Universidad de las Ciencias Informáticas

Facultad 4

Título: Subsistema de distribución de comensales para el sistema de gestión de servicios de alimentación en la UCI.

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Dalquerine Castillo Castillo.

Tutor: Ing. Yidier Romero Zaldivar.

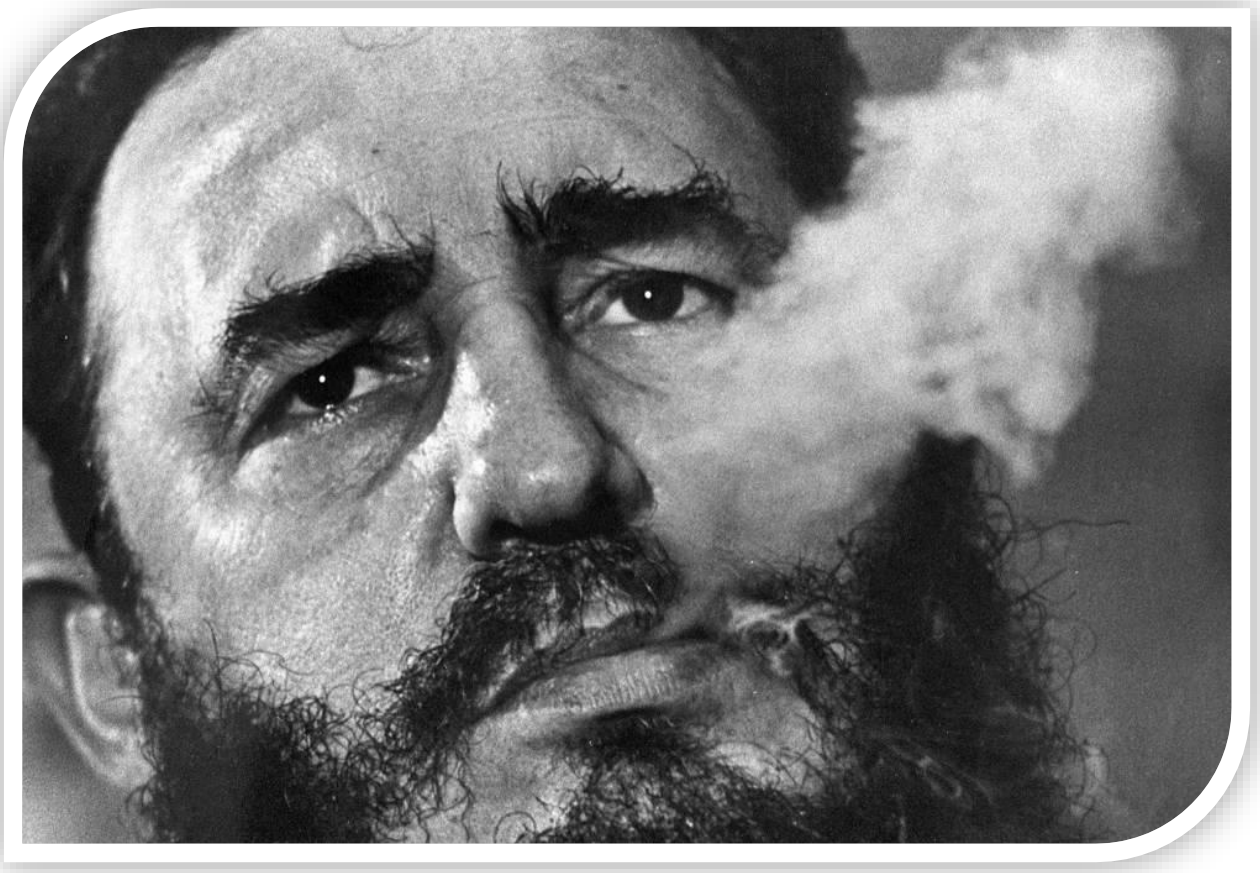
Ing. Lissette Soto Pelegrín.

Co-tutor: Yoan Céspedes Williams

Centro de Consultoría y Desarrollo de Arquitecturas Empresariales

La Habana, junio del 2018.

” Año 60 de la Revolución.”



¨ Si los jóvenes fallan, todo fallará. Es mi más profunda convicción que la juventud cubana luchará por impedirlo. Creo en ustedes. ¨

Fidel Castro Ruz

AGRADECIMIENTO

A mis padres, y en especial a mi madre por haberme apoyado en todo el transcurso de mi carrera dándome fuerzas para que siguiera adelante, cuidándome a mi hermosa hija Ashly para que me graduara. A mi hermano por estar mortificándome todo el tiempo, a mis tías por siempre preocuparse por cómo me iba en los estudios, a mi esposo porque a pesar que llevamos muy poco tiempo siempre tuve su ayuda comprensión, dedicación, ayuda y su amor. A mis tutores por guiarme en el transcurso de la realización de mi tesis. A mi cotutor Yoan Céspedes por siempre darme consejo, por ayudarme en todos estos años, por soportarme, por guiarme. A mis amigos Lamis, Jessi, Shirley, Daylen, Yailin, Gloria y Armando a pesar de que muchos ya no están tuve su cariño y su apoyo en todo lo que se podía, a todos los amistades que conocí en el transcurso de mi carrera como Jaime, Yanelis, Lijandy, entre otros que me quedan por mencionar. A todos gracias.

DEDICATORIA

A mis padres, por todo su amor y comprensión.

A mi hermano, por su cariño.

A mis tías, por todo su apoyo.

A mi esposo Yoan, por haberme ayudado en los momentos que más lo necesite.

A mis amigos que tanto me ayudaron en el transcurso de mi carrera.

DECLARACIÓN JURADA DE AUTORÍA

Declaramos ser autores del trabajo de diploma “Subsistema de distribución de comensales para el sistema de gestión de servicios de alimentación en la UCI” y otorgamos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo, así como, el derecho de uso en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

Firma del autor:

_____
Dalquerine Castillo Castillo

Firma del tutor:

_____
Ing. Yidier Romero Zaldivar

Firma del tutor:

_____
Ing. Lissette Soto Pelegrín

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) es una de las mayores universidades del país, en extensión y en personal. Cuenta con un sistema para gestionar el servicio de alimentación de los comensales de la institución. El sistema cuando se creó satisfacía las necesidades identificadas por la dirección de alimentos y se desarrolló en tecnologías propietarias (*framework* .Net). Con el paso de los años, el servicio de alimentación ha experimentado modificaciones y se identificaron nuevos requerimientos. Se decidió desarrollar un sistema de reservación de eventos (desayuno, almuerzo y comida) y al mismo tiempo realizar la migración a software libre del sistema en su totalidad. Lo anterior provocó la necesidad de migrar el subsistema de distribución de comensales a tecnologías libres, incorporando mejoras al mismo. El desarrollo se realizó sobre la plataforma *Java,* se utilizó Netbeans como entorno integrado de desarrollo (IDE por sus siglas en inglés), PostgreSQL como sistema gestor de base de datos y JPA para el Mapeo Objeto-Relacional (ORM por sus siglas en inglés).

Palabras claves: gestión de servicios de alimentación, sistema de distribución.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Con el transitar de los años, el desarrollo de las tecnologías de la información y las comunicaciones (TIC) ha provocado transformaciones en casi todos los ámbitos, se ha tratado de controlar la entrada y salida a lugares de forma eficiente para mejorar la calidad de vida de las personas dentro de un entorno social.

Con la implantación de nuevas tecnologías en diversas áreas, ha contribuido considerablemente al cambio de vida de la sociedad actual, lo que facilita el quehacer diario de todos los usuarios que de una forma u otra interactúa con estos sistemas. Con su uso se logran importantes mejoras, pues brinda una plataforma con toda la información necesaria para la toma de decisión. En Cuba desde muy temprano se ha identificado la necesidad de introducir a la sociedad las TIC para lograr una mejor cultura digital como una de las características propias de cada individuo.

Hoy en día, la progresiva evolución informática ha otorgado a distintas instituciones nacionales la implantación de nuevas tecnologías, un ejemplo de institución es la Universidad de las Ciencias Informáticas (UCI) una de las mayores universidades del país, en extensión y en personal, la cual cuenta con un sistema para gestionar el servicio de alimentación a los trabajadores y estudiantes de la institución.

El sistema cuando se creó satisfacía las necesidades identificadas por la Dirección de Alimentos. La tecnología utilizada para el desarrollo de este sistema fue el framework .Net, decisión tomada hace varios años y en ese momento las políticas de informatización de la UCI no potenciaban el desarrollo en software libre como lo está en la actualidad.

Con el paso de los años, el sistema de gestión de servicio de alimentación ha experimentado modificaciones y uno de los centros de desarrollo de la universidad, el Centro de Consultoría y Desarrollo de Arquitecturas Empresariales (CDAE), se dio a la tarea de realizar la migración a software libre de uno de los subsistemas que conforman el Sistema de Gestión de Servicios de Alimentación de la UCI, específicamente el **subsistema de distribución de comensales,** lo cual ayudaría a humanizar el proceso de distribución de personas por los diferentes comedores.

La aplicación utilizada actualmente permite una configuración limitada de la distribución de comensales por los distintos puntos de acceso a las diferentes áreas de alimentación de la universidad (comedores o *pantries*). El escenario más frecuente para la configuración de la distribución de comensales es cuando un especialista de la Dirección de Alimentos necesita planificar la distribución del día, el mismo puede configurar los comensales internos según su área de residencia (específicamente, según la manzana donde reside); y los comensales externos según su categoría como trabajador (todos juntos para que accedan a un mismo comedor, pasan por una puerta los trabajadores de servicio y por otra el resto de los profesores y especialistas externos). Existen otros escenarios donde la configuración a realizar no está incluida en el

anterior y donde el especialista tendrá que realizar dicha configuración de forma individual, según el comensal; tal es el caso de los comensales de entidades tercerizadas (XETID, SOFTEL, ETECSA) radicadas en el *campus* universitario. Además, de ese mismo modo se configuran todos los casos eventuales que no se ajusten al caso de comensales internos o externos.

La Dirección de Alimentos y la Dirección de Informatización han identificado un conjunto de escenarios donde los comensales podrían distribuirse por otro criterio distinto al criterio de si es "interno" o "externo". Un posible escenario es que todos los comensales de un área administrativa determinada puedan acceder a la misma puerta de comedor (esto trae el beneficio de fomentar unidad en los equipos de proyectos y departamentos). Otro posible escenario es que todos los comensales que trabajan en una misma edificación (docente, de residencia, de producción, etc.) puedan acceder a una misma puerta de comedor, de modo que la trayectoria de los comensales de su puesto de trabajo al comedor sea la menor posible. Estos escenarios pudieran activarse o no según las decisiones de la dirección de la universidad y para ese momento el sistema de distribución estaría habilitado con esas configuraciones. De esta manera el operador de la Dirección de Alimentos no tendría que realizar las configuraciones individuales a los comensales de una misma área, se humaniza la tarea y se minimiza la introducción de errores.

Con la realización del presente trabajo de diploma se pretende incidir en tres aspectos puntuales:

- La configuración de la distribución de comensales se encuentra limitada a solo un criterio: la condición de interno o externo. El resto de los comensales (terceros, eventuales, etc.) se configuran de manera individual.
- Se requiere la habilitación de una nueva configuración basada en el criterio del área administrativa donde labora el comensal.
- Se requiere migrar la tecnología que soporta el subsistema de distribución de comensales para que cumpla con la política de migración a software libre de la UCI.

A partir de lo expuesto anteriormente, se plantea como **problema a resolver** la siguiente interrogante: ¿cómo informatizar el proceso de distribución de comensales en el sistema de alimentación de la UCI?

La presente investigación tiene como **objeto de estudio** la informatización de procesos de distribución, enmarcando en el **campo de acción** el proceso de distribución de comensales en la UCI.

Se define como **objetivo general** el siguiente: desarrollar un sistema informático que contribuya a la informatización del proceso de distribución de comensales a los puntos de acceso de los comedores de la UCI.

Para dar cumplimiento al objetivo general planteado anteriormente se definen las siguientes **tareas de investigación**:

- Elaboración del marco conceptual para precisar los principales conceptos que se emplean en la investigación.
- Estudio del estado del arte de las herramientas existentes para la informatización de los procesos de distribución.
- Definición del ambiente de desarrollo a partir de las herramientas, tecnologías y lenguajes a utilizar.
- Definición de los requisitos funcionales y no funcionales del sistema.
- Descripción de la arquitectura a utilizar en la implementación de las funcionalidades.
- Implementación de la solución propuesta.
- Valoración de los resultados obtenidos.

En la presente investigación se hace uso de algunos **métodos teóricos y empíricos de investigación**, entre los que se encuentran:

Métodos teóricos

Permiten descubrir en el objeto de investigación las relaciones esenciales y las cualidades fundamentales. Por ello se apoya básicamente en los procesos de abstracción, análisis, síntesis, inducción y deducción(1).

- **Analítico-sintético**: Se emplea en el estudio de los sistemas de planificación, así como la búsqueda de información acerca de los lenguajes, herramientas y metodologías que son usados para la implementación de estos. También para estudiar cómo se desarrollan otros sistemas de planificación en un entorno de trabajo, analizando los resultados de sus deficiencias y características generales para así obtener un resultado eficiente que cumpla con los requisitos pedidos por el cliente.
- **Modelado:** Permitió realizar el modelado de todos los diagramas, esquemas y prototipos asociados.

Métodos empíricos

Su aporte al proceso de investigación es resultado fundamentalmente de la experiencia. Estos métodos posibilitan revelar las relaciones esenciales y las características fundamentales del objeto de estudio, a través de procedimientos prácticos con el objeto y diversos medios de estudio(1).

- **Observación:** se estudiaron soluciones informáticas similares a la que se desea desarrollar en la presente investigación, con el propósito de ver el funcionamiento de las mismas.

El trabajo de diploma se encuentra estructurado en 3 capítulos:

En el **Capítulo 1: Fundamentación teórica,** se define el marco conceptual donde se especifican los principales conceptos que se emplean en la investigación, se realiza un estudio del estado del arte de aplicaciones informáticas diseñadas para la informatización de procesos de distribución y se realiza una descripción de las tecnologías, herramientas y metodología de desarrollo seleccionadas.

El **Capítulo 2: Propuesta de solución**, presenta la descripción del proceso de distribución de comensales de la UCI, los requisitos funcionales y no funcionales del sistema desarrollado y la descripción de la arquitectura utilizada en la implementación de las funcionalidades.

Por su parte en el **Capítulo 3: Implementación y validación**, se muestran el modelo de implementación, diagrama de despliegue, diagrama de componentes; se detallan además las pruebas realizadas a las funcionalidades del sistema y sus resultados.

CAPÍTULO 1. FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se observan los elementos que permiten visualizar acerca del tema abordado en la investigación. Se innova en los conceptos fundamentales asociados al objeto de estudio para lograr un mejor entendimiento sobre el problema de investigación y los fundamentos teóricos que sustentan la investigación. Se realiza un estudio de la metodología, tecnologías, herramientas y lenguajes que serán usados en el desarrollo de la solución.

1.1 Marco conceptual

A lo largo de esta sección se espera poder dar al lector una mejor comprensión y entendimiento de la investigación desarrollada.

- ✓ **Distribución:** La distribución es la acción y efecto de distribuir a varias personas para que pueda acceder a un punto de control específico, durante la ocurrencia de un evento.
- ✓ **Comensales:** Es cada una de las personas que comen en un determinado lugar. En la universidad, es el personal que estudia y trabaja en ella, que acceden al servicio de alimentación correspondiente a cada comedor.
- ✓ **Control de acceso:** es un sistema automatizado que permite de forma eficaz, aprobar o negar el paso de personas o grupo de personas a zonas restringidas en función de ciertos parámetros de seguridad establecidos por una empresa, comercio, institución o cualquier otro ente. Los controles de acceso también hacen posible llevar un registro automatizado de los movimientos de un individuo o grupo dentro de un espacio determinado(2).

1.2 Sistemas homólogos

Después de definir y describir los principales conceptos relacionados con la investigación científica, se hace un estudio acerca de los principales sistemas homólogos, atendiendo particularmente la forma en la que realizan la distribución.

Nivel internacional

- ➢ ***Advanced Software* – Control de Comedores**

Desarrollado por la empresa *Advanced Software* que comercializa y distribuye el *software* de Control de Comedores, como módulo opcional e integrado en la aplicación de Control de Presencia o de Control de Accesos, ya sea en las versiones bajo base de datos Access o SQLServer/ORACLE, o bien de forma de aplicación independiente. Es una aplicación destinada a resolver con gran seguridad

el control de acceso del personal a los comedores de cualquier empresa con posibilidad de definir tipos de menú y valorarlos económicamente. De forma lógica y natural, hereda las máximas prestaciones en cuanto a control de accesos se refiere, en función de la aplicación base en la que se sostiene e integra o bajo la que ha sido adquirida como son la posibilidad de controlar centros remotos, con gestión de calendarios y puertas configurables. Incluye los elementos de gestión y clasificación de trabajadores por empresa, departamento, sección y categoría; la gestión de tarjetas de acceso; el registro y control de acceso a comedores, así como la creación de informes y estadísticas del sistema. De la misma forma pueden configurarse niveles de seguridad, indicando a qué puertas se tiene acceso y en qué franjas horarias independientes por día, asociándolos finalmente a los trabajadores. Se pueden agregar módulos opcionales como: Módulo de Exportación a Nómina, Módulo de Importación/Actualización de datos frecuentes y Módulo de Introducción de Marcajes mediante huella digital, entre otros(15).

Nivel nacional

- **Balance de Aguas**

Es un sistema para la asignación de las aguas disponibles en el país desarrollado por el Centro de *Software* de Holguín en el año 2010. La idea principal del sistema es controlar las entregas de agua que se establecen a diferentes entidades, a la vez que se obtienen las demandas de estos, teniendo en cuenta las condiciones de cada una de las fuentes de abasto al inicio del año. El sistema está estructurado en 4 opciones: Configuración, Gestión de datos, Gestión de procesos y Salidas del sistema o Reportes. Mediante la configuración se establecen los usuarios del sistema y se define el año del primer balance a realizar. Se establece además una gestión de provincias y municipios para la posibilidad de una nueva división político administrativa o cambios nacionales(16).

El sistema permite, además, generar e imprimir recuperaciones de la información obtenida en formato .pdf a través de diferentes criterios en dependencia del reporte que se escoja, ya sea de un año, trimestre, organismo o fuente de agua específicos; o según una zona, cuenca hidrográfica, sistema, provincia, dependiendo además de los permisos según el rol. Se obtienen salidas como(16):

- ✓ Modelo oficial de asignación de agua (para cada entidad).
- ✓ Propuesta de Balance de agua.
- ✓ Balance por tipo de fuente, por organismos y por entidades.

- ✓ Resumen general de utilización de las aguas.
- ✓ Balance general de fuentes y organismos para la Gaceta Oficial.

❖ **TEAMSOFT**

Es un sistema de soporte a la decisión diseñado con el propósito de apoyar el proceso de asignación de personal en las organizaciones de *software*. Fue desarrollado en el Instituto Superior Politécnico "José Antonio Echeverría" (CUJAE) en el 2010. La herramienta sustenta un modelo formal de asignación que integra factores que contribuyen a la asignación individual de roles en el proyecto y a la formación del equipo como un todo. Está desarrollada en Java y posee una interfaz amigable y flexible, capaz de tratar los factores ya sea como objetivos o como restricciones, y de ajustar algunos parámetros del modelo. La herramienta estructura el proceso de asignación de personal en dos etapas, asignación del jefe de proyecto y asignación del equipo. Antes de establecer el equipo de proyecto definitivo, se pueden evaluar tantas propuestas de equipos como deseen, considerado fijo al jefe de proyecto, quien fue previamente asignado(17).

Durante estas evaluaciones resulta posible ajustar algunos parámetros del modelo, seleccionar los objetivos y las restricciones a tomar en cuenta, así como seleccionar el método y el algoritmo de solución. A pesar que estas últimas facilidades están asociadas a cuestiones más técnicas, el hecho de que la herramienta posea una interfaz flexible y amigable permite que resulte fácil para el decisor interactuar con el sistema, aún sin ser conocedores de la temática de investigación de operaciones y de no dominar el modelo, y los métodos y algoritmos de solución que subyacen. Los resultados obtenidos en pruebas muestran que su aplicación contribuye a que el proceso de asignación, en general, se torne más objetivo y transparente y ha probado ser un instrumento especialmente útil durante el proceso de asignación de personas a los equipos de proyectos de software(17).

Nivel universidad

❖ **Sistema de Gestión de Alimentación**

Este sistema brinda la posibilidad de realizar la asignación de comensales en la UCI, atendiendo a una determinada estructura de comedores. Permite seleccionar entre distribuciones existentes para fechas y períodos especiales, además de ofrecer reportes como cantidad de comensales que han pasado y desglosarlo por puerta o por tipo. Actualmente se encuentra en uso y ha demostrado ser una aplicación funcional durante los años de explotación, sin embargo presenta un conjunto de deficiencias que dificultan

el trabajo de sus operadores. Entre las funcionalidades principales con que cuenta se pueden encontrar las siguientes:

- Gestión de eventos: muestra los eventos relacionados con la estructura de comedores con los siguientes datos: nombre, hora de inicio y fin, si el evento está activo, y la cantidad de raciones permitidas.
- Gestión de estructuras: permite construir y editar la estructura de los comedores, permitiendo la adición y eliminación de los puntos de control de acceso a los mismos.
- Gestión de grupos: posibilita organizar al personal de la universidad en grupos más pequeños que responden a determinadas características.

❖ **Subsistema de asignación de comensales para la Universidad de las Ciencias Informáticas**

El subsistema fue creado en el año 2012 por el Centro de Informatización Universitaria (CENIA), con el objetivo de asignar a los comensales por las diferentes puertas de los comedores para obtener una mayor organización y agilidad, utilizando para ello herramientas y tecnologías libres para mejorar la asignación de comensales. El subsistema se integra al Sistema de Planificación y Control de la Alimentación existente en la universidad. Este sistema está compuesto además por el Subsistema de control de acceso, el cual se utiliza en las puertas de los comedores para restringir y controlar el acceso de los comensales y el Subsistema de reservación de alimentación y planificación de menú, el cual brinda la posibilidad a los comensales de la universidad de reservar y cancelar el servicio de alimentación para los distintos eventos.

Entre las funcionalidades principales que se identificaron en esta aplicación se pueden citar las siguientes:

- **Gestión de eventos**: se gestionan los eventos (Desayuno, Almuerzo, Comida, entre otros) que se efectúan en la universidad.
- **Estructuras y puntos**: se diseña en forma de árbol la estructura de los comedores de la universidad, así como los puntos de control que pertenecen a las mismas.
- **Grupos y personas**: Se crean grupos para organizar a las personas según criterios definidos por el administrador de la aplicación.
- **Asignar grupos a puntos de control**: los grupos anteriormente creados y que contienen a un conjunto de personas, son asignados a los puntos de control por donde dichas personas tendrán derecho a consumir los platos que se oferten durante un determinado evento.

Asignar eventos a puntos: se establecen los eventos que tendrán lugar en cada una de las puertas, permitiendo especificar para cada grupo asignado a dicha puerta los eventos particulares a los que estos tendrán acceso.

Después de un estudio realizado a los diferentes sistemas informáticos semejantes a nivel nacional y a nivel universidad, se adquirió funcionalidades similares, ya que fueron creados con el mismo propósito de ayudar a los usuarios a tener una mejor distribución, ejemplo de estas funcionalidades es la gestión de eventos, la gestión de estructuras y la asignación de un grupo a un punto de control los cuales serán incluidos en el subsistema para la gestión de la distribución del servicio de alimentación donde se pretende cubrir las necesidades y mejorar los procesos de distribución de comensales en la universidad.

Los sistemas de Balance de agua y TEAMSOFT a pesar de estar desarrolladas sobre la base del código abierto, no son sistemas generales que se puedan implantar con otro objetivo, que no sea para el cual se desarrollaron.

1.3 Herramientas y tecnologías de desarrollo

Teniendo en cuenta las características del sistema, se estudiaron y seleccionaron las herramientas y tecnologías a utilizar en el desarrollo del mismo. Un factor importante en esta selección, lo constituyó el hecho de que las herramientas fueran de código abierto considerando la necesidad de los cambios de las instituciones cubanas, además de lenguajes de programación y gestores de bases de datos que respondan a las necesidades de la aplicación a implementar.

1.3.1 Lenguaje de programación

Un lenguaje de programación es un lenguaje que utiliza palabras especialmente definidas, gramática y puntuación que una computadora entiende. Si se intentara ejecutar instrucciones en seudocódigo, la computadora sería incapaz de entenderlas. Pero, si se intentaran ejecutar instrucciones en un lenguaje de programación (en código fuente), la computadora sí las entendería(3).
Así como hay muchos lenguajes que se hablan en el mundo (inglés, chino, hindi, etc.), también existe una infinidad de lenguajes de programación. Algunos de los más populares son *Visual Basic*, *C++* y *Java*. Cada lenguaje de programación define sus propias reglas de sintaxis(3).
Java es un lenguaje de programación con el que se realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía *Sun Microsystems* con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras(4).

Una principal característica de *Java* es que es un lenguaje independiente de la plataforma. Eso quiere decir que si hacemos un programa en *Java* podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo, *Windows, Linux, Apple*, etc. Esto lo consigue porque se ha creado una Máquina de *Java* para cada sistema que hace de puente entre el sistema operativo y el programa de *Java* y posibilita que este último se entienda perfectamente(4).
La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Java está desarrollándose incluso para distintos tipos de dispositivos además del ordenador como móviles, agendas y en general para cualquier cosa que se le ocurra a la industria(4).

1.3.2 Entorno de desarrollo integrado

Un entorno de desarrollo integrado es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código. Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o varios de estos.

Netbeans en su versión 8.0, es considerada como una plataforma ágil para desarrollar diferentes tipos de aplicaciones, ya que soporta diferentes lenguajes de programación PHP, C, C++, Java y JavaScript, entro otros. Es un entorno de desarrollo muy usado por los programadores desde hace algunos años porque es libre y abierto, además permite trabajar con sistemas operativos como: *Solaris, Linux, Mac, Windows*. El manejo de los proyectos realizados en *Netbeans* es ágil porque ofrece el trabajo con GUI (Interfaz Gráfica de Usuario), y logra profundizar sus datos e información de forma rápida y sencilla(5).
Una de las características más importantes y reconocidas de *Netbeans* IDE es la forma ágil y fácil que tiene para desarrollar software de escritorio *Java*, App web y móviles, también desarrolla App HTML5 con HTML, JavaScript y CSS. Cuenta con un grupo de herramientas con las cuáles los desarrolladores de *Netbeans* trabajan para desarrollos de *PHP, C, C++, Java*, entre otros. Se considera a *Netbeans* IDE como uno de los mejores soportes en las últimas versiones de la Tecnología *Java*, el IDE es considerado oficial para Java 8 pues sus herramientas de trabajo: editores, analizadores de código, convertidores podrán de forma rápida y sencilla actualizar las aplicaciones y de esta manera hacer uso de las nuevas construcciones (operaciones, funciones y referencias de métodos) de lenguaje *Java*(6).

Ventajas:

- Una de las grandes ventajas de la plataforma *Netbeans* es que permite el desarrollo de aplicaciones en varios lenguajes de programación.

- Es una de las plataformas más idóneas para programadores que desarrollan soluciones empresariales muy diversas.
- Su instalación y actualización es muy simple, cuenta con un completo sistema de ayuda y plantillas que permiten agilitar el desarrollo.
- Su uso es totalmente gratis.

1.3.3 Gestor de base de datos

PostgreSQL en su versión 9.4, es un gestor de bases de datos orientadas a objetos muy conocido y usado en entornos de software libre porque cumple los estándares SQL92 y SQL99, y también por el conjunto de funcionalidades avanzadas que soporta, lo que lo sitúa al mismo o a un mejor nivel que muchos SGBD comerciales. PostgreSQL se distribuye bajo licencia BSD(7).
PostgreSQL destaca por su amplísima lista de prestaciones que lo hacen capaz de competir con cualquier SGBD comercial:

- Cuenta con un rico conjunto de tipos de datos permitiendo, además, su extensión mediante tipos y operadores definidos y programados por el usuario.
- Su administración se basa en usuarios y privilegios.
- Sus opciones de conectividad abarcan TCP/IP, sockets Unix y sockets NT, además de soportar completamente ODBC.
- Es altamente confiable en cuanto a estabilidad se refiere.
- Puede extenderse con librerías externas para soportar encriptación, búsquedas por similitud fonética, etc.
- Control de concurrencia multi-versión, lo que mejora sensiblemente las operaciones de bloqueo y transacciones en sistemas multi-usuario.
- Soporte para vistas, claves foráneas, integridad referencial, disparadores, procedimientos almacenados, subconsultas y casi todos los tipos y operadores soportados en SQL92 y SQL99.
- Implementación de algunas extensiones de orientación a objetos. En PostgreSQL es posible definir un nuevo tipo de tabla a partir de otra previamente definida.

1.3.4 Pgadmin

Pgadmin en su versión 3, es una popular herramienta de código abierto para la administración y desarrollo en la plataforma PostgreSQL. Puede ser usado en Linux, FreeBSD, Solaris, Mac OSX y Windows. Pgadmin

III está diseñado para responder a las necesidades de los usuarios; desde la escritura de simples consultas SQL hasta para el desarrollo de bases de datos complejas. La interfaz gráfica soporta todas las características y hace fácil la administración. La conexión con el servidor puede hacerse a través del Protocolo de Control de Transmisión/Protocolo de Internet (TCP / IP) o *Unix Domain Sockets* (en plataformas Unix), y puede utilizar *Secure Sockets Layer* (SSL) para la seguridad(8).

Características

- ✓ En Pgadmin3 se puede ver y trabajar con casi todos los objetos de la base de datos, examinar sus propiedades y realizar tareas administrativas.
- ✓ Una característica interesante de Pgadmin3 es que, cada vez que se realiza alguna modificación en un objeto, escribe la(s) sentencia(s) SQL correspondiente(s), lo que hace que, además de una herramienta muy útil, sea a la vez didáctica. También incorpora funcionalidades para realizar consultas, examinar su ejecución y trabajar con los datos.

1.3.5 Herramienta de modelado

La herramienta recomendada a utilizar para el modelado de la propuesta de solución es Visual Paradigm, debido fundamentalmente a que es una herramienta multiplataforma que ayuda a una rápida construcción de aplicaciones de calidad. A continuación, se brindan detalles de esta herramienta.

Visual Paradigm en su versión 8.0, es una herramienta CASE: Ingeniería de *Software* Asistida por computación. La misma propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación(9).

Visual Paradigm ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas. Constituye una herramienta privada disponible en varias ediciones, cada una destinada a satisfacer diferentes necesidades. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de *software* de forma fiable a través de la utilización de un enfoque Orientado a Objetos(9).

Se caracteriza por:

- Disponibilidad en múltiples plataformas (*Windows, Linux*).
- Diseño centrado en casos de uso y enfocado al negocio que generan un *software* de mayor calidad.

- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, con diferentes especificaciones.
- Licencia gratuita y comercial.
- Soporta aplicaciones Web.

1.3.6 JSF (Java Server Faces)

Java Server Faces (JSF) en su versión 2.0, es un entorno de desarrollo de aplicaciones Web en Java. Esta tecnología está diseñada para simplificar el desarrollo Web con Java, y fomenta la separación de la presentación de las interfaces de usuario con la lógica de la aplicación. JSF utilizaba JSPs, añadiendo biblioteca que contienen componentes de alto nivel (menús, paneles, campos de texto…), cada uno de estos componentes puede interactuar con el servidor de forma independiente. En la actualidad, se utilizan "Facelets" que son páginas con extensión xhtml que sirven para lo mismo que las JSF, pero son una forma más sencilla de trabajar. Como Java Server Faces es un *framework*, simplifica el diseño de la estructura de la aplicación y también proporciona librerías que hacen más fácil el desarrollo de la aplicación(10).

Con Java Server Faces es posible utilizar Ajax (Asynchronous JavaScript and XML). Ajax es un enfoque en el que las diferentes acciones que solicita el usuario se realizan dentro de una misma página, de tal manera que el servidor no genera una nueva página sino sólo los datos. En las aplicaciones tradicionales, cada petición al servidor hace que éste genere una nueva página HTML/XHTML. Las aplicaciones RIA (Rich Internet Applications), como lo es JSF, intentan simular las aplicaciones de escritorio y, cuando usan Ajax, son más rápidas que las aplicaciones tradicionales(10).

1.3.7 JPA (Java Persistence API)

JPA es un framework que fue creado con Java EE 5 para resolver problemas de persistencia de datos. Proporciona un modelo de persistencia para mapear bases de datos relacionales. Se puede utilizar para acceder y manipular datos relacionales de Enterprise Java Beans (EJBs), componentes web y aplicaciones Java SE(11).

JPA es una abstracción que está por encima de Java Database Connectivity (JDBC) lo que permite ser independiente de SQL. Todas las clases y anotaciones de esta API se encuentran en el paquete javax.persistence. Los principales componentes de JPA son(11):

- Mapeo de base de datos relacionales (ORM). Es el mecanismo para mapear objetos a los datos almacenados en una base de datos relacional.
- Un API administrador de entidad para realizar operaciones en la base de datos tales como crear, leer, actualizar, eliminar (CRUD).
- El Java Persistence Query Language (JPQL) que permite recuperar datos con un lenguaje de consultas orientado a objetos.
- Las transacciones y mecanismos de bloqueo cuando se accedan a los datos concurrentemente, la API Java Transaction (JTA).

1.3.8 GlassFish

GlassFish en su versión 4.0, es un servidor de aplicaciones que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito y de código libre, se distribuye bajo la licencia CDDL y la GNU GPL(12).

- GlassFish tiene como base al servidor Sun Java System Application.
- GlassFish es la implementación de referencia (RI) de Java EE 5.
- GlassFish es un proyecto Open Source modular que permite incluir sus librerías como parte de otros frameworks, toolkits y productos.
- GlassFish es la base de código de las distribuciones estables, certificadas y con opción de contratar soporte y mantenimiento del Servidor de Aplicaciones de Sun: Sun Java System Application Server.
- GlassFish es una comunidad que contribuye mediante código, detección de bugs, foros de discusión, documentación, wikis, blogs y otros medios a la difusión y éxito del proyecto. La comunidad GlassFish colabora en la compatibilidad de otros frameworks Java Open Source(12).

1.3.9 PrimeFaces

PrimeFaces en su versión 4.0, es una biblioteca de componentes visuales de código abierto para el conjunto Java Server Faces 2.0 con más de 100 componentes enriquecidos. Su objetivo principal es ofrecer un conjunto de componentes para facilitar la creación y diseño de aplicaciones web(11).

Los componentes de PrimeFaces cuentan con soporte nativo de Ajax, pero no se encuentra implícito, de tal manera que se tiene que especificar los componentes que se deben actualizar al realizar una petición proporcionando así mayor control sobre los eventos. Cuenta también con un módulo adicional TouchFaces para el desarrollo de aplicaciones web para dispositivos móviles con navegadores basados en WebKit(11).

Las principales características de PrimeFaces son:

- Soporte nativo de Ajax, incluyendo *Push/Coment.*
- Kit para crear aplicaciones web móviles.
- Es compatible con otras bibliotecas de componentes como Jboss RichFaces.
- Uso de JavaScript no intrusivo.
- Es un proyecto *open source*, activo y estable.

1.4 Metodología de desarrollo de software

El Proceso Unificado Ágil (AUP) una versión simplificada de RUP. Esta describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. AUP se preocupa especialmente de la gestión de riesgos. Propone que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y sean abordados en etapas tempranas del mismo. Para ello, se crean y mantienen listas identificando los riesgos de las etapas del proyecto. Especialmente relevante en este sentido es el desarrollo de prototipos del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos. AUP reúne en una única disciplina las disciplinas de Modelado de Negocio, Requisitos y Análisis y Diseño. El resto de disciplinas (Implementación, Pruebas, Despliegue, Gestión de Configuración, Gestión y Entorno) coinciden con las restantes de RUP(13).

Al igual que en RUP, en AUP se establecen cuatro fases que transcurren de manera consecutiva y que acaban con hitos claros alcanzados:

- Inicio: El objetivo de esta fase es obtener una comprensión común cliente-equipo de desarrollo del alcance del nuevo sistema y definir una o varias arquitecturas candidatas para el mismo.
- Elaboración: El objetivo es que el equipo de desarrollo profundice en la comprensión de los requisitos del sistema y en validar la arquitectura.
- Construcción: Durante la fase de construcción el sistema es desarrollado y probado al completo en el ambiente de desarrollo.
- Transición: el sistema se lleva a los entornos de preproducción donde se somete a pruebas de validación y aceptación y finalmente se despliega en los sistemas de producción.

A partir de lo anteriormente se proponen utilizar la metodología de desarrollo AUP que propone la Universidad de las Ciencias Informáticas (UCI), debido a que es la metodología que ha escogido la universidad para los proyectos de desarrollo, y está determinada por el proyecto por el cual se desarrolla el proceso.

De las cuatro fases que propone AUP (Inicio, Elaboración, Construcción, Transición) se decide para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes tres fases de *AUP* en una sola, la cual se denomina Ejecución y se agrega la fase de Cierre(14).

Fases de AUP-UCI:

- **Inicio**: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planificación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo, costo y decidir si se ejecuta o no el proyecto.
- **Ejecución**: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
- **Cierre**: en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del mismo.

Escenarios de AUP-UCI:

- **Escenario No 1**: Proyectos que modelen el negocio con Casos de Uso del Negocio (CUN), solo pueden modelar el sistema con Casos de Uso del Sistema (CUS).
- **Escenario No 2**: Proyectos que modelen el negocio con Modelo Conceptual (MC) solo pueden modelar el sistema con CUS.
- **Escenario No 3**: Proyectos que modelen el negocio con Descripción de Proceso de Negocio (DPN), solo pueden modelar el sistema con Descripción de Requisitos por Proceso DRP.
- **Escenario No 4**: Proyectos que no modelen negocio solo pueden modelar el sistema con Historias de usuario (HU).

Para darle solución a la situación planteada, se hará uso del **escenario No 4** para modelar el sistema, el cual permite encapsular los requisitos funcionales en HU, por lo tanto, este escenario es aplicado en proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. Se recomienda en proyectos no muy extensos, pues una HU no debe poseer demasiada información.

1.5 Consideraciones del capítulo

Como resultado de la investigación en este capítulo concluimos que:

- Se realizó un estudio del estado del arte acerca de otras aplicaciones con propósitos similares a la propuesta, donde se adquirieron una serie de funcionalidades y características que son comunes en los sistemas de distribución estudiados para identificar cuál es la más indicada para darle solución a la problemática planteada además de aportar una visión más amplia del objeto de estudio de la investigación.
- Se realizó además un profundo estudio de las herramientas, tecnologías y lenguajes de programación idóneos para el desarrollo del subsistema, además de las metodologías de desarrollo de software.

CAPÍTULO 2. PROPUESTA DE SOLUCIÓN

En este capítulo se especifican los principales elementos que describen la propuesta de solución: los requisitos del software, los artefactos necesarios para la implementación del sistema, además se exponen las principales características del sistema, su diseño y arquitectura.

2.1 Descripción del proceso a automatizar

El Subsistema de Distribución de Comensales tiene como objetivo principal distribuir a los trabajadores y estudiantes por cada punto de acceso proporcionado de cada comedor. La aplicación debe permitir:

- La autenticación para los usuarios con roles definidos en la base de datos con acceso en el sistema a través del servidor de direcciones LDAP.
- El usuario autenticado que tenga el rol de administrador en el sistema, debe permitir la gestión de estructura que puede ser: complejo, comedor y pantry. Además de poder gestionar los puntos de acceso referentes a cada estructura, también se gestiona los usuarios y los roles que acceden en el subsistema.
- El usuario autenticado que posea el rol de planificador podrá gestionar las distribuciones de todos los trabajadores y estudiantes de la universidad, además de listar y aplicar cada distribución realizada. Podrá crear grupos y asignarlo a un punto de acceso determinado, mostrando al final un reporte con todos los datos de la distribución realizada por cada evento.

2.2 Requisitos funcionales

Los requisitos funcionales son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que éste reaccionará a entradas particulares y de cómo se debe comportar en situaciones particulares. En algunos casos, los requerimientos funcionales de los sistemas también declaran explícitamente lo que el sistema no debe hacer(18).

A continuación se muestran los requisitos funcionales obtenidos:

RF1: Autenticar usuarios.

RF2: Gestionar usuarios.

- RF2.1 Crear usuarios.
- RF2.2 Modificar usuarios.
- RF2.3 Eliminar usuarios.

RF3: Gestionar rol.

- RF3.1 Crear rol.
- RF3.2 Modificar rol.
- RF3.3 Eliminar rol.

RF4: Gestionar estructura complejo-comedor.

- RF4.1 Crear estructura complejo-comedor.
- RF4.2 Modificar estructura complejo-comedor.
- RF4.3 Eliminar estructura complejo-comedor.

RF5: Gestionar estructura comedor.

- RF5.1 Crear estructura comedor.
- RF5.2 Modificar estructura comedor.
- RF5.3 Eliminar estructura comedor.

RF6: Gestionar estructura pantry.

- RF6.1 Crear estructura pantry.
- RF6.2 Modificar estructura pantry.
- RF6.3 Eliminar estructura pantry.

RF7: Gestionar puntos de accesos.

- RF7.1 Crear puntos de accesos.
- RF7.2 Modificar puntos de accesos.
- RF7.3 Eliminar puntos de accesos.

RF8: Gestionar distribución de comensales.

- RF8.1 Crear distribución de comensales.
- RF8.2 Modificar distribución de comensales.
- RF8.3 Eliminar distribución de comensales.
- RF8.4 Listar distribución de comensales.

RF9: Aplicar distribución de comensales.

RF10: Crear grupos de comensales.

RF11: Asignar grupos a cada punto de acceso.

RF12: Mostrar reporte de la distribución de comensales.

2.2.1 Especificación de requisitos de software

Los requisitos para un sistema establecen con detalle las funciones, servicios y restricciones operativas del sistema. Estos requisitos reflejan la necesidad de los clientes de un sistema que ayuda a resolver problemas como el control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar esos servicios y restricciones se denomina ingeniería de requisitos(18).

Los requisitos funcionales identificados se representan a continuación:

Tabla 1 Requisitos funcionales

No	Nombre	Descripción	Prioridad	Complejidad
RF1	Autenticar usuario	El subsistema permitirá autenticar a los usuarios.	Alta	Baja
RF2	Crear usuario	Mediante esta funcionalidad se agregan nuevos usuarios.	Alta	Baja
RF3	Modificar usuario	El subsistema brindará la opción de modificar los datos pertenecientes a un usuario ya existente.	Alta	Baja
RF4	Eliminar usuario	Mediante esta funcionalidad se podrá eliminar un usuario seleccionado.	Alta	Baja
RF5	Crear rol	El subsistema debe permitir crearle un rol determinado a un usuario autenticado.	Alta	Baja
RF6	Modificar rol	El subsistema debe permitir la modificación del rol asignado a un usuario.	Alta	Baja

RF7	Eliminar rol	El subsistema debe permitir la eliminación de un rol asignado a un usuario dado.	Alta	Baja
RF8	Crear estructura complejo-comedor	Mediante esta funcionalidad se crean estructuras complejo-comedor.	Alta	Alta
RF9	Modificar estructura complejo-comedor	Mediante esta funcionalidad se modifica las estructuras complejo-comedor.	Alta	Alta
RF10	Eliminar estructura complejo-comedor	Mediante esta funcionalidad se elimina la estructura complejo-comedor.	Alta	Alta
RF11	Crear estructura comedor	Esta funcionalidad podrá crear una estructura comedor.	Alta	Alta
RF12	Modificar estructura comedor	Esta funcionalidad podrá modificar una estructura comedor.	Alta	Alta
RF13	Eliminar estructura comedor	Esta funcionalidad podrá eliminar una estructura comedor.	Alta	Alta
RF14	Crear estructura pantry	Esta funcionalidad podrá crear una estructura pantry.	Alta	Alta
RF15	Modificar estructura pantry	Esta funcionalidad podrá modificar una estructura pantry.	Alta	Alta
RF16	Eliminar estructura pantry	Esta funcionalidad podrá eliminar una estructura pantry.	Alta	Alta
RF17	Crear puntos de accesos	Esta funcionalidad permitirá crear un nuevo punto de acceso.	Alta	Alta

RF18	Modificar puntos de accesos	Esta funcionalidad permitirá modificar un punto de acceso seleccionado.	Alta	Alta
RF19	Eliminar puntos de accesos	Esta funcionalidad permitirá eliminar cualquier punto de acceso.	Alta	Alta
RF20	Crear distribución de comensales	El subsistema permitirá crear una nueva distribución.	Alta	Alta
RF21	Modificar distribución de comensales	El subsistema permitirá modificar cualquier distribución seleccionada.	Alta	Alta
RF22	Eliminar distribución de comensales	El subsistema permitirá eliminar una distribución seleccionada.	Alta	Alta
RF23	Listar distribución de comensales	El subsistema permitirá listar todas las distribuciones.	Alta	Media
RF24	Aplicar distribución de comensales	Esta funcionalidad permitirá aplicar la distribución a un grupo seleccionado.	Alta	Media
RF25	Crear grupos de comensales	El subsistema permitirá crear un nuevo grupo, editando correctamente los campos Nombre del grupo y Descripción.	Alta	Media
RF26	Asignar grupos a cada punto de acceso	Mediante esta funcionalidad se podrá asignar a cada grupo de comensales en los distintos puntos de acceso.	Alta	Alta
RF27	Mostrar reporte de la	Esta funcionalidad permitirá mostrar los reportes de cada distribución de comensales.	Alta	Media

	distribución de comensales			

2.2.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos requisitos que no se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. De forma alternativa, definen las restricciones del sistema como la capacidad de los dispositivos de entrada/salida y la representación de datos que se utiliza en la interface del sistema(18).

Tabla 2 Requisitos no funcionales

No.	Atributo de calidad	Sub-Atributo	Objetivo
RnF1	Usabilidad	RnF1.1	El subsistema debe presentar un menú, que permitan el acceso rápido a la información por parte de los usuarios que acceden al sistema.
RnF2	Soporte	RnF2.1	El subsistema contará con un grupo de soporte y asesoría al cliente.
RnF3	Software	RnF3.1	El subsistema debe funcionar sobre cualquier distribución de sistema operativo tanto en Linux como en Windows.
	Software	Rn3.2	Para la utilización del subsistema se necesitará el servidor web Grassfish y un gestor de Bases de datos.
RnF4	Hardware	RNF4.1	Para la utilización del subsistema se necesitará una PC con una memoria RAM de al menos 1GB y un disco duro de 150 GB.

RnF5	Seguridad	RnF5.1	El subsistema deberá permitir la autenticación mediante el directorio LDAP que proporciona la universidad.
	Seguridad	RnF5.2	El subsistema debe garantizar que la información sea vista y modificada por aquellos usuarios con los permisos que le han sido otorgados según su rol.

2.2.3 Historia de Usuario (HU)

Entre los artefactos que genera la metodología AUP se encuentran las Historias de Usuarios (HU) que permiten una descripción de los requisitos. Por tanto, se hará uso de esta técnica para encapsular los requisitos funcionales.

Una historia de usuario es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos. Las historias de usuario son una forma rápida de administrar los requisitos de los usuarios sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo para administrarlos. Las historias de usuario permiten responder rápidamente a los requisitos cambiantes(19).

2.2.4 Descripción de las Historia de Usuario (HU)

Durante el esbozo de la propuesta de solución se identificaron 12 historias de usuario que responden a las diferentes funcionalidades solicitadas por el cliente, además de una descripción que realiza el desarrollador para posteriormente conocer su implementación. A continuación se representan las Historias de Usuarios referentes a los requisitos funcionales “Crear grupos de comensales” y “Gestionar estructuras comedor”, el resto de las representaciones de las Historias de Usuario se encuentran en los anexos.

Tabla 3 Historia de Usuario 1: Crear grupos de comensales

Historias de usuario	
Número: HU 1	**Nombre del requisito**: Crear grupos de comensales
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días

Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción “Grupo”, se muestre un listado con los usuarios existentes y se cree un nuevo grupo con los usuarios elegidos, además de las opciones “Guardar” y “Cancelar”.	
Observaciones: NA	
Prototipo de interfaz: 	

Tabla 4 Historia de Usuario 2: Gestionar estructuras comedores

Historia de usuario	
Número: HU 2	**Nombre del requisito**: Gestionar estructura comedor
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado:** 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción “Gestionar estructura comedor”, se muestre un listado con las estructuras comedor existente en el sistema. Para cada estructura mostrada se habilitarán las opciones “Crear”, “Modificar” y “Eliminar”, además de la opción “Guardar” y “Cancelar”.	
Observaciones: NA	

Prototipo de interfaz

2.3 Modelo de diseño

El modelo de diseño debe definir todo lo que es necesario hacer para alcanzar el código final, se concentra en dos aspectos principales, el diseño de objetos y el diseño de sistemas. El diseño de objetos incluye la selección de algoritmos y estructuras de datos para satisfacer los objetivos de rendimiento y espacio del proyecto de software y el diseño de sistema define las decisiones estratégicas sobre cómo organizar la funcionalidad del sistema en torno al ambiente de implementación, incluyendo tanto hardware como software(20).

2.3.1 Diagrama de clases

Un diagrama de clases representa en un esquema gráfico, las clases u objetos intervinientes y como se relacionan en un escenario, sistema o entorno. Con estos diagramas, se logra diseñar el sistema a ser desarrollado en un lenguaje de programación, generalmente orientado a objetos. Estos diagramas los incorporan algunos entornos de desarrollo, tal es el caso de Eclipse con el plugin Papyrus o Netbeans con su respectivo plugin UML(21).

Los diagramas de clases juegan un papel muy importante ya que permiten visualizar a partir de las clases y sus vínculos, como los objetos interactúan en el entorno propuesto(21).

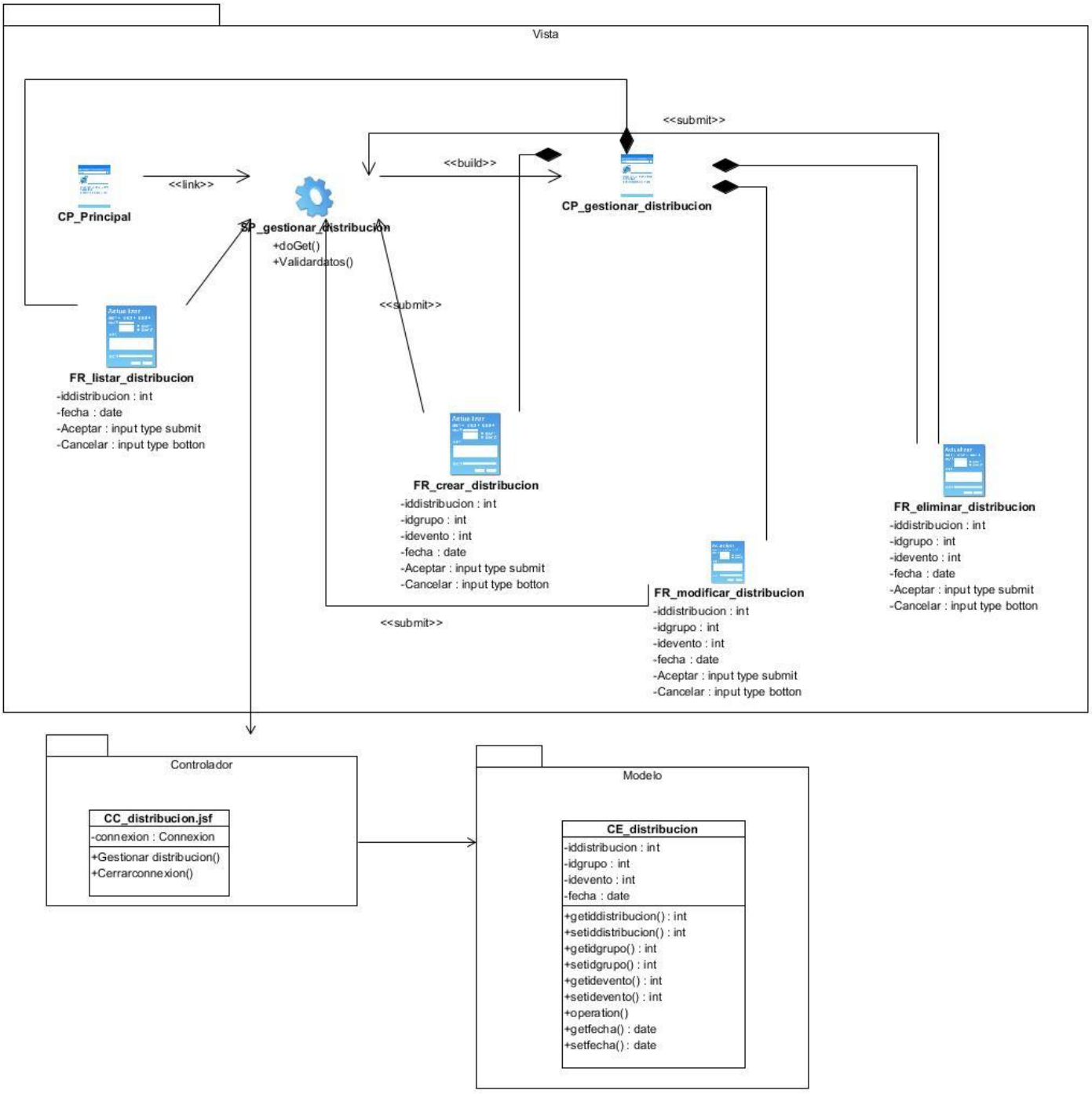


Figura 1: Diagrama de clase RF20, RF21, FR22 y RF23.

2.4 Estilo y patrones arquitectónico

El estilo arquitectónico **Llamada y retorno** es el utilizado en la solución propuesta ya que se consigue una estructura del programa que resulta relativamente fácil de modificar y cambiar de tamaño. Esta familia de

estilos enfatiza la modificabilidad y la escalabilidad. Son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas(22).

Los patrones arquitectónicos, por su parte, se han materializado con referencia a lenguajes y paradigmas también específicos de desarrollo, mientras que ningún estilo presupone o establece preceptivas al respecto(22).

Modelo- Vista- Controlador (MVC)

El patrón conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clasificaciones diferentes:

- ✓ Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- ✓ Vista: Maneja la visualización de la información.
- ✓ Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases. Esta separación permite construir y probar el modelo independientemente de la representación visual(23).

Ventajas:

- ✓ Soporte de vistas múltiples: Dado que la vista se encuentra separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.
- ✓ Adaptación al cambio: Los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación, o requerir soporte para nuevos dispositivos como teléfonos celulares o PDAs. Dado que el modelo no depende de las vistas, agregar nuevas opciones de presentación generalmente no afecta al modelo.

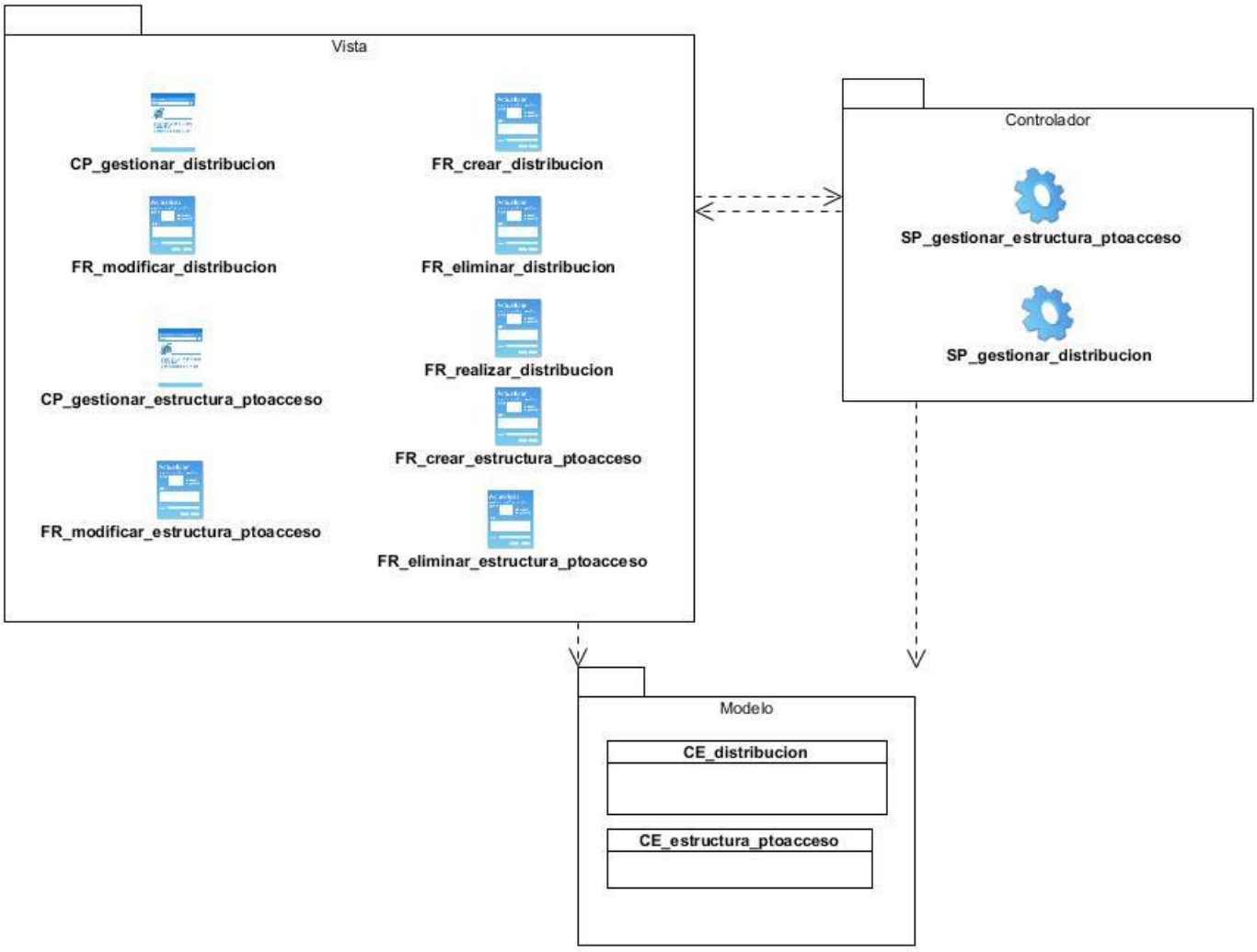


Figura 2: Diagrama de paquetes MVC del subsistema

2.5 Patrones de diseño

Un patrón de diseño es una solución repetible a problemas típicos y recurrentes en el diseño del software. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. No son un diseño terminado que puede traducirse directamente a código, sino más bien una descripción sobre cómo resolver el problema, la cual puede ser utilizada en diversas situaciones(24).

Patrones de diseño GRASP

Estos patrones son guías o principios que sirven para asignar responsabilidades a las clases. A continuación se muestran los utilizados:

Experto: Se encarga de asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para la realización de la asignación. Indica la responsabilidad de la creación de un objeto o la implementación de un método, debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo se obtiene un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento) (24).

Ejemplo: Este patrón se evidencia en la clase Usuario la cual tiene la información necesaria para gestionar los usuarios.

Bajo acoplamiento: Asigna las responsabilidades de manera que el acoplamiento (innecesario) se mantenga bajo(25).

Ejemplo: En la solución propuesta las clases de las vistas y los modelos de dominios, son totalmente independientes uno de otras, es decir, no existe una relación directa entre clases debido a la separación que hace el patrón arquitectónico Modelo-Vista-Controlador (MVC).

Alta cohesión: Asignar una responsabilidad de manera que la cohesión permanezca alta(24).

Asignar a las clases responsabilidades que trabajen sobre una misma área de aplicación y que no tengan mucha complejidad. Mejoran la claridad y facilidad con que se entiende el diseño.

Ejemplo: Este patrón permite agrupar de forma clara las vistas, la clase controladora y las clases modelos, estas últimas donde se implementa la lógica de negocio.

Controlador: es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control(24).

Ejemplo: Este patrón se evidencia en la clase controladora del subsistema, por ejemplo, la clase principal.

Creador: Se encarga asignar a la clase B la responsabilidad de crear una instancia de clase A(24). B es un creador de los objetos A. Este patrón es utilizado en las clases controladoras donde se encuentran las acciones definidas para el sistema y se ejecutan en cada una de ellas. En dichas acciones se crean los objetos de las clases que representan las entidades, lo que evidencia que las clases controladoras son creadoras de dichas entidades.

Ejemplo: Este patrón puede ser evidenciado en las clases *controladoras* que crean las instancias propias que se almacenar en la Base de Datos.

Patrones GOF

Los patrones GOF (*Gand of Four*, Banda de los Cuatro) son patrones de diseño que definen una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Estos patrones se dividen en tres categorías: los creacionales, los estructurales y los de comportamiento. Seguidamente se expondrán los patrones empleados(23).

Creacionales:

- ✓ **Singleton:** Está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. En este caso puede ser evidenciado en la clase Facade.

Estructural

- ✓ **Fachada:** Este patrón trata de simplificar la interfaz entre dos sistemas o componentes de *software*. Oculta un sistema complejo detrás de una clase que funciona como pantalla o fachada. Se puede encontrar la utilización de este patrón en la entidad Facade para acceder a la Base de Datos.

2.6 Modelo lógico y físico

Modelo de datos es un conjunto de conceptos que permite describir los datos, las relaciones entre ellos, la semántica y las restricciones de consistencia. El modelo entidad-relación es uno de los enfoques de datos más utilizado debido a su simplicidad y legibilidad, la cual se ve favorecida, ya que proporciona una notación diagramática muy comprensiva(26). En la figura 4 se muestran las diferentes tablas con sus atributos y sus respectivas relaciones.

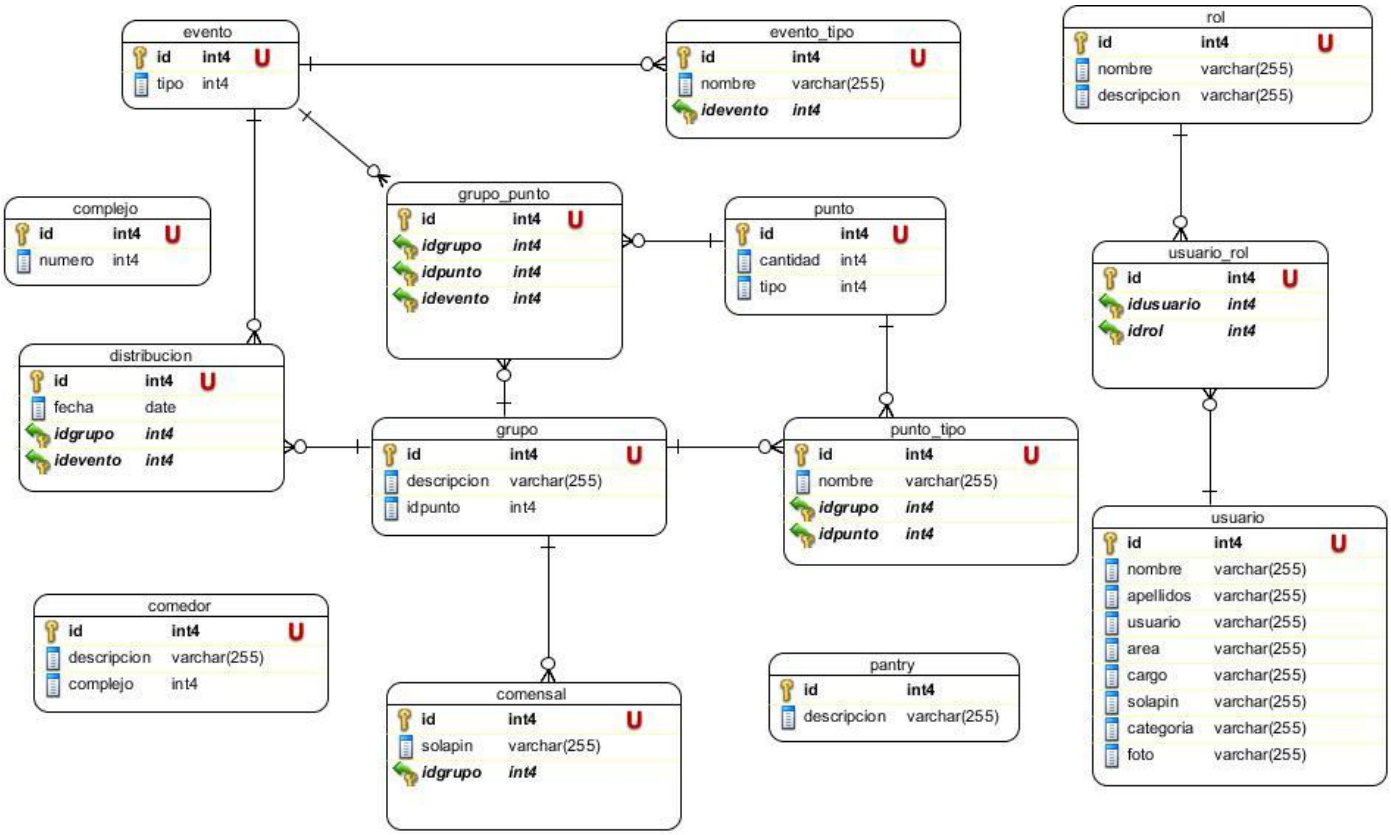


Figura 3: Diagrama Entidad-Relación

2.7 Consideraciones del capítulo

Una vez realizado el análisis y el diseño del subsistema se obtuvieron las siguientes consideraciones:

- Se obtuvo una mejor visión de las características del subsistema debido a que se explican una serie de conceptos y términos asociados al subsistema mediante un modelo de dominio permitiendo así identificar los requisitos de la propuesta de solución, los cuales se describen mediante artefactos que plantean las metas y condiciones que el subsistema debe cumplir.
- Se describió la arquitectura utilizada, así como los patrones de diseño empleados durante todo el desarrollo del subsistema.

CAPíTULO 3. IMPLEMENTACIÓN Y VALIDACIÓN

Una vez diseñada la propuesta de solución, se procede a la implementación de cada una de las funcionalidades, se muestra el código fuente de una de las principales clases y las pruebas aplicadas al sistema para demostrar la validez del mismo.

3.1 Modelo de implementación

El Modelo de Implementación es comprendido por un conjunto de componentes y subsistemas que constituyen la composición física de la implementación del sistema. Entre los componentes se encuentra datos, archivos, ejecutables, código fuente y los directorios. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos(9).

3.2 Diagrama de paquetes

Un paquete es un mecanismo utilizado para agrupar elementos de UML. Permite organizar los elementos modelados con UML, facilitando de ésta forma el manejo de los modelos de un sistema complejo. Permiten dividir un modelo para agrupar y encapsular sus elementos en unidades lógicas individuales. En general, pueden tener una interfaz (métodos de clases e interfaces exportadas) y una realización de éstas interfaces (clases internas que implementan dichas interfaces). Se pueden utilizar para plantear la arquitectura del sistema a nivel macro. Los paquetes pueden estar anidados unos dentro de otros, y unos paquetes pueden depender de otros paquetes(27).

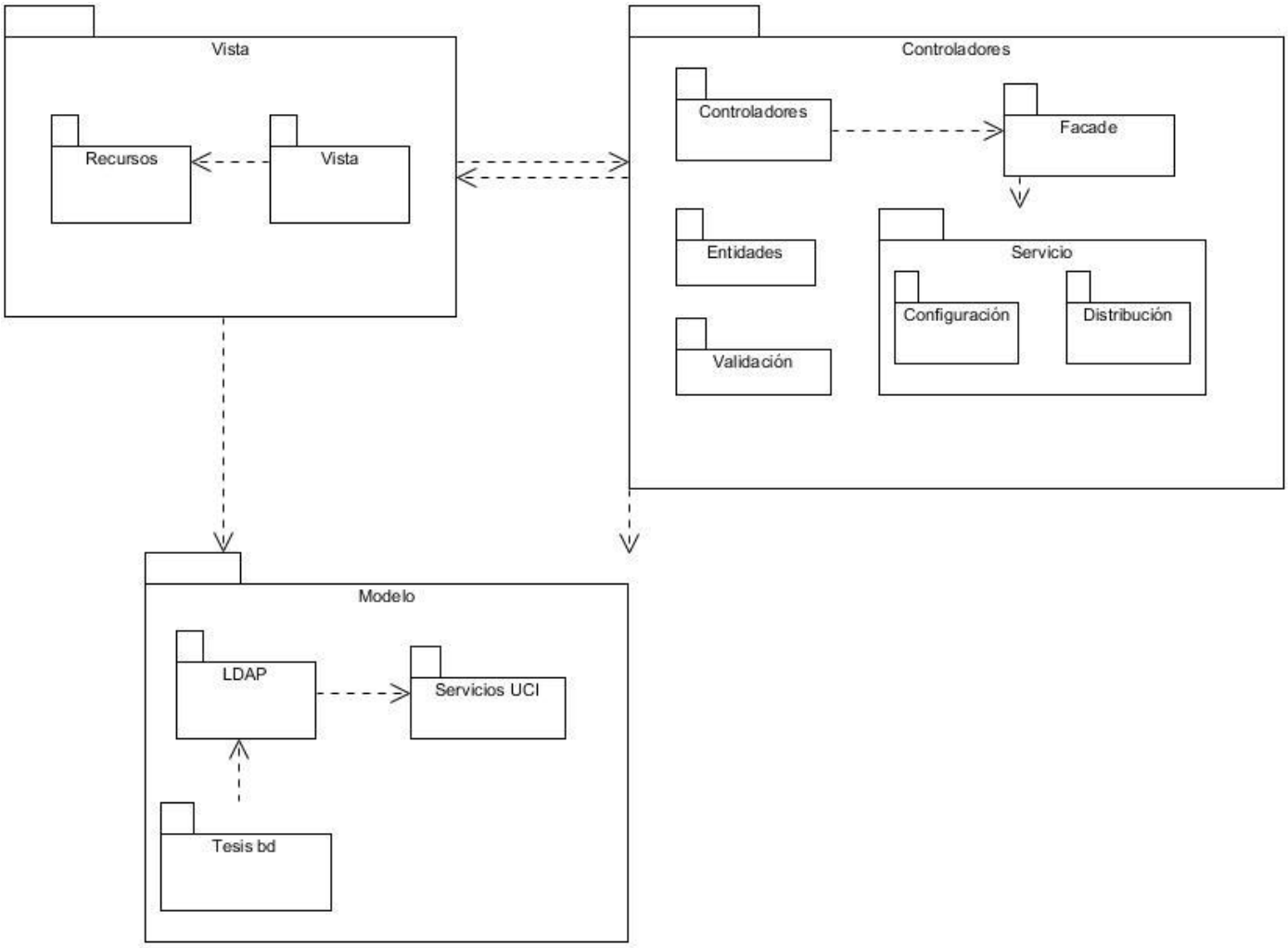


Figura 4: Diagrama de paquetes

3.3 Diagrama de componentes

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones de manera gráfica a través del uso de nodos y arcos entre estos.

Normalmente los diagramas de componentes contienen:

- ✓ Componentes
- ✓ Interfaces
- ✓ Relaciones de dependencia, generalización, asociaciones y realización.
- ✓ Paquetes o subsistemas
- ✓ Instancias de algunas clases

Visto de otro modo un diagrama de componentes puede ser un tipo especial de diagrama de clases que se centra en los componentes físicos del sistema(28).

A continuación, se muestra el diagrama de componentes correspondiente al subsistema:

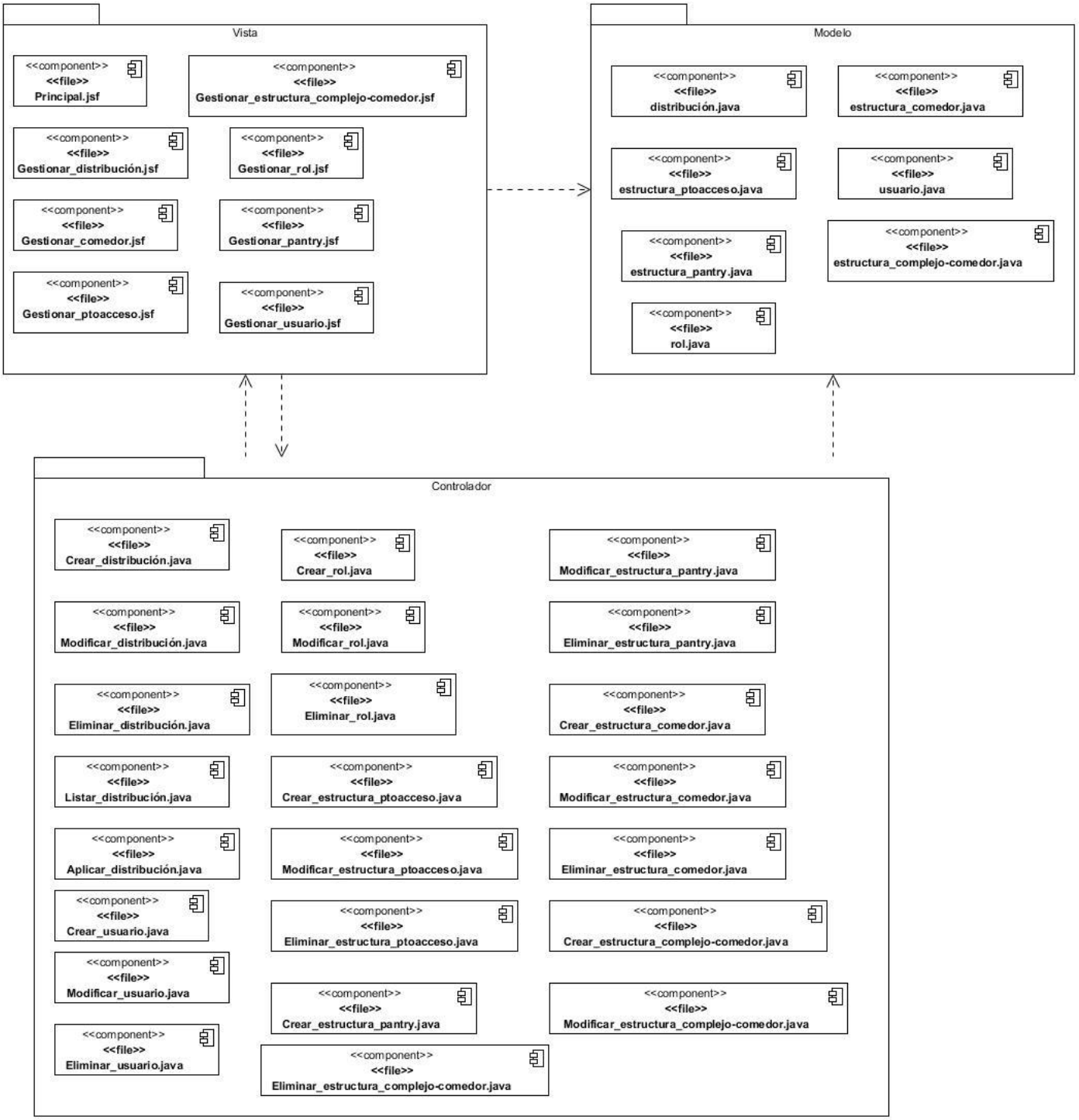


Figura 5: Diagrama de componentes

3.4 Diagrama de despliegue

Los Diagramas de Despliegue muestran las relaciones físicas de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Un nodo es un recurso de ejecución tal como un computador, un dispositivo o memoria(29).

Un diagrama de despliegue es un grafo de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes *software*, objetos, procesos (caso particular de un objeto). En general un nodo será una unidad de computación de algún tipo, desde un sensor a un *mainframe*. Las instancias de componentes *software* pueden estar unidas por relaciones de dependencia, posiblemente a interfaces (ya que un componente puede tener más de una interfaz) (29).

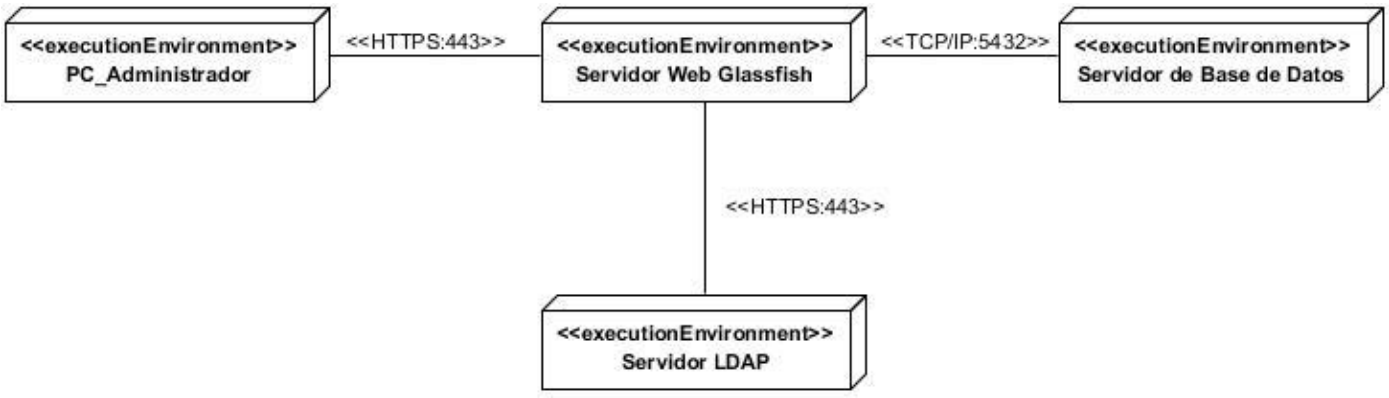


Figura 6: Diagrama de despliegue

Componentes del diagrama de despliegue:

- ✓ **Nodo:** Se representa con la figura de un cubo. El nodo se etiqueta con un nombre representativo de la partición física que simboliza. Se pueden asociar a los nodos subsistemas de construcción.
- ✓ **Conexiones:** Las conexiones se representan con una línea continua que une ambos nodos y pueden tener una etiqueta que indique el tipo de conexión. (ejemplo: canal, red, protocolo, etc.)
- ✓ **PC_Administrador:** Computadora en la cual se utilizará la aplicación.
- ✓ **Servidor de Base de Datos:** Simboliza el servidor donde estará el subsistema gestor de base de datos que dará respuesta a las peticiones hechas por la aplicación, además de almacenar los datos actualizados de las personas por cada distribución.
- ✓ **Servidor LDAP:** Representa el servidor de base de datos al cual se integra el subsistema para obtener los datos de las personas.

- ✓ **Servidor Web GlassFish:** ordenador en que se encuentra el servidor web GlassFish, este será el lugar en que se gestione todo el contenido de la aplicación. El mismo establecerá comunicación con los ordenadores clientes mediante protocolo HTTPS y con el servidor de base de datos por medio del protocolo TCP/IP.

3.5 Pruebas del software

Las pruebas de *software* son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo sistemáticamente. Por esta razón, se debe definir en el proceso de la ingeniería del *software* una plantilla para las pruebas del *software*: un conjunto de pasos en los que se puede situar los métodos específicos de diseño de casos de prueba. Las pruebas constituyen el último bastión desde el que se puede evaluar la calidad y de forma más pragmática descubrir los errores. La prueba del *software* es un elemento de un tema más amplio que, a menudo, es conocido como verificación y validación(9).

Para alcanzar el objetivo de las pruebas de *software* es necesario planear y ejecutar una serie de pasos (pruebas de unidad, integración, validación y sistema). Las pruebas de unidad e integración se concentran en la verificación funcional de cada componente y en la incorporación de componentes en la arquitectura del *software*. La prueba de validación demuestra el cumplimiento de los requisitos del *software* y la prueba del sistema valida el *software* una vez que sea incorporado a un sistema mayor(9).

3.5.1 Estrategias de prueba del *software*

Una estrategia de prueba del *software* integra las técnicas de diseño de caso de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción de *software*. La estrategia proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba, cuándo se deben planificar y realizar esos pasos, y cuánto esfuerzo, tiempo y recursos que se van a requerir. Por lo tanto, cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de las pruebas y la agrupación y evaluación de los datos resultantes(9).

Existen cuatro niveles o estrategias de pruebas según lo define Pressman:

- Prueba de unidad
- Prueba de integración
- Prueba de validación
- Prueba de sistema.

En el Subsistema de Distribución de comensales para la Dirección de Alimentos se define como estrategia o nivel de prueba a usar el nivel de unidad y el nivel de sistema.

La **prueba de unidad** hace un uso intensivo de las técnicas de prueba de caja blanca, ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de errores.

La **prueba del sistema** verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. Durante la prueba de sistema se usan exclusivamente técnicas de prueba de caja negra.

3.5.2 Pruebas del sistema

La prueba del sistema, realmente, está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas(9)**.**

Caja negra

Las técnicas de diseño de caja negra, también llamadas pruebas de comportamiento, son las que utilizan el análisis de la especificación, tanto funcional como no funcional, sin tener en cuenta la estructura interna del programa para diseñar los casos de prueba y, a diferencia de las pruebas de caja blanca, estas pruebas se suelen realizar durante las últimas etapas de la prueba(30).
Los métodos de caja negra permiten derivar casos de prueba que buscan encontrar los siguientes tipos de errores(31):

- ✓ Funciones incorrectas o faltantes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en acceso a BD externas.
- ✓ Errores de comportamiento o desempeño.
- ✓ Errores de inicialización o término.

Técnica de partición equivalente

La partición equivalente es un método de prueba de caja negra que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. Un caso de prueba ideal

descubre de forma inmediata una clase de errores (por ejemplo, proceso incorrecto de todos los datos de carácter) que, de otro modo, requerirían la ejecución de muchos casos antes de detectar el error genérico. La partición equivalente se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número total de casos de prueba que hay que desarrollar(9)**.**

1. Las clases de equivalencia se pueden definir de acuerdo con las siguientes directrices: Si una condición de entrada especifica un rango**,** se define una clase de equivalencia valida y dos no válidas.
2. Si una condición de entrada requiere un valor específico, se define una clase de equivalencia válida y dos no válida.
3. S una condición de entrada específica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
4. Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.

A continuación, se muestra una síntesis de los resultados obtenidos al aplicar las pruebas de Caja Negra (ver Tabla 5, 6 y 7), este método se aplicó a todos los requisitos funcionales, el cual mostró resultados satisfactorios.

Tabla 5: Caso de prueba Crear estructura comedor

Escenario	Descripción	Variable 1 id	Variable 2 descripción	Variable 3 complejo	Respuesta del sistema
EC 1.1 Crear estructuras	En este escenario se crean las estructuras correctamente.	V	V	V	Se muestra un mensaje: ¨Se creó la estructura correctamente¨.
		1	Comedor 1	1	
EC 1.2 Crear estructuras con datos incorrectos	En este escenario se realiza la creación de las estructuras con datos incorrectos.	V	I	V	Se muestra un mensaje: ¨El campo tipo es incorrecto ¨.
		1	Pantry 1	1	

EC 1.3 Crear estructuras con campos vacíos.	En este escenario se realiza la creación de la estructuras con campos vacíos.	V	V	NA	Se muestra un mensaje: ¨Existen campos vacíos¨.
		1	Comedor 1		
EC 1.4 Cancelar operación	En este escenario se cancela la operación de crear una nueva estructura.	NA	NA	NA	Se cancela la operación de adicionar una nueva estructura comedor.

Tabla 6: Caso de prueba Modificar estructura comedor

Escenario	Descripción	Variable 1 id	Variable 2 descripción	Variable 3 complejo	Respuesta del sistema
EC 2.1 Modificar estructuras	En este escenario se realiza la edición de una estructura comedor existente en el sistema correctamente.	V	V	V	Se muestra un mensaje: ¨Se modificó la estructura correctamente¨.
		1	Comedor 1	1	
EC 2.2 Modificar estructuras con datos incorrectos	En este escenario se realiza la edición de una estructura existente en el	V	V	I	Se muestra un mensaje: ¨Existen datos incorrectos¨.
		1	Comedor 1	tres	

	sistema con datos incorrectos.				
EC 2.3 Modificar estructuras con campos vacíos.	En este escenario se realiza la modificación de la estructuras con campos vacíos.	V	NA	V	Se muestra un mensaje: ¨Existen campos vacíos¨.
		1		1	
EC 2.4 Cancelar operación	En este escenario se cancela la operación de crear una nueva estructura.	NA	NA	NA	Se cancela la operación de modificar una estructura comedor.

Tabla 7: Caso de prueba Eliminar estructura comedor

Escenario	Descripción	Variable 1 id	Variable 2 descripción	Variable 3 complejo	Respuesta del sistema
EC 3.1 Eliminar estructuras correctamente	En este escenario se realiza la eliminación de cualquier estructura existente en el sistema correctamente.	V	V	V	Se muestra un mensaje: ¨Se eliminó la estructura correctamente¨.

EC 3.3 Cancelar operación	En este escenario se cancela la operación de crear una nueva estructura.	NA	NA	NA	Se cancela la operación de eliminar una estructura comedor.

3.5.3 Pruebas de unidad

Las pruebas de unicidad están enfocadas a los elementos más pequeños del *software*. Aplicable a componentes representados en el modelo de implementación para verificar que los flujos de control y de datos están cubiertos, y que ellos funcionen como se espera. La prueba de unidad está orientada a la técnica de prueba caja blanca(9)**.**

Caja blanca

La **prueba de caja blanca**, denominada a veces **prueba de caja de cristal** es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que(9)**:**

1. Garanticen que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo, programa o método.
2. Ejerciten todas las decisiones lógicas en las vertientes verdadera y falsa.
3. Ejecuten todos los bucles en sus límites operacionales.
4. Ejerciten las estructuras internas de datos para asegurar su validez.

Técnica de Camino básico

El método del camino básico permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida corno guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa(9).
A continuación se muestra un ejemplo realizado al método Autenticar usuario.

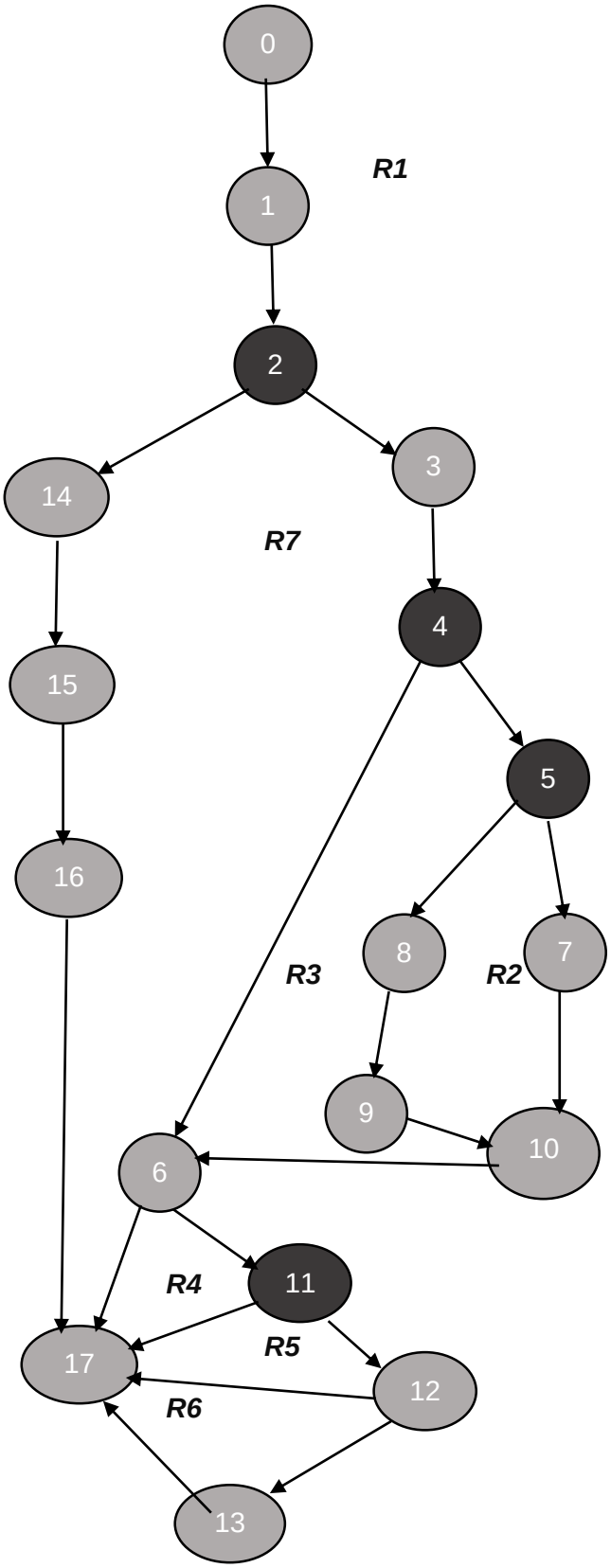
Tabla 8: Prueba unitaria Autenticar usuario

Método:

```
public void authentication() {//0
        int iduser = 0;
        int idrol = 3;//1
        try {//2
            AlUsuario User = logguer.login(usuario,
contra);//3
            if (User != null) {//4
                if
(!getFacade().ExistePersona(User.getUsuario
Usuario())) {//5
                    getFacade().create(User);
                    HttpSession session =
SessionUtils.getSession();
                    session.setAttribute("username",
User.getUsuarioUsuario());//7
                } else {//8
                    HttpSession session =
SessionUtils.getSession();
                    session.setAttribute("username",
User.getUsuarioUsuario());//9
                }
                iduser =
getFacade().getIdPersona(User.getUsuarioUs
uario());
                idrol =
getFacade().ObtenerRol(iduser);//10
            }
            if (idrol == 1) {//6
                    //pa la vista de admins
            } else if (idrol == 2) {//11
                //pa la vista de planif
            } else if (idrol == 3) {//12
                //pa la vista de usuario
                SessionUtils.redirectBrowser("faces/
welcomePrimefaces.xhtml");//13
            }
        } catch (Exception e) {//14
            e.printStackTrace();//15
        }

    }//16
```

Grafo resultante:



}//17	
Complejidad Ciclomática: V (G)= # de regiones = 7 V (G)=A-N+2= 23-18+2= 7 V (G)= P+1=6+1= 7	

Los cálculos realizados mediante las tres fórmulas anteriores dan el mismo resultado V(G) = 7, lo que revela que existen siete posibles caminos por donde el flujo puede circular y determina el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Se representan a continuación los caminos básicos:

- **Ruta 1**: **{**0, 1, 2, 3, 4, 6, 17**}**
- **Ruta 2**: **{**0, 1, 2, 3, 4, 5, 7, 10, 6, 17**}**
- **Ruta 3**: **{**0, 1, 2, 3, 4, 5, 8, 9, 10, 6, 17**}**
- **Ruta 4**: **{**0, 1, 2, 3, 4, 5, 8, 9, 10, 6, 11, 17**}**
- **Ruta 5**: **{**0, 1, 2, 3, 4, 5, 8, 9, 10, 6, 11, 12, 17**}**
- **Ruta 6**: **{**0, 1, 2, 3, 4, 5, 7, 10, 6, 11, 12, 13, 17**}**
- **Ruta 7**: **{**0, 1, 2, 14, 15, 16, 17**}**

A continuación, se plantean los casos de prueba que resguardan los caminos independientes presentados.

No	Entrada	Resultado
1	No se autentica ningún rol	El sistema no autentica ningún rol.
2	No existe usuario con el rol administrador.	El sistema crea el usuario con el rol de administrador.
3	Existe el usuario	El sistema crea la sección.
4	No existe el usuario y se crea el usuario con el rol de planificador.	El sistema crea el usuario con el rol de planificador.
5	No se autentica el usuario con rol de administrador ni planificador entonces va para la página index.	El sistema no autentica a los usuarios y entonces va para el index.

6	Existe el usuario con rol planificador.	El sistema le crea la sección al rol administrador.
7	No autentica ningún usuario.	El sistema lanza un error.

3.6 Resultados obtenidos

Fue necesario realizar 3 iteraciones de prueba para verificar el correcto funcionamiento de la solución.

En la primera iteración se detectaron 7 no conformidades (NC) asociadas a problemas funcionales, errores ortográficos y errores de diseño. Las NC detectadas fueron:

- No se almacenan ninguno de los datos del usuario que se autentica.
- No se adiciona un rol correctamente.
- No se listan los complejos comedores correctamente.
- No se listan los puntos de acceso correctamente.
- Errores ortográficos en los mensajes de notificación.
- No se muestra el logo de la interfaz de autenticación.
- No se cargaba el CSS de la plantilla administración.

En una segunda iteración se detectó 1 no conformidad:

- Se almacenan datos incompletos de los usuarios autenticados.

En una tercera iteración de pruebas no se detectaron nuevas no conformidades. La Figura 7 muestra la relación entre las nos conformidades detectadas y resuelta por cada prueba:

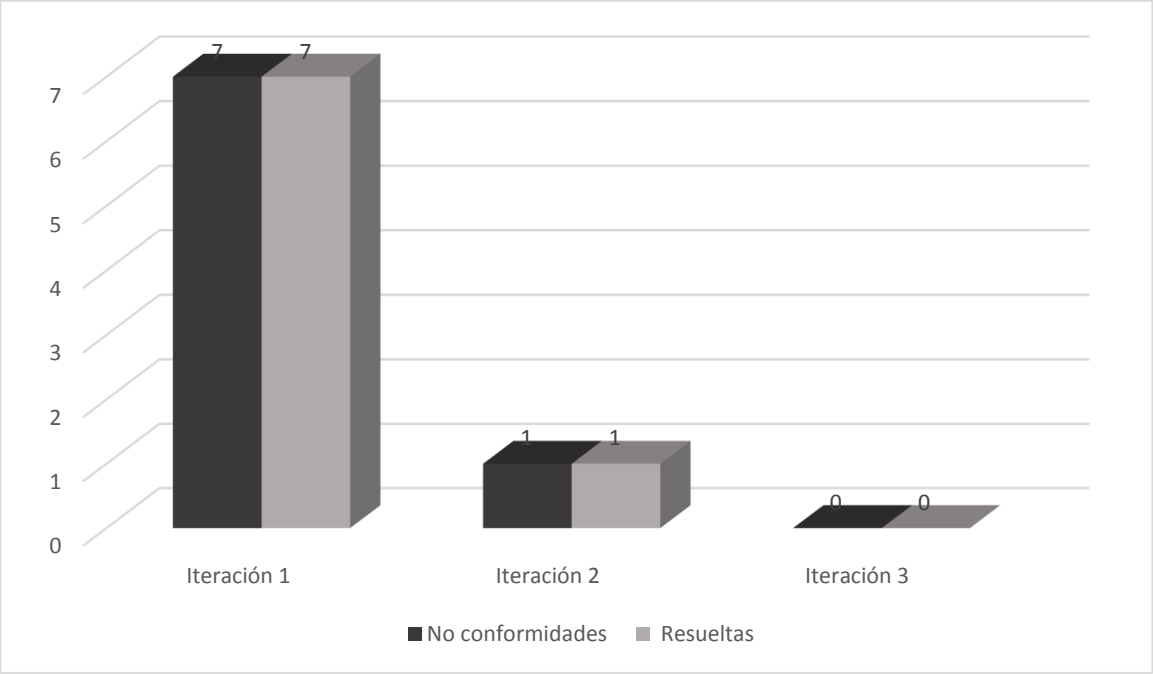


Figura 7: Representación de las iteraciones de las pruebas realizadas.

3.7 Consideraciones del capítulo

- En este capítulo se realizaron las tareas de ingeniería, se llevó a cabo la implementación del sistema y la descripción de la solución desarrollada.
- El diseño de los casos de prueba para los casos de uso identificados, posibilitó el desarrollo de las pruebas realizadas al sistema, lo que permitió brindarle una mejor solución a las no conformidades identificadas en las diferentes iteraciones.
- Se logró documentar las no conformidades mediante las pruebas realizadas y se corrigieron los errores de ejecución de los procesos no satisfactorios.

CONCLUSIONES GENERALES

- Con el desarrollo del presente trabajo se logró la implementación de un sistema informático para la distribución de comensales a los puntos de acceso de los comedores de la UCI, que contribuye a la informatización de dicho proceso.
- La utilización de las herramientas y tecnologías: PostgreSQL como gestor de base de datos, Java como lenguaje de programación y GlassFish como servidor de aplicaciones, permitieron la correcta migración a software libre del subsistema de distribución de comensales.
- Las pruebas de caja blanca y caja negra aplicadas al sistema, permitieron la detección de defectos y su corrección, para la obtención de un producto final con calidad.

RECOMENDACIONES

Los resultados alcanzados luego del desarrollo del presente trabajo satisfacen los requerimientos definidos. No obstante, para el desarrollo de futuras versiones se recomienda:

- Integrar el subsistema de distribución de comensales desarrollado, al sistema de gestión de servicio de alimentación de la UCI.
- Implantar el sistema informático desarrollado y constatar las mejoras que se obtienen en el proceso de distribución de comensales.
- Implementar el escenario para la configuración de la distribución según la ubicación geográfica de los comensales.

REFERENCIAS BIBLIOGRÁFICAS

1. **Martínes Péres, Raúl y Rodrígues Esponda, Eddy.** *Manual de metodología de la Investigación Científica.*
2. **¿Qué es un control de acceso?** [En línea] 26 de febrero de 2015. [Citado el: 19 de febrero de 2018.] Disponible en: http://sisca.co/que-es-un-control-de-acceso/.
3. **Dean, John S y Dean, Raymond H.** *Introducción a la programación con Java.* Ciudad de Mexico : McGRAW-HILL/INTERAMERICANA EDITORES, S.A. DE C.V., 2009. ISBN: 978-970-10-7278-3.
4. **Qué es Java. *Desarrollo web.*** [En línea] [Citado el: 24 de febrero de 2018.] Disponible en: https://www.desarrolloweb.com/articulos/497.php.
5. **Netbeans. (2018).** *Información Netbeans IDE 8.0*. Recuperado el 16 de marzo de 2018, de Netbeans.org: https://netbeans.org/community/releases/61/index_es.html
6. **Hebuterne, S**. (2014). *Guía de desarrollo de aplicación.* Barcelona: Ediciones ENI.
7. **Gibert Ginestá, Marc y Pérez Mora, Oscar.** *Base de datos en postgre.*
8. **Pgadmin PostgresSQL Tools. 2010.** Pgadmin PostgresSQL Tools. *Pgadmin PostgresSQL Tools.* [En línea] 2010. [Citado el: 5 de diciembre de 2017.] http://www.pgadmin.org/.
9. **Pressman, R. S**. *Ingeniería del Software, Un Enfoque Práctico.* 2005. ISBN 8448132149**.**
10. **Gómez Fuentes, Dra. María del Carmen y Cervantes Ojeda, Dr. Jorge.** *Introducción a la Programación Web con Java: JSP y Servlets, JavaServer Faces.* Ciudad de México : s.n., 2017. ISBN: 978-607-28-1069-3.
11. **Pech-May1, Fernando, y otros.** *Desarrollo de Aplicaciones web con JPA, EJB, JSF y PrimeFaces.* Tabasco : s.n.
12. **Torres Collaguazo, José Lizandro y Villagomez Cevallos, Jinson Oswaldo.** *Elaborar el manual para la configuración de un servidor Glassfish, utilizando el sistema operativo GNU/Linux.* Latacunga : s.n., 2009.
13. **Cordero L, Jorge Luis**. *Metodologías agiles, Proceso Unificado Ágil (AUP).* La Paz, El Alto- Bolivia : s.n.
14. ***SÁNCHEZ, T. R.*** *Metodología de desarrollo para la Actividad productiva de la UCI. 1.2 ed. Universidad de las Ciencias Informáticas, [Consultado el: 11/04/2018].*
15. **Advanced Software. 2011.** Módulo de Gestión y Control de Comedores. [En línea]. [Citado el: 13 de octubre de 2011.] http://www.advancedsoft.net/pdf/AdvComedor03.pdf.
16. **Saúco Peña, Aida María. 2010.** *Sistema de gestión de asignación de aguas disponibles.* [pdf] Holguín: Centro de Desarrollo de Software Holguín, 2010.

17. **André Ampuero, Margarita y Gulnara Baldoquí, María. 2010.** *Un sistema de soporte a la decisión para la asignación de recursos humanos a equipos de proyectos de software.* La Habana, Cuba: s.n., 2010. Vol. 31.

18. ***Sommerville, I.*** *Requerimientos. . septima ed. Madrid: Pearson Education S.A: 2005. vol. 7, 108 p. Ingeniería del Software ISBN 84-7829-074-5.*

19. ***Mike Cohn,*** *"User Stories Applied", 2004, Addison Wesley, ISBN 0-321-20568-5.*

20. **Weitzenfeld Ridel, Alfredo y Guardati Buemo, Silvia.** Ingeniería de software: el proceso para el desarrollo de software. 2008.

21. **Bonaparte, Ubaldo José.** *Proyectos UML. Diagramas de clases y aplicaciones Java en Netbeans.* Argentina : edUTecNe, 2012.

22. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires : s.n., 2004.

23. **Introducción a los patrones de diseño.** *Codecriticon.* [En línea] 11 de noviembre de 2014. [Citado el: 14 de febreo de 2018.] Disponible en: http://codecriticon.com/introduccion-patrones-diseno/.

24. **Patrones GRASP. Patrones GoF. Diferencia entre GRASP y GoF**. *TuxNots.* [En línea] 3 de diciembre de 2011. [Citado el: 8 de febrero de 2018.]

25. **Larman, Craig**. *UML y Patrones*. ^ eMadrid Madrid: Pearson Educación, 2003.

26. **Salazar C, Cristian.** Modelo de datos. Educación. [En línea] [Citado el: 20 de abril de 2018.] Disponible en: https://es.slideshare.net/csalazarc/modelo-de-datos-14506949.

27. **Gutierrez, Demián.** *UML Diagrama de Paquetes.* Venezuela : s.n., 2009.

28. **Ferré Grau, Xavier y Sánchez Segura, María Isabel.** *Desarrollo Orientado a Objetos con UML.* México: s.n., 2011.

29. **Marca Huallpara, Hugo Michael y Quisbert Limachi, Nancy Susana.** *Diagrama de Despliegue.*

30. **Sánchez Peño, José Manuel.** *Pruebas de Software. Fundamentos y Técnicas.* Madrid : s.n., 2015.

31. **Rodríguez Tello, Eduardo A.** *Estrategias y técnicas de prueba del software.* Tamaulipas : s.n., 2012.

ANEXOS

Anexo 1 Historia de usuario "Autenticar usuario"

Historias de usuario	
Número: HU 3	**Nombre del requisito**: Autenticar usuario
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Media	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir entra su usuario y su contraseña si los datos entrados son correctos se le mostrará la página inicial del sistema.	
Observaciones: NA	
Prototipo de interfaz: 	

Anexo 2 Historia de usuario "Gestionar rol"

Historias de usuario	
Número: HU 4	**Nombre del requisito**: Gestionar rol
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción "Roles", se muestra un listado con todos los usuarios en la base de datos con roles especificados, además	

muestra un enlace que permite agregar usuarios desde el servidor de direcciones LDAP de la UCI, para incluir este como un nuevo usuario con roles en el sistema.
Observaciones: Será asignado el rol al usuario adicionado.
Prototipo de interfaz: 

Anexo 3 Historia de usuario "Gestionar estructuras complejo-comedor"

Historias de usuario	
Número: HU 5	**Nombre del requisito**: Gestionar estructura complejo-comedor
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema permitirá que al hacer clic en la opción "Gestionar estructura complejo-comedor", se muestra un formulario en el que se introducirán los datos necesarios para la realización de una distribución. Se mostrarán las opciones "Guardar" y "Cancelar".	
Observaciones: NA	
Prototipo de interfaz:	

Anexo 4 Historia de usuario "Gestionar usuarios"

Historias de usuario	
Número: HU 6	**Nombre del requisito**: Gestionar usuarios
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción "Gestionar usuarios", se muestre un listado con los usuarios existentes. Para cada uno de los usuarios mostrados se habilitarán las opciones "Adicionar", "Modificar" y "Eliminar", además de la opción "Guardar" y "Cancelar".	
Observaciones: NA	

Prototipo de interfaz:

Anexo 5 Historia de usuario "Gestionar estructuras puntos de acceso"

Historias de usuario	
Número: HU 7	**Nombre del requisito**: Gestionar estructuras puntos de acceso
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema permitirá que al hacer clic en la opción "Gestionar estructuras puntos de acceso", se muestre un formulario con los datos necesarios para la Adicionar, Modificar y Eliminar puntos de acceso, además de mostrar las opciones "Guardar" y "Cancelar".	
Observaciones: NA	
Prototipo de interfaz:	

Anexo 6 Historia de usuario "Gestionar distribución de comensales"

Historias de usuario	
Número: HU 8	**Nombre del requisito**: Gestionar distribución de comensales
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción "Gestionar distribución de comensales", se muestre un listado con las distribuciones realizadas, además de Crear, Modificar, Eliminar, Listar y Aplicar dicha distribuciones. Además de las opciones "Guardar" y "Cancelar".	
Observaciones: NA	
Prototipo de interfaz:	

Anexo 7 Historia de usuario "Asignar grupos de comensales a cada punto de acceso"

Historias de usuario	
Número: HU 9	**Nombre del requisito**: Asignar grupos de comensales a cada punto de acceso
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema debe permitir que al hacer clic en la opción "Asignar grupos de comensales", se muestra un listado con todos los usuarios existentes, además muestra un enlace que permite agregar usuarios a un grupo, y ese grupo asignarlo a un punto de acceso determinado.	
Observaciones: NA	
Prototipo de interfaz:	

Anexo 8 Historia de usuario "Crear grupos de comensales"

Historias de usuario	
Número: HU 10	**Nombre del requisito**: Gestionar estructura pantry
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Alta	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas
Descripción: El subsistema permitirá que al hacer clic en la opción "Gestionar estructura complejo-comedor", se muestra un formulario en el que se introducirán los datos necesarios para la realización de una distribución. Se mostrarán las opciones "Guardar" y "Cancelar".	
Observaciones: NA	
Prototipo de interfaz: 	

Anexo 10 Historia de usuario "Mostrar reporte de la distribución de comensales"

Historias de usuario	
Número: HU 11	**Nombre del requisito**: Mostrar reporte de la distribución de comensales
Programador: Dalquerine Castillo Castillo	**Iteración Asignada**: 1
Prioridad: Media	**Tiempo Estimado**: 10 días
Riesgo en Desarrollo: Alto	**Tiempo Real**: 100 horas

Descripción: El subsistema debe permitir después de haber realizado la distribución, mostrar un reporte con todos los datos de dicha distribución, además de las opciones de “Guardar” y “Cancelar”.
Observaciones: NA
Prototipo de interfaz: