trabajo de la División de Servicios de información

CAPITULO 2

2 MARCO TEÓRICO

2.1 Universidad Industrial De Santander

2.1.1 Misión[1]

La Universidad Industrial de Santander es una organización que tiene como propósito la formación de personas de alta calidad ética, política y profesional; la generación y adecuación de conocimientos; la conservación y reinterpretación de la cultura y la participación activa liderando procesos de cambio por el progreso y mejor calidad de vida de la comunidad.

Orientan su misión los principios democráticos, la reflexión crítica, el ejercicio libre de la cátedra, el trabajo interdisciplinario y la relación con el mundo externo.
Sustenta su trabajo en las cualidades humanas de las personas que la integran, en la capacidad laboral de sus empleados, en la excelencia académica de sus profesores y en el compromiso de la comunidad universitaria con los propósitos institucionales y la construcción de una cultura de vida.

2.1.2 Visión

Como visión general en el año 2018, la Universidad Industrial de Santander se habrá fortalecido en su carácter público, aportando al desarrollo político, cultural, social y económico del país, como resultado de un proceso de generación y

[1] Tomado – https://www.uis.edu.co/webUIS/es/acercaUis/principios.html

adecuación de conocimiento en el cual la investigación constituye el eje articulador de sus funciones misionales.

La Universidad habrá desarrollado exitosamente una política de crecimiento vertical, mediante la cual se crearán y consolidarán programas de maestría y doctorado de alta calidad, sustentados en procesos de investigación pertinente para la región y el país.

La Institución habrá contribuido al desarrollo regional, mediante la formación del talento humano, la investigación y la extensión, reflejado en el mejoramiento de la calidad de vida, la competitividad internacional y el crecimiento económico. Como parte de este proceso, se ampliará la cobertura con la creación y consolidación de programas misionales pertinentes y soportes estratégicos en su sede central y en sus sedes regionales tanto a nivel profesional como a nivel tecnológico, atendiendo a la política de formación por ciclos aprobada por el Consejo Superior.

La Universidad habrá consolidado una política de articulación global que le ha permitido incrementar de manera significativa los resultados de sus procesos misionales mediante la cooperación con instituciones educativas y de investigación de alto prestigio, empresas, entidades gubernamentales, egresados y otros entes públicos y privados nacionales e internacionales.

La Universidad habrá fortalecido en toda su organización una cultura de gestión de alta calidad de los procesos misionales, estratégicos y de apoyo.

Como resultado de la actualización permanente de sus programas académicos, la Universidad forma personas con las competencias apropiadas para liderar el desarrollo económico y social y para realizar proyectos educativos e investigativos, que contribuyan al logro de las metas de desarrollo del país y a la consolidación de una sociedad del conocimiento a nivel regional, nacional e internacional.

La Institución habrá consolidado su estabilidad financiera y modernizado su infraestructura física y tecnológica.

2.1.3 Estructura organizacional de la UIS [2]

Como institución académica de educación superior enmarca su estructura organizacional en torno a los saberes en cinco facultades: Ingenierías Físico-Mecánicas, Ingenierías Físico-Químicas, Ciencias, Salud y Humanidades se conjugan los campos del conocimiento en los que la Universidad adelanta las actividades de docencia, investigación y extensión.

En el siguiente gráfico se aprecia la estructura organizacional de la UIS

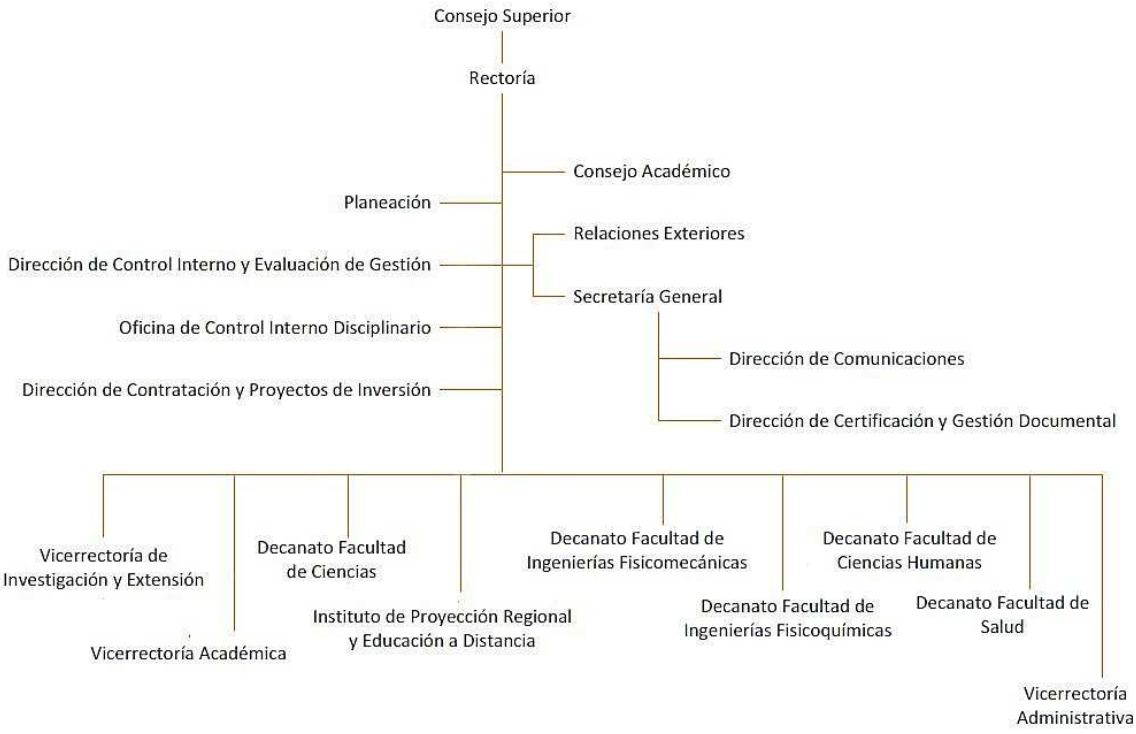

Figura 1 Estructura Organizacional de la UIS

[2] Tomado – https://www.uis.edu.co/webUIS/es/acercaUis/estructuraOrganizacional.html

Las Facultades son unidades académicas y administrativas que agrupan campos y disciplinas afines del conocimiento, profesores, personal administrativo, bienes y recursos, con el objeto de orientar, planificar, fomentar, coordinar, integrar y evaluar actividades de las Escuelas y Departamentos a su cargo, de conformidad con las políticas y criterios emanados del Consejo Superior (máximo órgano de dirección y gobierno de la Universidad) y del Consejo Académico (máxima autoridad académica). Cada Facultad está dirigida por el Decano y el Consejo de Facultad y tiene para la orientación, fomento y coordinación de las actividades de investigación y de extensión, un Director de Investigaciones dependiente del Decano.

Las Escuelas son unidades académicas y administrativas que agrupan uno o varios campos afines del conocimiento y desarrollan programas académicos de pregrado o postgrado, de investigación y de extensión. Cada Escuela tiene un Director quien está asesorado por el Consejo de Escuela y a su cargo se encuentra el personal docente y administrativo adscrito a ésta. Solamente la Escuela de Medicina tiene subdirector, por la cantidad de programas académicos de especialización que maneja.

De la Facultad de Ingenierías Fisicomecánicas dependen las Escuelas de Ingeniería Eléctrica, Electrónica y Telecomunicaciones; Ingeniería Mecánica; Estudios Industriales y Empresariales; Ingeniería Civil; Ingeniería de Sistemas y Diseño Industrial.

La Facultad de Ingenierías Fisicoquímicas está conformada por las Escuelas de Ingeniería Química, Ingeniería Metalúrgica, Ingeniería de Petróleos y Geología.
Hacen parte de la Facultad de Ciencias, las Escuelas de: Física, Química, Matemáticas y Biología.

De la Facultad de Salud, las Escuelas de: Medicina, Enfermería, Bacteriología y Laboratorio Clínico, Fisioterapia y Nutrición.
Conforman la Facultad de Ciencias Humanas, las Escuelas de: Trabajo Social, Idiomas, Educación, Artes, Derecho y Ciencia Política, Historia, Filosofía y Economía y Administración.

Los Departamentos son unidades académicas y administrativas dependientes de una Facultad o Escuela, que prestan servicios a una o varias Escuelas y desarrollan programas de investigación y extensión, de conformidad con las políticas y directrices de la Universidad.

Así, el Departamento de Deportes pertenece a la Facultad de Ciencias Humanas, y de la Escuela de Medicina dependen los Departamentos de Ciencias Básicas, Cirugía, Ginecobstetricia, Medicina Interna, Patología, Pediatría, Salud Mental y Salud Pública.

La Dirección General de Regionalización es la encargada de planificar, fomentar, dirigir, coordinar, evaluar y propender por la calidad académica de los programas de regionalización de la Universidad Industrial de Santander. Además, este organismo se constituye en la instancia correspondiente para la toma de decisiones directamente relacionadas con las sedes, para permitir una mayor agilidad y participación activa de las personas directamente relacionadas con su actividad.

Por su parte, del Instituto de Proyección Regional y Educación a Distancia (IPRED) unidad académica y administrativa adscrita a la Vicerrectoría Académica, dependen los programas de educación a distancia de la Universidad. El IPRED, ofrece además apoyo técnico y logístico para la utilización de metodologías convencionales en las distintas Escuelas. Al frente del Instituto se encuentra un Director General y los Coordinadores de los Programas Académicos que ofrecen.

2.2 División De Bienestar Universitario[3]

La División de Bienestar Universitario (DBU) es la dependencia administrativa de la Universidad Industrial de Santander que brinda apoyo directo a la actividad académica, contribuyendo activamente en la formación integral de los estudiantes a través del desarrollo de programas educativos y el ofrecimiento de servicios asistenciales que propenden por el mejoramiento de la calidad de vida de los estudiantes.

Figura 2 – Estructura Organizacional de la Vicerrectoría Administrativa de la UIS

[3] Tomado de https://www.uis.edu.co/webUIS/es/administracion/bienestarUniversitario/presentacion.jsp

La División de Bienestar Universitario tiene las siguientes funciones[4]:

- Dirigir, orientar, coordinar y ejecutar servicios para el desarrollo integral de los miembros de la comunidad universitaria en un proceso de mejoramiento continuo de la calidad de vida de quienes la conforman.
- Promover la creación de grupos de estudio, artísticos, culturales, deportivos y recreativos.
- Propiciar oportunidades para compartir experiencias.

2.2.1 Misión de Bienestar Universitario

Promover y contribuir al desarrollo integral de las personas que conforman la comunidad universitaria UIS y al mejoramiento de su calidad de vida, mediante la ejecución de Proyectos, Programas y Servicios orientados al desarrollo humano, la protección de la salud y el apoyo social y económico de los grupos vulnerables, con énfasis en la comunidad estudiantil.

2.2.2 Objetivos de Bienestar Universitario

- Ofrecer y mantener servicios y programas que promuevan la formación integral y el mejoramiento de la calidad de vida de la comunidad estudiantil.
- Prestar servicios de salud en el primer nivel de complejidad para favorecer las condiciones de salud y contribuir a la formación y desarrollo integral de los estudiantes
- Fomentar en la comunidad estudiantil la promoción de la salud, la prevención de enfermedades, el autocuidado y la adopción de estilos de vida

[4] Funciones aprobadas por Acuerdo Superior Nº 057 de 1994

saludables que propendan por una mejor calidad de vida y una nueva cultura de salud.

- Ofrecer y mantener servicios de alimentación, alojamiento y beneficios económicos a la Comunidad Universitaria para contribuir al mejoramiento de su calidad de vida.

2.2.3 Estructura Organizacional de la DBU[5]

La división de Bienestar Universitario tiene actualmente la siguiente estructura organizacional

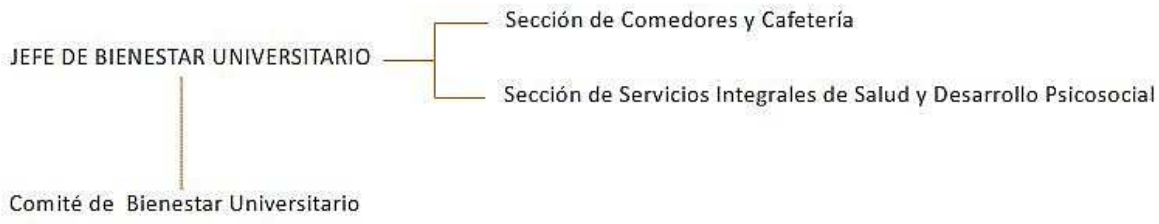

Figura 3 - Estructura Organizacional DBU

2.2.4 Sección Comedores Y Cafetería

Es la sección a través de la cual se participa en el proceso de formación integral del estudiante mediante acciones que permitan el logro de hábitos nutricionales adecuados.

2.2.5 Misión de la Sección de Comedores y Cafetería DBU

[5] Tomado de https://www.uis.edu.co/webUIS/es/administracion/bienestarUniversitario/estructuraOrganizacional.html

Contribuir al mejoramiento de las condiciones nutricionales de la comunidad universitaria a través de acciones educativas y servicios con calidad.
En la actualidad la sección ofrece los siguientes servicios:

- Cafetería
- Almuerzos Corrientes
- Eventos Especiales
- Comedores
- Combos Saludables

2.3Diagramación UML

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh. Estos autores fueron contratados por la empresa Rational Software Co. para crear una notación unificada en la cual basar la construcción de sus herramientas CASE. En el proceso de creación de UML han participado, no obstante, otras empresas de gran peso en la industria como Microsoft, Hewlett-Packard, Oracle o IBM, así como grupos de analistas y desarrolladores.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa (principalmente Booch, OMT y OOSE). UML ha puesto fin a las llamadas “guerras de métodos” que se dieron a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

En la División de Servicios de Información (DSI) se han establecido estándares concernientes al diseño de sistemas, los cuales deben ser desarrollados por módulos y deben ceñirse al estándar definido por el Lenguaje Unificado de Modelado 2.1 (UML).

El diseño de un módulo, desarrollado en la DSI, debe contar con los siguientes diagramas:

- Diagrama de Casos de Uso
- Diagrama de Clases
- Diagrama de Secuencia

2.3.1 Diagrama de Casos de Uso

El diagrama de casos de uso tiene el objetivo de evidenciar todas las funcionalidades del sistema y se diseña desde la perspectiva del usuario, ya que modela la interacción directa de este con el sistema; de su análisis se puede concluir si el sistema cumple satisfactoriamente con los requisitos de los usuarios.

Para el diagrama de casos de uso se identifican tres elementos básicos:

- ***Actores:*** Corresponden a los diferentes roles que los usuarios del sistema pueden representar. Cada rol debe ser único, es decir, distinguible de los demás, contando con un nombre específico y responsabilidades particulares al momento de interactuar con el sistema. No obstante, es necesario aclarar que un rol representa a un tipo o categoría de usuarios del sistema e incluso un usuario puede tener asignado más de un rol en el sistema. Un actor es usualmente identificado como se muestra la figura 2:

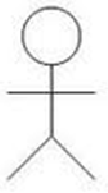

Figura 4 Actor

- ***Caso de Uso:*** Un caso de uso describe una funcionalidad del sistema que produce un resultado perceptible para el usuario, sin entrar en detalles sobre *cómo* la realiza. Su comportamiento puede especificarse como un flujo de eventos en texto formal, o en pseudocódigo. En su conjunto, los casos de uso son los que describen todas las funcionalidades del sistema. En la figura 3 indica cómo se grafican los casos de uso en el diagrama.

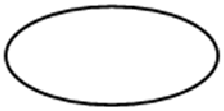

Figura 5 - : Caso de Uso

Un caso de uso especifica dos tipos de secuencias o comportamientos:

a) Secuencia Básica o Comportamiento Normal: Son las acciones que ejecuta el actor sobre el sistema en el orden establecido en cada caso de uso.

b) Secuencia o Comportamiento Alternativo: Es el camino que el Actor invoca cuando este no lleva a cabo la secuencia básica en forma satisfactoria, es decir, cuando el Actor comete errores al momento de interactuar con el sistema.

- ***Relaciones:*** Son los elementos que conectan los anteriores elementos en el diagrama (Actores y Casos de Uso), los cuales confieren un significado a cada vínculo que establecen.

  En un diagrama de casos de uso pueden existir los siguientes enlaces:

  - *Relación de Generalización entre Actores:* Se utiliza cuando un actor hereda ciertas características de otro más general.
  - *Relación de Generalización entre casos de Uso:* Indica cuando un caso de uso hereda y adiciona características de otro más general.

  - *Relación de Extensión:* Es un enlace entre casos de uso que define cuando un caso de uso es extendido por otro, es decir, cuando un caso de uso tiene variantes en su secuencia básica, la cual indica una extensión hacia la secuencia básica de otro.

  - *Relación de Inclusión:* Señala cuando el comportamiento normal de un caso de uso por su naturaleza incorpora el comportamiento normal de otro.

  - *Relación de Asociación:* Se utiliza para conectar un actor con un caso de uso e implica la existencia de una comunicación entre ellos.

  La siguiente figura 4 ilustra los diferentes tipos de relaciones que pueden presentarse en un diagrama de casos de uso:

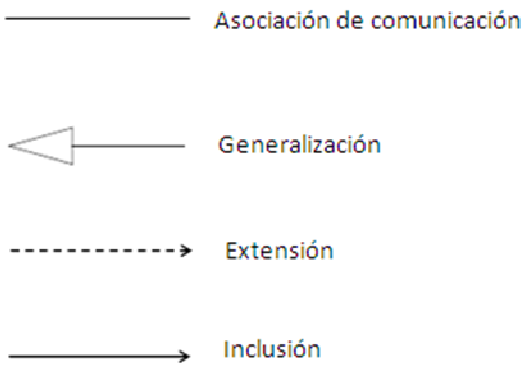

Figura 6 Tipos de Relaciones

2.3.2 Diagrama de Clases

Sirve para modelar las clases que involucra el sistema, pero es preciso aclarar que el diagrama de clase representa el prototipo estático del sistema, ya que no explica el comportamiento del sistema en el tiempo.

Un diagrama de clases se compone de dos elementos: Clases y Relaciones.

2.3.2.1 Clases

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Estas clases presentan ciertas características:

- *Estado:* Es definido por los atributos que contiene la clase y por sus posibles relaciones con otras.
- *Comportamiento:* Explica la funcionalidad de una clase, señalando todas las operaciones que esta puede realizar.

Una clase es representada en la figura 7:

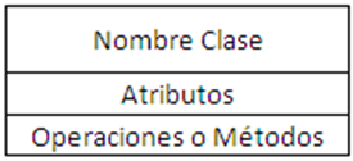

Figura 7 Representación de clase

2.3.2.2 Relaciones

Es la connotación entre los objetos de dos o más clases. Las clases pueden ser interrelacionadas por medio de las siguientes relaciones:

- ***Relación de Asociación:*** Representa un conjunto de enlaces entre dos clases distintas, los cuales pueden ser unidireccionales o bidireccionales. Además contempla la Multiplicidad de Asociaciones que es la cantidad de objetos de cada clase en una relación.

- ***Relación de Agregación:*** Es la relación que existe entre una clase que envuelve o incluye a otras clases, cuyos tiempos de vida no dependen del tiempo de vida de la clase que las incluye. Cuando el tiempo de vida de un objeto de una clase depende del tiempo de vida de otro objeto que lo incluye, se dice que es una ***Relación de Composición.***

- ***Relación de Herencia:*** Es el vínculo entre una Súper clase y una o más subclases hijas, quienes heredan atributos y comportamientos de su clase padre y pueden extender las propiedades de dicha Súper clase adicionando atributos que proporcionan un mayor detalle de lo que la clase padre representa.

La herencia puede darse de dos formas:

- *Generalizada:* Ocurre cuando la Súper clase encapsula las propiedades y comportamientos de una o más subclases. En este tipo de relación las subclases no pueden heredar.
- *Especializada:* Se da cuando las subclases heredan las propiedades y comportamientos de una clase mayor dándole a esta última un mayor nivel de detalle.

La figura 8 exhibe las relaciones que pueden existir en el diagrama de clases, anteriormente descritas:

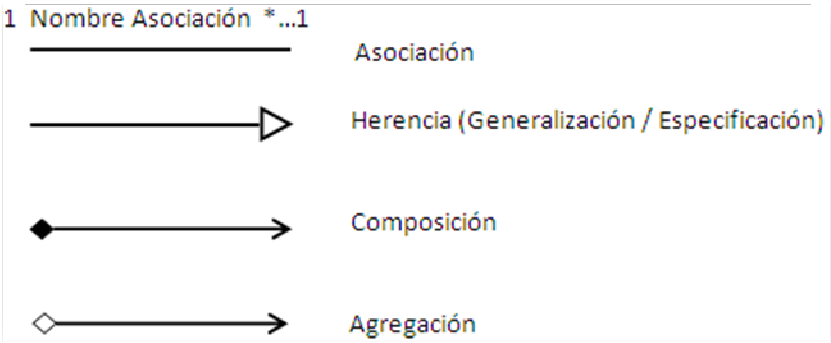

Figura 8 Relaciones entre Clases

2.4Tecnologías de Desarrollo de Aplicaciones Web

2.4.1 JAVA EE5

En los últimos años la gran mayoría de servicios y aplicaciones informáticas han migrado hacia los entornos web en pro de la ejecución en múltiples plataformas con diferentes capacidades de procesamiento e incluso la posibilidad de aplicaciones móviles, además de la reducción de costos y optimización de los

procesos. Sin importar la diversidad de las aplicaciones, los desarrolladores reconocen puntos comunes imprescindibles en ellas: Distribuidas, seguras, robustas, con alta capacidad transaccional y eficiente. La respuesta de Java para estas necesidades es Java Enterprise Edition, que busca agilizar el proceso de desarrollo brindándole a los desarrolladores toda una variada oferta de APIs, frameworks y tecnologías para responder a distintas necesidades.

Java Enterprise Edition 5 (Java EE 5) se centra en hacer más fácil el desarrollo. Sin embargo, conserva la riqueza de la plataforma J2EE 1.4. Reduce enormemente la necesidad de archivos XML de configuración (configuración del despliegue de las aplicaciones), lo que repercute en rapidez en el desarrollo ante la eliminación de código repetitivo. Ofrece la tecnología de servicios web API, haciendo que la codificación sea más simple y directa, pero manteniendo el poder que ha establecido a Java EE como la primera plataforma para servicios web y desarrollo de aplicaciones empresariales.

2.4.1.1 Modelo De Aplicación JAVA EE

La plataforma empresarial de Java fue diseñada para soportar aplicaciones que brinden servicios empresariales para clientes, empleados, proveedores, socios y demás personas o entidades que interactúan con la empresa.

El modelo de aplicación Java EE define una arquitectura para implementar servicios como aplicaciones de múltiples capas que distribuyen la escalabilidad, accesibilidad y facilidad de manejo que se necesita para aplicaciones empresariales.

2.4.1.2 Aplicaciones De Múltiples Capas Distribuidas

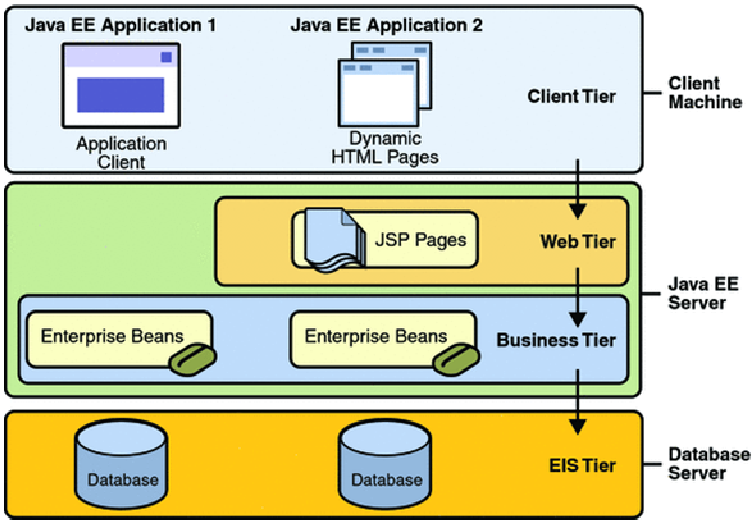

Figura 9 Aplicaciones de múltiples capas

La lógica de la aplicación se distribuye según su función en componentes, los cuales son instalados en diferentes máquinas de acuerdo a la capa en el ambiente de múltiples capas de Java EE a la cual pertenece el componente de aplicación.

La figura anterior muestra dos aplicaciones Java EE de capa múltiple dividida en sus capas las cuales están descritas en la siguiente lista.

- Los componentes de la capa de cliente se ejecutan en la máquina del cliente.
- Los componentes de la capa Web se ejecutan en el servidor Java EE.
- Los componentes de la capa de negocios se ejecutan en el servidor Java EE.

Una aplicación Java EE puede consistir en tres o cuatro capas, como se muestra en la figura, sin embargo las aplicaciones de múltiples capas de Java EE son consideradas de tres capas porque se distribuyen en tres ubicaciones: máquinas

clientes, la máquina servidor Java EE y la base de datos. Estas aplicaciones extienden el modelo cliente servidor, ubicando un servidor de aplicación entre la capa cliente y la de almacenamiento.

2.4.2 Servidor de Aplicaciones – Jboss

El servidor de aplicaciones JBoss, es una herramienta certificada para el desarrollo de aplicaciones empresariales Java. Su madurez y el esfuerzo de muchos desarrolladores, e incluso las sugerencias que han realizado sus usuarios, han permitido que JBoss AS (Application Server) se popularice ampliamente y sea común en los currículos de desarrolladores.

Es reconocido por soportar los estándares más recientes. De hecho, es el primer servidor de aplicaciones en alcanzar la certificación J2EE 1.4 cuando salió su versión 4.0. Pero JBoss no sólo marca la pauta en la adopción de estándares con su servidor de aplicaciones, sino en la imposición de los mismos. Hace parte del *Java Community Process* (JCP).

2.4.3 Aplicaciones web

Una aplicación web es una extensión dinámica de una web o servidor de aplicación. Hay dos tipos de aplicaciones web:

- Orientada a la presentación: Una aplicación web orientada a la presentación genera páginas web interactivas que contienen varios tipos de lenguajes de marcas (HTML, XML y demás) y contenido dinámico en respuesta a las peticiones.

- Orientadas a los servicios: Una aplicación web orientada a los servicios implementa el punto final de un servicio web. Las aplicaciones orientadas a la presentación son a menudo clientes de aplicaciones orientadas a servicios.

Desde la introducción de la tecnología de Servlets y JSP, han sido desarrollados marcos de trabajo (frameworks) para construir aplicaciones web interactivas. La figura muestra estas tecnologías y sus relaciones.

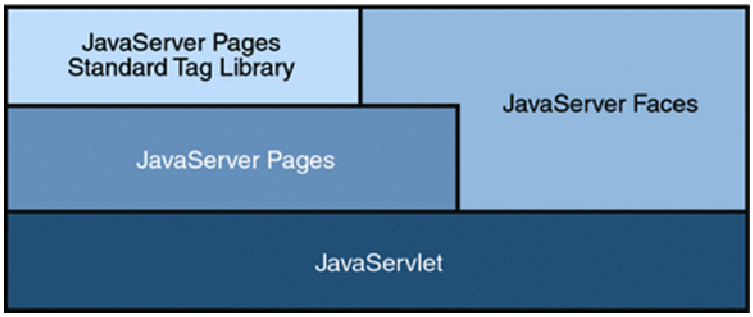

Figura 10 Tecnologías Java para aplicaciones web

Debe notarse que la tecnología Java Servlet es la base para todas las tecnologías de aplicaciones web. Cada tecnología agrega un nivel de abstracción que hace la creación de prototipos y el desarrollo más rápido y la aplicación web por sí misma es más fácil de mantener, de escalar y más robusta.

Los componentes web son soportados por los servicios de una plataforma en tiempo de ejecución llamada contenedor web. Un contenedor web proporciona los servicios como distribuir una petición, seguridad, concurrencia y manejo de ciclos de vida. También les da a los componentes web acceso a las APIs para nombrado, transacciones y correo electrónico.

2.4.3.1 Ciclo de vida de una aplicación web

Una aplicación web consiste en componentes web, ficheros de recursos estáticos como imágenes, clases de ayuda y librerías. El contenedor web proporciona muchos de los servicios de soporte para mejorar las habilidades de los componentes web y los hace fáciles de desarrollar. Sin embargo, dado que una aplicación web debe tomar estos servicios en una cuenta, el proceso de crear y ejecutar una aplicación web es diferente que el utilizado para las clases Java autónomas.

El proceso para crear, desplegar y ejecutar una aplicación web puede resumirse como sigue:

- Desarrollar el código del componente web.
- Desarrollar el descriptor de despliegue de la aplicación web.
- Compilar los componentes de la aplicación web y clases de ayuda referenciada por los componentes.
- Opcionalmente empaquetar la aplicación en una unidad que se pueda desplegar.
- Desplegar la aplicación en un contenedor web.
- Acceder a una URL que referencia la aplicación web.

2.4.3.2 Módulos web

En la arquitectura Java EE, los componentes web y los ficheros con contenido estático como imágenes son llamados recursos web. Un módulo web es la unidad más pequeña de un recurso web que se puede utilizar y desplegar. Un módulo web Java EE corresponde con una aplicación web como se define en la especificación de Java Servlet. (Numeral 2.4.4)

Además de los componentes web y los recursos web, un módulo web puede contener otros ficheros:

- Clases utilitarias del lado del servidor (beans para bases de datos, carritos de compras y demás). A menudo estas clases cumplen con la arquitectura JavaBeans.
- Clases del lado del cliente (applets y clases utilitarias).

Un módulo web tiene una estructura específica. El directorio más alto de la jerarquía de directorios de un módulo web es la raíz de documento de la aplicación. Es donde las páginas JSP, clases y archivos del lado del cliente y los recursos estáticos son almacenados.

La raíz de documentos contiene un subdirectorio llamado WEB-INF, que contiene los siguientes ficheros y directorios:

- web.xml: El descriptor de despliegue de aplicación.
- Los ficheros descriptores de las librerías de etiquetas.
- classes: Un directorio que contiene las clases del lado del servidor: componentes Servlets, clases utilitarias y JavaBean.
- tags: Un directorio que contiene ficheros de etiquetas, que son implementaciones de librerías de etiquetas.
- lib: Un directorio que contiene los archivos JAR de las librerías llamadas por las clases del lado del servidor.

Si su módulo web solo contiene páginas JSP y ficheros estáticos no se necesita incluir un fichero web.xml.

Se pueden crear subdirectorios específicos de la aplicación (esto es, directorios de paquete) tanto en la raíz de documentos como en el directorio WEB-INF/classes/.

Un módulo web puede ser desplegado como una estructura de ficheros sin empaquetar o puede ser empaquetado en un fichero JAR conocido como un archivo web (WAR). Dado que el contenido y uso de los ficheros WAR difieren de aquellos ficheros JAR, el nombre del fichero WAR utiliza una extensión .war. El módulo web descrito es portátil, se puede desplegar en cualquier contenedor web que cumpla con la especificación Java Servlet.

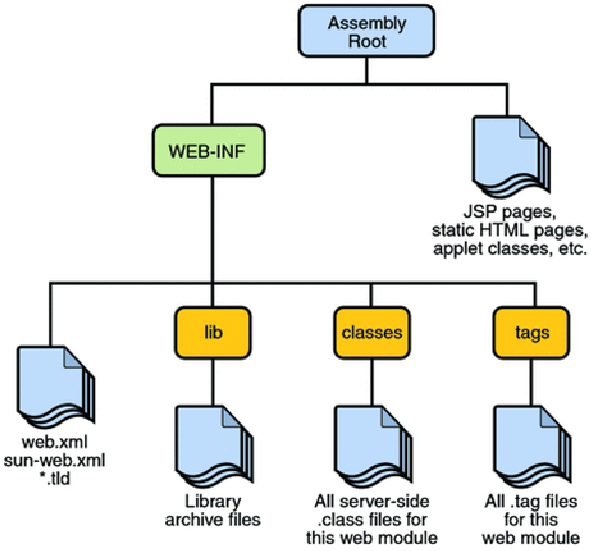

Figura 11 Estructura de un módulo web

2.4.4 Tecnología Servlet Java

2.4.4.1 ¿Qué es un Servlet?

Un servlet es una clase del lenguaje de programación Java que es utilizada para extender las habilidades de los servidores que guardan aplicaciones a las cuales se accede mediante el modelo petición-respuesta.

A pesar de que los servlets pueden devolver cualquier tipo de respuesta, estos son comúnmente utilizados para extender las aplicaciones almacenadas en

servidores web. Para estas aplicaciones, la tecnología Servlet Java define las clases servlets específicas para HTTP.

2.4.5 Java Server Faces

2.4.5.1 Características principales

La tecnología JavaServer Faces constituye un marco de trabajo (*framework*) de interfaces de usuario del lado de servidor para aplicaciones web basadas en tecnología Java y en el patrón MVC (Modelo Vista Controlador).

Los principales componentes de la tecnología JavaServer Faces son:

- Una API y una implementación de referencia para:
  - Representar componentes de interfaz de usuario (UI-*User Interface*) y manejar su estado.
  - Manejar eventos, validar en el lado del servidor y convertir datos.
  - Definir la navegación entre páginas.
  - Soportar internacionalización y accesibilidad.
  - Proporcionar extensibilidad para todas estas características.
- Una librería de etiquetas JavaServerPages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido junto con la librería de etiquetas para componentes UI facilita de forma significativa la tarea de la construcción y mantenimiento de aplicaciones web con UIs en el lado servidor. Con un mínimo esfuerzo, es posible:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos en el lado servidor.

- Construir una interfaz de usuario con componentes reutilizables y extensibles.

Como se puede apreciar en la figura, la interfaz de usuario que se crea con la tecnología JavaServer Faces se ejecuta en el servidor y se renderiza en el cliente.

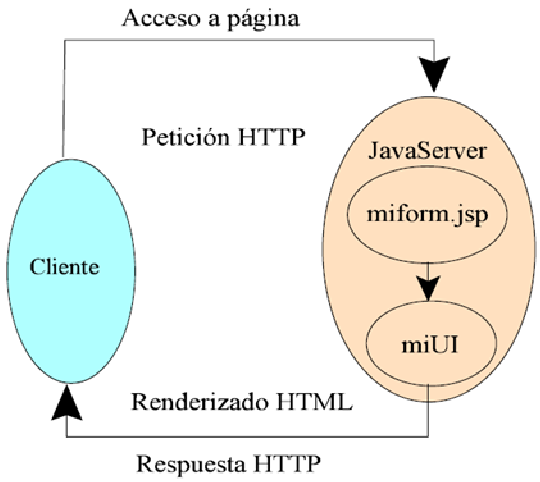

Figura 12 Diagrama de una aplicación JSF

En la figura, la página JSP (**miform.jsp**), especifica los componentes de la interfaz de usuario mediante etiquetas personalizadas definidas por la tecnología JavaServer Faces. La UI de la aplicación web (representada por *miUI* en la figura 12) maneja los objetos referenciados por la JSP, que pueden ser de los siguientes tipos:

- Objetos componentes que mapean las etiquetas sobre la página JSP.
- Oyentes de eventos, validadores y conversores registrados y asociados a los componentes.
- Objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación (lógica de negocio).

2.4.5.2 Beneficios de la Tecnología JavaServer Faces (JSF)

Una de las ventajas de que JSF sea una especificación estándar es que pueden encontrarse implementaciones de distintos fabricantes. Esto permite no vincularse exclusivamente con un proveedor concreto y permitir la selección del más adecuado según los requerimientos de la aplicación, según el número de componentes que suministra, el rendimiento de éstos, soporte proporcionado, precio, política de evolución, etc.

JSF trata la vista (la interfaz de usuario) de una forma algo diferente a lo que se está acostumbrado en las aplicaciones web, donde la programación de la interfaz se desarrolla a través de componentes y está basada en eventos (pulsación de un botón, cambio en el valor de un campo, etc.).

JSF es muy flexible, ya que permite personalizar tanto los componentes como la recarga de la vista de las páginas, con el fin elaborar interfaces de usuario en la forma que más convenga.

La tecnología JavaServer Faces permite construir aplicaciones web que introducen realmente una separación entre el comportamiento y la presentación, separación sólo ofrecida tradicionalmente por arquitecturas UI del lado del cliente y parcialmente por la tecnología JSP.

Separar la lógica de negocio de la presentación también permite que cada miembro del equipo de desarrollo de la aplicación web se centre en su parte asignada del proceso diseño, y proporciona un modelo sencillo de programación para enlazar todas las piezas.

Otro objetivo importante de la tecnología JavaServer Faces es mejorar los conceptos asociados con componente-UI y capa-web sin limitarse a una tecnología de *script* particular o un lenguaje de marcas. Aunque la tecnología JavaServer Faces incluye una librería de etiquetas JSP personalizadas para

representar componentes en una página JSP, las APIs de JavaServer Faces se han creado directamente sobre el API *JavaServlet*. Esto permite, teóricamente, hacer algunas cosas avanzadas: usar otra tecnología de presentación junto a JSP, crear componentes propios directamente desde las clases de componentes, y generar salida para diferentes dispositivos cliente; entre otras.

2.4.5.3 ¿Qué es una aplicación JavaServer Faces?

En su mayoría, las aplicaciones JavaServer Faces son como cualquier otra aplicación web Java. Se ejecutan en un contenedor de servlets de Java y, típicamente, contienen:

- Componentes JavaBeans conteniendo datos y funcionalidades específicas de la aplicación.
- Oyentes de Eventos.
- Páginas, (principalmente páginas JSP).
- Clases de utilidad del lado del servidor, como beans para acceder a las bases de datos.

Además de estos ítems, una aplicación JavaServer Faces también tiene:

- Una librería de etiquetas personalizadas para implementar componentes UI en una página.
- Una librería de etiquetas personalizadas para representar manejadores de eventos, validadores y otras acciones.
- Componentes UI representados como objetos con estado en el servidor.

Toda aplicación JavaServer Faces debe incluir una librería de etiquetas personalizadas que define las etiquetas que representan componentes UI, así como una librería de etiquetas para controlar otras acciones importantes, como

validadores y manejadores de eventos. La implementación de JavaServer Faces, de Sun Microsystems, proporciona estas dos librerías. La librería de etiquetas de componentes elimina la necesidad de codificar componentes UI en HTML u otro lenguaje de marcas, lo que se traduce en el empleo de componentes completamente reutilizables. Y la librería principal (core) hace fácil registrar eventos, validadores y otras acciones de los componentes.

Otra ventaja importante de las aplicaciones JavaServer Faces es que los componentes UI de la página están representados en el servidor como objetos con estado. Esto permite a la aplicación manipular el estado del componente y conectar los eventos generados por el cliente en el lado del servidor.

Finalmente, la tecnología JavaServer Faces permite convertir y validar datos sobre componentes individuales e informar de cualquier error antes de que se actualicen los datos en el lado del servidor.

Debido a la división de labores que permite el diseño de la tecnología JavaServer Faces, el desarrollo y mantenimiento de una aplicación JavaServer Faces se puede realizar muy rápida y fácilmente.

2.4.6 Modelo Vista Controlador En Jsf

El patrón MVC (Modelo Vista Controlador), nos permite separar la lógica de control (qué cosas hay que hacer pero no cómo), la lógica de negocio (cómo se hacen las cosas) y la lógica de presentación (cómo interactuar con el usuario).

Utilizando este tipo de patrón es posible conseguir más calidad, un mantenimiento más fácil, perder el miedo al folio en blanco (existe un patrón de partida por el que empezar un proyecto), etc. al margen de todo esto, una de las cosas más

importantes que permite el uso de este patrón consiste en normalizar y estandarizar el desarrollo de Software.

En la figura se muestra un esquema del patrón de diseño MVC:

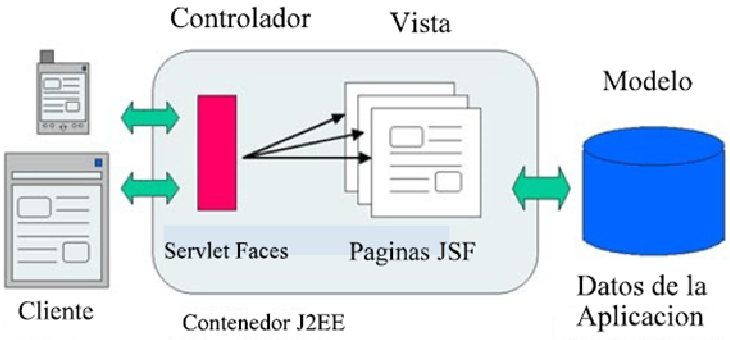

Figura 13 Modelo vista controlador

Este modelo de arquitectura presenta otras importantes ventajas:

- Hay una clara separación entre los componentes de un programa; lo cual admite implementarlos por separado.

- Se cuenta con una API muy bien definida; cualquiera que use la API, podrá reemplazar el modelo, la vista o el controlador, sin dificultad.

- La conexión entre el modelo y sus vistas es dinámica: se produce en tiempo de ejecución, no en tiempo de compilación.

2.4.6.1 Modelo

El modelo es el objeto que representa y trabaja directamente con los datos del programa: gestiona los datos y controla todas sus transformaciones. El modelo no

tiene conocimiento específico de los diferentes controladores y/o vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el modelo y sus vistas, y notificar a las vistas cuándo deben reflejar un cambio en el modelo.

2.4.6.2 Vista

La vista es el objeto que maneja la presentación visual de los datos gestionados por el Modelo. Genera una representación visual del modelo y muestra los datos al usuario e interactúa con el modelo a través de una referencia al propio modelo.

2.4.6.3 Controlador

El controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el modelo. Entra en acción cuando se realiza alguna operación, ya sea un cambio en la información del modelo o una interacción sobre la Vista. Se comunica con el modelo y la vista a través de una referencia al propio modelo.

Además, JSF opera como un gestor que reacciona ante los eventos provocados por el usuario, procesa sus acciones y los valores de estos eventos, y ejecuta código para actualizar el modelo o la vista.

El siguiente diagrama muestra la relación existente entre el modelo, la vista y el controlador cada vez que un usuario realiza operaciones sobre la página:

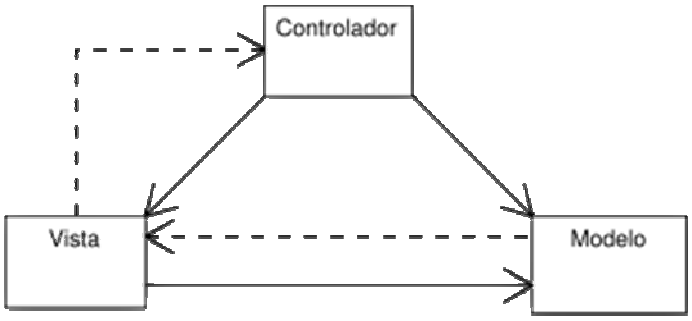

Figura 14 Modelo Vista-Controlador

En la figura 14 se puede observar las relaciones entre el Modelo, la Vista y el Controlador. Cabe destacar en la figura las líneas continuas que significan una relación directa como también las líneas discontinuas que implican una relación indirecta. También es importante tener en cuenta que aunque puede haber cierta "referencia indirecta" entre el Modelo y la Vista, el primero sigue sin saber nada del segundo.

El modo en que interactúan los tres elementos del modelo MVC se puede describir a continuación:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el

controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (del modelo) a la vista aunque puede dar la orden a la vista para que se actualice.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

2.4.7 Seam

Seam es un framework de aplicaciones de Java Enterprise Edition inspirado en los siguientes principios:

- Un tipo de cosas

Seam define un modelo de componentes uniformes para la lógica del negocio en su aplicación. Un componente Seam puede tener un estado, que se encuentra asociado a cualquiera de los diversos contextos bien definidos, incluyendo el de

larga duración, el contexto de persistencia, de procesos de negocio y el contexto de la conversación, que se conserva en todas las solicitudes múltiples en una interacción con el usuario.

En Seam no hay distinción entre los componentes del nivel de presentación y componentes de la lógica del negocio. El desarrollador puede estratificar la aplicación de acuerdo con la arquitectura que se haya diseñado. Los componentes Seam pueden simultáneamente tener acceso al estado asociado a la solicitud web y al estado de recursos transaccionales.

- Integrar JSF con EJB 3.0

  EJB 3 es un modelo de componentes a nivel de lógica de negocio y de persistencia, del lado del servidor, mientras que JSF es un modelo de componentes de la capa de presentación.

  JSF y EJB3 funcionan mejor juntos, a pesar de que Java EE5 no proporciona un estándar para integrar estos modelos de componentes. No obstante los creadores de ambos modelos (JSF y EJB3) proporcionan puntos de extensión estándar para permitir la integración con otros marcos.

  Seam unifica los modelos de componentes de JSF y EJB3, dejando al desarrollador el único problema de diseñar la lógica del negocio.

- Integrar AJAX

Seam soporta mejor las soluciones de código abierto basado en AJAX JSF: JBossRichFaces e ICEfaces. Estas soluciones le permiten agregar la capacidad de AJAX para la interfaz de usuario sin necesidad de escribir código JavaScript.

Por otra parte, seam proporciona una capa incorporada del Javascript que le permite llamar a los componentes JavaScript de forma asincrónica del lado cliente sin la necesidad de una capa de acción intermedia.

Estos enfoques funcionan bien porque Seam incorporó la gestión de concurrencia y el estado, que aseguran que las peticiones asincrónicas Ajax se manejan de forma segura y eficiente en el servidor.

- Procesos de negocio como el primer constructor de clase

Seam ofrece transparencia en la gestión de procesos de negocio a través de JBPM, incluso permite definir el flujo de páginas de la capa de presentación utilizando el mismo lenguaje que jBPM utilizado para la definición de procesos de negocio.

- Declarativa administración del estado

Tradicionalmente, las aplicaciones J2EE implementan la administración de estado de forma manual. Este enfoque es la fuente de muchos errores y pérdidas de memoria cuando las aplicaciones no pueden limpiar los atributos de sesión, o cuando los datos de sesión asociados a diferentes flujos de trabajo chocan en una aplicación de múltiples ventanas. Seam tiene el potencial de eliminar casi por completo esta clase de errores.

La declarativa de administración del estado se ha logrado gracias a que Seam extiende el modelo de contexto definido por la especificación de servlets (petición, sesión, aplicación) con dos nuevos contextos (conversación y procesos de negocio).

- Biyección

La biyección es dinámica y bidireccional, podríamos pensar en esto como un mecanismo de alias para variables contextuales (nombres en alguno de los contextos enlazados al hilo de ejecución actual) a atributos del componente.

La biyección permite auto ensamblaje de componentes con estado por el contenedor, lo que incluso permite a un componente asegurar y fácilmente manipular el valor de una variable contextual, simplemente asignándola a un atributo del componente.

- Preferir anotaciones a XML

La comunidad Java ha estado confundida sobre el tipo de meta información con la que cuenta la configuración; las anotaciones de Java han logrado cambiar esto.

EJB 3.0 abarca las anotaciones y la configuración de excepción como la forma más fácil de proporcionar información al contenedor en forma declarativa. Seam extiende las anotaciones de EJb3 con una serie de anotaciones para la administración del estado declarativo y demarcación del contexto declarativo.

- La prueba de integración es fácil

Los componentes de Seam, siendo simples clases Java, son por naturaleza comprobables. Sin embargo, para aplicaciones complejas, las pruebas unitarias por sí solas son insuficientes. Seam proporciona la capacidad de prueba de aplicaciones seam como una característica central del framework. El usuario puede escribir las pruebas JUnit o TestNG que reproducen una interacción completa con un usuario. Estas pruebas pueden ser ejecutadas directamente en

el IDE, donde Seam automáticamente implementa los componentes EJB con JBoss Embebido.

• Hay más de una aplicación web que sirve páginas HTML

Un framework de aplicaciones web realmente completo debe abordar problemas como la persistencia, la concurrencia, a sincronía, administración del estado, seguridad, correo electrónico, mensajería, pdf, generación de gráficos, servicios web, caché, entre otros.

Seam integra JPA e Hibernate3 para persistencia, EJB TimerService y Quartz para ligeras asincronías, jBPM para flujo de trabajo, JBoss rules para reglas del negocio, Meldware Mail para correo electrónico, HibernateSearch y Lucene para búsqueda de texto, JMS para mensajes y JBoss Caché para la página de almacenamiento en caché.

Seam funciona en cualquier servidor de aplicaciones Java EE y también en Tomcat. Si el IDE no admite EJB 3.0 puede utilizar Seam incorporado en la gestión de transacciones con JPA o Hibernate3 para la persistencia. También se puede utilizar JBoss embebido en Tomcat, y tener un soporte completo para EJB 3.0.

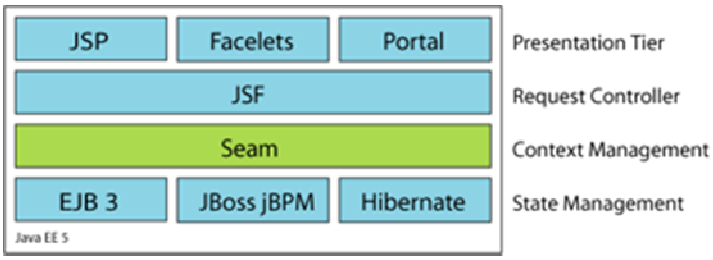

Figura 15 Arquitectura de Seam

2.4.7.1 Modelo De Componentes Contextuales

Para conocer el núcleo del Framework de Seam es necesario entender dos conceptos esenciales: los contextos y los componentes.

- Contextos de Seam*:*

Los contextos son contenedores gestionados por el mismo framework de Seam en los cuales residen todos los componentes instanciados y que pueden ser demarcados por medio de anotaciones.

Seam no sólo fue el primer framework que introdujo el concepto de Biyección de componentes sino que también es considerado como un Meta-Contenedor (Contenedor de Contenedores) que define y administra el ciclo de vida de cada componente inyectado o eyectado en un Contenedor o Contexto.

Contexto Sin Estado o Stateless*:* Es un contexto que contiene únicamente componentes sin estado (Stateless) lo que significa una ausencia de contexto ya que las resoluciones de una instancia de Seam no son almacenadas. No obstante se han desarrollado y utilizado ya que son una parte importante de cualquier aplicación de Seam.

Contexto de Evento: Considerado como el contexto de estado “más estrecho”, proporciona una generalización de la noción de contexto de petición Web para cubrir otros tipos de eventos. Un claro ejemplo de contexto de evento tiene que ver con el ciclo de vida de una petición JSF en el cual los componentes invocados por una solicitud JSF en un contenedor de evento, son eliminados inmediatamente después responder con la petición.

Contexto de Página: Este contexto permite asociar el estado de un componente con la instancia del llamado de una página, es decir, Seam crea el contexto al momento de direccionar una página. Es muy útil al momento de requerir listas de hacer clic (Combo Box) donde cada lista es devuelta por el cambio en los datos en el servidor. Los componentes perduran siempre y cuando no haya un redireccionamiento a otra página.

Contexto de Conversación: Es posiblemente el lugar ideal para guardar el estado de una aplicación ya que permite al desarrollador implementar casos de uso relativamente extensos o realizar transacciones relativamente largas. Una "conversación" es una unidad de trabajo desde el punto de vista del usuario, que abarca varias peticiones e incluso varias transacciones con la base datos. La conversación mantiene su estado asociándolo a cada ventana en forma individual con el fin de evitar posibles colisiones entre conversaciones, ya que un usuario puede estar manejando múltiples conversaciones al mismo tiempo (por ejemplo en el caso de tener la misma página en más de una instancia del navegador). Seam controla el estado de las conversaciones mediante un tiempo de espera que se puede configurar con el propósito de evitar el incremento de conversaciones inactivas de un usuario en sesión, dado el caso que un usuario abandone la conversación. Otra característica particular del contexto de conversación es la posibilidad de tener conversaciones anidadas; en otras palabras una conversación contenida dentro de otra mayor.

Contexto de Sesión: Mantiene el estado de los componentes asociados al inicio de sesión de un usuario. Este contexto puede ser asociado con la interface HttpSession, sin embargo el contexto de sesión de Seam fue diseñado para manejar la Sincronización (Serialización de peticiones para evitar colisiones de solicitudes) y la Clusterización (facilidad en la distribución de componentes); ventajas que le confieren una verdadera robustez a cualquier aplicación web.

Contexto de Proceso de Negocio: Se encuentra asociado con una ejecución de proceso de negocio largo que abarca múltiples interacciones entre múltiples usuarios lo que implica que el estado sea compartido, manejado y persistido por el motor BPM (Business Process Management). Seam carece de anotaciones para realizar la demarcación de este contexto por lo que se debe manejar en forma externa utilizando un Lenguaje de Definición de Procesos.

Contexto de Aplicación: Es el contexto más general de la especificación de servlets. Este contexto se utiliza generalmente para guardar información estática como los datos de configuración, de referencia o meta-modelos. Seam hace uso de este contexto al establecer su propia configuración y meta-modelos.

- *Variables de Contexto*

Corresponden al conjunto de objetos contenidos en el espacio de nombres definido por el Contexto. A una variable de contexto se le puede asociar cualquier valor deseado pero usualmente se realiza un enlace de las instancias de Componentes Seam a las variables de Contexto; de esta manera una instancia de un componente es identificada por su nombre dentro de un contexto.

- Prioridad de búsqueda de los Contextos Seam

Algunas veces las instancias de componentes son obtenidas desde un ámbito (Scope) particular conocido, pero cuando no se conoce el ámbito del componente a instanciar Seam consulta en todos los contextos con estado bajo un cierto orden de prioridad:

- Contexto de Evento.
- Contexto de Página.
- Contexto de Conversación.

- Contexto de Sesión.
- Contexto de Proceso de Negocio.
- Contexto de Aplicación.

- Componentes Seam:

Los componentes son objetos con estado (Stateful). Generalmente son EJBs (Enterprise JavaBeans), cuya instancia implica una asociación con un contexto en la cual a cada objeto se le asigna una única identidad.

Los componentes Seam son POJOs (acrónimo de Plain Old Java Object), es decir, son instancias de clases que no extienden o implementan nada adicional (como la implementación de interfaces en Hibernate). A pesar de que Seam es un framework desarrollo para integrarse profundamente con el estándar EJB 3.0, sus componentes pueden utilizarse por fuera del contenedor EJB 3.0.

Bean de Sesión con Estado: Estos componentes no solo son capaces de mantener el estado de la aplicación a través de múltiples invocaciones a un Bean sino también a lo largo de múltiples peticiones. Una de las características exclusivas de Seam es la manera como se mantiene la información de una conversación en curso, ya que esta es almacenada en variables de instancia de Sesión Stateful asociadas a este contexto, en lugar de adherirla directamente en el HttpSession.

A menudo los Bean de Sesión con Estado son utilizados como oyentes de acciones JSF aunque nunca deberían ser enlazados con el contexto de página ni con el contexto Stateless. Además en el ámbito de sesión, las peticiones concurrentes de Beans de Sesión Stateful estarán serializadas evitando posibles colisiones, siempre y cuando los interceptores de Seam no estén deshabilitados para el Bean.

Beans de Entidad: Los Beans de Entidad pueden ser ligados a una variable de contexto o a una función como componentes Seam. Como las entidades tienen una identidad de persistencia adicional a su identidad contextual, las instancias de entidad están explícitamente ligadas al código Java, en lugar de ser creadas implícitamente por el framework.

Como los Beans de Entidad son componentes que no soportan la biyección no suelen usarse como oyentes de acciones JSF pero si pueden emplearse como Beans de soporte para proveer propiedades a los componentes JSF tanto en el envío como en el despliegue de formularios.

Los Beans de Entidad están destinados al contexto de conversación por defecto y nunca debe pertenecer a un contexto Stateless; también hay que tener en cuenta que es más eficiente tener una referencia al Bean de entidad en un Bean de Sesión Stateful en ambientes distribuidos.

Java Beans: Estos componentes pueden ser utilizados con los Beans de Sesión Stateful, sin embargo no cuenta con las mismas funcionalidades de este último, entre las que se encuentran:

- Demarcación de transacciones declarativa.
- Seguridad declarativa.
- Replicación del estado distribuido eficiente.
- Persistencia EJB 3.0.
- Métodos de Tiempo de Espera.

2.4.7.2 Eventos, interceptores y el manejo de excepciones

- *Eventos Seam*

El modelo de componentes Seam fue desarrollado para su uso con aplicaciones orientadas a eventos. Los eventos en Seam son de varios tipos:

- Eventos JSF
- Eventos de transición jBPM
- Acciones de página de Seam
- Eventos de Seam orientados a componentes
- Eventos contextuales de Seam

- *Acciones de página*

Una acción de página Seam es un evento que ocurre antes de renderizar una página.

El método de acción de página puede devolver un resultado JSF. Si el resultado no es nulo, Seam utiliza las reglas de navegación que se hayan definido para navegar hacia una vista.

- *Parámetros de página*

Una petición a JSF Faces (envío de un formulario) encapsula una acción y los parámetros de entrada. Los parámetros de página pueden utilizarse con o sin acción.

- *Navegación*

Es posible utilizar las reglas estándar de navegación JSF definidas en el faces-config.xml en una aplicación de Seam. Sin embargo, las reglas de navegación estándar de JSF tienen una serie de limitaciones:

- No es posible especificar parámetros de la petición usada para ser usados al momento de redirigir a una página.
- No es posible iniciar o finalizar las conversaciones a partir de una regla de navegación.
- Las reglas de navegación funcionan evaluando el valor de retorno de un método; no es posible evaluar una expresión EL arbitraria.

- *Eventos orientados a componentes*

Los componentes de Seam pueden interactuar tan sólo llamando a sus métodos entre ellos. Los componentes con estado pueden incluso poner en práctica el modelo observador/observable. Pero para permitir que los componentes interactúen de una manera más débilmente acoplada de lo que es posible cuando los componentes llaman a los métodos de los otros directamente, Seam proporciona eventos impulsados por componentes.

- *Eventos Contextuales*

Seam define una serie de eventos que pueden ser usados para tipos especiales de integraciones con el framework. Algunas de ellas son:

- org.jboss.seam.validationFailed: Invocada cuando falla la validación JSF.
- org.jboss.seam.noConversation: Invocada cuando no hay una conversación larga y es requerida.
- org.jboss.seam.postDestroyContext.<SCOPE> : Invocada después de que el contexto (sesión, conversación, etc) es destruido.

- org.jboss.seam.beforePhase: Llamada antes de iniciar una fase JSF.
- org.jboss.seam.security.notLoggedIn: Llamada cuando el usuario no se ha autenticado y se requiere la autenticación.
- *Interceptores Seam*

El ciclo de vida de los interceptores con estado es el mismo del componente interceptado. Para los interceptores que no necesitan tener un estado, Seam deja optimizar el rendimiento mediante una anotación.

Mucha de la funcionalidad de Seam está implementada como un conjunto de interceptores de Seam predefinidos. No es necesario especificarlos explícitamente para todos los componentes que se desarrollen; estos interceptores existen para todos los componentes de Seam que son interceptables.

- *Administrar excepciones*

JSF es limitado en el manejo de excepciones. Por esta razón Seam permite especificar como ciertas clases de excepciones son tratadas bien sea con anotaciones o mediante declaraciones de estas clases en un archivo .xml.

2.4.7.3 Conversaciones y gestión de espacio de trabajo

- *Modelo de conversación Seam*

Seam propaga automáticamente el contexto de la conversación a través de los *postbacks* JSF y los redireccionamientos. Si no se hace nada especial, una solicitud tipo GET, por ejemplo, no propagará el contexto de la conversación y será procesada en una nueva conversación temporal. Por lo general este es el comportamiento deseado.

Si se quiere propagar una conversación Seam a través de una solicitud non-faces (no dirigida al árbol de componentes de JSF), se requiere especificar la identificación de la conversación como un parámetro de la petición.

El modelo de conversación de Seam facilita crear aplicaciones que se comporten correctamente con múltiples ventanas. Por lo general esto es suficiente salvo ciertos requerimientos adicionales de algunas aplicaciones como:

- La conversación se extiende por unidades más pequeñas, que se ejecutan en serie o a la vez.
- El usuario puede alternar en diferentes conversaciones dentro de una misma ventana, esto se llama gestión del área de trabajo.

- *Conversaciones anidadas*

Se crean invocando el método @Begin(nested=”true”) dentro del ámbito de la aplicación de una conversación existente. Una conversación anidada posee su propio contexto de conversación, pero puede leer los valores de contexto de la conversación exterior que en este caso son de solo lectura.

- La anidación de una conversación se inicia en un contexto dentro de la conversación externa, esta es considerada como la conversación padre.

- Los objetos cargados directamente en el contexto de la conversación anidada no afectan los objetos accesibles en la conversación padre.

- La inyección en un contexto de búsqueda en la primera conversación, busca el valor en el contexto de la conversación en curso, si no encuentra ningún valor continúa por la conversación en pila si la conversación es anidada.

La conversación anidada se destruye y se reanuda la conversación externa cuando encuentra la etiqueta @End.

Las conversaciones pueden ser anidadas a cualquier profundidad que se desee.

• *Iniciando conversaciones con la solicitud GET*

JSF no define ningún tipo de detector de acción cuando una página se accede a través de una solicitud non-faces, esto puede ocurrir si el usuario marca la página o si accede a ella a través de un link.

Cuando se desea iniciar una conversación al acceder a una página, dado que no existe un método de acción JSF, no se puede resolver el problema con la anotación @Begin.

Si la página necesita algún estado de la variable de contexto y este se lleva a cabo en un componente de Seam, el estado puede alcanzarse en el método @Create. Si no, puede definirse el método @Create.

Si ninguna de las opciones funciona, Seam permite definir una página de acciones en el archivo pages.xml.

• *Una conversación de larga duración*

Algunas páginas requieren una conversación de larga duración, para esto Seam ha incorporado un mecanismo para cumplir este requisito.

Cuando Seam determina que una página es solicitada fuera de una conversación de larga duración toma las siguientes medidas.

- Un evento contextual llamado org.jboss.seam.noConversation es levantado.
- Se registra un mensaje de alerta con el conjunto de claves org.jboss.seam.NoConversation.
- Si se define el usuario es redirigido a una página alternativa.

2.4.7.4 Navegacion En Seam

- *Modelo de navegación Stateless*

Existen dos formas para definir un modelo de navegación Stateless: (a) por medio de las reglas de Seam o (b) utilizando las reglas de JSF.

El modelo Stateless define un conjunto de salidas lógicas y salidas de nombre de un evento directamente a la página resultante de la vista. Las reglas de navegación son totalmente ajenas a cualquier estado en poder de la aplicación aparte de la página fuente del evento. Esto implica que los métodos de acción oyente (actionlistenermethods) deben en algunos escenarios tomar decisión con respecto al flujo de página, ya que sólo tienen acceso al estado actual de la aplicación.

Cuando una aplicación utiliza la paginación Stateless, Seam le permite al usuario navegar libremente por medio de los botones: *Regresar, Adelante* o *Refrescar*; siempre y cuando la aplicación se responsabilice de permanecer en el estado conversacional internamente consistente cuando se utilizan los botones anteriormente mencionados.

Una ventaja de implementar este tipo de navegación radica en consistir de reglas simples, de ser flexible, y además, de permitir el retorno de IDs de vista desde los métodos de acción oyente. No obstante, la paginación Stateless no soporta procesos de negocios complejos.

- *Modelo de Navegación Stateful*

Define un conjunto de transiciones entre un conjunto de estados lógicos y estados de nombres de la aplicación. Este modelo está orientado a los procesos de negocio en el que es posible establecer el flujo producido por cualquier interacción del usuario, en su totalidad, con la definición del flujo de página de JPDL e incluso escribir métodos de acción oyentes que desconocen por completo el flujo de la interacción.

JPDL es un lenguaje xml simple y de fácil lectura, el cual es instaurado por el Motor de Gestión de procesos de Negocios de Jboss (JBPM). JPDL logra resolver los siguientes problemas:

- Definición del flujo de página involucrado en complejas interacciones de usuario (definir el flujo de página de una conversación en particular).
- Definición de los procesos de negocio globales (cuando los procesos de negocios involucran múltiples conversaciones con múltiples usuarios simultáneamente).

El siguiente diagrama representa en forma básica un ejemplo de paginación Stateful:

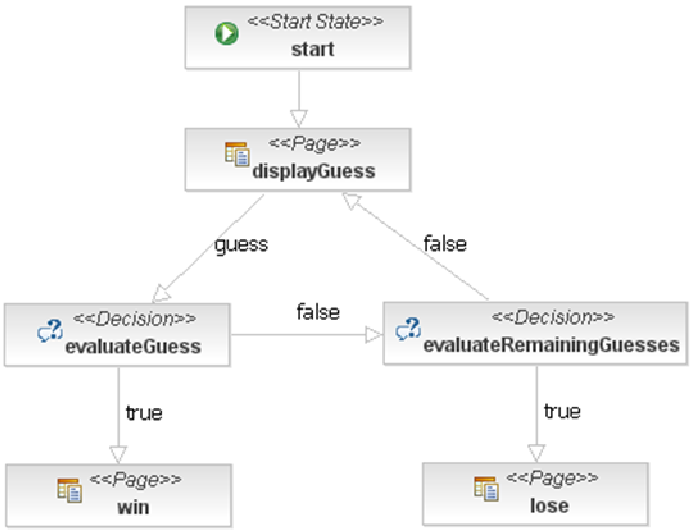

Figura 16 Modelo de Navegación Stateful

A pesar de que la navegación Stateful tiene un diseño para soportar cualquier interacción del usuario, al mismo tiempo es un poco más restringida que la Stateless, ya que todos los posibles eventos deben estar definidos por JPDL y no pueden interactuar con métodos de acción oyente que personalizan las interacciones del usuario sobre la aplicación.

2.4.7.5 Seam Y El Mapeo Objeto-Relacional

Seam provee un extenso soporte para las Arquitecturas de Persistencia más importantes de Java: (a) Hibernate 3 y (b) Java Persistence API presentada por EJB 3.0. Entender cuál es la relación de Hibernate 3 y EJB 3.0 con Seam, o cómo

se integran estos ORM con el Framework es de vital importancia para conocer las ventajas más importantes de Seam.

Una de las razones que impulsaron el desarrollo de Seam fue la dificultad que tenían otros Frameworks como Spring (que utilizaba Hibernate) para persistir los objetos, ya que cada transacción con la base de datos era atómica, es decir, que cada vez que una transacción finalizaba no sólo implicaba una interacción directa con la base de datos sino que también marcaba la pérdida del contexto de persistencia.

Muy pronto los desarrolladores que utilizaron tecnologías anteriores a Seam comenzaron a discrepar con el diseño del contexto de persistencia asociado a una transacción atómica, y es ahí donde Seam y EJB 3.0 surgen como una alternativa para solucionar este problema. Lo que se proponía era cambiar el concepto de Transacción Atómica por el de Transacción Optimista (una transacción que soporta más de una solicitud sin perder el contexto de persistencia vigente).

- *Transacciones gestionadas por Seam*

El ORM EJB 3.0 fue el primero en introducir componentes Stateful (Bean de Sesión Stateful) con un contexto de persistencia extendido asociado al tiempo de vida del componente; pero esta era una solución parcial del problema de la persistencia ya que EJB 3.0 tenía los siguientes inconvenientes:

- El ciclo de vida de un Bean de Sesión Stateful debía ser manejado manualmente por medio de código en la capa Web (un problema sutil que era más difícil en la práctica de lo que parecía).
- La propagación del contexto de persistencia entre componentes Stateful en las transacciones era posible pero difícil.

No obstante Seam resuelve el primer inconveniente asociando los componentes de sesión Stateful a la conversación, pero la segunda falencia en los componentes de EJB, Seam pudo resolverlo trabajando mancomunadamente con Hibernate (otro ORM) cuyos resultados proporcionaron las siguientes soluciones:

- Usar el contexto de persistencia extendido en el ámbito de la conversación, en lugar de asociarlo a la transacción.
- Usar dos transacciones por solicitud: la primera abarca desde la restauración de la fase de vista hasta el final de la invocación de la fase de aplicación; y la segunda transacción abarca la fase de devolución de la respuesta.

- Sincronización de la Transacción

La sincronización de transacciones tiene que ver con la devolución de llamadas (callbacks) que producen los eventos que inician y terminan dichas transacciones. Por defecto, Seam utiliza su propio componente de sincronización de transacciones el cual se usa explícitamente al momento de enviar una petición para que las devoluciones de llamada sean correctamente ejecutadas.

- *Contextos de Persistencia gestionados por Seam*

Cuando se utiliza Seam en otro ambiente diferente a Java EE5 no se puede confiar el manejo del ciclo de vida del contexto de persistencia al contenedor, e incluso, aún trabajando con la plataforma Java EE5 la propagación del contexto de persistencia entre los componentes es difícil y propensa a errores.

De cualquier manera se necesita un “Contexto de persistencia administrado” (especificado por JPA) o una “Sesión administrada” (de Hibernate) en los componentes.

El contexto de persistencia administrado por Seam es justo un componente Seam integrado que maneja una instancia del *EntityManager* (en JPA) o del *Session* (en Hibernate) el cual se puede inyectar en el momento deseado con la anotación *@In.*

Los contextos de persistencia que son gestionados por Seam son extremadamente eficientes en ambientes distribuidos, pues Seam es capaz de realizar una optimización de la especificación EJB3.0 y es que los contenedores se puedan usar para administrar contextos de persistencia extendidos. Seam también puede hacerse cargo de los errores en el contexto de persistencia extendido con la ventaja de llevar a cabo esta tarea en forma transparente en donde no existe la necesidad de replicar cualquier estado contexto de persistencia entre los nodos.

2.4.7.6 Seguridad Seam

La seguridad del API de Seam abarca las siguientes áreas:

- Autenticación: esta capa se basa en JAAS que permite a los usuarios autenticarse en cualquier proveedor de seguridad.

- Administrador de identidad: un API para la gestión de usuarios y sus funciones en tiempo de ejecución.

- Autorización: esta capa incluye apoyo a las funciones de usuario, permisos persistentes basados en reglas y una resolución de autorización para implementar fácilmente la lógica de seguridad.

- Permiso de gestión: conjunto de componentes integrados para facilitar la gestión de la política de seguridad de una aplicación.

- CAPTCHA: para ayudar en la prevención de software automatizado.

- *Desactivación De La Seguridad*

En caso que se desee implementar su propio enfoque de seguridad o por cualquier otro motivo se desee desactivar la seguridad que proporciona Seam simplemente se debe llamar al método estático Identity.setSecurityEnabled (false), que deshabilita la infraestructura de seguridad.

Otra opción para configurar la aplicación es controlando estos parámetros en el archivo components.xml:

- Entidad de seguridad
- Interceptor de seguridad Hibernate
- Interceptor de seguridad Seam
- Página de restricciones
- Servlet API de integración de seguridad

- *Autenticación*

La autenticación basada en JAAS (Java Authentication and Authorization Service) proporciona un API robusto y altamente configurable para manejar la autenticación de usuario. Si se necesita una autenticación menos compleja Seam ofrece un método simplificado de autenticación que oculta la complejidad de JAAS.

- Configurar un componente autenticador

Seam incorpora un módulo JAAS de entrada, SeamLoginModule, que delega la autenticación a los propios componentes de Seam; este módulo ya viene configurado y no requiere ningún archivo de configuración adicional. Permite escribir un método de autenticación utilizando las entidades de su aplicación, o, para autenticar con algún proveedor.
La configuración simplificada requiere el componente de identidad que se configura en el archivo components.xml.

- Escribir un método de autenticación

Este método no toma parámetros, y devuelve un booleano que indica si la autenticación fue realizada con éxito o no.

Cualquier rol que corresponda al usuario debe ser asignado mediante Identity.addRole ().

Para evitar que el método de autenticación sea invocado varias veces durante una única solicitud, cualquier código que deba ejecutarse en una autenticación debe ser escrito implementando un observador de eventos.

- Identity.addRole ()

Este método se comporta de manera diferente dependiendo de si las sesiones actuales se autentican o no. El siguiente diagrama de secuencia representa la lista de roles pre autenticados como un objeto de primera clase, para mostrar más claramente el proceso de autenticación.

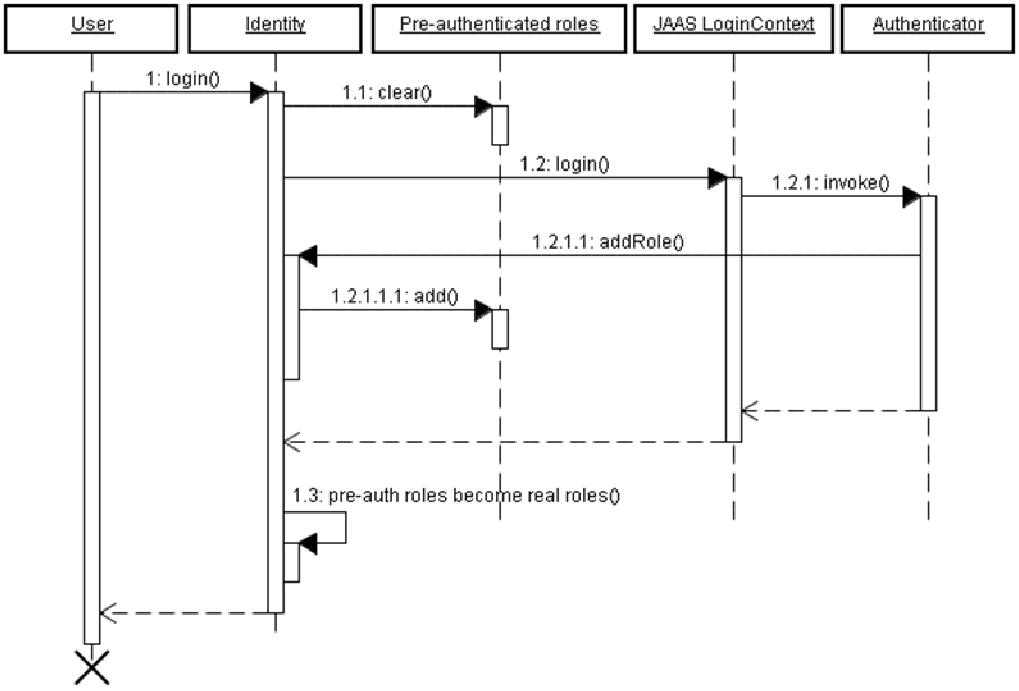

Figura 17 Diagrama de Secuencias Autenticación

- Manejo de excepciones de seguridad

Con el fin de evitar que los usuarios vean la página de error por defecto, se recomienda que en pages.xml se configure para redirigir los errores de seguridad a una página especial. Los principales tipos de excepciones producidas por el API de seguridad son:

- NotLoggedInException: se produce cuando el usuario intenta acceder a una acción restringida o cuando no se ha autenticado.

- AuthorizationException: se produce cuando el usuario ya está conectado y ha intentado acceder a una acción restringida para su rol.

- Redirección de entrada

Si un usuario no autenticado intenta acceder a una acción restringida, Seam puede configurarse para redirigirlo a una página de inicio de sesión y una vez el usuario se haya autenticado lo devuelve automáticamente a la página que intentaba acceder.

- Autenticación HTTP

Su uso no es recomendado a menos que sea absolutamente necesario, Seam proporciona los medios para la autenticación utilizando HTTP básico o HTTP Digest(RFC 2617). Para cualquier forma de autenticación el filtro de autenticación debe estar habilitado en components.xml.

- *Escribir un autenticadorDigest*

Si usa la autenticación implícita, la clase autenticador debe ampliar la clase abstracta org.jboss.seam.security.digest.DigestAuthenticatory, y el uso de validatePassword () para validar la contraseña del usuario.

- Características avanzadas de autenticación

Estas características son proporcionadas por el API de seguridad para abordar requisitos de seguridad más complejos.

- *Configuración del contenedor JAAS*

Si delega la seguridad en el sistema por defecto, proporciona una propiedad JAAS-config-nombre en components.xml. Por ejemplo, si está utilizando JBoss AS y desea utilizar la política de otros (que utiliza el módulo de entrada

UsersRolesLoginModule proporcionada por JBoss AS), la entrada en components.xml se vería así:

<security:identity jaas-config-name="other"/>

- Gestión de identidad

Seam proporciona un estándar API para la gestión de usuarios de acuerdo al rol que desempeñe cada uno, el centro de gestión de identidad es el componente identityManager, que proporciona los métodos para crear, modificar y eliminar usuarios, concediendo y eliminando funciones.

2.4.7.7 Almacenamiento En Caché De Seam

En la mayoría de las aplicaciones empresariales, la base de datos es el principal cuello de botella como también es la capa menos escalable en el entorno de ejecución. En conclusión es casi imposible que una aplicación sea escalable mientras la interacción con la base de datos sea ostensible. Por lo tanto la escalabilidad de cualquier aplicación se puede favorecer si se disminuyen las transacciones directas con la base de datos, y esa es la principal misión de la caché.

- *Almacenamiento en Caché multi-capa*

Con Seam se puede planificar para configurar un almacenamiento en una caché de manera individual para cada una de las capas de la aplicación.

- *Caché de la Base de Datos*

La base de datos tiene su propia caché asignada pero a diferencia de la caché en la capa de Aplicación no es tan escalable.

- *Caché de segundo nivel*

Independientemente del ORM que se emplee, en una aplicación Seam se dispone de una caché de segundo nivel de los datos de la base de datos. Su diseño la hace favorable en ambientes distribuidos en donde múltiples usuarios podrían utilizar los datos de esta caché siempre y cuando las modificaciones en dichos datos sean muy pocas.

- *Caché a nivel de Contexto*

El contexto de conversación en Seam es una caché del estado conversacional. Los componentes que son inyectados en el contexto de conversación pueden mantener almacenado el estado relacionado con la interacción del usuario actual.

En particular, el contexto de persistencia actúa como un caché de los datos que han sido leídos en la conversación actual. Seam optimiza las respuestas de sus propios contextos de persistencia en un ambiente distribuido y no hay ningún requisito para mantener la consistencia transaccional con la base de datos.

Las aplicaciones pueden guardar en estado transaccional el componente *cacheProvider* (Proveedor de caché) de Seam y este estado puede ser visible si la caché soporta el trabajo en un ambiente clusterizado. También pueden almacenar un estado no transaccional en el contexto de aplicación de Seam, el cual es invisible para los otros nodos del clúster.

- *Caché a nivel de JSF*

Seam permite el almacenamiento en caché de los fragmentos recargados de una página JSF. A diferencia del segundo nivel de caché del ORM, este caché no es automáticamente inhabilitado cuando los datos cambian, así que su invalidación se debe ser explícita en el código de la aplicación o establecer las políticas de expiración apropiadas.

2.4.8 EJB 3.0

Enterprise JavaBeans (EJB) es una arquitectura de componentes para la construcción de aplicaciones empresariales ejecutadas en servidores. Tiene por propósito proveer una forma estándar de implementar este tipo de aplicaciones, haciéndose cargo de aspectos comunes y repetitivos como la persistencia, la integridad transaccional y la seguridad, permitiendo que el desarrollador pase a preocuparse exclusivamente por la lógica del negocio en sí.

JBoss AS fue de los primeros servidores de aplicaciones en adoptar las especificaciones de EJB 3.0. Este modelo de EJB simplifica el desarrollo eliminando la necesidad de una interfaz “Home” y descriptores de despliegue (reemplazándolos por anotaciones), y facilita la implementación de la persistencia de una nueva manera por medio de JPA.

2.4.9 JPA

Más conocida por su sigla **JPA**, es la API de persistencia desarrollada para la plataforma Java EE e incluida en el estándar EJB3. Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional. El objetivo que persigue el diseño de esta API es no perder las ventajas de la

orientación a objetos al interactuar con una base de datos, como sí pasaba con EJB2, y permitir usar objetos regulares.

Una característica fascinante de JPA, es que permite que se hagan cambios al diseño de la base de datos sin tener que reescribir enteramente las aplicaciones. Para esto JPA introduce JPQL.

2.4.10 JPQL

Java Persistence Query Language es el lenguaje de consultas que se usará en el desarrollo de este trabajo. Con él, el desarrollador jamás se refiere a las entidades directamente al momento de hacer las consultas, sino a los objetos mismos que fueron mapeados. Si el diseño de la BD cambiara, sólo habría que modificar las anotaciones de las entidades. El resto del código quedaría intacto y seguiría funcionando tal y como antes. Esta facilidad permite optimizar las bases de datos cuando se considere necesario, migrar a bases diferentes, o sencillamente corregir diseños defectuosos con un esfuerzo mínimo.

CAPITULO 3

3. METODOLOGÍA DE DESARROLLO

3.1Ciclo de vida del proyecto

A continuación se realiza una descripción de las diferentes actividades que se llevaron a cabo durante el transcurso del proyecto, para definir la metodología que se aplicó en el desarrollo del nuevo sistema de información, orientado a la web, soporte para el proceso del servicio de cafetería ofrecido por Bienestar Universitario.

3.1.1 Análisis de Requerimientos:

En el análisis de requerimientos se especificó, junto con los funcionarios de Bienestar Universitario de la Universidad Industrial De Santander, la función y comportamiento que debe tener el sistema de información a desarrollarse en el transcurso del proyecto, se indicó la interfaz con otros elementos del sistema, se estableció la integración con el sistema de información institucional y se fijaron los estándares de diseño que debe cumplir el sistema.

El análisis de requerimientos permite la representación de la información y las funciones que pueden ser traducidas en datos, arquitectura y diseño procedimental. Finalmente, la especificación de requerimientos suministra los medios para valorar la calidad de los programas, una vez que se haya construido.

Es en esta etapa donde se hicieron reuniones periódicas con los funcionarios de los servicios de Cafetería de Bienestar Universitario de la Universidad Industrial De Santander, para que fueran ellos los que definieran las características y

alcances del software que se iba a desarrollar y que se adaptará plenamente a las exigencias de los procesos de Bienestar Universitario de la Universidad Industrial De Santander.

3.1.2 Diseño:

El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos: estructura de datos, arquitectura de software, representaciones de interfaz y detalle procedimental (algoritmo).

En esta etapa de diseño se hace una traducción de los requisitos a una representación del software donde se pueda evaluar su calidad antes de que comience la codificación.
El diseño se efectuó, mediante modelos UML (Lenguaje de Modelado Unificado) que incluyó los diagramas que han sido seleccionados dentro de los estándares de desarrollo de software utilizados en la División de Servicios de Información, que son: de casos de uso, de clases y de secuencia, utilizando la herramienta de modelaje Enterprise Architect.

3.1.3 Implementación de la Aplicación:

En esta etapa se procedió a generar el software que se ha diseñado, teniendo en cuenta los parámetros establecidos por la división de Servicios de Información, en cuanto a los estándares técnicos y de calidad que caracterizan las aplicaciones que son generadas para el servicio de la Universidad, teniendo como base el Lenguaje de programación JAVA 5 y la plataforma Informix como motor de base de datos.

3.1.4 Pruebas del software:

Corresponden a los procesos que permiten verificar la calidad de un producto software y el cumplimiento de los requerimientos establecidos en la fase de análisis de requerimientos.
Las pruebas de software se integran dentro de las diferentes fases del ciclo de desarrollo del software establecidas en la ingeniería de software.
Una vez generado el código, comenzaron las pruebas, proceso utilizado para identificar posibles fallos de implementación, calidad, o usabilidad de un programa. Las pruebas se centraron en los procesos lógicos internos del software, asegurando que todas las sentencias sean probadas y garantizando que la entrada definida produce los resultados esperados.

Las pruebas fueron de carácter permanente a lo largo del desarrollo del proyecto por parte del equipo de trabajo, y hubo un periodo de tiempo para que los usuarios finales interactuaran con la aplicación y detectaran posibles ajustes.

3.1.5 Ajustes

Después de las pruebas, cada uno de los errores detectados o las observaciones hechas por los usuarios, debidamente analizadas, fueron tenidos en cuenta para ajustar el sistema, de tal manera que se adaptara plenamente a los requerimientos establecidos.
También estos se hicieron permanentemente a lo largo del desarrollo del proyecto, a la par con la realización de las pruebas, en la medida en que se detectaron errores o inconsistencias en el software que se había desarrollado.

3.2Metodología De Desarrollo

Teniendo como base las actividades anteriormente descritas, se propuso como metodología de desarrollo de éste proyecto el **MODELO DE CONSTRUCCIÓN POR PROTOTIPOS.**

Se eligió esta metodología debido a que es muy frecuente que los usuarios que están solicitando el sistema, en este caso la sección de Cafetería de Bienestar Universitario de la Universidad Industrial De Santander, definan un conjunto de objetivos generales para el software, pero no identifican los requisitos detallados de entrada, proceso o salida. En otros casos, el responsable del desarrollo del software puede no estar seguro de la eficacia de un algoritmo, o no haber comprendido plenamente el requerimiento del usuario. Para éstas y otras muchas situaciones, un *paradigma de construcción por prototipos* puede ofrecer el mejor enfoque, ya que la entrega de prototipos, que hacen parte integral del proyecto en su conjunto, permitirán la corrección temprana de errores o la redefinición del sistema en caso de ser necesario, y los prototipos funcionales permiten la familiarización del usuario con el sistema que se está desarrollando.

A continuación se observa la estructura del MODELO:

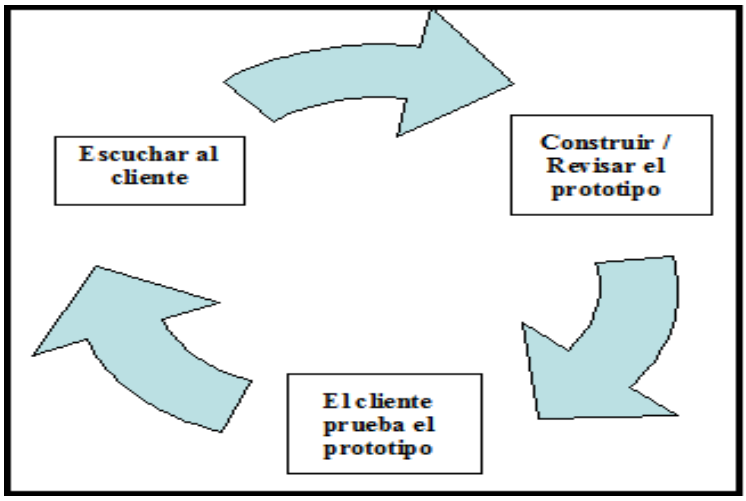

Figura 18 Estructura (Ciclo)

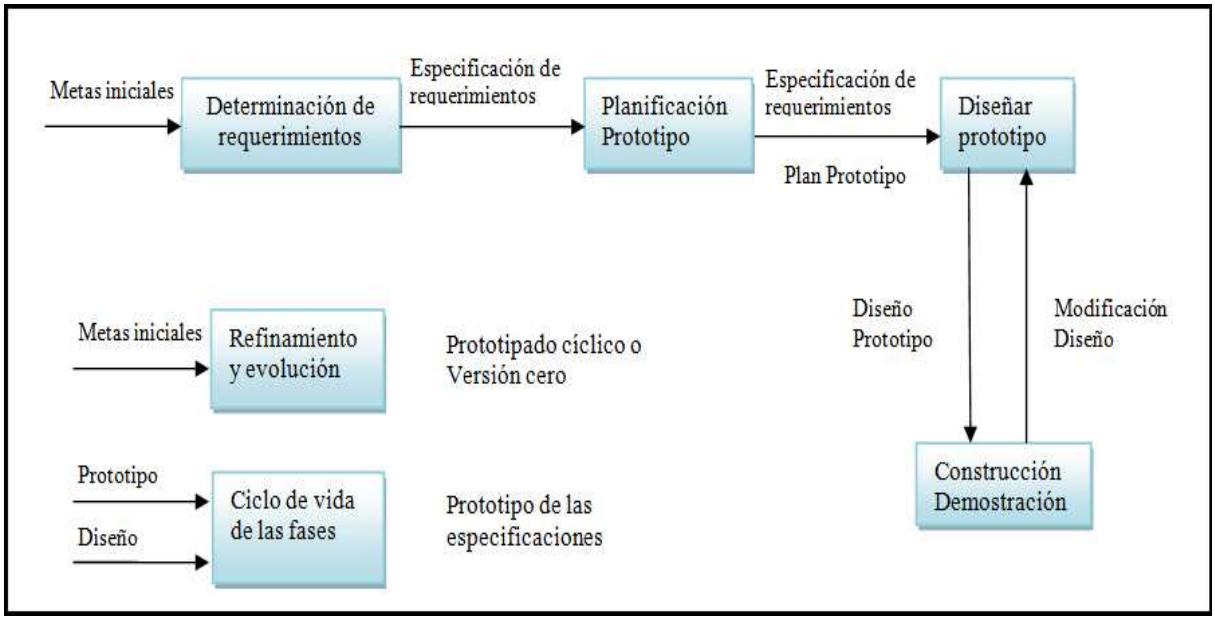

Figura 19 Estructura (Versión imagen)

Esta metodología es viable para el desarrollo del proyecto debido a:

- En la creación de los prototipos iníciales se debe trabajar con unas ideas aproximadas de lo que desea Bienestar Universitario de la Universidad Industrial De Santander relacionado con el servicio de cafetería, para presentar un producto o prototipo inicial, el cual puede evolucionar y refinarse dando como resultado un prototipo más maduro, que cumpla con todos los requerimientos de los usuarios.

- Con el uso del modelo de prototipos se da la facilidad de mejorar, de manera temprana los prototipos, teniendo en cuenta las sugerencias del usuario solicitante del proyecto, de tal forma que se cubran a cabalidad sus requerimientos.

- En este sistema de desarrollo se debe validar la versión actual del prototipo, para proceder a generar una nueva versión que contemple los nuevos requerimientos, con el fin de evitar retrocesos en el proceso de desarrollo.

3.2.1 Procedimiento a seguir para la metodología planteada:

a. La construcción de prototipos comenzó con la recolección de los requisitos, con los encargados del servicio de Cafetería de Bienestar Universitario de la Universidad Industrial De Santander.

b. El grupo de desarrollo y el usuario se reunieron y definieron los objetivos globales del software, identificaron todos los requisitos conocidos y perfilaron las áreas en donde sería necesaria una mayor definición.

c. Se produjo el diseño del prototipo que se enfocó sobre la representación de los aspectos del software visibles al usuario (por ejemplo, métodos de entrada, esquemas de navegación y formatos de salida). En ésta etapa el usuario adquirió mayor claridad de las funcionalidades del sistema y dio sugerencias necesarias para la construcción de un prototipo más avanzado.

d. Se procedió con la construcción del prototipo que se ha propuesto.

e. El prototipo fue evaluado por el usuario y se utilizó para refinar los requisitos del software a desarrollar.

f. Se produjo un proceso iterativo en el que el prototipo es “afinado” (Refinamiento del prototipo) para que satisficiera las necesidades del usuario, al mismo tiempo que facilitó al desarrollador una mejor comprensión de la labor a realizar , permitiendo entregar el producto final requerido.

3.3Plan De Trabajo

El plan de trabajo del proyecto fue definido por FASES que son descritas a continuación y que con éstas pudo ser medido el nivel de avance alcanzado.

3.3.1 *Fase Previa: De Acoplamiento*

- Esta fase consistió en la realización de reuniones previas del usuario con el grupo desarrollador, para dar a conocer los alcances del proyecto y definir los objetivos.

- Se realizó el planteamiento de prioridades sobre las cuales se iba a trabajar y se estudió la metodología y estándares a ser utilizados en el proyecto establecidos por la División de Servicios de Información.

3.3.2 Primera Fase: Recolección Y Análisis De Requerimientos.

- Se procedió con reuniones periódicas con los funcionarios de Bienestar Universitario de la Universidad Industrial de Santander, con el objetivo de establecer las necesidades que se requerían satisfacer con el desarrollo del proyecto.

- En esta Fase se mostró todo lo que el sistema debe hacer más todas las restricciones sobre la funcionalidad que se deben tener en cuenta.

- Se especificaron el conjunto completo de resultados a ser obtenidos al utilizar el sistema.

- La importancia de esta fase radicó en que se mostró a los desarrolladores y a los usuarios qué se necesitaba del sistema, en un lenguaje que todos comprendieran, por lo tanto, fue primordial la excelente comunicación entre usuarios y desarrollador.

- Al terminar esta fase se tuvo un modelo completo que represente el sistema total en algún nivel de abstracción mental que puede ser especificada a nivel de detalle en un documento denominado “Especificación de Requerimientos”.

3.3.3 Segunda Fase: Diseño Del Prototipo

- Teniendo la especificación de los requerimientos en un lenguaje natural a nivel de detalle, se procedió entonces a abstraer esa información y traducirlo en un lenguaje de modelado unificado (UML).

- Dentro de ésta fase se definió la interfaz del usuario y la interfaz de comunicación entre los programas.
- En esta etapa se evaluó el grado de aceptación y satisfacción de los usuarios frente a los modelos de entrada y salida de datos y la interfaz de usuario propuesta.

- Durante el proceso de refinamiento del prototipo se llegó a esta fase para realizar las mejoras identificadas en los diseños planteados en el primer prototipo.

3.3.4 Tercera Fase: Desarrollo Del Prototipo.

- Durante esta fase se realizó la programación de los requerimientos que habían sido modelados previamente.

3.3.5 Cuarta Fase: Refinamiento Del Prototipo

- Dentro de esta fase se llevaron a cabo la corrección de las fallas que surgieron tras la realización de las pruebas funcionales de las secciones de automatización de los módulos o de la realización de la evaluación por parte de los usuarios en cuanto al prototipo y su aceptación o de nuevas sugerencias para un mejoramiento o maduración del prototipo.

- Se realizaron los ajustes y refinación del funcionamiento de los servicios, regresando a las fases de diseño y desarrollo hasta obtener el producto final.

3.3.6 Quinta Fase: Implantación Y Entrega Del Producto Final

- Se Implementó el sistema de Información, orientado a la Web, soporte para el proceso del servicio de cafetería ofrecido por Bienestar Universitario.

- Se Capacitó a los Usuarios en el funcionamiento del sistema que se estaba entregando.

4 APLICACIÓN DE LA METODOLOGÍA

4.1 Levantamiento De Requerimientos

Se realizaron reuniones periódicas con el cliente DBU, en las cuales se planteó la idea general de lo que se requería y se fueron estableciendo los requerimientos, a medida que se avanzaba en las reuniones se logró definir todas las funcionalidades que se deseaban para comenzar con el análisis y diseño del sistema de información, consiguiendo de esta manera un desarrollo óptimo para el cliente

4.1.1 Descripción General

Cuando se tuvo conocimiento del entorno en el que se iba a desarrollar el sistema e identificados los principales problemas, se planteó una solución partiendo de los requisitos o condiciones extraídas en el levantamiento de requerimientos.

Se propuso una solución para evitar los problemas que se tienen en la cafetería de BU, debido a la forma que se implementa el control en sus procesos ya que estas actividades se hacen de manera manual, dando la oportunidad a pérdida de información la cual en estos tiempos es considerada como un activo más en las organizaciones.

Por lo cual se propuso la oportunidad de tener un sistema completamente orientado a la web, donde se pueda contar con toda la información en línea y totalmente actualizada, y de esta forma solucionar de forma más oportuna los inconvenientes presentados al no contar con un sistema de información que administre los procesos realizados en cafetería.

4.1.2 Funciones Del Sistema

El sistema de información de cafetería de Bienestar Universitario está compuesto por los módulos de pedidos, ventas y devoluciones, turnos, inventarios físicos, productos, reservas y tablas soporte. Todos estos módulos permiten al administrador gestionar los inventarios de los diferentes puntos de venta de forma eficiente y eficaz y la reserva de menús por parte de las unidades académico administrativas.

A continuación se describe la funcionalidad de cada módulo:

1. **Productos**: Este módulo permite administrar los inventarios de los productos no preparados y menús programados en los diferentes puntos de venta.

   - Administrar productos cafetería : Filtra de todos los elementos que tiene disponible Bienestar Universitario los que se van a vender en las diferentes cafeterías asociándole características adicionales como la agrupación , estado de vigencia y valor del IVA si hubiera lugar.

   - Administrar productos punto de venta: Asigna a cada punto de venta los productos a vender en la misma asociándole características como el precio, la existencia mínima y cantidad del mismo en el punto de venta. Esto permite administrar el inventario de los productos por punto de venta.

   - Trasladar productos: Permite realizar traslados de productos entre los diferentes puntos de venta, ofreciendo una posibilidad distinta a la de

realizar un pedido y esperar hasta que llegue, logrando suplir una escases de algún producto agotado en un punto de venta.

- Dar de baja: Este modulo brinda al jefe del punto de venta la posibilidad de descontar del inventario cualquier producto, debido a alguna circunstancia presentada como por ejemplo un saqueo o robo, la cual cree un desfase entre el inventario real y el inventario que tiene el sistema, logrando de esta manera tener un inventario totalmente ajustado a la realidad.

- Asignar menús programados a los puntos de venta: A través del sistema SIMIN el jefe de Bienestar Universitario programa una cantidad de menús para cada semana disponibles para la venta, es el jefe del punto de venta la persona encargada de distribuir dicha cantidad entre los puntos de venta que desee.

2. **Ventas y devoluciones**: Permite realizar la venta de productos no preparados, menús programados y cancelar anticonceptivos de la farmacia de Bienestar Universitario. También permite realizar devoluciones de productos no preparados y menús programados por diferentes causas que hubiera lugar.

- Arqueo de los puntos de venta: Permite controlar los estados financieros de cada punto de venta al término de una jornada, permitiendo generar un reporte de los dividendos exactos con los que debe contar el cajero al culminar su turno.

3. **Pedidos**: El sistema ofrece la posibilidad de realizar 2 tipos de pedidos, un pedido que requiere una aprobación de las cantidades previamente solicitadas y está sujeto a la misma, la segunda es un pedido el cual no requiere aprobación previa de los productos solicitados y sirve para satisfacer la demanda de un producto de forma inmediata.

Estas dos modalidades permiten abastecer el inventario de los puntos de venta.

4. **Control de inventarios**: Mantener un inventario lo más ajustado a la realidad es una de las necesidades más importantes para el negocio. Para esto el sistema ofrece la posibilidad que realizar inventarios periódicos por parte de algún empleado los cuales serán revisados por el jefe del punto de punto o administrador quien procederá a investigar y aceptar posteriormente dicho control de inventarios ajustando así el inventario de los productos auditados.

5. **Turnos**: El sistema de información de Cafetería de Bienestar Universitario ofrece un apoyo completo en la administración del personal que laborar en los puntos de venta. Este módulo permite gestionar el personal de la siguiente forma :

    a. Primero se crea el personal a laborar en cada punto de venta

    b. Posteriormente al empleado se le asignan los posibles roles que puede desempeñar en los diferentes puntos de venta. Es importante aclarar que dichos roles solo son una ayuda que permiten al jefe de punto de venta conocer las funciones que desempeña cierta persona en su lugar de trabajo pero no corresponden a los roles que puede desempeñar dicha persona dentro del sistema de información.

    c. Luego se procese a asignar un turno al empleado en un punto de venta asociándole un rol de los posibles asignados previamente, una fecha de inicio y fin y hora de inicio y fin.

    d. Si un empleado requiere un permiso laborar o presenta una excusa que no le permita realizar su turno, este puede ser reemplazado a través de la funcionalidad de turnos extras la cual permite asignar un empleado

que tenga el mismo rol y posea la disponibilidad de tiempo para realizar dicho turno.

6. **Reservas**: Brinda la posibilidad de consultar los productos no preparados, menús programados o especiales con su respectiva información nutricional y precio a la comunidad universitaria en general. Son los jefes de cada unidad las personas que puede reservar dichos menús para un evento o reunión.

Este módulo es muy dinámico gracias a que permite conocer el estado de la solicitud realizada. El sistema enviará un correo electrónico al jefe de la unidad en el caso que algún menú no se pueda despachar para que este a su vez ingrese y modifique la reserva.

7. **Tablas soporte**:

Este modulo permite alimentar el sistema con la información necesaria para el funcionamiento adecuado del mismo, logrando tener un sistema muy dinámico en el cual el administrador no dependerá del desarrollador para realizar cambios en los parámetros establecidos inicialmente. Dado que los valores creados por el administrador pueden cambiar a medida que sea necesario.

Las tablas soporte manejadas dentro del sistema de cafetería son:

- Punto de venta: Tabla soporte utilizada para crear los diferentes puntos de venta que maneja el BU

- Agrupación cafetería: En esta tabla soporte se ingresan los diferentes grupos en los que se van a clasificar los elementos de cafetería, como por ejemplo: lácteos, gaseosas, galletas etc.

- Causa de devolución: Son las posibles causas por las cuales se pueden devolver un producto de cafetería después de realizada una venta ya sea directa o por reservas, como por ejemplo: vencido, dañado etc.

- Estado del pedido: Posibles estados en los cuales se puede encontrar un pedido o reserva dependiendo la etapa donde se encuentren, por ejemplo: por aprobar, aprobado, entregado etc.

- IVA cafetería: Permite crear los diferentes valores de IVA que puedan aplicárseles a los productos.

- Personal: Permite crear el personal que labora en BU.

- Rol: Permite crear todos los posibles roles que puede desempeñar el personal que trabajo en B.U. Estos roles no corresponden a los roles del esquema de seguridad. Estos roles representan las funcionen que desempeña un trabajador.

8. **Consultas**: Contiene diferentes tipos de consultas que permitirán al administrador monitorear y gestionar el personal y el inventario en cada punto de venta.

4.2Estándares De La División De Servicios De Información.[6]

4.2.1 Aspectos Generales

- Interfaz de desarrollo

El IDE de desarrollo a utilizar es el JBoss Developer Studio, el cual debe ser instalado en la carpeta por defecto del instalador.
Este y todos los programas necesarios se pueden descargar, por el personal autorizado, del equipo establecido para tal fin.

- Servidor de aplicaciones

En cuanto al servidor de aplicaciones de desarrollo se debe utilizar el mismo que se encuentra en los servidores de producción y desarrollo, el cual debe ser instalado en la carpeta C:\jboss-5.0.0.GA.

- JAVA

La versión del compilador de JAVA debe ser la 1.5, la cual debe ser instalada en el directorio C:\java1.5.

- SEAM

La versión del seam a utilizar es la 2.1.2.GA. (jboss-seam-2.1.2.GA.zip).

Espacio de trabajo

El espacio de trabajo se debe crear en C:\workspace.

[6] Fuente: Estándares de la División de Servicios de Información de la Universidad Industrial de Santander.

El nombre del proyecto para los JPA debe estar conformado de la siguiente manera:

[Sistema]Entidades

Por ejemplo: AcademicoEntidades (La primera letra de cada palabra en mayúscula).

El repositorio para los JPA debe estar conformado de la siguiente manera: [Sistema]JPA

Por ejemplo: AcademicoJPA (La primera letra de cada palabra en mayúscula). El Server name en el IDE de desarrollo se debe llamar JBoss 5 y el nombre del JBoss Runtime Enviroment se debe llamar JBoss 5.

- Servidor de versiones

Para su configuración se debe instalar en el JBoss Developer Studio los siguientes paquetes:

- subclipse-site-1.4.7
- ajdt_1.6.1a_for_eclipse_3.4
- org.tmatesoft.svn_1.2.1.eclipse

Una vez instalados se debe solicitar el nombre de usuario y la contraseña al Ingeniero encargado.

Cualquier inquietud acerca de la instalación y configuración del servidor de versiones, dirigirse con el Ingeniero encargado.

También se debe instalar el siguiente programa: TortoiseSVN-1.5.8.15348-win32-svn-1.5.5.msi para conectarse con el servidor de versiones de la documentación.

- Plantillas, estilos, imágenes y formateador

Descargar del servidor de versiones de documentación las carpetas Estilos, Plantillas e Imágenes y copiarlas en la careta view de la aplicación.

- Patrones a utilizar

Los patrones a utilizar corresponden a cada una de las capas implementadas:

- Capa de presentación: Modelo Vista Controlador, el cual es implementado por Java Server Faces (JSF).
- Capa de lógica de negocio: Session Façade
- Capa de persistencia: Entity Access Object (EAO)

- Rich Faces

La versión a utilizar es la incluida en el jboss-seam-2.1.2.

- Código HTML

Está prohibido el uso de etiquetas HTML en las páginas.

Anotaciones en la entidad

Las anotaciones en la entidad se deben colocar antes del método get correspondiente y no en la declaración del atributo.

- Llaves compuestas

Se debe utilizar la anotación EmbeddedId, la cual obliga a declara un objeto del tipo de la llave dentro del EJB de entidad.

- HashCode e Equals en EJB de entidad

Se debe utilizar únicamente los campos que conforman la llave primaria. En el caso de las llaves compuestas, el atributo que hace referencia a ésta.

- JPA externos

Para utilizar los JPA externos (academico.jar, recursos-humanos.jar, etc), éstos deben copiarse en la carpeta raíz. En el caso de Windows C:\, para Linux \).
En cada uno de los archivos persistence.xml de su aplicación, se debe utilizar

<jar-file>file:/jpa.jar</jar-file>.
Por ejemplo:
<jar-file>file:/academico.jar</jar-file>
<jar-file>file:/recursos-humanos.jar</jar-file>
<jar-file>file:/general-UIS.jar</jar-file>

- Servicios

Para crear un servicio se debe tener en cuenta las siguientes reglas:
- La interfaz debe contener la anotación Remote

- La implementación de la interfaz debe contener la anotación RemoteBinding con su respectivo nombre jndi (igual al utilizado por la anotación Name). Por ejemplo:

@RemoteBinding(jndiBinding = "ConsultarGenerales")

Para utilizar el servicio se debe utilizar la anotación EJB indicando el nombre jndi. De acuerdo al ejemplo anterior sería:
@EJB(mappedName = "ConsultarGenerales")

4.2.2 Documentación de los Diagramas de Diseño

- Casos de Uso.

Para el desarrollo del modelo de casos de uso, se debe realizar un diagrama de casos de usos por módulo del sistema a implementar siguiendo el estándar propuesto por el Lenguaje Unificado de Modelado 2.1 (UML). Se deben tener en cuenta los siguientes los siguientes puntos:

- Identificación de Actores

Se identifican con el rol que desempeñan en el sistema.

- Diagrama de Casos de Uso

El diagrama de casos de uso se tiene que realizar con la herramienta Enterprise Architect. Cada caso de uso constituye un flujo completo de eventos especificando la interacción que toma lugar entre el actor y el sistema.

Casos de Uso

Se deben identificar con una acción. Los verbos que se pueden utilizar se encuentran al final del documento.

La información mínima requerida por cada caso de uso es la siguiente:

- Descripción completa
- Precondiciones y postcondiciones
- Descripción del escenario básico y alternos
- Diagrama de clases
- Si el caso de uso es complejo se debe incluir:
- Diagrama de secuencia y/o diagrama de actividades

4.2.3 Sintaxis de Nombres en Java

Reglas de sintaxis generales

- En esta sección se especifican las reglas de sintaxis generales para todos los identificadores (nombres de variables, clases, métodos, etc.)
- Todos los nombres de los identificadores deben estar en español.
- Siempre se deben utilizar nombres que sean claros, concretos y libres de ambigüedades. Usando palabras completas evitando acrónimos y abreviaturas.
- Los nombres deben estar definidos sin espacios en blanco, sin guiones (_ , -), ni comillas (“,’), sin operadores (+, - , / , *), sin tildes, utilizar la n en vez de la ñ y sin caracteres especiales.
- No se debe utilizar la mayúscula para diferenciar entre identificadores distintos. Ejemplo: contador y Contador.
- No se deben diferenciar dos identificadores solo con numerales en cualquier posición. Ejemplo: contador1, contador2, 1contador, 2contador.

- Las siguientes partículas están prohibidas en la declaración de los nombres de identificadores: artículos (el, la, los, unos, unas, un), determinantes demostrativos (este, ese, aquel, aquellos), cardinales (uno, dos, etc.), pronombres de cualquier tipo (yo, tu, el, me, te, se, este, ese, mi, tu, su, etc.).
- Se deben utilizar máximo 5 palabras por nombre, las 3 primeras palabras van completas, a partir de la cuarta palabra se quitan las vocales a la palabra exceptuando la última vocal y la primera si la palabra empieza por vocal. No se utiliza ningún separador entre las palabras, se separa cada palabra utilizando su primera letra en mayúscula. Ejemplo: hacerMantenimientoConsultaUsros, hacerMantenimientoAsignaturasCntxto.

Clases

- Las clases con lógica de negocio y con acceso a la capa de presentación debe llevar el estado @Stateful
- Todos los tipos de datos primitivos, objetos y estructuras declaradas a nivel de clase, deben llevar el modificador de visibilidad prívate. Además deben ser ordenados de la siguiente manera
    - Interfaz de servicios
    - Tipos de datos primitivos
    - Tipos de datos objeto
    - Tipos de datos entidad
    - Listas
    - Clases HTML (si se usan)
    - Constructores

- Los métodos de la clases deben ser ordenados de la siguiente manera:
    - Métodos privados
    - Métodos públicos
    - Métodos get/set de Objetos

- Métodos get/set de Estructuras

- Los nombres de las clases deben iniciar siempre en mayúscula, deben ser simples y descriptivos.
- Los nombres de las clases de Entidad deben ser sustantivos en singular.
- Para las clases de Entidad siempre se debe definir la anotación @Table que indica la tabla de la base de datos relacionada para su persistencia. Además se debe utilizar el parámetro name de la anotación @Entity para identificar el EJB (@Entity(name = “xxx”)). El nombre de la clase puede ser distinto al nombre de la tabla y debe seguir el estándar de identificadores mencionado en el numeral anterior.
- Los nombres de los EJB utilizados por el patrón Session Façade está conformado por un verbo autorizado (Ver anexo de verbos). Además debe incluir al final las letras “EJB”. Ejemplo: RegistrarMatriculaEstudianteEJB.
- La interfaz asociada al EJB debe llevar el mismo nombre del EJB sin el sufijo “EJB”. Ejemplo: RegistrarMatriculaEstudiante.
- Para los EJB de entidad, cuando la clase representa una relación entre dos entidades, el nombre se forma uniendo los nombres de las entidades involucradas.
- Los nombres de las clases que representan excepciones terminaran siempre con el sufijo “EXCEPCION”.
- El nombre de la clase no contendrá detalles sobre la implementación interna de la misma. Por ejemplo ArrayEstudiantes no es nombre válido.

Métodos

- Se deben utilizar los verbos autorizados para su codificación.
- El nombre de los métodos debe iniciar con minúscula la primera palabra, las siguientes palabras inician en mayúscula, sin separadores.

- La primera palabra del nombre de los métodos debe ser un verbo en infinitivo y debe representar una acción o comportamiento de la clase.
- El nombre del método debe describir claramente el comportamiento del mismo.
- En lo posible no se deben usar verbos genéricos aplicables a todo como: procesar, gestionar, manejar. Ejemplo: procesarEstudiante(), gestionarCliente(), en este caso el verbo no aclara el cometido real del método.

Paquetes

El nombre de los paquetes debe iniciar con minúscula la primera palabra, las siguientes palabras inician en mayúscula, sin separadores.

- La estructura de los paquetes es la siguiente:
  co.edu.uis.[sistema].[aplicación].[módulo].[caso de uso]
  Ejemplo:co.edu.uis.financiero.egresos.contratacion.ordenarPrestacionServicio.
- La estructura para el paquete donde van a estar las entidades comunes para todos los sistemas es el siguiente:
  co.edu.uis.sistema.entidades
- La estructura para el paquete donde van a estar los servicios comunes para todos los sistemas es el siguiente:
  co.edu.uis.sistema.servicios

Variables

Para el nombre de las variables utilizar palabras completas en singular (máximo tres palabras). El nombre deber ir en plural cuando la variable representa una lista o un conjunto de elementos.
Si las palabras no son suficientes para la descripción, se debe hacer un comentario, al frente de la variable.

Las constantes (final) van en mayúscula sostenida separando las palabras por guión de piso (_).
Para los argumentos, deben iniciar siempre con la letra “a” y posteriormente el nombre según lo establecido anteriormente.
Para las instancias de las clases, las entidades llevan el mismo nombre de la clase, solo que la palabra inicial va en minúscula. Si se necesita más de una instancia, para diferenciarlas se debe adicionar una palabra que identifique el rol que desempeña. Ejemplo: Estudiante estudiantePregrado, Estudiante estudiantePostgrado.
La variable del EntityManager se debe llamar em.
La variable de tipo FacesMessages se debe llamar facesMessages.

Librerías (JAR)

El nombre de las librerías no sigue el mismo estándar de las clases. Éstas se deben escribir en minúscula, separando cada palabra con un guión (-). Ejemplo: recursos-humanos.jar.

Nombres de archivos

Se sigue el mismo estándar descrito en las reglas de sintaxis generales.

4.2.4 Documentación

La documentación relacionada con el diseño del sistema reside en la base de datos de Enterpirse Architect.

La documentación del código fuente se debe hacer en cada una de las clases, siguiendo el estándar de JAVADOC y documentando la definición de la clase,

descripción de los métodos get y set y descripción de los parámetros de entrada y salida de cada uno de los métodos que componen la clase.

4.2.5 Capa de Presentación

Plantilla principal

La plantilla principal de una página es la siguiente:

Figura 20 Plantilla Principal División de Servicios de Información

Contenido

Esta sección se refiere al caso de uso implementado. La estructura de esta sección es:

Figura 21 Plantilla de Contenido División de Servicios de Información

1. Mensajes: En la parte superior se visualizan los mensajes asociados a la creación de un registro. Los mensajes producto de las transacciones de edición y eliminación se visualizan en un modal panel en el centro de la página.

2. Título: Nombre del módulo seguido de el nombre del submenú seleccionado.
3. Cuerpo del formulario:
    a. Nombre del formulario: Acción que se está realizando las cuales pueden ser crear, modificar, eliminar entre otras .
    b. Campos: Información a diligenciar por el usuario. La información requerida se encuentra con un * al derecho del campo.
    c. Ayuda: El usuario dispone ayuda asociadas a cada campo en caso de no saber qué información colocar en el mismo.

4. Listados: Cada transacción que se realice como guardar, modificar o eliminar un registro se podrá visualizar en la tabla que aparece en la parte inferior del formulario. Asociado a cada elemento se encuentran las acciones disponibles para este.

Paginación

Para la paginación se debe tener en cuenta el tipo de consulta que se va a realizar y la memoria consumida por ésta en el servidor de base de datos. De acuerdo a esto la aplicación decide el número de registros máximos que debe retornar la consulta.

Por ejemplo, se quiere consultar los estudiantes de la sede Bucaramanga. De acuerdo al análisis realizado, no hay un consumo alto de memoria, por lo que se decide retornar 100 estudiantes máximos. Esto indica que cuando se implemente el método de consulta en JPQL, el parámetro setMaxResults debe tener el valor de 100.

Texto

Existen cinco tipos de plantillas para mostrar texto en la página. Éstas se encuentran dentro de la carpeta Plantillas.

- etiqueta.xhtml: Texto o datos.
- etiquetaColumna.xhtml: El título de la columna en una tabla
- titulo.xhtml: Titulos
- etiquetaNavegacion.xhtml: Utilizado en la barra de navegación
- mostrar.xhtml: Permite visualizar dos etiquetas (etiqueta, dato) al mismo tiempo. Su objetivo es la de visualizar datos como si fuera un formulario.

La sintaxis para utilizar las plantillas son:

```
<s:decorate template="../Plantillas/[nombrePlantilla]">
        <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
        </ui:define>
</s:decorate>
```

Donde nombrePlantilla puede ser etiqueta.xhtml, etiquetaColumna.xhtml, etiquetaTitulo.xhtml o etiquetaNavegacion.xhtml.

Para la plantilla mostrar.xhtml:

```
<s:decorate template="../Plantillas/mostrar.xhtml">
        <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
        </ui:define>
        <h:outputText   value="#{formatoWeb.usuario.primerNombre}"/>
</s:decorate>
```

Edición

Para capturar cualquier tipo de dato en una caja de texto se debe utilizar la plantilla edicion.xhtml de la siguiente manera:

```
<s:decorate id="dcr[identificador]"  template="/Plantillas/edicion.xhtml">
    <ui:define name="label">
                #{FormatoWeb.mostrarMensaje('primerNombre', true)}
    </ui:define>

    <h:inputText id="txt[nombreCajaDeTexto]" value="[valor]"
                required="true">

        <a:support event="onblur"        reRender="dcr[identificador]"/>
    </h:inputText>
</s:decorate>
```

dcr[identificador] es el nombre del elemento decorate. Por ejemplo: dcrPrimerNombre.
txt[nombreCajaDeTexto] es el nombre de la caja de texto. Por ejemplo: txtPrimerNombre.

Tablas estáticas

Se debe usar el control panelGrid de Java Server Faces.

Tablas dinámicas

Se debe usar el control dataTable de Rich Faces, agregando las siguientes líneas de programación en su definición:

```
onRowMouseOver="this.style.backgroundColor='#E1E1E1'"
 onRowMouseOut="this.style.backgroundColor='#{a4jSkin.tableBackgroundColor}'
```

Las cuales indican el color de la fila cuando el puntero del mouse esta sobre ésta.

Listas desplegables

Para cualquier tipo de lista se debe utilizar el control de Java Server Faces.

Mensajes del sistema

Los mensajes del sistema son textos enviados por los EJB de sesión, ya sea de confirmación de una acción o errores en el procedimiento. Dichos errores se deben mostrar en la etiqueta FacesMessages la cual debe estar definida al comienzo de la página después del menú si existiera éste. El código es el siguiente:

```
<h:messages globalOnly="true" styleClass="message"/>
```

Para mostrar errores de validación en los formularios, éstos deben aparecer al frente de cada control y no globalmente.

4.3Roles Y Usuarios

La seguridad del sistema de información esta manejada mediante los diferentes roles existentes en los procesos llevados dentro del mismo, de esta forma se restringe los accesos por parte de los diferentes usuarios que utilizan el sistema.
Los diferentes roles son los encargados de determinar hasta qué nivel de navegación puede tener un determinado usuario en la sesión, algunos de los roles contemplados en el sistema de cafetería son:

Administrador:

Tiene acceso a todo el sistema de información, algunas de las funciones a realizar sobre el sistema es crear los productos a vender, aprobar pedidos, realizar entradas de elementos en cualquier momento, dar de baja elementos, realizar un control de inventarios etc.

Almacenista:

Tiene acceso a la parte relacionada con las entradas de productos en las etapas de despacho de un pedido o una entrada inesperada de elementos a los puntos de venta, algunas de sus funciones es despachar los elementos de un pedido que previamente fue solicitado y después aprobado.

Cajero:

El tiene acceso a todo lo relacionado con la venta de los productos ofrecidos por cafetería, entre sus funciones dentro del sistema esta vender elementos del punto de venta.

Jefe de unidad:

Son los jefes de las unidades académico administrativas los cuales pueden realizar reservas de menús y productos ofrecidos por la cafetería, entre sus funciones está realizar reservas de los productos de cafetería, cancelar las reservas que todavía no están en estado aprobado, consultar menús disponibles.

Usuario invitado:

Es aquel que tiene solo acceso a alguna información del sistema de información, entre sus permisos esta consultar menús y productos disponibles en las diferentes cafeterías.

Jefe puntos de venta

El jefe puntos de venta, es la persona encargada de administrar los inventarios de los puntos de venta entre los módulos que tiene disponible se encuentra el modulo de inventarios, pedidos y ventas.

4.4Diagramas UML

En el proceso de análisis y diseño del sistema de información se construyeron los diagramas UML. Durante el transcurso del proyecto se fueron adaptando en la medida que se fueron presentando avances del mismo desde la presentación del prototipo inicial.

Los esquemas básicos de cada uno de los diagramas que se generaron, producto del diseño, se presentarán a continuación, con el fin de ilustrar el fondo y la forma como fueron producidos.

4.4.1 Diagrama: Casos de Uso

4.4.1.1 Actores

Como se puede visualizar en la figura que aparece a continuación se han identificado 6 actores para el sistema de Cafetería de Bienestar Universitario.

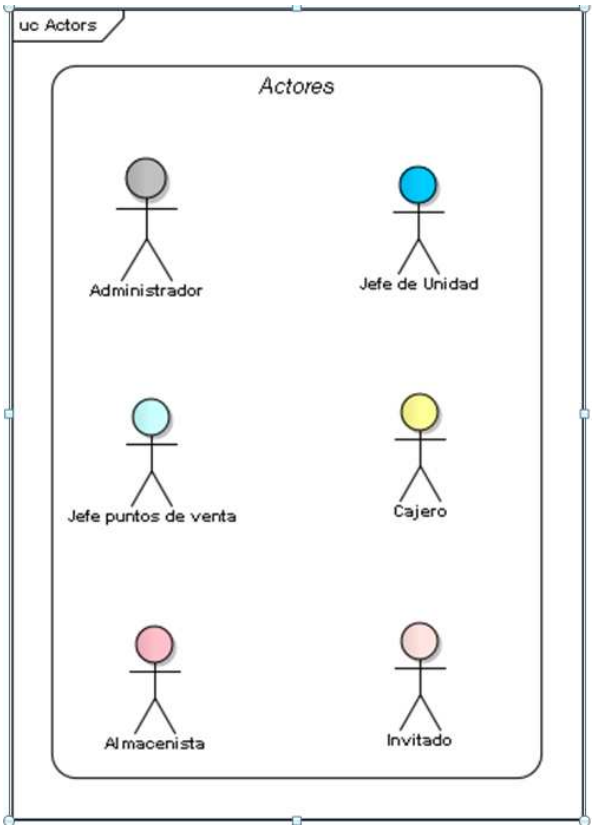

Figura 22 Actores del Diagrama de Casos de Uso

Actor	Descripción
Administrador	El administrador, es el Encargado o el Jefe de Sección de Comedores y Cafetería, quien administra todos los procesos que realiza el sistema de Cafetería.
Jefe de Unidad	El Jefe de Unidad, son los jefes de las unidades académico administrativas de la Universidad Industrial de Santander quienes tienen permiso para acceder al módulo de reservas.
Jefe Puntos de Venta	El jefe puntos de venta, es la persona encargada de administrar los inventarios de los puntos de venta.
Cajero	El Cajero , es la persona encargada de registrar las ventas de productos ofrecidos por las cafeterías de Bienestar Universitario
Almacenista	El Almacenista, es la persona encargada del ingreso y despacho de productos del almacén de Bienestar Universitario.
Invitado	El invitado, comunidad universitaria en general la cual puede ingresar a consultar los productos disponibles para la venta en los diferentes puntos de venta sin necesidad de estar registrado en el sistema.

Tabla 1 Actores del sistema

4.4.1.2 Casos de uso

En el modelo de casos de uso UML para el Sistema de información de Cafetería, se pretende integrar toda la información recopilada y analizada y estructurarla en 7 diagramas.

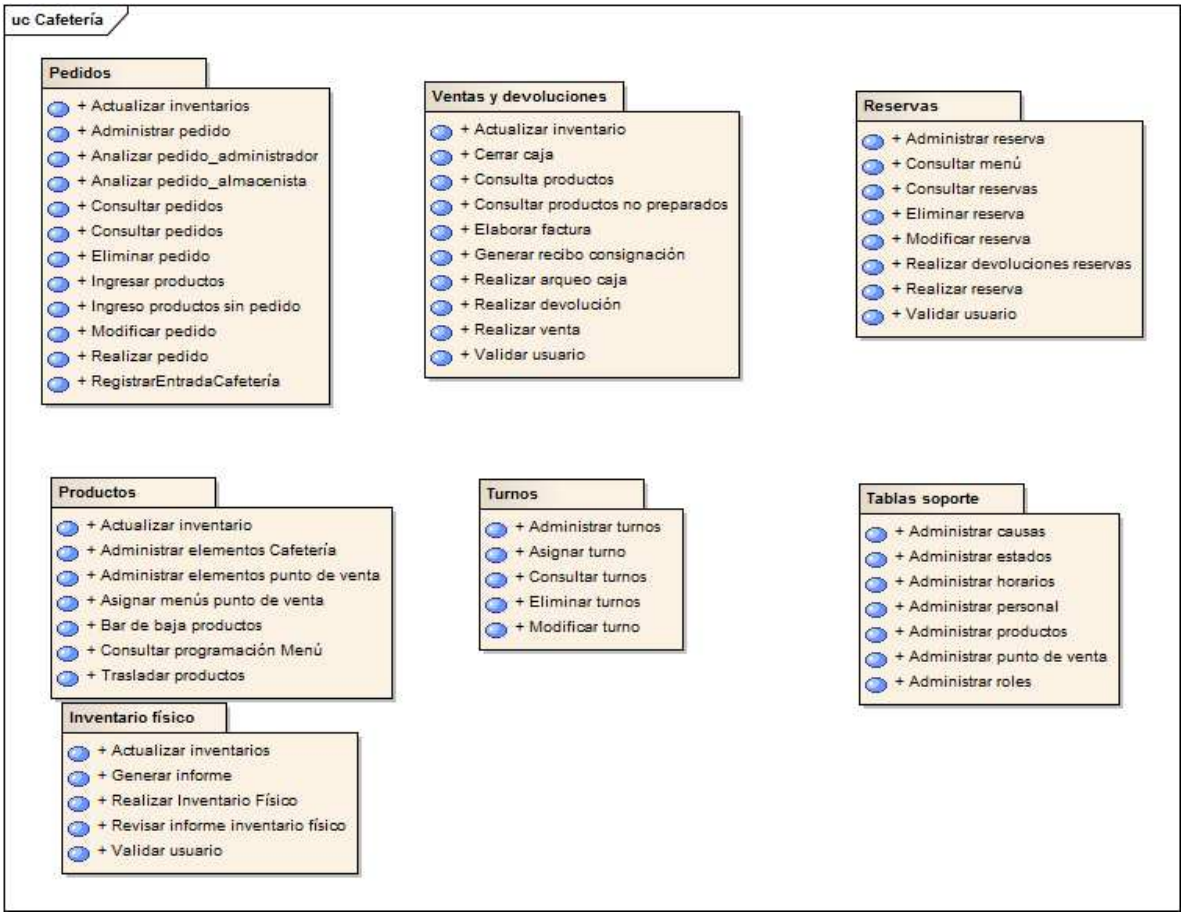

Figura 23 Casos de uso sistema de cafetería

Por lo extenso de los casos de uso se explicará detalladamente el caso de uso Ventas y devoluciones.

CASO DE USO : Ventas y Devoluciones

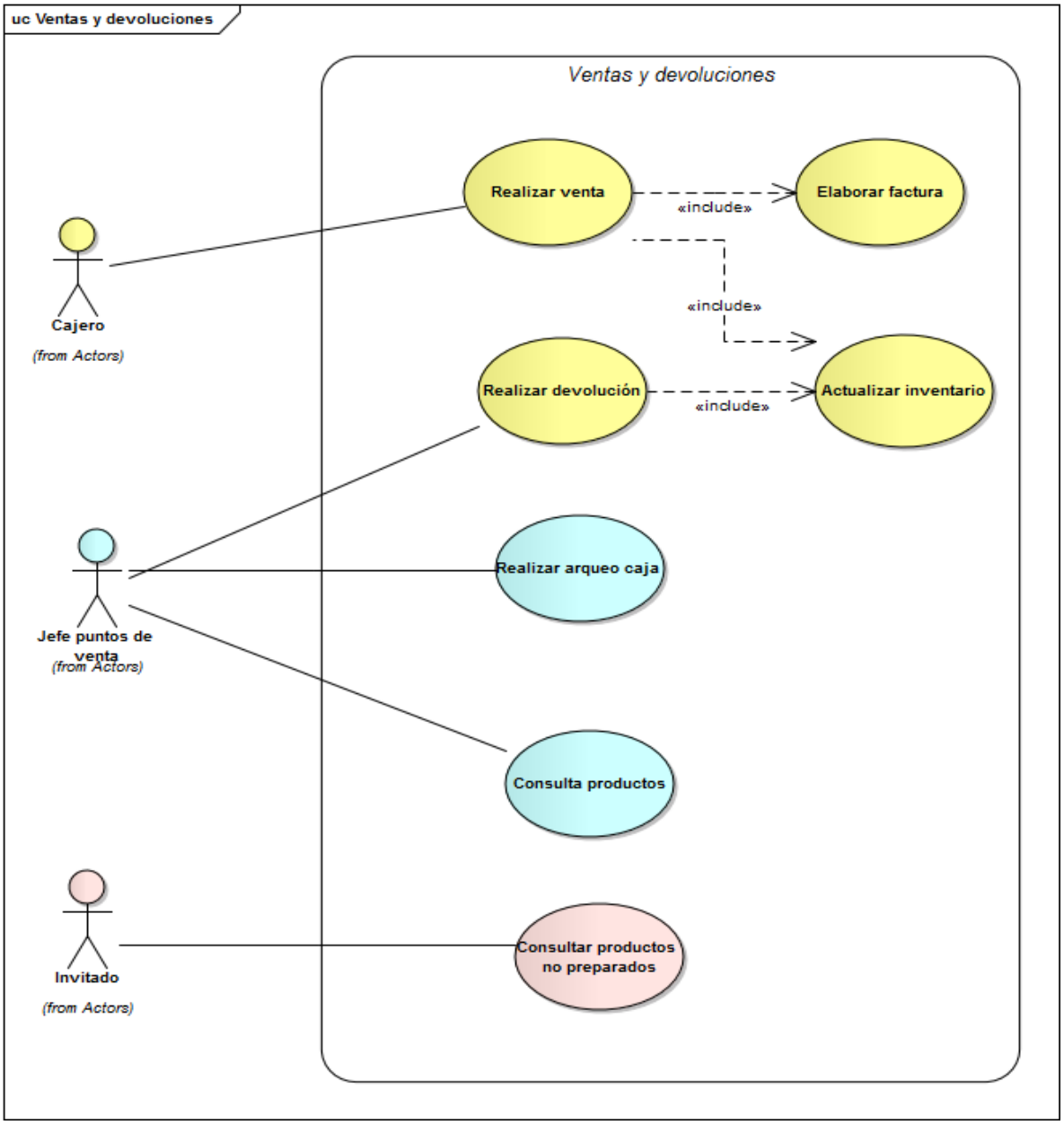

Figura 24 Caso de uso de ventas y devoluciones

A continuación se describe cada caso de uso

CASO DE USO	DESCRIPCIÓN
Realizar venta	Permite realizar la venta de los productos no preparados y menús programados disponibles en cada punto de venta. También permite cancelar el valor de los anticonceptivos ofrecidos por la farmacia de B.U.
Efectuar factura	Permite generar una factura que contiene la información del punto de venta donde se está realizando la transacción como el detalle de los elementos vendidos.
Realizar devolución	Permite realizar la devolución de uno o más elementos por diversas causas asociados a una factura de venta
Actualizar inventario	Aumenta o disminuye la cantidad de un elemento en el inventario del punto de venta donde se efectuó la transacción.
Realizar arqueo	Permite al finalizar cada turno en una caja de un punto de venta realizar arqueo al mismo y generar el valor a consignar.
Consultar productos	Permite consultar a través de criterios como el punto de venta ,nombre producto o nombre de la agrupación información asociada al mismo , la cual le permite al administrador tomar decisión a la hora de realizar un pedido o traslado de productos.
Consultar productos no preparados	Permite consultar los productos no preparados disponibles en los puntos de venta de B.U.

Tabla 2 caso de uso de realizar venta y devoluciones

Detalle del caso de uso realizar venta

CASO DE USO	**REALIZAR VENTA**	
Objetivo:	Permite realizar la venta de los productos no preparados y menús programados disponibles en cada punto de venta. También permite cancelar el valor de los anticonceptivos ofrecidos por la farmacia de B.U.	
Actores:	Cajero	
Pre condiciones	Validarse como Cajero	
Post Condiciones	Generar recibo de venta	
Pasos	**Acción del Actor**	**Respuesta del Sistema**
	Ingresar al Sistema	
		Muestra el formulario para realizar una venta
	Escoge el tipo de venta a realizar	
		Se muestra el formulario para el tipo de venta seleccionado.
	Escoger un elemento	
		Se escoge un elemento, se asigna una cantidad.
	Finaliza la venta	
		El cajero después de registrar lo que desea el cliente , ingresa el dinero con el cual va a cancelar y envía a imprimir el recibo
Variaciones:	El cajero puede realizar ventas de productos no preparados o menús programados.	

Extensiones:	El sistema accede a la base de datos para listar los datos que corresponden al punto de venta donde se está realizando la transacción.

Tabla 3 Documentación caso de uso realizar venta

4.4.2 Diagrama de Clases

El diagrama de clases resultado de la etapa de análisis y diseño, el cual se ha modificado durante el desarrollo del proyecto satisfaciendo las necesidades del cliente. Debido a la extensión del mismo y para efectos de una visualización adecuada de la imagen se presenta únicamente la sección relacionada con las ventas.

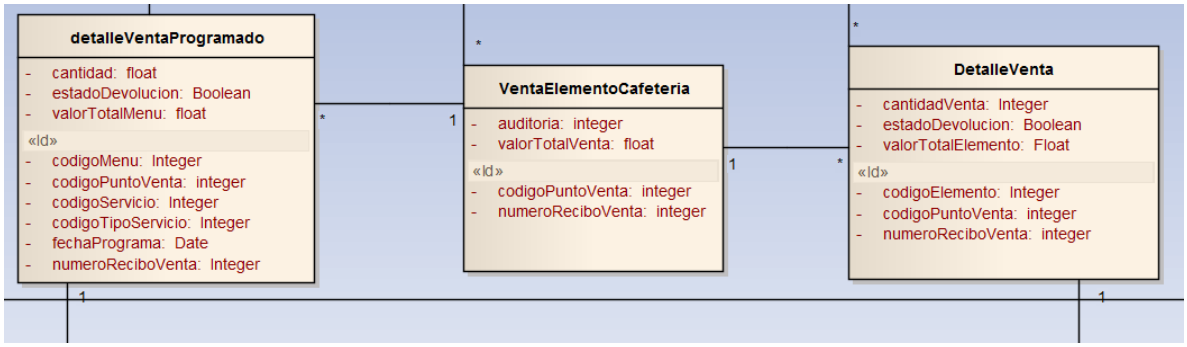

Figura 25 Fragmento del diagrama de clases

Clase : Venta elemento Cafetería			
Descripción: Representa las ventas de elementos preparados y menús programados en los diferentes puntos de venta			
Pk	**Nombre del atributo**	**Tipo de dato**	**Descripción**
True	codigoPuntoVenta	serial	Indicativo que identifica un punto venta
True	numeroReciboVenta	serial	Indicativo que identifica un recibo de venta.
False	valorTotalVenta	Float	Valor total de la venta realizada
False	Auditoria	Integer	Estos campos son los

			relacionados con la estructura de seguridad del sistema.

Tabla 4 Documentación de la clase venta elemento cafetería

Clase : Detalle venta programado			
Descripción: Representa la discriminación de los menús programados, vendidos en cada una de las respectivas ventas realizadas.			
Pk	**Nombre del atributo**	**Tipo de dato**	**Descripción**
True	codigoServicio	serial	Indicativo que identifica el servicio del cual se está vendiendo menús.
True	codigoTipoServicio	serial	Indicativo que identifica el tipo de servicio.
True	codigoMenu	serial	Indicativo que identificar el menú.
True	codigoPuntoVenta	serial	Indicativo que identifica el código del punto de venta donde se realiza la venta.
True	fechaPrograma	Date	Fecha de programación del menú
True	numeroReciboVenta	Serial	Indicativo que identifica un recibo de venta
False	Cantidad	Integer	Cantidad solicitada de un menú.
False	estadoDevolucion	Boolean	Permite determinar si un menú devuelto o no. Tiene 2 posibles estados • No devuelto • Devuelto

False	valorTotalMenu	Float	Valor total de un mismo menú vendido.

Tabla 5 Documentación detalle venta programados

Clase : Detalle venta			
Descripción: Representa la discriminación de los elementos no preparados vendidos en cada una de las ventas realizadas.			
Pk	**Nombre del atributo**	**Tipo de dato**	**Descripción**
True	codigoElemento	serial	Indicativo que identifica un elemento no prepararado
True	codigoPuntoVenta	serial	Indicativo que identifica el código del punto de venta donde se realiza la venta.
True	numeroReciboVenta	Serial	Indicativo que identifica un recibo de venta
False	cantidadVenta	Integer	Cantidad de un mismo elemento vendido.
False	estadoDevolucion	Boolean	Permite determinar si un menú devuelto o no. Tiene 2 posibles estados • No devuelto • Devuelto
False	valorTotalElemento	Float	Valor total de un mismo elemento vendido.

Tabla 6 Documentación clase detalle venta

4.4.3 Diagramas de Secuencia

Los diagramas de secuencia amplían la visión general del funcionamiento e interacción del sistema con el usuario.

A continuación se presentan los diagramas de secuencia de los casos de uso más complejos

- Realizar venta

Permite al cajero realizar la venta de productos no preparados y menús programados en los diferentes puntos de venta.

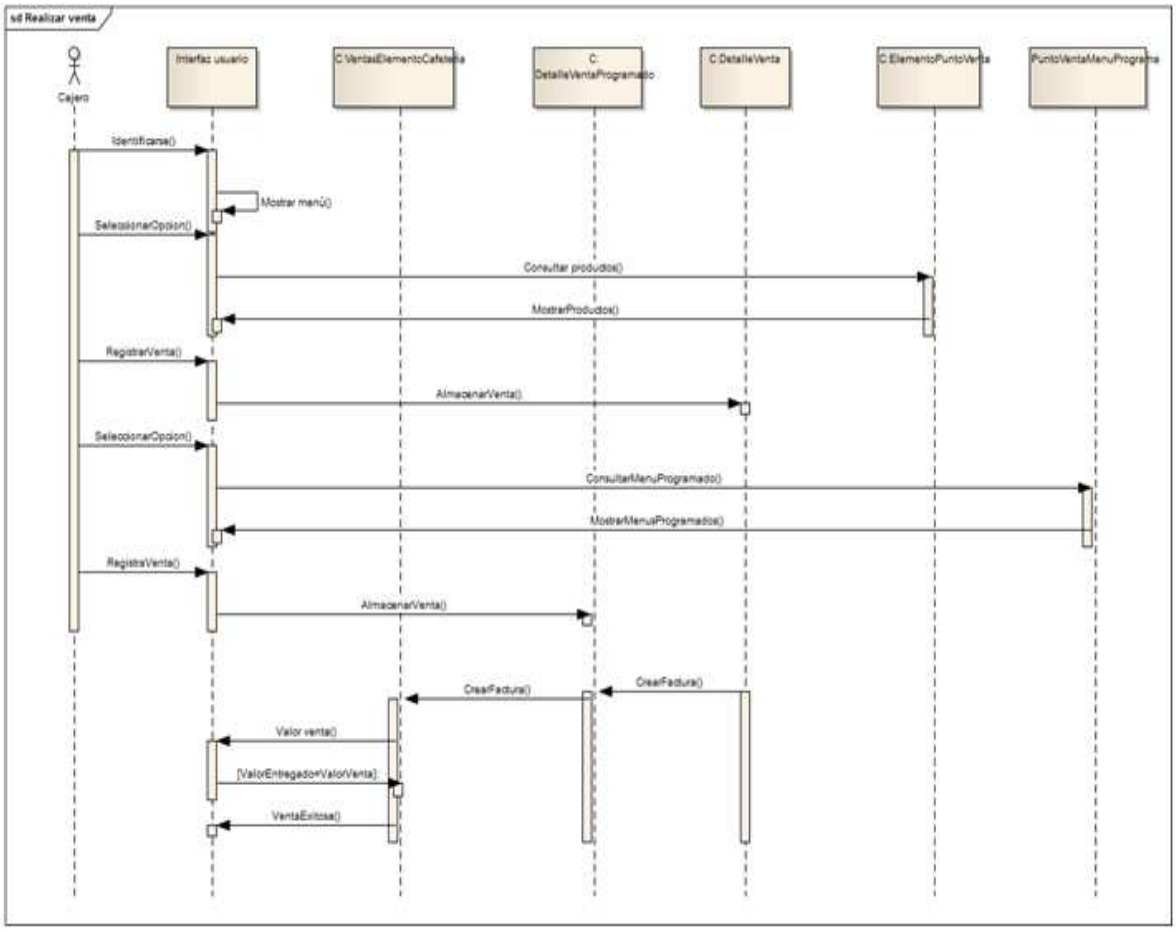

Figura 26 Diagrama Secuencia: Realizar venta

- Realizar reserva

Permite a los Jefes de Unidad realizar reservas de productos no preparados, menús programados o especiales.

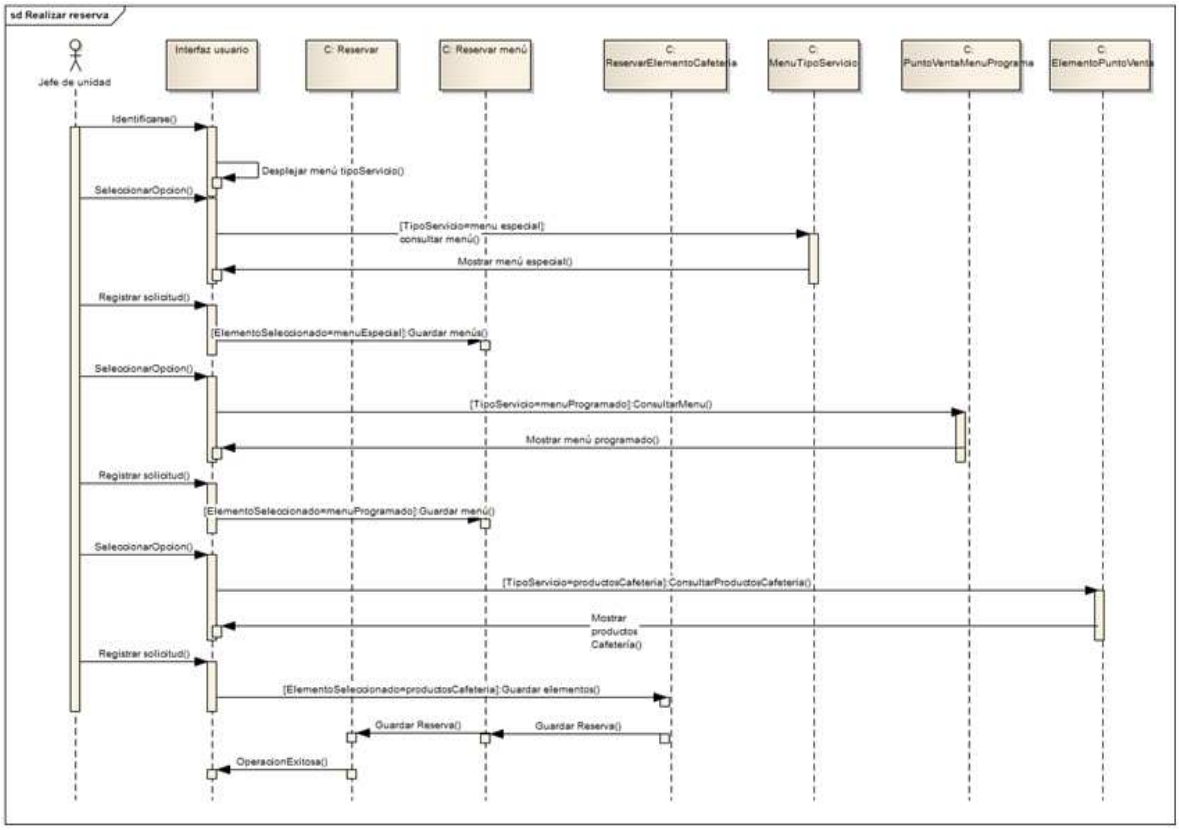

Figura 27 Diagrama Secuencia: Reservas

4.5Prototipo Inicial

4.5.1 Generalidades del prototipo inicial

El proceso de construcción del prototipo inicial comenzó desde el momento en el cual la División de Bienestar Universitario manifestó el interés de desarrolla el proyecto y continuó con una serie de encuentras periódicos, con la nutricionista Consuelo Serrano Vega adscrita a la División de Bienestar Universitario.

En los encuentros realizados se escuchó y analizó cada uno de los requerimientos deseados para el sistema de información. Dicha información permitió construir un prototipo inicial no funcional el cual sirvió como guía para el desarrollo del software posteriormente.

El prototipo no funcional se creó como una serie de páginas HTML sin ninguna funcionalidad, las cuales permitirían visualizar la estructura de navegación, facilitando la labor de determinar la información a solicitar al usuario, el contenido de cada una de las vistas y aclarar algunos procesos que se llevan a cabo dentro de la cafetería de Bienestar Universitario.

Se manejo un proceso cíclico de perfeccionamiento y refinamiento que permitiera llegar a un nivel de acercamiento detallado de lo que quiere alcanzar el usuario con el sistema, logrando reducir el tiempo de perfeccionamiento del prototipo final agilizando de esta forma el ciclo de vida del proyecto.

Durante el transcurso de desarrollo del prototipo no funcional se fue construyendo el diseño integrado por los diagramas de casos de uso, de clase y de secuencia.

4.5.2 Interfaz resultado del prototipo no funcional

Debido a la extensión del sistema, a continuación se visualizarán algunas interfaces diseñadas en el mismo.

Figura 28 Pantalla de Bienvenida (Prototipo Inicial)

Módulo Reservas.

Esta funcionalidad permite a los jefes de las unidades académico administrativas reservas menús programados o especiales para un evento o reunión. Una de las funcionalidades que ofrece este módulo es la consulta de menús.

- Consultar menú

En esta parte del sistema el usuario puede consultar en cualquier momento los menús programados o especiales existentes con su información nutricional.

Figura 29 Pantalla de consultar menú (Prototipo Inicial)

- Reservar

Luego de visualizar los menús disponibles es el jefe de la unidad Académico Administrativa quien ingresa y diligencia el siguiente formato

Figura 30 Formulario reservas

Luego de realizar una reserva el sistema le permite consultar el estado en el cual se encuentra esta. Una reserva tiene los siguientes estados

- Por aprobación
- Aprobado
- Despachado
- Entregado
- Cancelado

Gracias a esto el sistema es muy dinámico y flexible ofreciéndole al usuario la posibilidad de realizar un ajuste siempre y cuando la reserva no haya sido aprobada en su totalidad. Esto se puede observar en la siguiente imagen donde

las acciones de edición son visibles cuando el estado de la reserva es por aprobación.

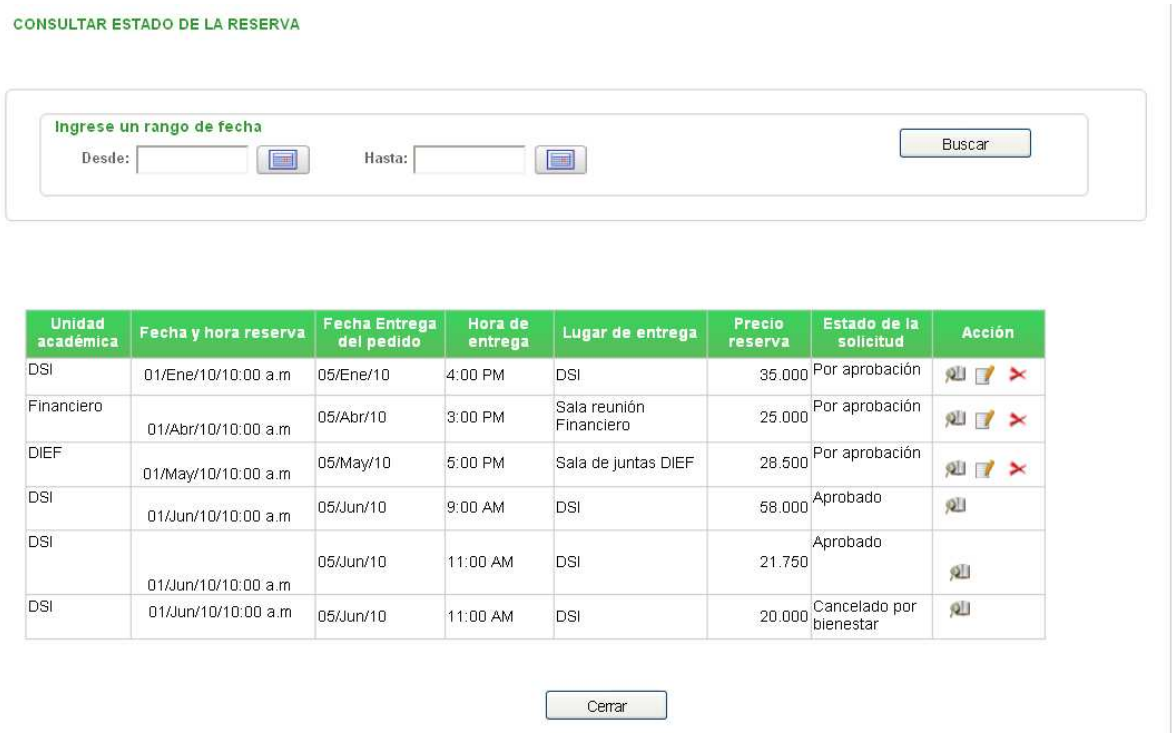

Figura 31 Consulta estado reserva

4.6Prototipo Final

4.6.1 Generalidades del prototipo Final

Al término del desarrollo del sistema de información orientado a la web, para soportar el proceso del servicio de cafetería ofrecido por el Bienestar Universitario, se obtuvo un prototipo que se adapta plenamente a las características que se han definido desde el comienzo del desarrollo del proyecto.

El producto final cumplió con todos los requerimientos, y después de revisión por parte del cliente, ha sido aceptado por éste, manifestando su plena conformidad con el mismo.

Debido a la extensión del producto final se describirá de forma general algunas funcionalidades del modulo productos y ventas y devoluciones, propias del sistema de información para cafetería de Bienestar Universitario.

Módulo productos

La primera actividad a realizar por el administrador luego de ingresar los datos requeridos en las tablas soporte es la creación de los productos de cafetería lo cual consiste en seleccionar de todos los productos que tiene disponible Bienestar Universitario, los que se van a ofrecer en los diferentes puntos de venta. Como se puede observar en la gráfica se ha agregado a cafetería el producto jugo tampico.

.

Figura 32 Creación producto cafetería

Posteriormente de los productos previamente asignados a cafetería se asignan los productos se que van a vender en cada punto de venta. Para el ejemplo que se está desarrollando se ha asignado el jugo tampico al punto de venta Bienestar Universitario, lo cual implica que dicho producto se podrá vender en dicho lugar.

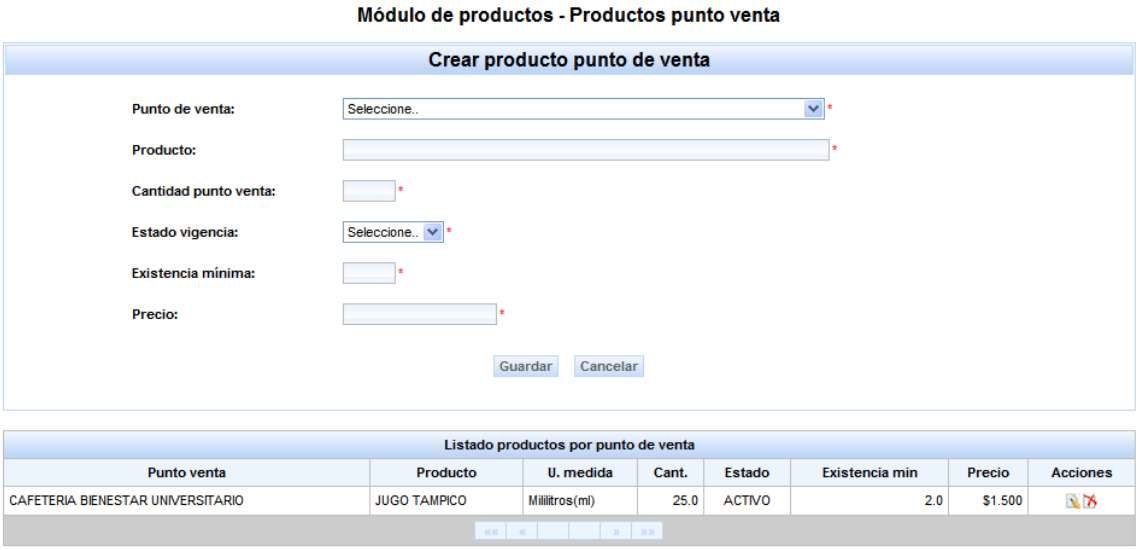

Figura 33 Creación producto por punto de venta

Módulo Ventas y Devoluciones

Permite realizar la venta de productos no preparados y menús programados. Para efectos del ejemplo se realizará la venta del jugo tampico y el menú muté santanderiano.

Figura 34 Modulo de ventas

Realizada la venta, el sistema genera el recibo de dicha transacción.

El jefe de punto de venta es la persona encargada de realizar la devolución de un producto, para esto se debe seleccionar el punto de venta donde realizó la transacción y ingresar el número de recibo. Con esta información el sistema el sistema muestra los productos asociados a dicha factura. En la columna acción permite realizar la devolución del producto deseado.

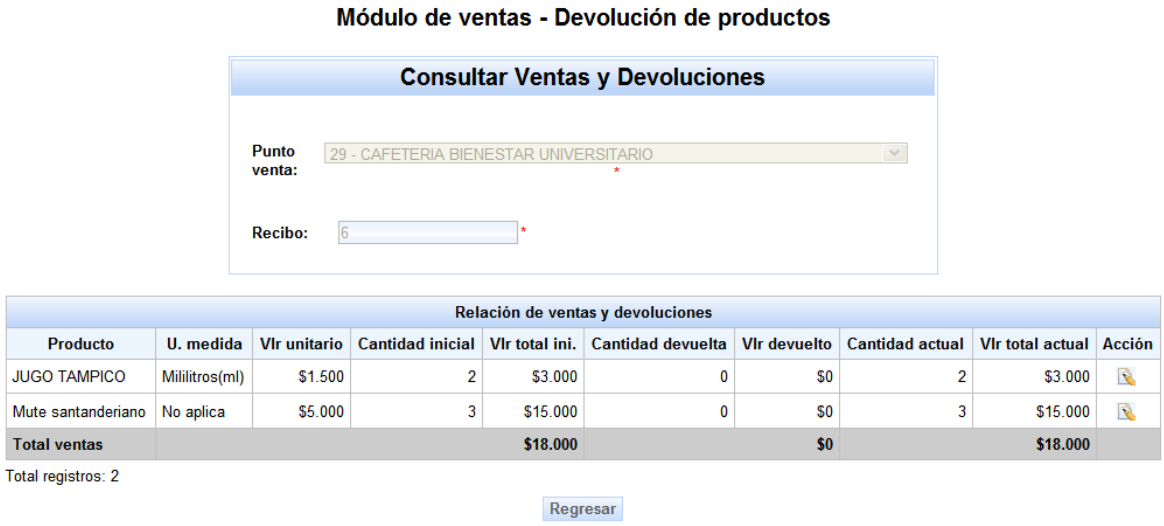

Figura 35 Modulo de devoluciones

Un producto puede ser devuelto por las siguientes razones:

- Exceso de pedido
- Producto equivocado
- Producto en mal estado
- Producto vencido

Dependiendo de la opción escogida el sistema aumentará o disminuirá la cantidad de este producto en el inventario del punto de venta.

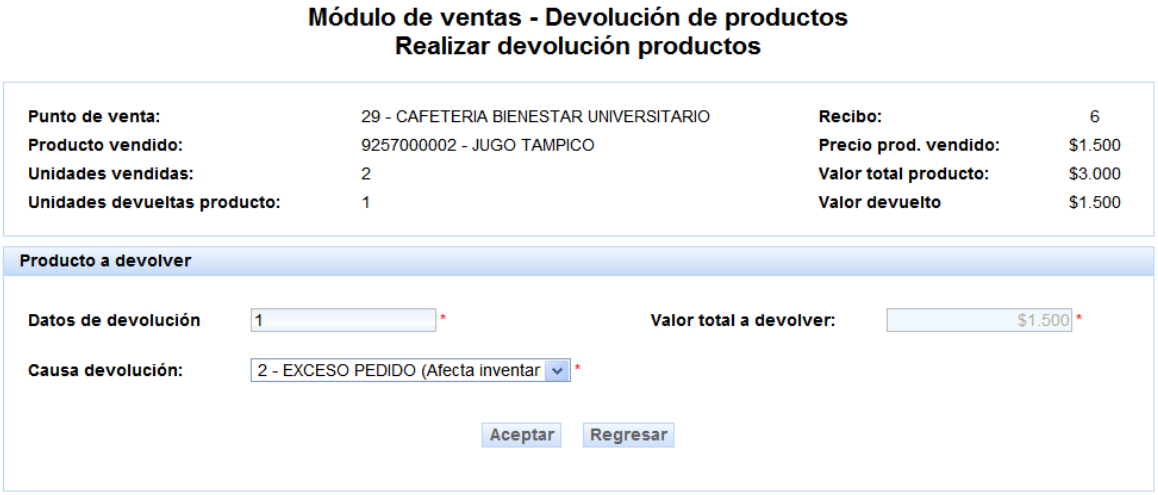

Figura 36 Detalle de la devolución

Luego de realizada la devolución el sistema muestra los datos de la factura pero con las devoluciones que sean realizado a los productos. No permitiendo más devoluciones del mismo cuando ya se ha devuelto la totalidad del mismo.

Figura 37 Resumen de la devolución

4.6.2 Esquema de Seguridad de la UIS

Para este proyecto se utiliza el esquema de seguridad definido por la División de Servicios de Información para los diferentes sistemas de información que apoyan la gestión de la Universidad Industrial de Santander, el cual está basado en la estructura de roles – usuarios.

Los roles se establecen de acuerdo a las actividades que realizan los usuarios. A cada uno de los roles definidos se le asocian los usuarios de acuerdo a las funciones que desempeñen.

4.6.2.1 Estructura de la Base de Datos soporte

La base de datos que soporta el esquema de seguridad contempla básicamente las siguientes tablas:

Sistema: Contiene información de los sistemas de información de la universidad. Para cada sistema se específica: Nombre, descripción del sistema, fecha y hora de creación en la base de datos, fecha y hora de inicio de vigencia del sistema, fecha y hora de cierre de vigencia del sistema.

Rol: Contiene información de los diferentes roles definidos para cada sistema de información, como: Nombre asignado al rol, descripción del rol, fecha y hora de creación, fecha y hora de inicio de vigencia del rol, fecha y hora de cierre de vigencia del rol.

Usuario: Contiene información de los posibles usuarios de los sistemas de información. Entre esta información está: tipo y número de documento de

identidad del usuario, fecha y hora de creación del usuario, fecha y hora de inicio de vigencia del usuario, fecha y hora de cierre de vigencia del usuario.
Sistema–rol: Contiene los roles definidos para cada uno de los sistemas de información, indicando: rol, sistema, fecha y hora de creación del rol – sistema, fecha y hora de inicio de vigencia del rol en el sistema, fecha y hora de cierre de vigencia del rol en el sistema.

Rol–usuario: Contempla los usuarios asociados a cada uno de los roles definidos, considerando: Rol, usuario, fecha y hora de creación del rol – usuario, fecha y hora de inicio de vigencia del usuario en el rol, fecha y hora de cierre de vigencia del usuario en el rol.

Menú–rol–sistema: Contiene los menús asociados a los roles en los distintos sistema de información, contemplando: Sistema de información, nombre del menú, descripción del menú, fecha y hora de creación del menú, fecha y hora de inicio de vigencia del menú asociado al rol, fecha y hora de cierre de vigencia del menú asociado al rol.

Opción–menú–rol: Contempla las opciones definidas para cada una de los posibles menús establecidos para cada sistema de información. Contiene: Nombre de la opción, descripción de la opción, nombre del menú superior, nombre del menú que contiene la opción, nombre del programa a ejecutar cuando la opción es la de más bajo nivel, fecha y hora de creación de la opción del menú, fecha y hora de inicio de vigencia de la opción, fecha y hora de cierre de la opción.

Tabla–sistema: Contiene información de las tablas que conforman la base de datos que soporta cada uno de los sistemas de información. Considera: Sistema de información, nombre de la tabla, descripción de la tabla.

Tipo–permiso: Establece para cada tabla de un sistema de información, los roles que tienen permisos para incluir registros, para modificar registros o para eliminar registros en ella. Contiene: Sistema de información, nombre de la tabla, clase de permiso (inclusión, modificación, eliminación de registros), fecha y hora de creación del permiso, fecha y hora de inicio de vigencia del permiso, fecha y hora de fin de vigencia del permiso.

Acceso–tabla: Define para las tablas de un sistema de información si un rol tiene permiso sobre toda la información de la tabla o sobre una parte de esta. Considera: Sistema, nombre de la tabla, clase de acceso (total, parcial), fecha y hora de creación del permiso, fecha y hora de inicio de vigencia del permiso, fecha y hora de fin de vigencia del permiso.

Atributo–tabla: Establece los atributos sobre los cuales se debe controlar el acceso a una tabla, cuando a un rol se le concede permiso para hacer uso parcial de la información existente en una tabla. Contiene: Sistema de información, nombre de la tabla, nombre del atributo sobre el cual se controla el acceso a la tabla, descripción del atributo, fecha y hora de creación del atributo, fecha y hora de inicio de vigencia del atributo, fecha y hora de fin de vigencia del atributo.

Valor–atributo-proceso: Contiene los valores que deben tener los atributos definidos en cada tabla en la tabla atributo – tabla que permiten el acceso a la información asociada a estos valores. Específica: Sistema de información, nombre de la tabla, nombre del atributo, valor del atributo, descripción, fecha y hora de creación del valor del atributo, fecha y hora de inicio de vigencia del valor del atributo, fecha y hora de fin de vigencia del valor del atributo.

Acceso-sistema: Contempla el histórico de acceso que un usuario ha realizado a un sistema, identificando las opciones que ha seleccionado. Contiene: Login de

usuario, rol, identificación de la sesión, sistema, opción seleccionada, fecha y hora de ingreso, fecha y hora de salida.

4.6.2.2 Entorno de Navegación

Para cada sistema de información, la UAA responsable define los roles necesarios para el adecuado uso del sistema de información de acuerdo a las funciones que realice y establece los usuarios asociados a cada uno de ellos.

Para cada rol se define el menú de inicio, el cual le permite a cada usuario que hace parte de este rol, empezar la navegación por las distintas opciones que le ofrece el sistema, hasta llegar al nivel más bajo en el cual se ejecuta el proceso que soporta la actividad que desea realizar.

Este entorno está soportado por las siguientes tablas de las base de datos del esquema de seguridad: Rol, usuario, sistema, sistema-rol, usuario-rol, menú-rol, opción-menú rol, descritas en “ESTRUCTURA DE LA BASE DE DATOS SOPORTE”.

4.6.2.3 Entorno de Control de Datos

Para los roles definidos en cada uno de los sistemas de información se especifican las tablas a las cuales puede accesar, el tipo de transacción que puede realizar sobre estas tablas (inclusión, modificación o eliminación de registros), si tiene acceso total o parcial a la información que contiene la tabla.
Para el acceso a la información de la tabla de manera parcial, se debe establecer el atributo o atributos seleccionados, los valores que estos atributos deben tener para autorizar el acceso solicitado.
Este entorno está soportado por las siguientes tablas de las base de datos del esquema de seguridad: Rol, usuario, sistema, sistema-rol, usuario-rol, tabla–sistema, tipo-permiso, acceso-tabla, atributo-tabla, valor atributo proceso, descritas en “ESTRUCTURA DE LA BASE DE DATOS SOPORTE”.

4.6.2.4 Auditoria

Todas las tablas que conforman la base de datos soporte del esquema de seguridad tienen el historial de las transacciones realizadas sobre cada una de ellas.

El historial de las transacciones de cada tabla contiene información de los registros incluidos en la tabla, de los registros modificados y de los registros eliminados. Adicionalmente, en cada transacción se especifica: Fecha de la transacción, hora de la transacción, tipo de transacción (I/U/D), tipo y número de documento de identidad del usuario que realizó la transacción, login, rol asociado, dirección IP y MAC del equipo desde el cual llevó a cabo la transacción.

4.6.3 Documentación Programas Fuente

En la labor desarrollo la documentación juega un papel muy importante para el mantenimiento de software. Esto es aun mas cierto en proyectos grandes, en los que pueden estar involucrados demasiados desarrolladores, todos con culturas de desarrollo diferentes. Aunque todos estos desarrollos cuentan con estándares muy definidos, siempre puede ser necesaria la aclaración sobre ciertos fragmentos o secciones de código.

Históricamente todos los lenguajes de programación han contado con facilidades para la inserción de comentarios dentro del código fuente. Estos comentarios no son compilados puesto que no tienen utilidad funcional pero cuando otro desarrollador ve el código fuente, son necesarios para entender partes que no son claras a simple vista. Java heredó de C la forma de hacer tales comentarios:

Para líneas	// comentario
Para bloques de comentarios	/*Para bloques enteros de comentarios, con varias líneas.*/

Sin embargo estas herramientas no son necesarias para documentar proyectos largos, como es el caso de las aplicaciones empresariales. Para esto Java ofrece Javadoc, que es un generador de documentación a partir de código Java. El código que Javadoc utiliza es aquel que encuentre entre:

/**

*documentación.

*/

El propósito de este modelo de documentación no es deprecar las formas tradicionales, que siguen siendo válidas sobre secciones pequeñas de código. Lo que busca Javadoc es generar documentación clara y precisa sobre clases, sus miembros y sus métodos. Así pues, los comentarios especificados mediante el formato anterior van en la defición de clases, sus métodos y miembros.

Ejemplo de documentación básica de un EJB de sesión

```
package co.edu.uis.cafeteria.mantenimientos;

import java.util.List;

import javax.ejb.EJB;
import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.Query;
```

```
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Destroy;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.international.StatusMessages;

import co.edu.uis.cafeteria.productos.ConsultarGeneralVenta;
import co.edu.uis.excepciones.DSIException;
import co.edu.uis.minutas.entidades.PuntoVenta;
import org.jboss.seam.international.StatusMessage.Severity;
import co.edu.uis.servicios.FormatoWeb;

/**
 *
 * @author Carlos Javier Prieto y Jonathan Carrero
 * Este EJB permite crear, modificar y eliminar un punto de venta )
 *
 */
@Stateful
@Scope(ScopeType.CONVERSATION)
@Name("administrarPuntoVenta")
public class AdministrarPuntoVentaEJB implements AdministrarPuntoVenta {

 @In
 private EntityManager    em;

 @In
 private StatusMessages     statusMessages;

 @EJB
 private ConsultarGeneralVenta    consultarGeneralVentaEJB;


 private boolean      transaccionExitosa;
```

```
  private boolean        elementoRepetido;
  private PuntoVenta     puntoVenta;
  private List<PuntoVenta>   puntosVentas;

  @Out(required = false)
  private PuntoVenta     puntoVentaTransaccion;


  /**
   * Este método asigna un punto de venta para que pueda ser
modificado
   *
   * @param aPuntoVenta
   */
  public void asignarPuntoVenta(PuntoVenta aPuntoVenta) {
    this.puntoVentaTransaccion = aPuntoVenta;
  }

  /**
   * Este método visualiza un punto de venta
   *
   * @param aPuntoVenta
   */
  public void verPuntoVenta(PuntoVenta aPuntoVenta) {
    this.puntoVentaTransaccion = aPuntoVenta;
  }

  /**
   * Este método crea un nuevo punto de venta
   */
  public void crearPuntoVenta() {
    try {
      try {
        if (this.puntoVenta != null &&
!this.puntoVenta.getNombrePuntoVenta().trim().equals("")) {
```

```
this.puntoVenta.setNombrePuntoVenta(this.puntoVenta.getNombrePuntoVenta()
.trim().toUpperCase());
        this.em.persist(puntoVenta);
        this.em.flush();
        this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.INFO,FormatoWeb.get("creacionExitosa", true));
        this.puntoVenta = new PuntoVenta();

      } else {

       this.statusMessages.addToControl("txtNombre", "Información
requerida");
      }

    } catch (Exception e) {
      throw new DSIException(e);
    }
   } catch (DSIException e) {
    this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.ERROR, e.getMessage());

   }
  }


  /**
   * Este método permite modificar el punto de venta
   */
  public void editarPuntoVenta() {
   try {
    try {
     this.transaccionExitosa = false;

     if (this.puntoVentaTransaccion.getNombrePuntoVenta() != null &&
!this.puntoVentaTransaccion.getNombrePuntoVenta().trim().equals("")) {
```

```
this.puntoVentaTransaccion.setNombrePuntoVenta(this.puntoVentaTransaccion
.getNombrePuntoVenta().trim().toUpperCase());
       this.em.merge(this.puntoVentaTransaccion);
       this.em.flush();
       this.em.refresh(this.puntoVentaTransaccion);
       this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.INFO,FormatoWeb.get("edicionExitosa", true));
       this.transaccionExitosa = true;

      } else {
       this.statusMessages.addToControl("txtNombre", "Información
requerida");
      }

    } catch (Exception e) {
      throw new DSIException(e);
    }
   } catch (DSIException e) {
    this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.ERROR, e.getMessage());
   }
  }

  /**
   * Este método elimina un punto de venta creado
   */
  public void eliminarPuntoVenta() {
   try {
    try {
     this.em.remove(this.em.merge(this.puntoVentaTransaccion));
     this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.INFO,FormatoWeb.get("eliminacionExitosa", true));
      this.em.flush();

    } catch (Exception e) {
      throw new DSIException(e);
```

```
    }
   } catch (DSIException e) {
    this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.ERROR, e.getMessage());
   }
  }

  /**
   * Este método borra el formulario de creación de puntos de venta
   */
  public void cancelarCreacion() {
   this.puntoVenta = new PuntoVenta();
  }

  /**
   * Este método determina si una transacción se realizó exitosamente o no
   *
   * @return transaccionExitosa
   */
  public boolean isTransaccionExitosa() {
   return transaccionExitosa;
  }

  /**
   * Este método asigna la bandera de deterinación si una transacción fue
   * exitosa o no
   *
   * @param transaccionExitosa
   */
  public void setTransaccionExitosa(boolean transaccionExitosa) {
   this.transaccionExitosa = transaccionExitosa;
  }

  /**
   * Este método obtiene los puntos de venta creados
   *
```

```
   * @return puntosVentas
   */
  public List<PuntoVenta> getPuntosVentas() {
    try {
      this.puntosVentas =
this.consultarGeneralVentaEJB.getPuntosDeVenta(null);
    } catch (DSIException e) {
      this.statusMessages.instance().addToControl("mensajesValidacion",
Severity.ERROR, e.getMessage());
    }
    return puntosVentas;
  }

  public void setPuntosVentas(List<PuntoVenta> puntosVentas) {
    this.puntosVentas = puntosVentas;
  }

  /**
   * Este método obtiene el punto de venta a crear
   *
   * @return puntoVenta
   */
  public PuntoVenta getPuntoVenta() {
    if (this.puntoVenta == null) {
      this.puntoVenta = new PuntoVenta();

    }
    return puntoVenta;
  }

  /**
   * Este método asigna valores al punto de venta
   *
   * @param puntoVenta
   */
  public void setPuntoVenta(PuntoVenta puntoVenta) {
```

```
    this.puntoVenta = puntoVenta;
  }

  @Destroy
  @Remove
  public void destroy() {
  }

}
```

CAPITULO 5

5. CONCLUSIONES

- La utilización de la metodología UML para el diseño del software fue de vital importancia para el posterior desarrollo del sistema de información, pues permitió tener una visualización clara y muy bien documentada de lo que el cliente solicitó en la fase de análisis de requerimientos.

- Con el nuevo sistema de información que soporta los procesos de cafetería se logra controlar y administrar de una forma mucho más eficiente y eficaz los procesos de cafetería teniendo en cuenta la cobertura del nuevo sistema y la disponibilidad en todo momento de la información actualizada,

- Este sistema de información fue desarrollado para soportar el manejo dinámico y flexible del servicio de cafetería.

- El modelo de construcción por prototipos permitió agilizar el desarrollo del sistema de información propuesto teniendo en cuenta que el usuario siempre tuvo una participación directa en las diferentes fases de cobijaron la realización de este proyecto.

- El sistema de información de Cafetería de Bienestar Universitario contribuye a la mejora continua de los procesos de la misma, aumentando la satisfacción de la comunidad universitaria UIS que hace uso de sus servicios-

- El proceso llevado entre la División de Servicios de Información y los estudiantes de Ingeniería de Sistemas alcanzó las expectativas esperadas por las partes interesadas, Para los estudiantes les permitió trabajar en un entorno desarrollo de software de altos requerimientos y para la DSI un producto de calidad cumpliendo con los estándares establecidos para el desarrollo de software de la universidad.

CAPITULO 6

6. RECOMENDACIONES

- La división de Bienestar debe realizar todos los esfuerzos posibles para que este sistema de información entre en funcionamiento teniendo en cuenta los beneficios que ofrece.

- La división de Bienestar debe hacer los ajustes en los procesos y procedimientos que soportan el funcionamiento del servicio de cafetería para aprovechar todo el potencial que ofrece el software.

- Dar a conocer, más ampliamente a las unidades académico administrativas y a la comunidad universitaria en general los nuevos servicios que ofrece este sistema.

- Dotar a la División de Bienestar Universitario de la infraestructura tecnológica necesaria para facilitar el uso de este sistema.

- Aprovechar que la Universidad cuenta con una División de Servicios de Información y una escuela de ingeniería de Sistemas, para seguir generando procesos de formación y fortalecimiento de los conocimientos adquiridos a lo largo de la carrera, logrando de esta manera un gran beneficio para las partes.

CAPITULO 7

7. BIBLIOGRAFÍA

[1] PRESSMAN, Roger. Ingeniería del Software, un enfoque práctico. Mc GraW Hill, 2005.

[2] PIATTINI Velthuis, Mario G, Aplicaciones Informáticas de Gestión. Una perspectiva de Ingeniería del Software. Alfaomega Ra-Ma. México. 2004.

[3] COBOS, Carlos Alberto; MENDOZA, Martha Eliana. Manual De Informix – SQL . Universidad Industrial de Santander, 1998.

[4] LÓPEZ GUSTAVO, Blanco Ignacio. Tesis en Ingeniería Informática

[5] GROFF, James R. – WEINBERG, Paul N. APLIQUE SQL. OsBORNE-McGrawHill, 1991.

[6] FERRÉ GRAU, Xavier - SÁNCHEZ S. María Isabel. Desarrollo Orientado a Objetos con UML. Facultad de Informática - UPM.

[7] Estándares de desarrollo. División de Servicios de Información. Bucaramanga, UIS.

Enlaces sitios WEB

- http://livedemo.exadel.com/richfaces-demo/richfaces/comboBox.jsf?tab=usage&cid=170149

- http://www.sparxsystems.com.ar/

- http://download.oracle.com/docs/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/pdldocs/facelets/index.html