UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

ARQUITECTURA JAVA EE PARA LA ADMINISTRACIÓN DE UN RESTAURANTE

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMPUTACIÓN

PRESENTAN

GALICIA BADILLO NÉSTOR ENRIQUE
LÓPEZ ORDÓÑEZ TOMÁS EDUARDO
MARTELL AVILA REYNALDO

DIRECTOR DE TESIS:
FIS. ADAN ZEPEDA GOROSTIZA

CIUDAD UNIVERSITARIA, D.F. FEBRERO 2013

Agradecimientos

A nuestra familia, pilar de nuestra preparación e impulso hacia el futuro. Ustedes son la razón de todo este esfuerzo y esperamos que con nuestras acciones merezcamos ser depositarios de su orgullo y merecedores de su inquebrantable confianza.

A la UNAM, Facultad de Ingeniería y toda su comunidad universitaria que fueron capaces de proporcionarnos el ambiente germinal para imbuir en nosotros el espíritu profesional que todo ingeniero debe presentar durante su desempeño profesional.

A aquellos amigos y compañeros que compartieron las aulas con nosotros, pues junto a ellos, y a través de ellos, pudimos empaparnos de experiencias enriquecedoras en esta etapa de nuestra vida.

Al Físico Adán Zepeda, quien confiando en nosotros aceptó ser el director de este trabajo y se encargó de encaminarlo a buen puerto.

Al Ingeniero Amaury Perea, quien fue un apoyo fundamental en la concepción y realización de este trabajo.

A todos ustedes les extendemos un agradecimiento eterno esperando corresponderles al comprometernos a desempeñar una vida profesional acorde a los valores éticos y morales que con tanto esfuerzo inculcaron en nuestras personas.

Dedicatoria

Especial dedicatoria para mi padre

Para mi madre y hermanos

Para mis amigos

Para Marisol

Néstor Enrique Galicia Badillo.

Para mis padres y hermano que tanto me han dado.

Para mi tía que tanto me dio.

Tomás E. López Ordóñez.

Para mi madre por darme la vida y dedicarnos la suya sin recibir nada a cambio.

Para mi padre por ser un hombre bueno que siempre supo enseñarme las obligaciones de un padre, de un amigo y de un hombre.

Para mis hermanas por todo su apoyo y consejos.

Para mis sobrinos y primos.

Para Stephanie por tu fortaleza, por tus consejos y por todo los momentos felices.

Para mi familia especialmente a mi tío Hugo por ser el mejor ejemplo.

Para mis amigos por todos sus consejos.

Reynaldo Martell Avila.

Índice

1. Introducción.

Hoy en día las comunicaciones electrónicas se encuentran jugando un importante papel en las interacciones sociales y a medida que se extiende su uso se puede apreciar como su influencia ha ido incrementando hasta el punto de trastocar incluso la forma en que éstas se realizan, un ejemplo sencillo puede ser el de la evolución de las compras por internet ya que de tener una interacción eminentemente física entre un proveedor y un comprador se ha alcanzado un punto en donde ya no es necesario una interacción presencial sino que es posible realizar y pagar un pedido desde ubicaciones remotas sin que estas dos entidades hayan tenido contacto físico alguno y no sólo eso también es posible mantener simultáneamente otras interacciones comerciales con clientes totalmente distintos. Desgraciadamente lo que se gana en ubicuidad desemboca en un aumento en la complejidad de la interacción debido a que ahora se tienen que implementar mecanismos que permitan garantizar la autenticidad de cada una de las entidades participantes, así como la gestión de la interacción misma haciendo inminentemente necesario que todos los aspectos de la interacción funcionen en una forma coordinada, exacta, pero sobre todo, confiable ya que de no lograr alguno de estos aspectos la interacción se podría considerar corrupta y por lo tanto su fin no se llevaría a cabo de forma correcta.

Como podrá observarse esta dinámica evolutiva de las relaciones sociales naturalmente terminará por extenderse a las transacciones comerciales, ya que en sí mismas son interacciones sociales, y con ello se abrirá la puerta a una ingente cantidad de novedosos servicios tecnológicos que puedan ser capaces de solventar los diversos problemas, tanto de comunicación y transporte como de administración y seguridad, que ineludiblemente traerán consigo los nuevos esquemas de transacciones comerciales. Así mismo se hace

patente que una gran área de explotación comercial es la creación de servicios que sean capaces de sustituir viejos paradigmas de transacciones comerciales por métodos innovadores que permitan desarrollar un proceso eficiente, controlado y repetible, capaz de replicar esas transacciones comerciales mediante algún tipo de comunicación electrónica.

En este trabajo se intentará llevar a cabo esto último proponiendo un nuevo modelo para la gestión de las órdenes de los comensales de un restaurante basado en una arquitectura web e implementado bajo marcos de trabajos desarrollados en tecnologías Java EE.

2. Objetivo.

El trabajo presente pretende implementar un sistema web para la administración de comensales en un restaurante que permita automatizar el proceso de la toma y gestión de pedidos a través de una arquitectura basada en tecnologías Java EE que agilizan el desarrollo de aplicaciones robustas y eficientes.

3. Justificación.

Entre los distintos servicios comerciales susceptibles a ser automatizados destaca el que brindan los restaurantes ya que es un modelo de servicio bastante enraizado en la cultura humana y es tan típico que se puede encontrar alrededor de prácticamente todo el globo sin presentar apenas variaciones. Por esta razón es que consideramos a este servicio comercial como un arquetipo de muchos otros que por sus distintas características involucrarían procesos más complejos y por ende un candidato idóneo para presentar nuestra propuesta de automatización de servicios comerciales que respondan a características comunes a la administración de un restaurante.

3.1. Modelo tradicional.

Tradicionalmente la forma en que un restaurante ofrecería su servicios a un cliente sería la siguiente: un administrador se encargaría de ubicar al comensal en alguna de las mesas que el restaurante tuviera disponible y le proporcionaría una carta para que fuera capaz de emitir su orden, posteriormente mediante la intermediación de un mesero se tomaría la orden del comensal y, por medio del mismo mesero se transportaría la orden a la cocina donde sería procesada. Una vez procesada se hace necesaria una nueva intermediación del

mesero para llevar el pedido al comensal para su degustación. Por último, para efectuar el pago, la intervención del mesero es nuevamente requerida ya que en él cae la responsabilidad de notificarle la cuenta al cliente y, en algunas ocasiones, hasta de gestionar el cobro.

Como puede observarse este modelo descansa fuertemente sobre los recursos humanos que son los meseros e incluso demanda, de éstos, un fuerte compromiso de concentración y eficiencia con el fin de mantener al cliente satisfecho en todos los aspectos. Este modelo funciona estupendamente bien en ambientes pequeños donde la cantidad de meseros se relaciona directamente con la cantidad de clientes de tal forma que un mesero no tenga que distribuir su atención entre demasiados clientes. Lamentablemente es muy fácil suponer que tal situación puede no presentarse con regularidad dejando al restaurante en un estado altamente propenso a la comisión de errores como el no tomar la orden completa, tardar demasiado tiempo en tomar una orden, mala gestión del preparado de los alimentos, etc. Errores que al repetirse constantemente terminarían por afectar la satisfacción del cliente y por ende a los intereses mismos del negocio.

3.2. Modelo propuesto.

En esta propuesta se pretende aprovechar el auge de las comunicaciones electrónicas asumiendo que hoy en día una vasta cantidad de clientes potenciales cuenta entre sus pertenencias con un teléfono celular inteligente capaz de conectarse a una red inalámbrica por medio de la especificación Wi-Fi. Con esto en mente, se propone que algunas de las obligaciones del mesero sean sustituidas por medio de un sistema capaz de desplegar el menú de alimentos en el teléfono celular de cada comensal, tomar su orden y transferirla a

la cocina para su pronta atención, todo esto vía web. Con esto el mesero quedaría únicamente con las obligaciones del transporte de alimentos y a la atención de los clientes sin distraerse por tomar las órdenes de éstos.

Este modelo además representa un ahorro considerable para el restaurante pues elimina la necesidad de imprimir grandes cantidades de menús y con ello los costos de conservación y almacenamiento de los mismos. Mención aparte merece el hecho de que la implementación del sistema es por demás económica dado que la mayor parte de la infraestructura requerida descansa en el lado del cliente, pues es su teléfono celular el que se encarga de la visualización del menú y de configurar su orden.

3.3. Arquitectura propuesta.

El diseño, programación e implantación de un sistema como el propuesto involucra un alto grado de complejidad pues durante su funcionamiento se involucrarán factores críticos para una aplicación distribuida como son:

- Concurrencia de las transacciones que se hacen con la base de datos.
- Gran carga de usuarios en el servidor.
- Adecuado manejo de las conexiones con las base de datos.
- Alto consumo de recursos.
- Baja portabilidad del sistema.
- Baja seguridad.
- Desarrollo complicado y difícil de mantener.

Para remediar estos factores se pretende desarrollar el sistema bajo una arquitectura web construida a partir de tecnologías libres que se encarguen de administrar los recursos adecuadamente permitiéndonos abolir la mayoría de los problemas enumerados. En concreto se propone una arquitectura elaborada a partir de *frameworks* libres de la siguiente forma: Hibernate 3 para poder administrar las sesiones con la base de datos y su integración a la aplicación mediante objetos persistentes, Spring 3 para gestionar los servicios del negocio y Struts 2 para resolver la comunicación entre la vista y el controlador en un patrón de arquitectura MVC. Como podrá observarse a lo largo de este desarrollo la colaboración entre estas tecnologías proporcionará una arquitectura de desarrollo lo suficientemente robusta como para realizar una programación ordenada, coherente y extremadamente eficiente de una manera ágil y rápida.

4. Alcances.

Dado que los *frameworks* empleados poseen una funcionalidad vasta en distintas áreas del desarrollo de una aplicación, el presente trabajo está limitado a la explotación de las características que en su conjunto permiten la obtención de una arquitectura robusta, cuidando aspectos como la funcionalidad y seguridad, dejando de lado aquellas que proveen operatividad secundaria.

Como una consecuencia de la interacción entre las tecnologías empleadas se obtiene una arquitectura portable que permite la construcción de aplicaciones web totalmente independiente al servidor de aplicaciones.

Como ejemplificación de la funcionalidad del conjunto de las tecnologías empleadas se muestra una aplicación con la capacidad de gestionar las órdenes de los comensales de un restaurante. Este sistema sólo pretende facilitar un medio para visualizar el menú de platillos vía web, con base en él emitir un pedido y cuando lo desee el comensal pedir su cuenta, así mismo se facilita una interfaz gráfica que permite al administrador del restaurante gestionar los lugares disponibles dentro del restaurante y la administración del menú que se le mostrará al comensal. En el trabajo presente no se tiene contemplado ningún tipo de pago electrónico ni de procesamiento de la orden del comensal en la cocina del restaurante.

5. Marco teórico.

Antes de comenzar el desarrollo de un sistema informático se requiere determinar la forma en que será programado. Al desarrollar un sistema es común encontrarse con una multitud de problemas derivados de una deficiente administración del código escrito así como de una falta de estandarización en los métodos de programación empleados. Para minimizar estos inconvenientes es práctica común definir un marco de trabajo que permita el desarrollo en forma sistemática, ordenada, modular y eficiente, al mismo tiempo de que garantice la seguridad del sistema.

Con la intención de agilizar el proceso de definición de un marco de trabajo óptimo para el desarrollo del sistema propuesto se ha optado por hacer uso de varias tecnologías de código abierto disponibles en el mercado e integrarlas en una única arquitectura que brinde las características previamente mencionadas. En el apartado presente se desgranarán las diversas tecnologías usadas para construir la arquitectura en la que se soporta el desarrollo del sistema de gestión de un restaurante vía web propuesto.

5.1. Wi-Fi.

Para garantizar la compatibilidad entre dispositivos inalámbricos se creó en 1999 la WECA (Wireless Ethernet Compatibility Alliance) quien posteriormente, al desarrollar la primera certificación de interoperabilidad entre equipos según la norma IEEE 802.11b, cambiaría su nombre a Wi-Fi Alliance. Dicha certificación fue creada bajo la marca Wi-Fi y por un uso indebido de los términos el nombre del estándar se ha confundido con el nombre de la certificación.

A los dispositivos certificados por la Wi-Fi Alliance se les permite usar el logotipo:

Imagen 5.1 Logotipo Wi-Fi

Originalmente la certificación Wi-Fi sólo se refería a los productos capaces de funcionar bajo el estándar 802.11b pero actualmente el término Wi-Fi se aplica a todo aquel producto que cumpla con el estándar 802.11 del IEEE el cual regula a las redes inalámbricas de área local. Este estándar ha ido evolucionando a través del tiempo dando lugar a lo que ahora se conoce como familia 802.11 dentro de la cual destacan tres especificaciones: la 802.11a, la 802.11b y la 802.11g.

El estándar 802.11a tiene un flujo de datos de 54 Mbps y un rango de 30 metros aproximadamente. Se basa en tecnología OFDM (multiplexación por división de frecuencias ortogonales), transmite en un rango de frecuencia de 5 GHz y utiliza 8 canales no superpuestos.

El estándar 802.11b tiene una transferencia máxima de datos de 11 Mbps en un rango de 100 metros aproximadamente en ambientes cerrados y de más de 200 metros al aire libre.

El estándar 802.11g permite un máximo de transferencia de datos de 54 Mbps en rangos comparables a los del 802.11b.

Es importante mencionar que los dispositivos fabricados con la especificación 802.11a son incompatibles con los dispositivos 802.11b y 802.11g. Sin embargo, existen artefactos que incorporan ambos chips llamados “de banda dual”. En cambio, el estándar 802.11g es completamente compatible con el 802.11b debido a que utiliza el rango de frecuencia de 2.4 GHz con codificación OFDM,

Las diferencias entre estos estándares se pueden apreciar mejor en la tabla siguiente:

Estándar	Frecuencia	Velocidad	Rango
Wi-Fi A (802.11a)	5 GHz	54 Mbit/s	10 m
Wi-Fi B (802.11b)	2,4 GHz	11 Mbit/s	100 m
Wi-Fi G (802.11g)	2,4 GHz	54 Mbit/s	100 m

Tabla 5.1 Estándares Wi-Fi más comunes.

Un aspecto importante a considerar en las redes inalámbricas estructuradas a través de tecnología Wi-Fi es la seguridad. En general se utiliza un mecanismo de encriptación y autenticación especificado en el estándar IEEE 802.11 llamado WEP (Wired Equivalent Privacy) que utiliza un esquema de cifrado simétrico en el que la misma clave y algoritmo se utilizan tanto para el cifrado de los datos como para su descifrado.

Aunque WEP es actualmente el mecanismo más extendido en la implementación de las redes Wi-Fi es cierto, también, que presenta algunas vulnerabilidades de seguridad por lo que, con el fin de resolverlas, se creó la certificación WPA (Wi-Fi Protected Access), la cual, al incrementar el tamaño de las claves, el número de llaves en uso, y al agregar un sistema de verificación de mensajes hace que la entrada no autorizada a redes inalámbricas sea mucho más difícil.

En la práctica se puede decir que Wi-Fi admite cualquier tipo de dispositivo de alta velocidad dentro de un radio de varias decenas de metros en ambientes cerrados o dentro de un radio de cientos de metros al aire libre.

5.2. Java EE.

Java fue desarrollado en 1995 por James Gosling cuando trabajaba para la empresa Sun Microsystems. La denominación Java la reciben tanto el lenguaje de programación como la plataforma de desarrollo. Ésta se compone a su vez de una máquina virtual y una interfaz

de programación de aplicaciones (API). Como lenguaje de programación se puede decir que es un lenguaje de alto nivel orientado a objetos.

Uno de los objetivos principales de Java es que las aplicaciones creadas con él sean independientes del *hardware* en el que se ejecutan, para lograrlo, al compilar un programa en Java en vez de generar código máquina se genera un tipo de código intermedio, conocido como *bytecode,* que no necesita conocer las características de la máquina en donde se usará. Para ejecutar el *bytecode* se requiere la máquina virtual de java (JVM) quien se encarga de interpretarlo y generar el código máquina del procesador en el que se esté ejecutando la aplicación; esto implica que para cada entorno debemos contar con una JVM diferente. Una vez instalada la JVM las aplicaciones se pueden ejecutar sin tener que modificar ni recompilar el código lo cual hace de Java un lenguaje muy apropiado para aplicaciones corporativas y de Internet pues permite ejecutar la misma aplicación en diferentes entornos sin cambio alguno.

Java se distribuye en tres plataformas:

- **Java SE (Java Standart Edition).** Versión estándar de la plataforma, es decir, la que se usa para desarrollar aplicaciones de escritorio. Es la plataforma base para Java EE y Java ME.
- **Java EE (Java Enterprise Edition):** Anteriormente conocida como J2EE (Java 2 Platform Enterprise Edition), es una versión de Java usada para la creación de aplicaciones grandes de cliente/servidor y para el desarrollo de *WebServices*. Para poder usar Java EE se requiere tener instalado previamente Java SE.

- **Java ME (Java Micro Edition).** Plataforma de desarrollo de aplicaciones para dispositivos con recursos limitados, como pueden ser los dispositivos móviles. Existen 2 versiones de esta plataforma: la *Connected Limited Device Configuration* (CLDC) y la *Connected Device Configuration* (CDC). La CLDC es mucho más limitada que la CLC.

Java EE es, como ya se mencionó, la plataforma Java para el desarrollo de aplicaciones empresariales. El modelo que propone sirve para desarrollar aplicaciones multinivel facilitando su escalabilidad, accesibilidad y manejabilidad. En concreto propone dividir la aplicación en dos partes fundamentales: la parte referente al negocio y lógica de presentación que debe ser asumida por el desarrollador y la parte que hace hincapié en los servicios estándar de las aplicaciones multinivel que será provista por la plataforma Java EE.

La plataforma Java EE usa un modelo multinivel que permite dividir la lógica de negocio en diversos componentes cada uno de los cuales se especializa en una función específica e incluso son capaces de funcionar en máquinas distintas dependiendo del nivel en que se desempeñen y según lo especificado por el ambiente multinivel que se esté desarrollando. En un buen diseño de un sistema multinivel, cada nivel superior sólo debe depender de uno inferior, pero no al contrario. Por ejemplo una división muy común de un ambiente multinivel es la que se muestra a continuación:

- **Nivel EIS**. El nivel EIS o también conocido como nivel de integración, es donde se gestiona la información empresarial de los sistemas, incluyendo sistema de base de datos (DBMS). Los recursos del nivel EIS suelen ser transaccionales. El nivel EIS

se encuentran fuera del control del servidor de aplicaciones, sin embargo el servidor es capaz de manejar las transacciones y agrupación de conexiones de manera estándar.[1] Java EE posee potentes capacidades para interactuar con los recursos del nivel de integración, como el API de JDBC para el acceso a base de datos relacionales, JNDI para el acceso a servicios de directorio y JCA que permite la conectividad con otro nivel EIS de otro sistema.

- **Nivel de Negocio**. Este nivel contiene los objetos de la lógica de negocio de las aplicaciones y tiene la responsabilidad de interactuar con los recursos del nivel EIS. El nivel de negocio se beneficia de muchos servicios de los contenedores Java EE como administración de transacciones y agrupación de conexiones.
- **Nivel de presentación**. Este nivel agrupa los componentes de visualización que hacen posible la comunicación del sistema con el usuario final. En una aplicación web Java EE el nivel se compondría de *servlets*, clases *helpers* utilizadas por los *servlets* y componentes de visualización como los JSPs y JSFs.

En la imagen 5.2 se puede apreciar mejor la integración de estos niveles en un desarrollo web bajo la una plataforma Java EE.

Una aplicación Java EE está creada mediante componentes. Los componentes Java EE son piezas funcionales de software capaces de ser ensambladas en una aplicación Java EE y que a su vez pueden comunicarse con otros componentes. La diferencia entre una clase Java común y un componente Java EE reside en que éste último se encuentra ensamblado en una aplicación Java EE y ha sido verificado que cumpla con la especificación de Java EE de tal

[1] Rod Johonson, Expert one-one-one J2EE Design and Development

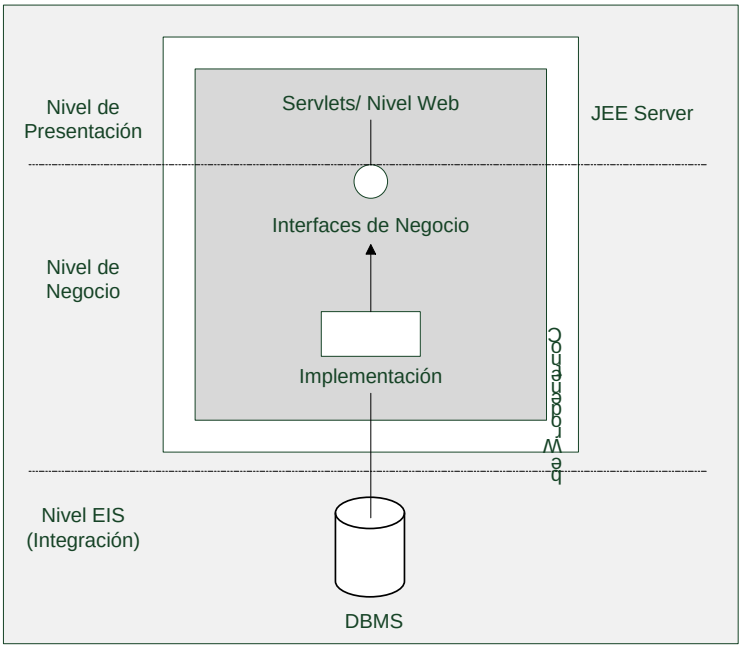


Imagen 5.2 Niveles de un desarrollo en java EE.

forma que puedan ser manejados por un servidor Java EE. La especificación Java EE define los componentes siguientes:

- *Applets* y aplicaciones cliente son componentes que corren en la máquina cliente.
- Tecnologías como son: *Java Servlet, JavaServer Faces*, y *JavaServer Pages* (JSP) son componentes Web que corren en el servidor.
- *Enterprise JavaBeans* (EJB) son componentes de negocio que corren en el servidor.

Como se mencionó anteriormente la plataforma Java EE se encarga de proveer un conjunto de servicios subyacentes comunes para todos los componentes de un tipo en específico. Esto lo hace mediante los llamados contenedores. Los contenedores son la interfaz entre un componente y la funcionalidad de bajo nivel que soporta al componente. Por lo tanto, antes

de poder ser ejecutado un componente debe ser ensamblado en un módulo Java EE y desplegado en su contenedor. Algunos de los servicios que provee Java EE son configurables por lo que los componentes de una aplicación Java EE pueden tener diferente comportamiento basados en dónde son desplegados, por ejemplo, un EJB puede tener una configuración de seguridad que le permita tener cierto nivel de acceso a la información de la base de datos en un entorno de producción y otro nivel de acceso a la base de datos en otro entorno de producción. El proceso de despliegue *(deploy*) instala los componentes de la aplicación Java EE en los contenedores Java EE.

Los principales contenedores Java EE son:

- **Servidor de Aplicaciones Java EE**. Entorno de ejecución de aplicaciones Java EE. Un servidor de aplicaciones Java EE se compone del contenedor Web y el contenedor de EJBs. Cabe mencionar que se pueden desarrollar aplicaciones que no hagan uso del contenedor de EJBs.
- **Contenedor de EJB**. Maneja la ejecución de los EJB. Ambos el contenedor como los *beans* corren en el servidor.
- **Contenedor Web**. Controla la ejecución de las páginas web, *servlets*, etc. Los componentes web y su contendor corren en el servidor.

Al finalizar el desarrollo de una aplicación Java EE se debe empaquetar en una o más unidades para su posterior desplegado en cualquier plataforma que cumpla con las especificaciones Java EE. Cada una de estas unidades debe contener los componentes

funcionales y un descriptor de despliegue opcional. Una vez desplegado en la plataforma local la aplicación esta lista para su uso.

5.3. Arquitecturas por capas.

La separación por capas de un sistema es una técnica adoptada por los diseñadores de *software* para dividir complicados sistemas de *software*. Consiste en imaginar los subsistemas principales del *software* en forma de pastel, donde cada una de las capas depende de una capa inferior. En este esquema las capas superiores usan varios servicios definidos por una capa inferior, pero una capa inferior no puede depender de una capa superior, además cada capa oculta sus capas inferiores de las capas superiores. Por ejemplo la capa 4 utiliza servicios de la capa 3 la cual utiliza servicios de la capa 2, pero la capa 4 no puede acceder a los servicios de la capa 2. Es muy importante distinguir los conceptos de Capas y Niveles debido a que es muy común confundir estos dos conceptos y aplicarlos de forma errónea. En el punto anterior se definió que los niveles se encargan de hacer una distribución física de los componentes, en cambio las capas realizan una división lógica sin tomar en cuenta su separación real.

Un sistema dividido en capas tiene los siguientes beneficios.

- Se puede tener conocimiento de una sola capa sin saber mucho de las otras capas.
- Se puede sustituir una capa por una implementación alternativa que cuente con servicios similares.
- Las capas permiten una buena estandarización.

- Una vez que se tiene una capa construida se pueden utilizar sus servicios para las capas superiores.

La desventaja de la utilización de capas es que, debido a que las capas no encapsulan todas las funcionalidades, a veces se tienen que realizar los cambios en cascada. Por ejemplo, cuando es necesario mostrar un campo en la interfaz de usuario éste tiene que ser consultado en la base de datos y debe ser propagado por cada una de las capas superiores hasta llegar a la de presentación.

Existen infinidad de formas de dividir un sistema en capas, pero la industria del desarrollo de *software* ha definido un patrón llamado arquitectura por capas, en el cual se muestra que, por lo menos, se debe contar con una capa de presentación, una capa de aplicación, una capa de negocio y una capa de persistencia. Los nombres de las capas pueden variar en las diferentes arquitecturas, alguna capa puede estar o no estar, o simplemente se puede dividir en dos o más capas, pero lo importante es que nunca debe faltar la capa de negocio. A continuación se explica cada una de estas capas principales.

- **Capa de Presentación**. Esta capa es la encargada de proporcionar los objetos de la lógica de negocio a los clientes (clientes HTML, *Applets* y otros clientes web). La capa de presentación recibe la petición del cliente, gestiona el inicio de sesiones, controla el acceso a los servicios de negocio, construye la respuesta y la entrega al cliente.
- **Capa de Aplicación**. La capa de aplicación es una capa delegada que administra las actividades de la aplicación; es importante que esta capa no contenga lógica de

negocio. Los servicios que se encuentran en esta capa son los encargados de coordinar los servicios que se proveen en la capa de negocio.

- **Capa de Negocio**. Esta capa proporciona los servicios de negocio requeridos por los clientes de la aplicación ignorando la responsabilidad de la persistencia de los datos. La capa contiene los datos de la lógica de negocio y es la encargada de interactuar con los servicios de la capa de persistencia.
- **Capa de Persistencia**. Esta capa es nombrada capa de persistencia debido a que se encuentra ligada a ciertas tecnologías de acceso a datos. Los componentes de persistencia proporcionan acceso a los recursos externos de base de datos para recuperar información y suministrarla a las capas superiores.

5.4. Patrones de software.

Durante un desarrollo los programadores constantemente se encuentran con problemas similares tanto en el diseño como en la implementación. Algunos de estos problemas son sutiles y producen efectos en cascada ocasionando errores que pueden terminar en el bloqueo del sistema.

Un patrón de software proporciona una solución a la aparición constante de problemas semejantes mediante una plantilla que puede ser reutilizada en distintas situaciones. No es un código completo que puede agregarse en el cuerpo del desarrollo como tal, son un conjunto de reglas que solucionan un problema reiterativo. Cada patrón consta de tres partes: el contexto, el problema repetitivo y una solución.

5.4.1. Patrones de diseño.

Durante el año de 1970, Christopher Alexander escribió algunos libros de patrones aplicados a Ingeniería civil y arquitectura, posteriormente la comunidad de desarrolladores de *software* adoptaría la idea de aplicar patrones documentados que auxiliaran en el desarrollo de *software*. En 1994 se publicó el libro *"Design Patterns: Elements of Reusable Object Oriented Sofware"* que fue escrito por un grupo denominado *Gang of Four* (GoF, en español la pandilla de los cuatro) conformado por 4 pioneros expertos diseñadores de *software* orientado a objetos (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides), en él se documentaron 23 patrones de diseño que normalmente los arquitectos y diseñadores aplican, cabe mencionar que ellos no fueron los primeros en aplicar dichos patrones, ni los únicos, sin embargo a partir de dicha publicación se difundió a gran media el uso de patrones de diseño.

El término de patrón de diseño ha sido intentado definir por varios arquitectos y diseñadores de *software* y por ende se cuenta con una variada multitud de definiciones, cada una respondiendo a un enfoque particular y específico. No obstante todas ellas coinciden en que por lo general se puede definir a los patrones de diseño como soluciones para problemas que suceden una y otra vez en nuestro entorno, de tal forma que éstas puedan ser reutilizadas sin tener que hacer nunca lo mismo dos veces.

Habitualmente un patrón de diseño cuenta con las siguientes características:

- **Nombre del Patrón**. El nombre del patrón debe ser usado para describir el problema del diseño, su solución y consecuencias en una o dos palabras. Nombrar un patrón de diseño incrementa inmediatamente el vocabulario del diseño. Esto nos permite contar con diseños de un alto nivel de abstracción y tener un vocabulario de

patrones que proporciona un lenguaje entre colaboradores así como en la documentación.

- **El Problema**. Describe cuándo aplicar un patrón así como su contexto. Explica específicamente los problemas del diseño tales como la forma de representar algoritmos como objetos. Algunas veces en el problema se incluirá una lista de condiciones que se deben cumplir antes de aplicar el patrón.
- **La solución.** Describe los elementos que integran el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño o implementación en particular debido a que un patrón es una plantilla que puede ser utilizada en varias ocasiones diferentes, en cambio proporciona una descripción abstracta referente a un problema de diseño que une ciertos elementos que pueden resolverlo.
- **Las consecuencias**. Se dan a conocer los resultados de la posible aplicación. El uso del patrón es determinado a partir de sus ventajas y desventajas incluyendo su impacto en la flexibilidad, portabilidad y reutilización del sistema. [2]

5.4.1.1. Patrones creacionales.

Los patrones creacionales nos proporcionan un proceso abstracto para la instanciación de objetos, estos patrones nos ayudan a implementar sistemas independientes capaces de crear, componer y representar objetos. Los patrones creacionales delegan la instanciación a otro objeto, lo cual implica una codificación de un conjunto fijo de comportamientos a un conjunto más específico que pueden ser compuestos por otros más complejos.

[2] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Elements of Reusable Object-Oriented Software

La siguiente tabla muestra y describe los patrones creacionales.

Nombre	Nivel	Descripción
Factory Method	Creacional de clase	Patrón que contiene un método que tiene como fin la instanciación de objetos donde no es necesario especificar una clase en concreto.
Abstract Factory	Creacional de objeto	Patrón que proporciona una interfaz con el objetivo de crear familias de objetos que están relacionados o son dependientes sin tener que especificar su clase concreta.
Builder	Creacional de objeto	Separa la construcción de un objeto complejo de su representación, con el fin de que el mismo proceso de construcción pueda crear diferentes representaciones.
Prototype	Creacional de objeto	Especifica la creación de objetos de una instancia ya existente.
Singleton	Creacional de objeto	Asegura que solo exista una instancia de un objeto y le provee un punto de acceso.

Tabla 5.2 Patrones creacionales.

5.4.1.2. Patrones estructurales.

Los patrones estructurales definen cómo se pueden combinar objetos y clases para formar grandes estructuras a partir de otras estructuras que pueden ser simples o complejas proporcionándoles nuevas funcionalidades.

La siguiente tabla muestra y describe los patrones estructurales.

Nombre	Nivel	Descripción
Adapter	Estructurales de clase	Convierte una interfaz a una interfaz que el cliente está esperando, este patrón les permite trabajar juntas debido a que no todas las interfaces son compatibles.
Bridge	Estructurales de objeto	Este patrón separa la abstracción de su implementación, de manera que puedan cambiar de forma independiente.
Composite	Estructurales de objeto	Este patrón consiste en crear objetos a partir de otros más específicos, también puede contener otros objetos. Se crean objetos en estructura de árbol para representar jerarquías.
Decorator	Estructurales de objeto	Consiste en extender dinámicamente nuevas funcionalidades a un objeto.
Facade	Estructurales de objeto	Consiste en una interfaz unificada para un conjunto de interfaces de un sistema.
Flyweight	Estructurales de objeto	Patrón que utiliza un objeto compartido para soportar un gran número de objetos finitos eficientemente.
Proxy	Estructurales de objeto	Provee un objeto como medio para controlar el acceso a otro objeto.

Tabla 5.3 Patrones estructurales.

5.4.1.3. Patrones de comportamiento.

Los patrones de comportamiento describen la forma de comunicación entre objetos y clases. Estos patrones se caracterizan por contar con un flujo de control complejo que es difícil de seguir en tiempo de ejecución. El objetivo del patrón de comportamiento es el control de la interconexión entre objetos.

La siguiente tabla muestra y describe los patrones de comportamiento.

Nombre	Nivel	Descripción
Chain of Responsibility		Evita el acoplamiento del emisor de la petición de su receptor, dando a más de un objeto la oportunidad de manejar la petición.
Command	Comportamiento de Objeto	Consiste en parametrizar a los clientes en distintas peticiones, así como tener un registro de las peticiones y poder deshacer las operaciones, esto se lleva a cabo gracias al encapsulamiento de cada petición en un objeto.
Interpreter		Dado un lenguaje define una representación de su gramática además de un intérprete que permite interpretar las sentencias del lenguaje.
Iterator		Permite recorrer una colección de objetos si tener que saber su orden interno.
Mediator		Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento para evitar que los objetos se refieran unos a otros explícitamente.
Memento		Captura y externaliza el estado interno de un objeto sin violar las propiedades de encapsulamiento para poder recuperar el estado posteriormente.
Observer		Este patrón define una dependencia de uno a muchos para cuando un objeto cambie su estado se le notifique y actualizan de manera automática todos los objetos relacionados.
State		Permite que un objeto modifique su comportamiento cada que cambie su estado interno, esto da la apariencia de que cambia la clase del objeto.
Strategy		Define una familia de algoritmos, encapsulando cada uno y permitiendo que sean intercambiados. Permite que un algoritmo varíe independientemente de los clientes que lo usen.
Template Method		Define el esqueleto de un algoritmo delegando a una subclase que permite

	redefinir ciertos pasos del algoritmo sin cambiar su estructura.
Visitor	Representa una operación a realizar sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin tener que realizar algún cambio a las clases de los elementos sobre las que opera.

Tabla 5.4 Patrones de comportamiento.

5.4.2. Patrones de arquitectura.

Los patrones de arquitectura establecen una estructura fundamental en la organización esquemática que deben tener los sistemas y proporcionan un conjunto de subsistemas predefinidos, sus responsabilidades y un conjunto de reglas y guías para organizar las relaciones existentes entre ellos.

5.4.2.1. Patrones Java EE

Debido a la evolución y demanda del desarrollo de aplicaciones web, aparece un catálogo de patrones para el desarrollo de aplicaciones empresariales Java EE. En un libro publicado por SUN llamado *"Core J2EE Patterns"* se encuentran documentados los patrones orientados al diseño de aplicaciones empresariales J2EE. Hoy en día estos patrones aún prevalecen, sin embargo con la reciente evolución a la plataforma Java EE aparece un nuevo catálogo de patrones de diseño.

La siguiente tabla muestra los patrones de diseño clasificados por capas.

Capa	Patrón	Descripción
Presentación	View Helper	Este patrón permite encapsular en un objeto la información antes de ser enviada hacia la vista o el modelo
Aplicación	Business Delegate	Es un patrón que se utiliza para enviar información entre la capa de presentación y los servicios de negocio. Los Business Delegate ocultan los detalles de la implementación de los servicios de negocio.
Negocio	Bussines Object	Este patrón permite crear un objeto que separa la información empresarial de la lógica de negocio. Este objeto es encargado de procesar información, realizar operaciones y evaluar reglas de negocio.
Persistencia	Data Transfer Object	El patrón DTO se usa para transferir datos entre las diferentes capas. La diferencia con el Business Object es que el DTO no contiene ninguna lógica de negocio.
	Data Access Object	Patrón que permite utilizar un objeto para el acceso a las fuentes de información. El DAO es el encargado de gestionar la conexión con la fuente de datos.
	Template	El patrón Template permite crear una plantilla que define una serie de pasos estándar que se repiten. Por ejemplo, cuando trabajamos con un *framework* de persistencia de datos como Hibernate es necesario crear primero una nueva sesión de Hibernate, posteriormente iniciar una transacción antes de ejecutar cualquier operación, cuando la operación haya terminado es necesario cerrar una conexión y opcionalmente confirmar o deshacer una transacción. El patrón Template evita repetir siempre los mismos pasos comentados anteriormente.

Tabla 5.5 Patrones Java EE

5.4.2.2. Modelo Vista Controlador (MVC)

El patrón MVC (*Model View Controller*) es uno de los patrones de diseño más usado en la actualidad. El patrón MVC inició tras el desarrollo de *Smalltalk* creado por Trygve

Reenskaug durante los años 70's. Desde entonces, ha servido como base para el desarrollo de la mayoría de los *frameworks* actuales orientados al diseño MVC.

El MVC es un patrón de arquitectura que separa la interfaz del cliente de la lógica de negocio en tres componentes distintos.

A continuación se describen los componentes del patrón MVC.

- **Modelo**. Un modelo generalmente es un *JavaBean* que contiene la información empresarial. El objeto del modelo no debe saber cómo recuperar los datos de la capa de negocio, en su lugar el modelo contiene los atributos que permitan ser inicializados por el controlador. Por lo tanto la presentación de las vistas nunca tendrá errores por falta de acceso a la información. Esta funcionalidad del modelo permite la reducción de errores en la capa de presentación.
- **Vista**. Es un componente usado para visualizar los datos de un modelo. La función de una vista es únicamente la visualización de la información, por lo que nunca debe contener lógica de negocio ni contener otros datos que no sean proporcionados al modelo.
- **Controlador**. Un controlador es una clase Java que permite manejar las peticiones del cliente, interactúa con los objetos de negocio, crea los modelos y los envía a las vistas de su petición correspondiente. La petición al controlador no implementa la lógica de negocio, la responsabilidad del controlador es interactuar con las implementaciones de la capa de negocio.

La imagen 5.3 muestra el control de flujo del patrón MVC. El *servlet* controlador es un *servlet* que ofrece algún *framework* como Struts, el controlador de peticiones es una clase que pertenece a la capa de presentación y es la encargada de interactuar con las interfaces de negocio para crear el modelo. La vista puede ser un JSP o un XSLT.

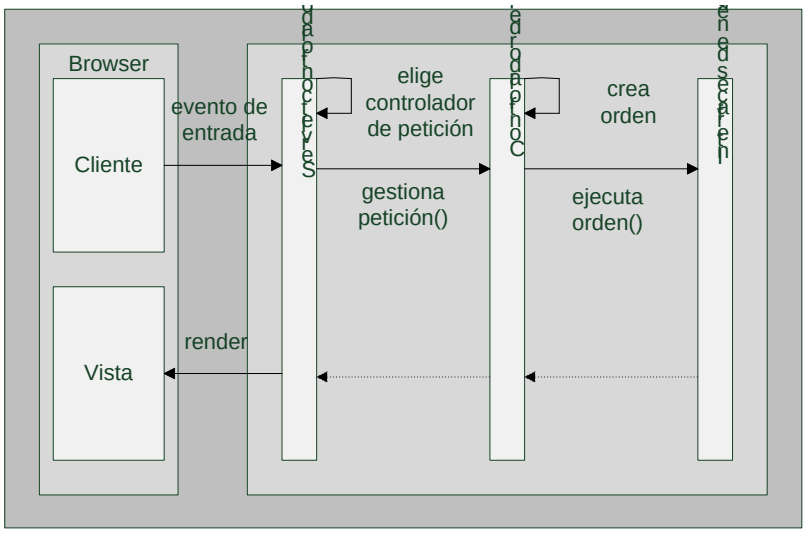


Imagen 5.3 Ejemplo MVC

5.4.2.3. Inversión de Control (IoC) e Inyección de dependencias (DI)

IoC es un patrón de arquitectura que exterioriza la creación y gestión de las dependencias de los componentes. Consideremos un ejemplo donde tenemos una clase A que depende de una instancia de la clase B para realizar algún tipo de procesamiento. Tradicionalmente la clase A crea una instancia de la clase B usando el operador *new* u obteniéndola con algún tipo de clase *factory*. El patrón IoC permite que una instancia de la clase B (o una subclase) le sea proporcionada a la clase A por un componente externo. El patrón IoC es una implementación del principio de Hollywood (no nos llames, nosotros te llamamos).

Existen múltiples formas de implementar la IoC, entre ellas se encuentra la inyección de dependencias que consiste en proporcionarle la referencia de una dependencia a una clase en tiempo de ejecución a través de propiedades o por argumentos del constructor.
Existen dos principales maneras de implementar la inyección de dependencias:

- **Inyección por constructor**. Este método se basa en inyectar las dependencias de los componentes por su constructor.
- **Inyección por método *setter***. En este método el contenedor IoC inyecta las dependencias en el componente mediante métodos *setter* al estilo *JavaBean*.

Para el diseño de aplicaciones la inyección de dependencias tiene los beneficios siguientes:

- Reduce drásticamente el código necesario para acoplar los diferentes componentes de una aplicación. Comúnmente este código es trivial, debido a que la creación de una dependencia es una simple instanciación de un objeto. Sin embargo el código puede resultar complicado cuando aumenta la complejidad de la invocación de las dependencias o simplemente no se pueden invocar.
- Aplicaciones simples de configurar. La inyección de dependencias puede utilizar anotaciones o XML para configurar las clases que son inyectables a otras.
- Gestiona las dependencias comunes en un solo repositorio.
- Facilita el cambio de dependencias. Esto afecta radicalmente a la capacidad de realizar pruebas a la aplicación.
- Se cuenta con un buen diseño de aplicaciones.

5.5. Struts 2.

Struts 2 es un *framework* de presentación basado en el patrón MVC con una amplia capacidad de configuración. Este *framework* está diseñado para agilizar el ciclo de desarrollo: construcción, implementación y mantenimiento de las aplicaciones a través del tiempo.

El proceso común que sigue una petición en Struts 2 es:

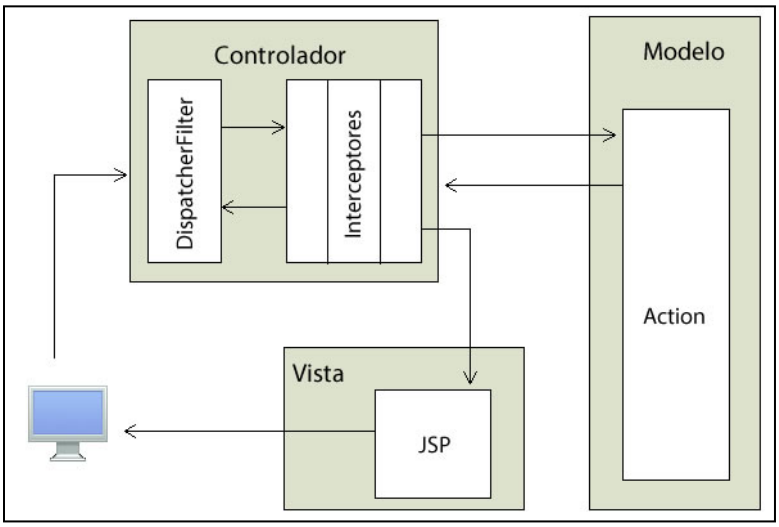


Imagen 5.4 Flujo de una petición en Struts 2.

1) El cliente hace una petición para un recurso de la aplicación. El filtro de Struts2 revisa la petición y determina el *Action* apropiado para servirla.
2) Se aplican los interceptores los cuales realizan algunas funciones como validaciones, flujos de trabajo y carga de archivos.
3) Se ejecuta el método adecuado del *Action* con la lógica del negocio.
4) El *Action* indica cuál *Result* debe ser aplicado. El *Result* genera la salida apropiada dependiendo del resultado del proceso.

5) Se aplican al resultado los mismos interceptores que se aplicaron a la petición, pero en orden inverso.
6) El resultado vuelve a pasar por el *DispatcherFilter* aunque este ya no hace ningún proceso sobre el resultado.
7) El resultado es enviado al usuario y este lo visualiza.

5.5.1. Componentes.

Los principales componentes del *framework* son:

- **DispatcherFilter**

  Es el punto de entrada, a partir de él se lanza la ejecución de todas las peticiones que involucran al *framework*. Sus principales responsabilidades son:

  - Ejecutar los *Actions* correspondientes a las peticiones.
  - Comenzar la ejecución de la cadena de interceptores asociados a la petición.
  - Limpiar el contexto sobre el cual se ejecuta el *Action* para evitar fugas de memoria.

- **Interceptores**

  Son clases que permiten implementar funcionalidades para ejecutarse antes y después de la llamada a un *Action*. Incluso pueden prevenir la ejecución del *Action* correspondiente.

Struts2 define un conjunto de interceptores por defecto y con un orden en particular (cadena de interceptores) que le permite realizar funcionalidades sobre la petición de los *Actions*. Estas funcionalidades son, por ejemplo: evitar el doble envío de un formulario, conversión de tipo de datos, inicialización de objetos, validaciones, subida de archivos y más.

- **Actions**

  Son los encargados de ejecutar la lógica necesaria para manejar una petición determinada y regresar un resultado. Estrictamente hablando, el único requisito para que una clase sea considerada como *Action* es que debe tener un método que no reciba argumentos y que devuelva una cadena o un objeto de tipo *Result*. Sin embargo, los *Actions* también pueden implementar la interfaz *Action* o heredar de una clase base que proporciona Struts2, *ActionSupport*, la cual hace más sencilla la creación y el manejo de dicho *Action*.

  ActionSupport es una clase de la librería *XWork* que proporciona interfaces para realizar validaciones e internacionalización. Dado que muchos procesos requieren ambas, es conveniente que dichos procesos se hagan por defecto. La utilización de ésta clase no es requerida en el *framework* de Struts2, sin embargo provee utilidades que facilitan el desarrollo. Por ejemplo, en la interfaz *Action* implementa un único método público que regresa una cadena y proporciona constantes con los nombres de los resultados más comunes (*success*, *error*, *input*, *login*).

- **Results**

  Los *Results* son objetos que encapsulan el resultado de un *Action*. Después de que un *Action* ha procesado la lógica, debe enviar la respuesta de regreso al usuario en forma de *Result*. El tipo del resultado indica cómo debe ser tratado el resultado que se regresará al cliente (JSP, archivo, JSON, redirección, entre otros). Un *Action* puede tener más de un *Result* asociado, esto nos permitirá enviar una vista distinta dependiendo de la lógica de ejecución del *Action*.

  A pesar de que Struts 2 cuenta con una gran variedad de tipos de resultados, también provee la posibilidad de crear resultados propios, sólo se necesita la creación de la lógica de la funcionalidad y dar de alta el nuevo tipo de resultado en el archivo de configuración, indicando el nombre y la clase a la que hace referencia.

5.5.2. Tags.

Struts2 proporciona un conjunto de etiquetas para la vista. Se pueden dividir en dos tipos: generales y de interfaz gráfica. Las generales son usadas para controlar la ejecución del flujo cuando se genera la página. También sirven para obtener información de lugares distintos del *Action*, como *JavaBeans*.

Las etiquetas de interfaz gráfica se enfocan en el manejo de la información proveniente de la pila de valores o de un *Action* para poder ser desplegadas. Además de su funcionalidad, la mayor diferencia entre estos dos tipos es que las de interfaz gráfica soportan plantillas y temas.

Etiquetas generales.

Etiqueta	Funcionalidad
if	Realiza un condicional básico. Puede usarse solo o en conjunto con la etiqueta else o else if.
elseif	Realiza un condicional básico en conjunto con la etiqueta if.
else	Realiza un condicional básico en conjunto con la etiqueta if y elseif.
iterator	Itera sobre un valor que puede ser un objeto de cualquier colección.
sort	Ordena una lista utilizando un objeto comparador. Tanto la lista como el comparador se proporcionan mediante atributos de la etiqueta.
subset	Indica el rango que utilizará la etiqueta *iterator* sobre la lista.
a	Crea un ancla HTML.
action	Permite llamar a un *Action* desde una JSP especificando el nombre y el *namespace* al que pertenece.
date	Llena un objeto *Date* de acuerdo al formato especificado como atributo de la etiqueta.
include	Incluye la salida de una JSP o *Servlet* en la página actual.
param	Se utiliza para pasar parámetros a otras etiquetas para su construcción.
property	Proporciona el valor de una propiedad, que será la primera en la pila de valores si ninguna ha sido especificada.
push	Agrega un valor a la pila de valores.
set	Crea una variable y la inicializa con el valor especificado como parámetro.
url	Crea una URL.

Tabla 5.6 Etiquetas generales de Struts 2.

Etiquetas de interfaz gráfica

Etiqueta	Funcionalidad
checkbox	Genera una etiqueta HTML de tipo *checkbox*.
checboxlist	Genera una serie de etiquetas HTML de tipo *checkbox* con los valores especificados en una lista.
combobox	Genera una etiqueta HTML de tipo *select* y sus respectivas opciones obtenidas a partir de una lista.
head	Genera el encabezado de un documento HTML.
file	Genera una etiqueta HTML de tipo input file.
form	Genera un formulario HTML.
hidden	Genera una etiqueta HTML de tipo *input hidden* inicializada con el valor especificado.
label	Genera una etiqueta HTML de tipo *label* para usar en conjunto con aquellas etiquetas que especifiquen el atributo *label*.
optgroup	Genera un grupo de opciones para una etiqueta *select*.
password	Genera una etiqueta HTML de tipo input password.
radio	Genera un elemento HTML de tipo radio.
reset	Genera un botón de tipo *reset*.
select	Genera un elemento HTML de tipo *select* y sus opciones correspondientes de acuerdo a una lista proporcionada como parámetro de la etiqueta.
submit	Genera un botón de tipo *submit*.
textarea	Genera una caja de texto.
textfield	Genera un elemento HTML de tipo *input text*
div	Genera un elemento HTML de tipo *div*

Tabla 5.7 Etiquetas de interfaz gráfica de Struts 2.

5.6. Spring 3.

Spring es un *framework* mediante el cual se puede simplificar la construcción de aplicaciones Java y que, a diferencia de otros *frameworks,* nos permite el desarrollo de prácticamente cualquier aplicación Java sin estar supeditado únicamente a la elaboración de aplicaciones web. Entre sus características más importantes están su ligereza (no requiere de muchos recursos para su ejecución) y la gran cantidad de servicios que ofrece. Ambas características han hecho que Spring se haya convertido en una de las soluciones para desarrollo informático más aceptadas dentro de la industria actual del *software*.

El inicio de Spring se encuentra en la publicación del libro *"Expert One-to-One J2EE Design and Development"* escrito por Rod Johnson. En este libro el autor explica su propio *framework* llamado Interface 21 que a la postre, al pasar a formar parte del mundo del *software* de código abierto, adoptaría el nombre de Spring.

En la plataforma Java EE hay ocasiones en las que el uso de la tecnología EJB puede llegar a ser complicado debido a que está orientada a una arquitectura distribuida e invocaciones de transacciones remotas. Spring intenta reducir el uso de esta operación mediante la integración de diferentes tecnologías en un solo *framework* que brinde una solución simplificada y portable entre servidores de aplicaciones. Aun así, si el desarrollador lo requiere, Spring cuenta con una forma de integrarse con la tecnología EJB.

La versión de Spring 3.0.x cuenta con 20 módulos empaquetados dentro de 20 archivos JAR. En la tabla siguiente se muestran la descripción de los módulos con los que estará construida la arquitectura que se pretende proponer.

Módulo	Descripción
aop	Clases necesarias para utilizar las características de AOP en una aplicación con Spring.
beans	Clases que Spring proporciona para la manipulación de *beans*.
context	Extensión del *core* de Spring. Contiene clases para JNDI.
core	Núcleo para las aplicaciones de Spring.
jdbc	Incluye todas las clases para JDBC. Este módulo es requerido para las aplicaciones que requieran acceso a una base de datos.
orm	Amplia los servicios JDBC de Spring para la integración con una herramienta ORM como Hibernate. Cualquier clase de este componente depende del módulo de JDBC.
transaction	Proporciona las clases de transacciones.
web	Contiene las clases para utilizar Spring en una aplicación web.

Tabla 5.8 Módulos de Spring 3 usados en la arquitectura propuesta.

Existen también diferentes librerías que ayudan a que herramientas de terceros puedan adaptarse a desarrollos realizados por medio de Spring. A continuación se muestra una tabla que contiene las herramientas más usadas en el desarrollo de sistemas web y la forma de integrarlas con Spring.

Tecnología	Descripción
Hibernate	El *framework* ORM de Hibernate puede ser integrado a una arquitectura basada en Spring. Esta integración necesita de los archivos JAR que proporciona la distribución de dicho *framework*.
log4j	Esta integración puede ser utilizada para que Spring sea el encargado de configurar el log4j.
Struts2	El uso de Spring junto con el *framework* de Struts2 se lleva a cabo con las librerías de la distribución del *framework* así como el *plugin* de integración de Spring con Struts2.

Tabla 5.9 Herramientas de desarrollo y su integración con Spring.

Es importante mencionar que Spring es un *framework* bastante grande y es muy complicado definir todas sus funcionalidades, razón por la que en el presente trabajo únicamente se definirán los conceptos básicos utilizados durante el diseño y construcción del sistema a realizar.

5.6.1. Inyección de dependencias.

El núcleo de Spring está basado en el patrón de diseño llamado Inyección de dependencias, que como se comentó anteriormente es una forma de la IoC. Los métodos de inyección de dependencia que Spring proporciona son dos: Inyección por método *setter* e inyección por constructor.

La IoC en Spring se realiza por medio de una interfaz `BeanFactory` que representa el contenedor para la inyección de dependencias. Esta interfaz es la responsable de manejar los componentes, incluyendo las dependencias y su ciclo de vida. Es importante mencionar que en Spring el término *bean* es usado para referirse a cualquier componente que sea manejado por el contenedor de inyección de dependencias. También existe dentro de Spring la interfaz `ApplicationContext`, ésta hereda de la interfaz `BeanFactory` y ofrece una serie de características mucho más robustas por lo que es altamente recomendable utilizarla en el desarrollo de la mayoría de las aplicaciones.

En una aplicación web la configuración de Spring se realiza colocando un *listener* en el archivo *web.xml* para que automáticamente arranque el `ApplicationContext` e inicialice el archivo de configuración de Spring. La clase *listener* especificada en el archivo *web.xml* debe tener el comportamiento de la clase `RequestContextListener`. La

configuración de los *beans* y sus dependencias se especifica en un archivo XML de configuración cuyo nombre se encuentra definido como un parámetro en el `web.xml`.

Existen dos formas de configurar Spring: XML y Anotaciones. La configuración por XML exterioriza la configuración, mientras las anotaciones permiten programar la configuración dentro del código mismo. Existen sus pros y sus contras para las dos, sin embargo, se considera buena práctica que el diseño de la aplicación exteriorice cualquier configuración de tal forma que sea mucho más fácil visualizar las dependencia declaradas en archivos XML que si estuvieran incluidas dentro del código desarrollado. Por esta razón en el presente documento sólo se detallará la configuración con XML.

La estructura del XML necesita estar definida por el DTD o por los *namespaces* que la aplicación requiera. Todos los *beans* susceptibles de ser instanciados por Spring se deben declarar dentro de la etiqueta `beans` del archivo de configuración. No hay que olvidar que para la creación y configuración de un *bean* en Spring se debe de contar con tres elementos:

- Una interfaz que contiene todos los métodos que el diseño del *bean* necesite.
- Una clase que implemente los métodos de la interfaz del punto anterior.
- La definición del *bean* dentro del XML de configuración.

Anteriormente se mencionó que Spring utilizaba dos formas de realizar la inyección de dependencias a continuación se explica cómo se define el uso de cada una de estas formas en el archivo de configuración de Spring.

- Inyección por método setter

  La configuración de inyección por *setter* necesita especificar una etiqueta `property` contenida dentro de la etiqueta `bean` de la dependencia a la que se le quiera inyectar. En el *bean* se debe especificar además un identificador, la clase a la que hace referencia el *bean* y las propiedades que se deben inyectar.

- Inyección por constructor

  En este tipo de inyección se especifica el identificador, la clase y los argumentos del constructor.

5.6.2. Inyección de parámetros.

Spring soporta una gran cantidad de opciones para inyectar parámetros, permite inyectar parámetros simples, otros componentes, colecciones Java, propiedades definidas externamente e incluso *beans* definidos en otra *factory*. Se puede usar cualquiera de estos tipos de inyección de parámetros para la inyección *setter* así como para la inyección por constructor usando su correspondiente etiqueta `property` y `constructor-args` respectivamente.

Los principales tipos de parámetros que se pueden inyectar así como la forma de configurar su inclusión se explican a continuación:

- Inyección de parámetros simples

  Para inyectar parámetros simples únicamente se especifica el valor en la configuración de la etiqueta `bean`. Esto se hace a través de la etiqueta `value`.

El valor predeterminado es un tipo String, sin embargo se puede especificar cualquier tipo de dato primitivo de Java.

- Inyección de *beans*

  En la mayoría de veces tenemos la necesidad de inyectar un *bean* dentro de otro, para lograrlo usamos la etiqueta `ref` dentro de la definición del *bean* al que se le desee inyectar el parámetro.

- Inyección de colecciones

  A menudo, los *beans* necesitan tener acceso a colecciones de objetos en lugar de sólo *beans* o valores simples. Spring permite inyectar una colección de objetos en un *bean*. Se pueden inyectar las colecciones: `Map`, `Properties`, `Set` y `List`.

5.6.3. Ciclo de vida de Spring.

Spring no sólo instancia objetos, también se ocupa de manejar el ciclo de vida de cada objeto. Esto permite a los *beans* realizar cierto procesamiento relevante en algún punto de su ciclo de vida. En general existen dos eventos importantes en el ciclo de vida de un *bean*: después de la inicialización y antes de la destrucción.

Para definir un método de inicio, es necesario especificar un atributo `init-method` en el *bean*. Del mismo modo para especificar la lógica de destrucción del *bean* se coloca el nombre del método en el atributo `destroy-method`.

Además de los procesos que se pueden realizar al momento de crear o destruir un *bean*, es importante mencionar los alcances o modos de instanciación que están relacionados con el ciclo de vida de un *bean*.

5.6.4. Modos de instanciación.

Por defecto, todos los *beans* de Spring son *singletons*. Esto significa que Spring mantiene una sola instancia del *bean* y todos los objetos dependientes usan la misma instancia. El término *singleton* es usado indistintamente en Java para hacer referencia a dos conceptos distintos: un objeto que tiene una única instancia en la aplicación y el patrón de diseño Singleton. Es importante mencionar que en este caso el término es utilizado para el primero de éstos.

En general, el modo *singleton* debería de ser usado en los siguientes escenarios:

- **Objetos compartidos sin ningún estado**: Debido a que no se necesita sincronizar si no hay estado, no es realmente forzoso crear una nueva instancia del *bean* cada vez que un objeto dependiente requiera usarlo.
- **Objetos compartidos de solo lectura**: Este punto es similar al anterior, pero tiene un estado de sólo lectura. En este caso, no es necesario la sincronización, por lo que crear una instancia del *bean* para satisfacer cada solicitud solo causaría sobrecarga adicional.
- **Objetos compartidos con estado compartido**: Si se tiene un *bean* que tiene estado compartido, el modo *singleton* es el ideal.

- **Alto rendimiento de objetos con estado compartido**: Si se tiene un *bean* que se usa mucho en una aplicación, entonces encontramos que mantener un *bean* en modo singleton, permite un mejor rendimiento que estar creando constantemente varias de instancias.

Por lo general se busca que la mayoría de veces el modo de instanciación sea *singleton* pero no siempre se puede lograr este objetivo por lo que Spring provee, además de *singleton*, otros modos de instanciación o alcances para una aplicación web. A continuación se muestran estos modos:

- **Singleton**. El alcance predeterminado.
- **Prototype**. Una nueva instancia se crea cuando la aplicación lo solicite.
- **Request**. Consiste en una sola instancia por cada petición HTTP.
- **Session**. Consiste en una sola instancia por cada sesión HTTP.

5.6.5. Herencia de *beans.*

Spring permite definir un *bean* que hereda las propiedades de otro *bean* en el mismo `AplicationContext`. En esta definición el *bean* padre puede proporcionar a cada *bean* hijo una configuración base, el *bean* hijo puede descartar algunas propiedades según sea necesario.

La configuración del *bean* hijo se realiza colocando el id del *bean* padre en el atributo `parent` de la etiqueta `bean`.

5.6.6. Spring AOP.

Además de la inyección de dependencias, otra característica fundamental del *framework* de Spring es la posibilidad de desarrollar aplicaciones orientada a aspectos (AOP). AOP es una técnica de programación que se presenta normalmente al usar el *framework* de Spring. El uso de la programación orientada a aspectos trata de modularizar el comportamiento que separa la división de responsabilidades como el manejo de transacciones y la seguridad de la aplicación.

Spring AOP puede ser utilizado para ciertos elementos de una aplicación como son:

- Inicio de sesión.
- Seguridad
- Debuggin.
- Persistencia.
- Manejo de Transacciones.

Una característica importante de Spring AOP es la portabilidad entre servidores de aplicación. Spring AOP se basa en puntos de ejecución donde se cuenta con un control en específico de la petición que se realiza, a esto se le conoce como *Pointcuts Dinámicos*.

5.7. Hibernate 3.

5.7.1. Object-Relational Mapping (ORM).

ORM es una solución que hace posible manipular objetos sin la necesidad de considerar cómo dichos objetos interactúan con la fuente de datos, separando así, la lógica de negocios con el manejo del modelo relacional en la capa de persistencia.

Se han desarrollado una gran cantidad de soluciones ORM. Básicamente un *framework* de esta naturaleza mapea objetos cuando son persistidos a parámetros de una sentencia por medio de JDBC. También proveen mapeos más complejos, como la jerarquía en herencia y las relaciones entre objetos, *"lazy loading"* y el manejo temporal de objetos persistentes. Esta última característica permite a los *frameworks* ORM mantener en memoria aquella información que es consultada constantemente para que, en lugar de realizar la búsqueda en la base de datos en las siguientes peticiones, causando deficiencias y lentitud de las respuestas, los objetos sean obtenidos de la memoria. El *"Lazy loading"* es otra característica importante que permite obtener un objeto sin tener que inicializar los objetos con los que se relaciona hasta el momento en que son requeridos.

Los *frameworks* ORM usualmente requieren las definiciones de los mapeos que determinan cómo deben ser mapeados cada uno de los objetos persistentes en las tablas y columnas de una base de datos. Esta configuración se realiza de forma declarativa, lo cual permite el desarrollo con un código más flexible.

Algunas de las soluciones ORM proveen un lenguaje de consulta mediante objetos que permiten realizar consultas de una forma orientada a objetos, evitando así el uso directo de las tablas y columnas mediante SQL.

5.7.2. Hibernate como una solución ORM.

Hibernate es un intermediario que relaciona un ambiente orientado a objetos a uno relacional. Provee un servicio de persistencia para las aplicaciones ejecutando todas las operaciones requeridas entre dichos ambientes. Incrementa la efectividad y el desempeño de las aplicaciones, disminuye la cantidad de código y permite a los desarrolladores centrarse en la lógica de negocio.

Hibernate trabaja con un modelo de programación basado en POJOs *(Plain Old Java Object*) minimizando así la dependencia de la aplicación en los *framworks* y haciendo el desarrollo más productivo y portable.

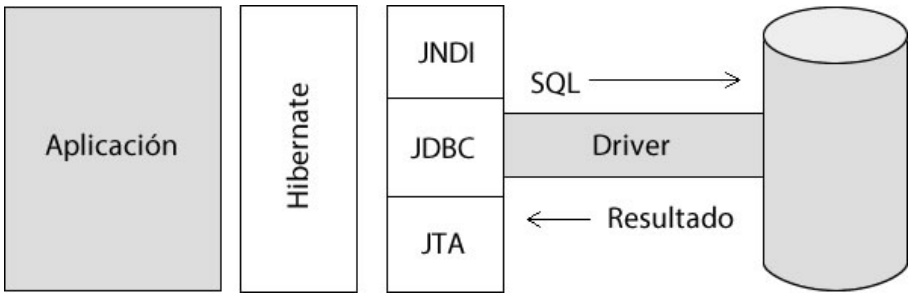


Imagen 5.5 Intervención de Hibernate en una aplicación.

El funcionamiento de Hibernate se define básicamente en tres partes:

- Archivos de configuración.

  La configuración siempre se presenta en archivos XML o en propiedades que incluyen la información necesaria para llevar a cabo la conexión y el manejo de la base de datos (usuario, contraseña, URL, driver, dialecto SQL).

- Mapeos de la base.

  Son archivos XML que establecen cómo se deben mapear los objetos de la aplicación respecto a la base de datos. Estos documentos especifican la tabla a la que corresponden, el mapeo de cada atributo de la clase a su respectiva columna en la base y en ocasiones, otro tipo de información como relaciones a otros objetos y herencia. Estos archivos tienen una sintaxis simple, haciéndolos fácil de desarrollar y de mantener.

- Objetos persistentes.

  Los objetos persistentes son aquellos que permiten la persistencia de la información con la base de datos. Estas entidades y sus clases únicamente deben seguir las reglas de una clase POJO.

5.7.3. Ciclo de vida de los objetos persistentes.

Las clases persistentes únicamente siguen las reglas de un POJO, no es necesario que invoquen un API de Hibernate. En otras palabras, no es necesario que una clase persistente contenga un indicador de su persistencia. Esta es la razón por la que Hibernate es un proveedor transparente de persistencia. Cada POJO puede ser manejado por Hibernate si:

- El objeto puede ser guardado en una tabla de la base de datos.
- Los atributos del objeto pueden ser guardados en una base de datos.
- Existe una definición que determina cómo se mapeará cada atributo del objeto a su correspondiente columna de la tabla.

Para persistir los objetos, la aplicación debe llamar a Hibernate mediante sus interfaces, generalmente instancias de la clase `Session`. Los objetos persistentes que son enviados y regresados mediante las sesiones tienen los siguientes estados en su ciclo de vida:

- Transitorio

  Se refiere a aquellos objetos que aún no son persistentes y no están asociados a alguna columna de una tabla. Estos objetos no serán guardados hasta que se realice una petición explícita. Hibernate considera a los objetos transitorios en un contexto no transaccional y no son monitoreados por si sufren un cambio (creados, modificados o eliminados), así que no existe un mecanismo de *roll-back* activado para ellos. Cuando la aplicación pide que se persista un objeto transitorio Hibernate le asigna un identificador, lo guarda en la base de datos y lo sitúa en un estado de persistencia.

- Persistente

  Los objetos transitorios son cambiados a persistentes cuando se guardan en la base de datos. Persistente se refiere a aquellos objetos que tienen una representación en la base de datos con una llave primaria apropiada y están asociados a una sesión válida. Antes de que las sesiones asociadas sean cerradas, Hibernate realiza los cambios de los objetos en la base de datos para mantenerla actualizada. El cambio de estado a Aislado es transparente, se realiza al cerrar la sesión asociada al objeto persistente.

- Aislado

  Al igual que los objetos persistentes, los objetos aislados son objetos que ya han sido persistidos, con la diferencia de que su sesión asociada ya ha sido cerrada. Hibernate ya no se hace cargo de ellos y no monitorea las modificaciones que puedan sufrir. Sin embargo, cuando son asociados a otra sesión válida, son regresados a un estado persistente.

- Eliminado

  Se refiere a aquellos objetos persistentes que han sido dispuestos para su eliminación de la base de datos por una sesión. Los objetos eliminados son borrados de la base de datos y cambiados a un estado de trascendencia en cuanto la sesión ejecuta las transacciones. Esto se puede hacer explícitamente invocando una operación de borrado para esos objetos o implícitamente invocando una operación de borrado para objetos relacionados cuando los objetos persistentes fueron mapeados con la opción de eliminación en cascada y son cambiados a un estado transitorio.

5.7.4. Persistencia con Hibernate.

El servicio de persistencia de Hibernate puede ser utilizado por distintas interfaces, incluyendo las de `Session`, `Query`, `Criteria` y `Transaction`. De todas estas, la interfaz `Session` juega un rol crucial, ya que la interacción de Hibernate involucra al menos un objeto `Session`.

El objeto `Session` realiza las operaciones básicas de persistencia, esto incluye desde guardar un objeto que ha sido creado, hasta cargar, actualizar y eliminar un objeto ya persistente. Incluye un manejador de transacciones para realizar las operaciones de persistencia como una única tarea. Adicionalmente, provee de una caché para que se realicen las cargas, actualizaciones y guardados a través de ella.

El API de `Session` es utilizado generalmente para los siguientes propósitos:

- **Realizar operaciones CRUD**: El término CRUD significa realizar operaciones para guardar, cargar, actualizar y eliminar mediante los métodos `save()`, `persist()`, `get()`, `load()`, `update()` y `saveOrUpdate()`.
- **Manejar las transacciones**: Delimitar las transacciones mediante los métodos `beginTransacion()`, `commit()` y `rollback()`.
- **Trabajar con filtros**: Aplicar, activar y desactivar filtro en los datos obtenidos en una consulta con los métodos `createFilter()`, `enableFilter()` y `disableFilter()`.
- **Crear consultas**: Crear las consultas expresadas con el API de Criteria o HQL, mediante los métodos `createQuery()`, `createSQLQuery()`, `createNamedQuery()` y `createCriteria()`.
- **Trabajar con los recursos de Hibernate**: Dar acceso a los recursos que una sesión utiliza en segundo plano, incluyendo la conexión a la base de datos y la ejecución de las transacciones, y el manejo del objeto `SessionFactory` perteneciente a la sesión.

5.7.5. Consultas en Hibernate.

Las consultas son una parte vital en cualquier código de acceso a información. La mayoría de las aplicaciones requieren realizar búsquedas en objetos persistentes que satisfagan un criterio determinado. SQL es un poderoso lenguaje para realizar consultas en información persistente y determinar criterios mediante las clausulas `WHERE` y `JOIN`. Mediante SQL se puede expresar cualquier restricción que se requiera y según el número de restricciones, la expresión de consulta se vuelve más compleja y es más complicada de expresar, probar y mantener.

Hibernate facilita una serie de soluciones para realizar consultas con objetos persistentes. Provee de una gran cantidad de métodos, desde el nativo SQL hasta consultas avanzadas con un API propio de Hibernate:

- Hibernate Query Language (HQL): HQL es un método de consulta flexible, poderoso y similar a SQL, sin embargo fueron diseñados para trabajar con un tipo de información distinta. Las consultas con HQL trabajan con las propiedades de objetos persistentes. En otras palabras, SQL se expresa en términos de tablas, columnas, vistas, etc., mientras que HQL lo hace en términos orientados a objetos, utilizando clases y propiedades. Tanto SQL como HQL se construyen a base de cadenas compuestas de las palabras reservadas `select`, `from` o `where`.
- SQL nativo: Hibernate permite el uso de SQL, sin embargo se debe considerar el costo de perder algunos de los beneficios como la portabilidad por lo que no es recomendado salvo en ocasiones especiales. Una buena razón para utilizar HQL en lugar de SQL nativo es que HQL es un lenguaje de consultas más compacto.

- API de Criteria: Es una alternativa a HQL que permite expresar las consultas y sus restricciones en forma de programación, a diferencia de HQL, que se construyen como cadenas. Cada consulta con el API de Criteria se representa como objeto de tipo Criteria y éstos utilizan objetos `Criterion` anidados para representar sus restricciones que actúan como filtros para seleccionar los objetos persistentes. La utilidad de Criteria proporciona una muy útil propiedad llamada "Consulta mediante ejemplo" (QBE) que permite realizar la búsqueda proporcionando un objeto como ejemplo.

5.7.6. Hibernate frente a otras soluciones de persistencia.

A diferencia de otros *frameworks* como iBatis, TopLink, entre otros o los EJBs con su solución en persistencia, Hibernate es de fácil aprendizaje y de un uso simple y comprensivo, y en algunos casos (como los EJBs), no requiere de un servidor de aplicaciones. Hibernate tiene mucha documentación y se encuentran recursos muy fácilmente. Para usar Hibernate únicamente se requiere java SE 1.2 o posterior y puede ser utilizado de forma autónoma o en aplicaciones distribuidas.

Hibernate soluciona muchos problemas en el mapeo de objetos a un ambiente relacional, sin embargo es importante mencionar que no es un sustituto de JDBC. Más aún, es una herramienta que se conecta a la base de datos mediante JDBC y presenta a la base de datos como un ambiente orientado a objetos.

5.8. Vistas.

En un sistema web una de las partes fundamentales es la encargada de la generación de la interfaz de usuario pues es precisamente a través de esta interfaz que el sistema se comunica con el operador que lo va a utilizar. Hablando de sistemas web enfocados a una operación manual, esta parte cobra mucha mayor relevancia pues además de tener que cumplir con los objetivos primarios del sistema lo tiene que hacer de una manera que resulte atractiva, amigable y transparente para el usuario humano.

Para cumplir ambas metas el desarrollo de software se beneficia de una infinidad de tecnologías que facilitan el desplegado visual de interfaces de usuario sobre navegadores web. Entre las más destacadas se pueden mencionar las que se enumeran en este apartado y que fueron las seleccionadas para solventar la capa de presentación de la arquitectura propuesta en este trabajo.

5.8.1. XHTML.

El *Extensible HyperText Markup Language* (XHTML) es un lenguaje de etiquetas derivado del *HyperText Markup Language* (HTML) que era usado para la construcción de páginas web. El HTML desciende del *Standard Generalized Markup Language* (SGML) quien, a su vez, se originó del *Generalized Markup Language* (GML) desarrollado por IBM como una implementación de un conjunto de macros basadas sobre el concepto de etiquetas para su formateador de texto denominado SCRIPT. El SGSML no es más que la normalización del GML bajo el estándar ISO 887, fue originalmente diseñado para permitir la compartición de archivos que fueran capaces de permanecer legibles por décadas sin importar los cambios de tecnologías. El HTML es un estándar reconocido por el *World*

Wide Web Consortium (W3C) y por lo mismo se puede asegurar que una página escrita en este lenguaje va ser visualizada exactamente de la misma forma en cualquier navegador web. Actualmente se aconseja preferir el XHTML sobre el HTML porque al poseer una sintaxis más estricta que la del HTML permite que sea más sencillo la búsqueda de errores o la realización de cambios posteriores además de que posibilita el uso de procesadores genéricos de XML para su análisis e interpretación.

HTML es un lenguaje de marcado, es decir, a través de etiquetas es posible indicar las características propias de un objeto del texto que se esté escribiendo. Las etiquetas en realidad consisten en palabras claves rodeadas por los símbolos < y >; para distinguir una etiqueta de apertura de una de clausura simplemente se le añade el carácter / al símbolo de cierre de la etiqueta de clausura, así, se tendría que para etiquetar un objeto, éste se debe encerrar entre una etiqueta de apertura y una de clausura de la forma siguiente:

```
<etiqueta> objeto <etiqueta/>
```

Incluso es posible anidar varias etiquetas para indicar varías características del objeto como a continuación se muestra:

```
<etiqueta1>
     <etiqueta2> objeto <etiqueta2/>
<etiqueta1/>
```

Se puede observar que esta forma de presentar la información tiene como principal ventaja la sencillez en su escritura y lectura haciéndola ideal para el procesamiento de información por parte de las personas y de los sistemas electrónicos, aunque a su vez presenta la

desventaja de aumentar considerablemente el tamaño del documento, cuestión por la cual se suelen utilizar etiquetas con nombres muy cortos.

El XHTML, por su parte, es en realidad una adaptación del HTML a los estándares de XML. También se encuentra regulado por el W3C quien publicó la primera versión del estándar el 26 de enero del año 2000 bajo la denominación XHTML 1.0. Como adaptación, XHTML mantiene casi todas las etiquetas y características del HTML pero añade restricciones y elementos propios de los documentos XML bien formados. En síntesis XHTML presenta una forma de codificación mucho más estricta que el HTML, estas restricciones son las siguientes:

- La declaración del tipo de documento (DOCTYPE) es obligatoria.
- La definición del atributo `namespace` en la etiqueta `html` es obligatoria.
- Las etiquetas `html`, `head`, `title` y `body` son obligatorias
- Las etiquetas deben de estar correctamente anidados, es decir, la que se abre primero es la última en cerrarse.
- Todas las etiquetas, incluso las vacías, deben ser siempre cerradas.
- Las etiquetas se escriben en minúsculas siempre.
- Los atributos de las etiquetas se escriben siempre en minúsculas.
- Los atributos se encierran entre comillas ya sean simples o dobles.
- Se prohíbe la compactación de atributos.
- Todos los símbolos deben ser escritos utilizando su nombre de entidad aún en las URLs, por ejemplo escribir (`&`) en vez de sólo `&`.

- El atributo `name` ha sido formalmente desaprobado para las etiquetas `a`, `applet`, `form`, `frame`, `iframe`, `img`, y `map`, y puede ser excluido en futuras versiones.

A continuación se presenta una lista de las etiquetas XHTML más comunes y su función.

Etiqueta	Descripción
!DOCTYPE	Tipo de documento.
html	Raíz de todo el documento.
head	Encabezado del documento.
body	Cuerpo del documento.
link	Enlace a otros archivos-
meta	Metainformación sobre el documento.
script	Script.
style	Hoja de estilo incluida en el documento.
title	Título del documento.
<!-- ... -->	Comentario.
a	Hiperenlace.
br	Salto de línea.
div	División en el documento.
img	Imagen.
span	Fragmento de texto (elemento en línea).
h1 a h6	Encabezados (de nivel 1 a 6).
p	Párrafo.
em	Énfasis.

strong	Énfasis (mayor que `em`).
ol	Lista ordenada.
ul	Lista no ordenada.
li	Elemento de lista.
table	Tabla.
col	Columna.
thead	Cabecera de tabla.
tbody	Cuerpo de tabla .
tfoot	Pie de tabla.
tr	Fila de la tabla.
th	Celda de cabecera.
td	Celda de la tabla.
form	Formulario.
label	Etiqueta de un control.
button	Botón.
input	Entrada de información.
option	Opción de un menú.
optgroup	Grupo de opciones en un menú.
select	Menú.
textarea	Área de texto.
b	Texto en negrita.
i	Texto en itálicas.

Tabla 5.10 Principales etiquetas XHTML.

5.8.2. CSS.

Hojas de estilo en cascada (CSS) es un lenguaje creado para organizar el aspecto de los documentos descritos por medio de los lenguajes HTML y XHTML. CSS brinda una inmejorable manera de separar el contenido de la forma en que éste se presenta en las páginas web, esto es deseable pues, de otra forma, se complicaría demasiado el mantenimiento de las páginas web ya que comúnmente la responsabilidad de la creación de los contenidos y el diseño recaen en personas distintas. Una ventaja añadida de la separación de los contenidos y su presentación es que los documentos XHTML creados son más flexibles adaptándose mejor a las diferentes plataformas que los presentan: pantallas de ordenador, pantallas de dispositivos móviles, impresoras y dispositivos utilizados por personas discapacitadas.

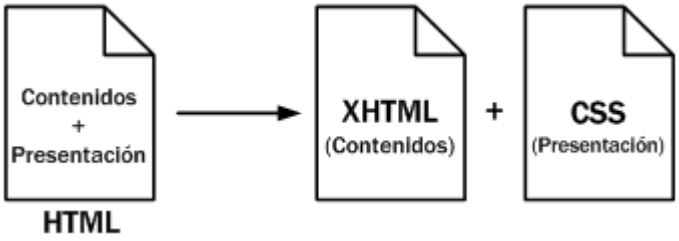


Imagen 5.6 Separación del contenido y presentación de una página web

CSS fue estandarizado por primera vez por el W3C en 1995 publicándose a finales de 1996 la primera recomendación oficial, la CSS nivel1 (CSS1). Posteriormente el 12 de mayo del año 1998 se publicó una segunda recomendación oficial conocida como CSS nivel 2 (CSS 2) con una revisión posterior (CSS2.1) que es la que hoy en día usa la mayoría de navegadores web, no obstante el W3C continua trabajando en una recomendación más conocida como CSS nivel 3 (CSS3) aunque de ésta sólo se han presentado borradores hasta

el momento. Cada nivel no es excluyente del anterior, más bien un nivel superior añade nuevas funcionalidades al nivel previo.

A grandes rasgos una CSS consiste en una serie de reglas que describen el aspecto con que se presentará un elemento determinado. Cada regla se compone de selectores, propiedades y atributos. Los selectores ayudan a discernir sobre qué elementos se aplicará el estilo indicado y las propiedades y atributos sirven para describir las características con las que se presentarán dichos elementos. A su vez cada regla se define entre los caracteres `{` y `}`, con lo que es posible definir varias reglas en un mismo documento. A continuación se presenta la estructura de una regla escrita en CSS:

```
selector [, selector2, ...] [:pseudo-class][::pseudo-element] {
 propiedad: valor;
 propiedad2: valor2;
 ...
}
```

Por ejemplo teniendo el siguiente código en HTML:

```
<html>
<body>
  <h1>Titular de la página</h1>
  <p>Un párrafo de texto no muy largo.</p>
</body>
</html>
```

Si se quisiera que la etiqueta `h1` se mostrara en color rojo con la fuente Arial y la etiqueta `p` se mostrara en color gris con una fuente Verdana se tendrían que aplicar los siguientes estilos:

```
h1 { color: red;  font-family: Arial;}
p  { color: gray; font-family: Verdana;}
```

Para incluir los estilos en un archivo HTML se puede hacer de dos formas: o se incluye directamente en el documento HTML o se escribe un documento aparte conteniendo los estilos y se enlaza hacienda referencia a él en el documento HTML. La primera forma se realiza encerrando la descripción de los estilos en la etiqueta `style` que debe encontrarse anidada a su vez en la cabecera del documento HTML. La segunda forma hace uso de la etiqueta `link` y enlaza con el documento que contiene los estilos. También es posible agregar un estilo directamente en un elemento HTML pero es una forma altamente desaconsejable y no se recomienda su uso. Por lo general al desarrollar una página web primero se utiliza el lenguaje XHTML para marcar el contenido y posteriormente se utiliza CSS para definir los aspectos de sus elementos como pueden ser: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición dentro de la página, etc.

El estándar CSS 2.1 define 115 propiedades, cada una con su propia lista de valores permitidos. Por su parte, los últimos borradores del estándar CSS 3 ya incluyen 239 propiedades. Algunas de las más importantes se describen a continuación:

Propiedades	Descripción
background-color	Color de fondo.
background-image	Utilizar una imagen como fondo.
background-attachment	Dejar fija la imagen de fondo.
background-position	Ubicar una imagen en un lugar determinado.
text-align	Centrar un texto.
text-decoration	Definir un texto tachado o subrayado.
text-transform	Convertir un texto a minúsculas o mayúsculas.

letter-spacing	Controlar el espacio entre letras.
word-spacing	Controlar el espacio entre palabras.
color	Color del texto.
font-family	Definir un tipo de fuente.
font-style	Estilo de la fuente.
font-size	Definir el tamaño de la fuente.
border-width	Establecer el ancho del borde del elemento.
border-color	Color del borde.
border-style	Estilo del borde.
margin	Margen alrededor de un elemento.
padding	Relleno entre el borde y el contenido.
table-layout	Tamaño de las celdas de una tabla.
width	Ancho de un párrafo.
height	Altura de un elemento.
overflow	Insertar una barra de desplazamiento en caso que el contenido sea superior a la caja que lo contiene
visibility	Mostrar u ocultar un texto
position	Posiciona un elemento dentro de la página.
top, right, bottom, left	Ubicar un elemento por medio de estas propiedades.
vertical-align	Alineación vertical del texto.

Tabla 5.11 Principales propiedades CSS.

5.8.3. AJAX.

El término *Asynchronous Javascript And XML* (AJAX) denomina al conjunto de tecnologías que auxilian en el desarrollo de aplicaciones web interactivas. La característica

principal de éste concepto es que nos permite desarrollar páginas web capaces de actualizar partes de su información sin la necesidad de tener que refrescar la página en su totalidad.

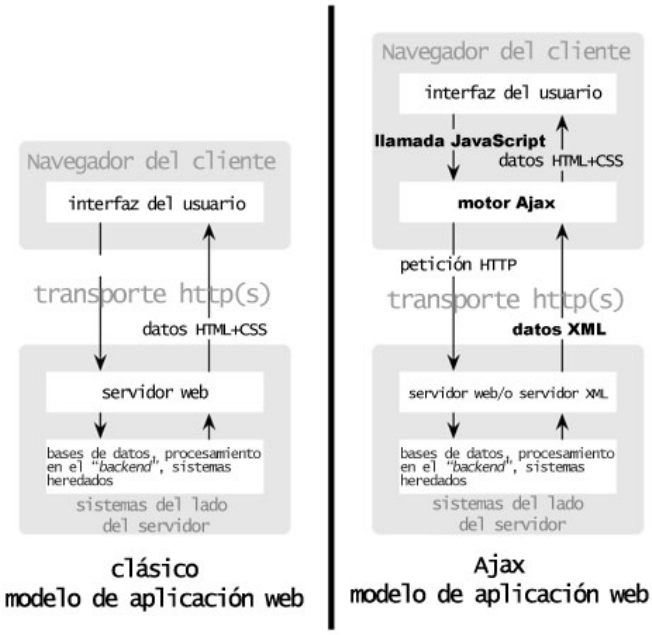


Imagen 5.7 Modelo clásico y modelo usando tecnologías AJAX

En el modelo clásico de las aplicaciones web las acciones de los usuarios disparan eventos a través de una interfaz de usuario quien construye un requerimiento HTTP y lo envía al servidor. El servidor recibe el requerimiento, procesa la petición y retorna el resultado al usuario en forma de una página web. Como se puede observar si la aplicación realiza múltiples peticiones HTTP se vuelve bastante incómoda y lenta desde la perspectiva del usuario.

Con AJAX evitamos la recarga de la página completa cada vez que se atiende una petición HTTP, en vez de eso el intercambio de peticiones con el servidor se realiza en un segundo plano y el usuario apenas si nota este intercambio de información. Esto se logra al agregar una capa intermedia entre el usuario y el servidor que será la encargada de gestionar todas

las peticiones mejorando así la respuesta del servidor y por consiguiente dotando de una percepción de alta agilidad al usuario.

En la imagen 5.8 se comparan el comportamiento de las peticiones en un modelo tradicional y en un modelo desarrollado con AJAX. Como podrá observarse la comunicación en AJAX se hace de una manera asíncrona acelerando la percepción de respuesta en el usuario final.

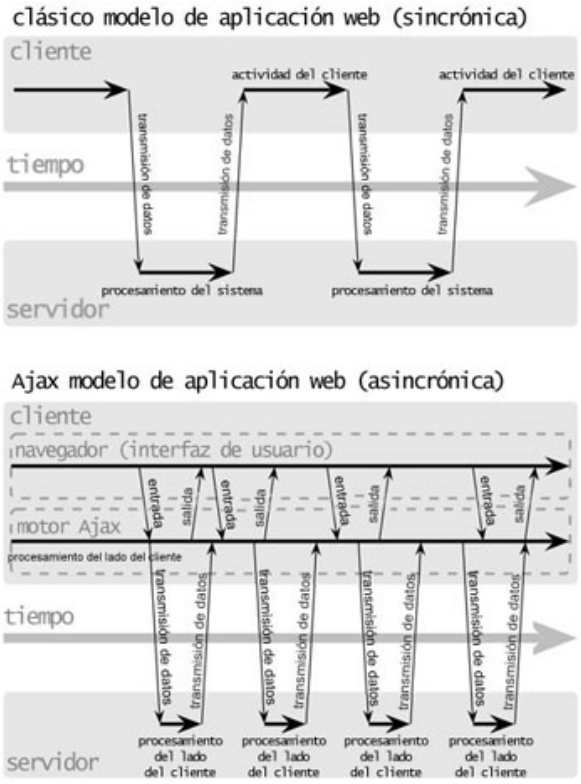


Imagen 5.8 Comunicación en un modelo tradicional y en AJAX.

En el modelo AJAX las peticiones HTTP al servidor se realizan con *Javascript* mediante un objeto encargado del transporte bidireccional de la información y la manipulación del éxito o fracaso de dicha petición. El conjunto de tecnologías que conforman AJAX son:

- XHTML y CSS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- *XMLHttpRequest*, para el intercambio asíncrono de información.
- *Javascript*, para unir todas las demás tecnologías.

La primeras tecnologías ya han sido abordadas en el presente trabajo por lo que a continuación se procederá a dar una breve descripción de lo que son DOM, JSON, *XMLHttpRequest* y *Javascript* las cuales nos permitirán completar una aplicación web en modelo AJAX.

DOM

Document Object Model (DOM) es una API que nos proporciona un conjunto estándar de utilidades para la manipulación de documentos XML y por extensión también se pueden aplicar al XHTML.

La forma de funcionar de DOM consiste en transformar el documento XML en una estructura jerárquica de nodos en forma de árbol que resulte mucho más sencilla de manejar que el propio documento XML original. Esta estructura tiene las ventajas de no sólo representar el contenido del documento original sino que también representa las relaciones entre los elementos del contenido. Una vez construida la estructura es posible interactuar con ella por medio de las utilidades del API, comúnmente a través de algún lenguaje de programación como puede ser *Javascript*. Una desventaja importante a considerar en el

manejo de documentos XML a través de DOM es que al requerir construir una estructura jerárquica es imperativo que se haya cargado la totalidad del documento.

JSON

Por lo común se utilizan dos formas para intercambiar información con un servidor: XML y JSON. XML, como ya se vio, es un lenguaje de marcado y no se procederá a su explicación. *Javascript Object Notation* (JSON) es un formato mucho más compacto que el XML por lo que es mucho más fácil procesarlo en un navegador.

Un objeto en JSON se define por un conjunto de pares "nombre":"valor" encerrado entre unos corchetes. Por ejemplo, para definir a una persona llamada Juan de 22 años y con estudios de primaria y secundaria se podría utilizar el siguiente objeto JSON:

```
{
  'nombre':'Juan',
  'edad':22,
  'estudios':['primaria','secundaria']
}
```

Una vez hecha una petición al servidor de un objeto JSON mediante la propiedad `responseText` del *XMLHttpRequest* éste se retorna en texto plano por lo que es imperativo hacer uso de la función `eval()` que transformará la cadena de texto recibida en un objeto JSON. Una vez transformado ya es posible acceder a los métodos y propiedades del objeto mediante la notación de puntos tradicional de *Javascript*.

XMLHttpRequest

La historia de AJAX está íntimamente relacionada con un objeto de programación llamado *XMLHttpRequest*. El objeto *XMLHttpRequest* es un elemento fundamental para la

comunicación asincrónica con el servidor. Este nos permite enviar y recibir información en cualquier formato y en segundo plano sin que el usuario note el procedimiento.

La forma de crear el objeto XMLHttpRequest varía entre los navegadores, razón por la cual un procedimiento estándar para crear este objeto sería el descrito en el código siguiente:

```
if(window.XMLHttpRequest) {
    peticion_http = new XMLHttpRequest();
} else if(window.ActiveXObject) {
    peticion_http = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Una vez creado el objeto *XMLHttpRequest* podemos manipularlo para realizar nuestras peticiones asíncronas a través de *Javascript*.

Las propiedades definidas para el objeto *XMLHttpRequest* son:

Propiedad	Descripción
readyState	Valor numérico (entero) que almacena el estado de la petición
responseText	El contenido de la respuesta del servidor en forma de cadena de texto
responseXML	El contenido de la respuesta del servidor en formato XML. El objeto devuelto se puede procesar como un objeto DOM
status	El código de estado HTTP devuelto por el servidor (200 para una respuesta correcta, 404 para "No encontrado", 500 para un error de servidor, etc.)
statusText	El código de estado HTTP devuelto por el servidor en forma de cadena de texto: "OK", "Not Found", "Internal Server Error", etc.

Tabla 5.12 Propiedades del XMLHttpRequest.

Los valores definidos para la propiedad `readyState` son los siguientes:

Valor	Descripción

0	No inicializado (objeto creado, pero no se ha invocado el método open)
1	Cargando (objeto creado, pero no se ha invocado el método send)
2	Cargado (se ha invocado el método send, pero el servidor aún no ha respondido)
3	Interactivo (se han recibido algunos datos, aunque no se puede emplear la propiedad responseText)
4	Completo (se han recibido todos los datos de la respuesta del servidor)

Tabla 5.13 Valores del readyState del XMLHttpRequest.

Javascript

Javascript es un lenguaje de programación interpretado y orientado a objetos que es usado principalmente para ejecutar programas en un navegador web del lado del cliente permitiéndonos la creación de páginas web dinámicas.

Javascript fue desarrollado originalmente por Brendan Eich de Netscape con el nombre de Mocha, el cual fue renombrado posteriormente a *LiveScript*, para finalmente quedar como *Javascript*. Su sintaxis es bastante similar a la usada por el lenguaje C aunque adopta los nombres y convenciones del lenguaje Java.

El principal inconveniente que tiene el uso de *Javascript* es que los distintos navegadores tienen distintas formas de implementarlo por lo cual es difícil escribir un código que se ejecute de la misma forma en todos los navegadores posibles. Estas diferencias en el funcionamiento de *Javascript* han marcado tanto a esta tecnología que se han tenido que desarrollar varias alternativas para solucionarlas, así han surgido muchos productos, como los *frameworks Javascript*, que ayudan a realizar funcionalidades avanzadas sin tener que preocuparse en hacer versiones distintas de los scripts para cada uno de los navegadores posibles.

Por comodidad y ya que *Javascript* es demasiado extenso y no es el objetivo de trabajo presente en este documento sólo se proporcionará esta breve descripción y no se ahondará más en el mismo.

5.8.4. JQuery.

JQuery es una librería *Javascript* creada por John Resig. Actualmente es software libre y de código abierto, posee un doble licenciamiento bajo la Licencia MIT y la Licencia Pública General de GNU v2, permitiendo su uso en proyectos libres o privados. Es una librería bastante eficiente y ligera, su versión comprimida pesa 20 KB.

JQuery consiste en un único archivo *Javascript* que contiene las funcionalidades comunes de DOM, eventos, efectos y AJAX. Una vez incluido este archivo se puede empezar a hacer uso de las funciones JQuery sin ningún requerimiento más. La función básica de JQuery es `$()`, a través de ella se pueden seleccionar elementos del DOM para realizar procesamiento con ellos. Este selector es sumamente potente pues entre sus características incluye permitir selecciones a través de Xpath o de selectores CSS. Al llamar a la función `$()` con un selector como argumento, el valor devuelto será otro objeto JQuery representando la colección de elementos DOM seleccionados de donde podemos obtener uno de los elementos seleccionado utilizando el operador `[]`, como si de un arreglo normal se tratara.

Una vez que hemos seleccionado los elementos que deseamos podemos, a través de funciones de JQuery, modificar sus atributos, agregarles una nueva clase o una nueva propiedad CSS, añadirles contenido, eliminarlos, ocultarlos o hacerlos visibles. Toda esta manipulación se verá reflejada en algún cambio visual en la página web que se esté

manipulando. Así mismo JQuery también incluye una vasta cantidad de funciones para responder a eventos determinados o incluso hasta algunas animaciones. JQuery también contiene funciones enfocadas a AJAX, el método principal para realizar la peticiones es `$.ajax()` y a partir de él se despliegan una serie de otras funciones relacionadas, de más alto nivel y especializadas en tareas concretas como son: `$.get()`, `$.post()`, `$.load()`, etc.

Es importante comentar que JQuery no es el único *framework* disponible en el mercado para la manipulación de *Javascript*, pero tiene una gran aceptación por parte de los desarrolladores y un grado de penetración en el mercado muy amplio, lo que lo hace una de las mejores opciones.

5.8.5. JQuery Mobile.

JQuery Mobile es un *framework Javascript* para desarrollo de aplicaciones web enfocadas a dispositivos móviles basado en JQuery. El *framework* nos permite crear aplicaciones web optimizadas para ser desplegadas en dispositivos móviles haciendo que los elementos de éstas puedan ser soportados en un amplio rango de plataformas y dispositivos.

Entre las principales características de JQuery Mobile destacan:

- Es capaz de trabajar con HTML5.
- Tiene una gran automatización de procesos. Por ejemplo, si detecta que se puede hacer una conexión Ajax en lugar de una convencional, la hace automáticamente por esa vía.
- Soporta la gestión de eventos de dispositivos táctiles.

- Permite la personalización de temas.

JQuery Mobile permite construir aplicaciones rápidas, en donde el usuario no tenga largos tiempo de espera, esto se logra al almacenar varias pantallas lógicas en un único archivo HTML e ir mostrándolas y ocultándolas. Para saber la separación entre estas páginas y su estructura se hace uso del atributo `data` aplicado en etiquetas `div` con los valores de `page, header`, `body` y `footer` para acotar la página y definir la cabecera, el cuerpo y el pie. Para navegar de una página a otra, simplemente debemos crear un enlace y establecer el identificador de la página destino en su atributo `href`. A grandes rasgos esta sería la forma de construir una aplicación mediante JQuery y sólo faltaría por decir que para manipular los elementos de la página se seguiría un procedimiento muy similar al del JQuery.

5.8.6. JSP.

Java Server Pages (JSP) es una tecnología Java que permite generar contenido web dinámico en forma de documentos HTML, XML o de otro tipo. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar *scripts* de servidor en sintaxis Java. JSP fue desarrollado por Sun Microsystem y actualmente existen dos especificaciones la JSP 1.2 y la JSP 2.1.

El motor de las páginas JSP está basado en los *servlets* de Java. Al generar un archivo JSP se escribe una estructura de etiquetas HTML que incluye también sentencias de código Java que serán ejecutadas en el servidor. Estos archivos JSP son traducidos por el motor JSP en un *servlet* que será el encargado de generar la página web que se deba mostrar.

La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y por lo tanto posibilita la separación de la lógica de negocio y el acceso a datos de la parte encargada de la presentación de la información pues el archivo JSP puede quedar encargado de sólo generar el código HTML de la página que visualizará el usuario final.

5.8.7. Tiles 2.

Tiles 2 es un *framework* de capa de presentación para las aplicaciones Java EE que permite separar las páginas web en fragmentos reusables. Para usar Tiles 2 es necesario tener instalados el JRE de Java SE 5.0 y un contenedor de *servlets* que soporte Servlet 2.4 y JSP 2.0 o superior. Tiles 2 se encuentra auspiciado por la *Apache Foundation* y se distribuye baja la licencia *Apache Software License Version 2.0*.

Tiles 2 permite a los desarrolladores definir fragmentos de páginas web que pueden ser ensamblado en una página web en tiempo de ejecución. Estos fragmentos ayudan a reducir la duplicidad de elementos comunes en las páginas que se deseen desarrollar o inclusive nos permite reutilizar estos fragmentos dentro de otros fragmentos para crear plantillas reusables. Estas plantillas permiten conservar un mismo estilo de presentación a lo largo de toda una aplicación.

El funcionamiento de Tiles 2 se basa principalmente en 3 componentes: las plantillas, los atributos y las definiciones. Las plantillas son documentos, por ejemplo JSP, que indican la estructura que va a tener la página web, los atributos sirven para hacer referencia en las plantillas a los componentes que se van a reutilizar y las definiciones se usan para

especificar los documentos HTML a los que hacen referencia los atributos, estas definiciones se establecen a través de un archivo de configuración semejante al siguiente:

```
<definition name="myapp.homepage" template="/layouts/classic.jsp">
  <put-attribute name="header" value="/tiles/banner.jsp" />
  <put-attribute name="menu" value="/tiles/common_menu.jsp" />
  <put-attribute name="body" value="/tiles/home_body.jsp" />
  <put-attribute name="footer" value="/tiles/credits.jsp" />
</definition>
```

Tiles se puede integrar con los *frameworks* Struts 1, Struts 2 y Shale así como con las tecnologías de vista Free Marker y Velocity. Lo que lo convierte en una herramienta ampliamente versátil con bastante aceptación en el mundo del desarrollo de aplicaciones web

5.9. Servidor de aplicaciones.

Un servidor de aplicaciones es básicamente un *software* que se encarga de servir de intermediario entre el usuario y las operaciones de capa de negocio que realiza una aplicación de un sistema distribuido. También suele proveer servicios de seguridad, acceso a datos, transacciones, manejo de carga y gestión de recursos a dichas aplicaciones.

La diferencia entre un servidor web y un servidor de aplicaciones se encuentra en que mientras el servidor web únicamente está encargado de responder peticiones HTTP el servidor de aplicaciones es capaz de proveer servicios de lógica de negocio a través de varios protocolos sin excluir el HTTP.

Con el auge de Java EE se han desarrollado varias alterativas de servidores de aplicación que permiten la utilización de sistemas web con este estándar, el cual establece una serie de condiciones para considerar un servidor de aplicaciones apto para desplegar aplicaciones de

esta índole, por lo tanto se puede decir que un servidor de aplicaciones Java EE es en realidad una implementación de la especificación Java EE.

Existen diversos servidores de aplicaciones Java EE en el mercado, entre los principales se encuentran:

- GlassFish
- JBoss Application Server
- IBM WebSphere
- Oracle WebLogic
- Oracle BPM

El concepto de servidor de aplicaciones no está restringido únicamente a tecnologías Java EE, por ejemplo Microsoft tiene el *Internet Information Server* para su plataforma .Net y, adicionalmente, se pueden encontrar servidores de aplicación de código abierto y comerciales de otros proveedores; algunos ejemplos son Base4 Server y Zope.

Un concepto que debe quedar claro es que no todas las aplicaciones web necesitan un servidor de aplicaciones para funcionar. Puede haber sistemas como, por ejemplo, una que acceda a una base de datos no muy compleja y sin grandes requerimientos de seguridad y disponibilidad, que únicamente necesite un servidor web mediante el uso de *servlets* y JSP.

6. Desarrollo.

6.1. Herramientas de desarrollo.

Si bien es cierto que es posible escribir un programa computacional en un editor de texto simple, hacerlo de esta forma es extremadamente rústico y poco práctico, pues actualmente existe una gran variedad de herramientas que nos facilitan la explotación de las funcionalidades que proveen las nuevas tecnologías y, consecuentemente, disminuyen el tiempo de desarrollo de una manera considerable. Por otro lado, la construcción de aplicaciones se ha visto altamente beneficiado por una creciente aparición de *frameworks* y APIs que permiten la reutilización de código y proporcionan, a su vez, implementaciones robustas de las funcionalidades informáticas más comunes en el desarrollo *software*. Por lo tanto se hace patente que para poder efectuar un desarrollo adecuado de una aplicación es importante recurrir a una correcta integración de todas aquellas tecnologías que se encuentren disponibles y permitan realizar un desarrollo óptimo en el menor tiempo posible.

A continuación se presentan las herramientas y *software* empleados para llevar a cabo el desarrollo de la arquitectura presentada.

6.1.1. IDE Eclipse.

Eclipse es un IDE (*Integrated Development Environment*) que incluye un área de trabajo y un sistema de *plugins* que lo hacen una potente, robusta y completa herramienta de desarrollo. Está escrito principalmente en Java y mediante extensiones puede soportar desarrollos en lenguajes como C, C++, PHP y *Groovy*. El proyecto de Eclipse consta de 3 principales áreas:

- **Proyecto Eclipse:** Es responsable de desarrollar el IDE (la plataforma para integrar las demás herramientas), las herramientas de desarrollo para Java (JDT) y el ambiente de *plugins* (PDE) que extienden las funcionalidades de la plataforma.
- **Proyecto de herramientas**: Se enfoca en la creación de extensiones para aumentar el soporte y la funcionalidad de la plataforma.
- **Proyecto de tecnología:** tiene como objetivo la búsqueda de nuevas tecnologías, su incubación y su enseñanza mediante la plataforma Eclipse.

Entre las plataformas más usadas durante el desarrollo de aplicaciones web se encuentran la plataforma de herramientas web (WTP) que cuenta con soporte para aplicaciones construidas con Java EE, incluye editores gráficos y de código para una variedad de lenguajes, APIs para el soporte de publicación, ejecución y prueba de aplicaciones, entre otras características; y la plataforma de servidor con soporte para Tomcat, Glassfish, etc., destacando la opción de *debug* que permite al programador rastrear variables paso a paso en la ejecución de una aplicación.

6.1.2. JBoss Tools.

JBoss Tools es un conjunto de *plugins* para Eclipse, entre estos *plugins* cabe destacar a Hibernate Tools que brinda el soporte para Hibernate.

Hibernate Tools es una colección de herramientas para Hibernate 3. Proporciona la posibilidad de realizar ingeniería inversa, generación de código, visualización e integración con Hibernate. Se necesita la versión de JBoos Tools 3.5.1.

La siguiente tabla muestra las principales características de Hibernate Tools.

Característica	Beneficio
Asistentes para la creación y generación de código.	Un conjunto de asistentes para crear archivos de configuración `cfg.xml`, archivos de mapeo y `reveng.xml`. El asistente ayuda a la generación de código y a realizar una ingeniería inversa de todo un esquema de base de datos.
Editores de archivos de configuración y de mapeo.	Soporta autocompletar y resaltar la sintaxis. Los editores también soportan autocompletar semántica para nombre de clases y propiedades.
Herramientas para organizar y controlar la ingeniería inversa.	El asistente de generación de código ofrece potentes funcionalidades para generar clases, archivos de mapeo y el editor del archivo `reveng.xml` proporciona control sobre estos proceso.
Consola de Hibernate.	Se trata de una nueva perspectiva en Eclipse que proporciona una visión general de las configuraciones de la consola de Hibernate, donde se puede visualizar las clases persistentes y sus relaciones. Permite ejecutar consultas HQL para visualizar directamente en Eclipse.
Editor de HQL y Hibenrate Criteria.	Editores en donde se puede escribir, editar y ejecutar consultas HQL y Criteria.

Tabla 6.1 Principales características de Hibernate Tools.

6.1.3. Ant.

Apache Ant es una herramienta de línea de comandos que tiene como fin el manejo de los procesos especificados en un archivo de configuración. Los procesos se dividen en objetivos llamados *targets* y puntos de extensión que dependen unos de otros. El uso principal de Ant es la construcción de aplicaciones Java. Ant proporciona una serie de tareas que permiten compilar, ensamblar, probar y ejecutar aplicaciones Java.

Eclipse integra funcionalidades de Ant que permiten crear, importar, exportar, configurar y ejecutar tareas usando su área de trabajo. Cualquier configuración de Ant que se haya

creado puede ser utilizada como parte de la construcción de un proyecto y se ejecutará en el orden especificado.

6.1.4. Log4j.

Log4j es una herramienta libre, desarrollada en el proyecto *Jakarta* de Apache, diseñada para registrar eventos de interés en una aplicación. La importancia de contar con una herramienta de este tipo reside en la pronta localización de los problemas que surjan tanto en el desarrollo como en el ambiente de producción del sistema. Con log4j es posible activar el registro de eventos en tiempo de ejecución sin la necesidad de modificar el código fuente. Log4j fue creado con tres metas principales: confiabilidad, velocidad y flexibilidad.

6.1.5. SVN.

SVN es una herramienta de control de versiones con la misión de permitir a los desarrolladores trabajar en un mismo proyecto de forma ordenada y óptima. Eclipse cuenta con un *plugin* de desarrollo llamado Mylyn, que le proporciona al IDE una perspectiva para manejar los repositorios permitiendo comparar, subir, descargar y sincronizar los cambios que tenga el desarrollador. Otra herramienta que permite el manejo de SVN es Subclipse, que además de proveer las características antes mencionadas, añade la posibilidad de revisar las versiones gráficamente entre las distintas ramas.

6.1.6. Apache Tomcat 7.

Apache Tomcat es un servidor libre, contenedor de aplicaciones web basadas en Java. Concretamente sirve para ejecutar aplicaciones desarrolladas con *servlets* y JSPs. Es un servidor estable con las características que cualquier otro contenedor de aplicaciones web comercial posee.

La compatibilidad con la máquina virtual de Java, de la implementación del API de los *servlets* y los JSPs depende de la versión de Tomcat, para este caso (versión 7), se soporta el JDK 1.6, *Servlet* API 3.0 y JSP API 2.2.

Tomcat incluye un manejador vía web al cual se puede acceder en la ruta /manager. Mediante esta aplicación se pueden realizar las acciones básicas de control sobre las aplicaciones que se estén ejecutando en el servidor. Algunas de las funcionalidades que abarca son la instalación, publicación, detención y eliminación de aplicaciones. Mediante este manejador también es posible consultar estadísticas básicas sobre el servidor e información acerca del consumo de memoria por la máquina virtual de Java.

6.1.7. MySQL.

Es un sistema manejador de base de datos (DBMS) relacional rápido y flexible de alto desempeño con soporte para múltiples usuarios. Es uno de los DBMS más populares en las aplicaciones web por ser libre y estar disponible para casi la mayoría de las plataformas. MySQL puede ser ejecutado en sistemas Unix, Windows y Mac. Puede ser utilizado en distintos tipos de aplicaciones, sin embargo, su mayor uso reside en aplicaciones web gracias a su rapidez y seguridad. Entre sus características destacan:

- No requiere recursos especiales en el servidor.
- La línea de comandos es una muy poderosa herramienta con soporte para realizar consultas mediante SQL.
- Soporta indexación y objetos binarios.
- Permite realizar cambios a la estructura de una tabla cuando el servidor está corriendo.
- Se encuentra disponible como un programa independiente según el ambiente en el que se requiera (cliente o servidor).

6.2. Arquitectura.

Habiendo establecido las bases para la construcción de una aplicación robusta en un ámbito web, se plantea como punto de partida una arquitectura MVC, con la siguiente división de capas:

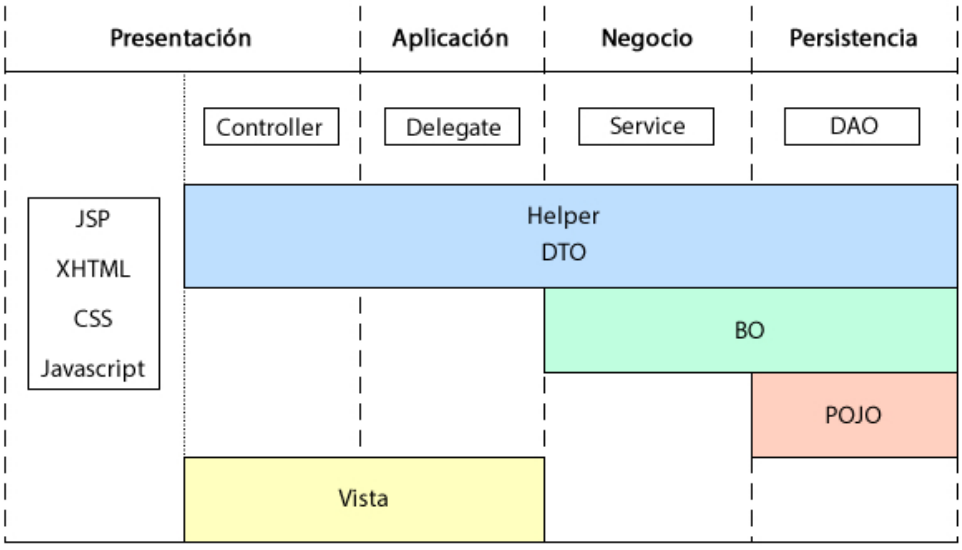


Imagen 6.1 Capas de la arquitectura propuesta.

En la imagen 6.1 se observa que además de los bloques representativos de cada capa se incluyen otros elementos auxiliares con patrones de diseño determinados que permiten complementar el desarrollo y tener bien identificadas las tareas que se realizan en cada área.

Para poder obtener un funcionamiento óptimo entre cada una de las capas presentadas en esta arquitectura se recurre al empleo de tecnologías robustas que tienen una tarea específica a resolver y que en conjunto dan como producto un sistema de alto rendimiento. Se proponen los siguientes *frameworks*:

- Struts 2: Para implementar la capa de Presentación y su acoplamiento con la capa de Aplicación. Es el encargado de supervisar las acciones que el usuario ejecute en el sistema, así como las vistas que deberán presentarse.
- Spring 3: Para desacoplar la dependencia entre los diferentes componentes de la arquitectura mediante la inversión de control y la inyección de dependencias, ganando la reutilización de recursos y asegurando el correcto desempeño del sistema.
- Hibernate 3: Para implementar el modelo relacional de la base de datos y con ello persistir los datos a través de objetos en la aplicación, logrando un mejor desempeño de la base de datos.

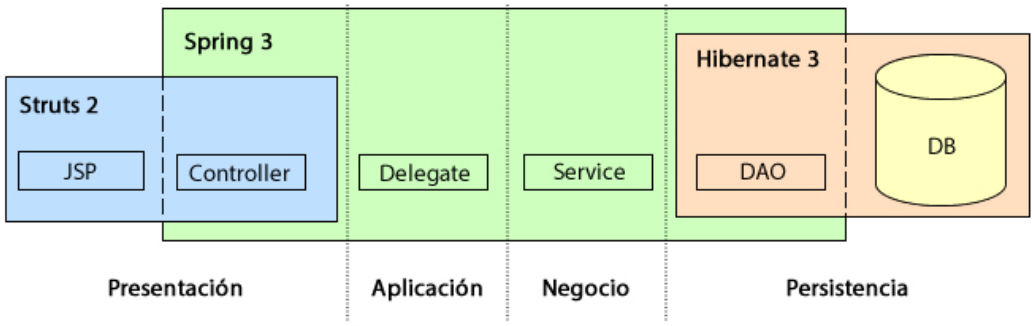


Imagen 6.2 Tecnologías utilizadas en la arquitectura.

6.3. Configuración.

6.3.1. Servidor Tomcat.

Aun cuando la arquitectura es portable entre servidores de aplicaciones, es necesario realizar configuraciones en el servidor para poder iniciarlo, pararlo, establecer valores de arranque y conexiones a fuentes de información.

Los puntos que se muestran a continuación son las configuraciones necesarias para el servidor Tomcat.

- Administrador de aplicaciones.

En muchas ocasiones es útil contar con la capacidad de desplegar una aplicación web o parar alguna que ya exista, sin tener que reiniciar el servidor. Tomcat proporciona un sistema de seguridad en el cual se debe especificar los roles con su correspondiente usuario y contraseña. Los roles que son necesarios para desplegar o parar una aplicación son: `manager-gui`, `manager-script` y `manager-jmx`.

A continuación se muestra la configuración del archivo `tomcat-users.xml`.

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<user password="tomcat" roles="manager-gui,manager-script,manager-
jmx" username="tomcat"/>
```

Con la etiqueta `role` se indica el rol para la administración del servidor y con la etiqueta `user` se establece el nombre de usuario, su contraseña y se le asignan roles en el servidor.

- Perfil de arranque.

Para poder utilizar un perfil de arranque en el servidor web es necesario configurar las variables de ambiente, esto se realiza modificando el archivo `catalina.bat` que se encuentran en el directorio `bin` de Tomcat.

```
set JAVA_OPTS=%JAVA_OPTS% -DSIR_ENV=tomcat/qa %LOGGING_CONFIG%
```

La línea anterior establece una variable que servirá como parámetro de inicio para el proyecto, la variable es identificada por `SIR_ENV` y su correspondiente valor es `tomcat/qa`. Con esta variable podemos contar con una forma de identificar el tipo de arranque que tendrá la aplicación.

- Conexión a base de datos.

Antes de iniciar la configuración de la base de datos se debe colocar la librería de conexión `mysql-connector-java-5.1.21-bin.jar` en el directorio `lib` de Tomcat.

La conexión con la base de datos se realiza mediante un recurso "*Java Naming and Directory Interface*" (*JNDI*). Esta forma de conexión proporciona una mejor administración de conexiones hacia la base de datos, ya que exterioriza la configuración de conexión para

los programadores delegando a los encargados de administrar los servidores dicha configuración. Tomcat no proporciona una interfaz gráfica para administrar los recursos *JNDI*, sin embargo proporciona archivos de configuración para crear el recurso.

El recurso *JNDI* de conexión a la base de datos se define en el archivo `server.xml`. Este XML contiene una etiqueta `GlobalNamingResources` en donde se coloca la conexión con la siguiente estructura.

```
<Resource auth="Container" driverClassName="com.mysql.jdbc.Driver"
      factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
      initialSize="10"
      maxActive="120"
      maxIdle="5"
      minIdle="5"
      name="jdbc/restaurante"
      password="restaurante"
      type="javax.sql.DataSource"
      url="jdbc:mysql://localhost:3306/restaurante"
      username="restaurante"
      validationQuery="select 1 from control.client limit 1"/>
```

La etiqueta `Resource` contiene la configuración de un recurso que se pondrá a disposición de la aplicación. Las propiedades para su configuración son las siguientes:

- `auth`: Indica cómo va ser administrado el recurso, debe ser `Container` para que sea el contenedor el que lo maneje.
- `driverClassName`: Indica el nombre completo de la clase Java del Driver JDBC.
- `factory`: Este atributo es requerido y su valor debe ser `org.apache.tomcat.jdbc.pool.DataSourceFactory`.

- `initialSize`: Indica el número de conexiones que se crean cuando el pool es iniciado.
- `maxActive`: Indica el número máximo de conexiones activas que se pueden asignar al mismo tiempo.
- `maxIdle`: Indica el número máximo de conexiones inactivas que se deben mantener.
- `minIdle`: El número mínimo de conexiones inactivas con las que se debe contar.
- `name`: Establece el nombre del recurso.
- `password`: Indica la contraseña de conexión a la base de datos.
- `type`: La clase java que espera la aplicación cuando se realice la búsqueda de este recurso.
- `url`: Indica la URL de conexión a la base de datos.
- `username`: Indica el usuario de conexión de la base de datos.
- `validationQuery`: La consulta SQL usada para validar la conexión.

Una vez creado el recurso en el servidor se debe crear un enlace de acceso para el recurso. El enlace de acceso se define en el archivo `context.xml` dentro del directorio `conf` de Tomcat.

```
<ResourceLink global="jdbc/restaurante" name="jdbc/restaurante"
type="javax.sql.DataSource"/>
```

La etiqueta `ResourceLink` agrega un enlace a un recurso definido en el contexto global de *JNDI*. Las propiedades de esta fuente de acceso son:

- `global`: Nombre del enlace correspondiente al recurso.
- `name`: Nombre del enlace en relación al contexto `java:comp/env`.
- `type`: Clase Java que espera la aplicación cuando se realice la búsqueda de este recurso.

6.3.2. Aplicación.

La aplicación consiste en un proyecto de tipo "*Dynamic Web Application*" el cual, para facilitar su distribución, es agrupado en la siguiente estructura:

- src: Este directorio contiene el código fuente, los archivos de configuración de Struts2 y log4j.
- ant: Contiene los archivos de Ant para construir el proyecto.
- WebContent/html: Contiene los recursos externos de la aplicación como hojas de estilo, imágenes y código *Javascritpt*.
- WebContent/jsp: Contiene los archivos JSP de la aplicación.
- WebContent/applicationContext: Este directorio contiene los archivos de la configuración de Spring.
- WebContent/lib: Este directorio contiene las librerías en archivos JAR que serán utilizados por la aplicación.

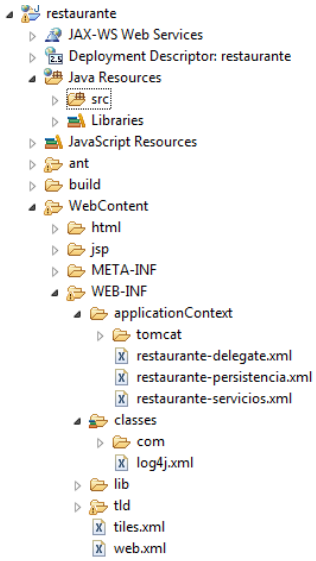


Imagen 6.3 Estructura de la aplicación desarrollada.

6.3.2.1. Descriptor de despliegue.

El archivo `web.xml` se conoce como descriptor de despliegue. Este descriptor especifica varios parámetros de configuración para la aplicación, en los que se encuentran los siguientes:

- Parámetros de contexto: Los parámetros de contexto que son necesarios para la aplicación son: localización de archivos que configuran a Spring, log4j y tiles.
- Clases oyentes: Las clases oyentes que se configuran en la aplicación se ejecutan una vez que se despliegue o detenga la aplicación, estas clases *listener* proporcionan la configuración del `ApplicationContext` de Spring, un proceso que se encarga de limpiar el cache del JDK al momento de finalizar la aplicación y por último especificar el manejo de Tiles.

- Filtro de Struts 2: La definición de este filtro le permite a Struts 2 el manejo de la petición.
- Página de bienvenida: Esta configuración es útil para ocultar los archivos estáticos o archivos JSP del WAR debido a que el usuario que visite la URL de la aplicación automáticamente abrirá la página de inicio.
- Controladores de error: Por medio de esta declaración se puede personalizar lo que se le presentara al usuario en caso de error. En general se contemplan dos tipos de errores: 404 (Error de página no encontrada) y 500 (Error de solicitud del servidor).

6.3.2.2. Integración de Hibernate y Spring.

Una aplicación convencional donde se utiliza Hibernate necesita tener acceso a su base de datos y a las entidades de mapeo. El punto de acceso para Hibernate es la `SessionFactory`, de la cual podemos obtener un objeto `Session`. Spring proporciona una `SessionFactory` que ofrece algunas opciones adicionales para configurar los recursos de una aplicación.

La configuración de la `SessionFactory` se realiza con la clase `LocalSessionFactoryBean`, la cual contiene una propiedad llamada `dataSource` a la que se le puede inyectar un *bean* de la clase `JndiObjectFactoryBean` que se encarga de buscar el recurso *JNDI*.

A continuación se muestra la configuración de la clase `JndiObjectFactoryBean` que representa la fuente de datos.

```
<bean id="dataSource"
class="org.springframework.jndi.JndiObjectFactoryBean">
```

```
        <property name="jndiName"
        value="java:comp/env/jdbc/restaurante"></property>
        <property name="resourceRef" value="true" />
    </bean>
```

La siguiente configuración importante es la creación de la `SessionFactory` a la cual se le debe inyectar además de la propiedad `dataSource` lo siguiente:

- `hibernateProperties`: Son las propiedades de configuración de Hibernate.
    - `hibernate.dialect`: El dialecto de la base de datos para los querys que Hibernate debe de utilizar.
    - `hibernate.show_sql`: Indica la salida de la consulta SQL para un log o la consola.
    - `hibernate.format_sql`: Habilita el registro en la consola de todas las sentencias SQL.
    - `hibernate.max_fetch_depth`: Establece la profundidad máxima de asociaciones externas cuando los objetos tienen asociaciones con otros objetos mapeados.
    - `hibernate.jdbc.batch_size`: Indica a Hibernate el número de operaciones que deben realizarse en lote.
- `mappingJarLocations`: Esta propiedad define una lista de archivos JAR que contienen los mapeos de las clases persistentes.

La siguiente configuración muestra la creación del *bean* `SessionFactory`.

```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
	<property name="dataSource">
		<ref bean="dataSource"/>
	</property>
	<property name="hibernateProperties">
		<props>
			<prop
			key="hibernate.dialect">org.hibernate.dialect.MySQL5Dia
			lect</prop>
			<prop key="hibernate.show_sql">true</prop>
			<prop key="hibernate.format_sql">true</prop>
			<prop key="hibernate.max_fetch_depth">1</prop>
			<prop key="hibernate.jdbc.batch_size">20</prop>
		</props>
	</property>
	<property name="mappingJarLocations">
		<list>
			<value>WEB-INF/lib/restaurante-persistencia.jar</value>
		</list>
	</property>
</bean>
```

6.3.2.3. Hibernate Template.

Spring ofrece un `HibernateTemplate` que permite manejar la mayor parte de las operaciones repetidas por Hibernate. `HibernateTemplate` proporciona una propiedad `SessionFactory` la cual configuramos en la sección anterior y se le inyecta a la definición del *bean* `HibernateTemplate` como se muestra a continuación.

```
	<bean id="hibernateTemplate"
		class="org.springframework.orm.hibernate3.HibernateTemplate">
		 <property name="sessionFactory">
		 <ref bean="sessionFactory"/>
		 </property>
	</bean>
```

6.3.2.4. Transacciones declarativas.

Spring proporciona a través de la programación orientada a aspectos (AOP) el manejo de transacciones declarativas. Esta opción es la más usada debido a que reduce la generación de código de la aplicación, es decir, el programador no tiene que iniciar transacciones ni generar un *commit* o un *rollback*.

La forma habitual de configurar el manejo de transacciones declarativas es como se indica a continuación:

1) Definir un manejador de transacciones para Hibernate. Esto se lleva a cabo con la declaración del *bean* `TransactionManager` al que además se le debe inyectar el *bean* de la `SessionFactory`.
2) Configurar un proxy para envolver el objeto del cual se requiera gestionar las transacciones. Esta configuración necesita declarar un *bean* `TransactionProxyFactoryBean` al cual se le debe definir una referencia al *bean* `TransactionManager`.
3) Finalmente, se necesita crear un *bean* que tenga las propiedades de `TransactionProxyFactoryBean`. Esto se realiza con el atributo `parent` y la correspondiente referencia al *bean* `TransactionProxyFactoryBean`. Este *bean* requiere establecer el objeto al cual se quiera manejar las transacciones a través del atributo `target` y los atributos de transacción, `transactionAttributes`. Estos atributos contienen la semántica de la transacción así como los métodos donde se aplica.

La siguiente configuración establece el *bean* del manejador de transacciones de Hibernate y el *bean* `TransactionProxyFactoryBean` que envuelve el objeto al que se necesita manejar las transacciones.

```
<bean id="transactionManager"
     class="org.springframework.orm.hibernate3.HibernateTransactio
nManager">
      <property name="sessionFactory">
```

```
            <ref bean="sessionFactory"/>
        </property>
</bean>


<bean id="abstractTxDefinition"
class="org.springframework.transaction.interceptor.TransactionProx
yFactoryBean" lazy-init="true">
      <property name="transactionManager">
            <ref bean="transactionManager"/>
        </property>
</bean>
```

El siguiente ejemplo muestra cómo establecer el objeto al cual se le requiere gestionar las transacciones:

```
<bean id="loginDelegate" parent="abstractTxDefinition">
      <property name="target">
            <bean
class="com.restaurante.delegate.impl.LoginDelegateImpl">
                  <property name="controlSesionesService">
                        <ref bean="controlSesionesService" />
                  </property>
            </bean>
      </property>
      <property name="transactionAttributes">
            <props>
                  <prop key="valida*">PROPAGATION_SUPPORTS</prop>
                  <prop key="consulta*">PROPAGATION_SUPPORTS</prop>
                  <prop key="guarda*">PROPAGATION_REQUIRED</prop>
            </props>
      </property>
</bean>
```

6.3.2.5. Definición de Tiles.

La creación de una plantilla consiste en crear la página principal con la definición de los fragmentos del código, esta definición se realiza dentro de la etiqueta `tiles-definitions`. El tile principal contiene la siguiente estructura.

```
<definition name="baseLayout" template="/jsp/base/baseLayout.jsp">
      <put-attribute name="body" value="" />
</definition>
```

Esta definición indica que el archivo `baseLayout.jsp` es la plantilla de trabajo que a su vez contiene la declaración de la inserción del fragmento con el *taglib* `<tiles:insertAttribute name="body" />`. Una vez teniendo la plantilla `baseLayout` solo resta definir el tile que tenga las características de la plantilla e insertarle el fragmento de código como se muestra a continuación.

```
<definition name="/inicio.tiles" extends="baseLayout">
      <put-attribute name="body">
      /jsp/control/index.jsp
      </put-attribute>
</definition>
```

6.3.2.6. Configuración de Struts 2.

La configuración de Struts2 se realiza por medio del archivo `struts.xml`, este archivo contiene la definición de las constantes para el desarrollo y también cuenta con la agrupación de los *actions*.

La agrupación de los *actions* está diseñada de tal forma que separe las funcionalidades de cada negocio así como la tecnología JSON, es decir, se cuenta con la agrupación de *actions* para el negocio del administrador, la aplicación móvil y un grupo que permite manejar JSON.

Cada grupo de *actions* se definen dentro de la etiqueta `package`, la cual contiene un nombre de agrupación y permite que se herede la configuración de uno o más grupos, los grupos de *actions* para la aplicación deben heredar de `struts-default` así como de `json-default` que sirven para las funcionalidades de Struts2 y el manejo de JSON respectivamente.

Dentro de cada grupo de *actions* existen tres elementos importantes:

- *Action*: Es la definición del nombre, la clase Java y el método que se debe ejecutar. Dentro de este *action* se crean los *result* de cada *action*, es importante mencionar que el *result* puede ser de varios tipos: jsp, tile, json e incluso un tipo personalizado.
- Resultados globales: Los resultados globales son útiles para ser aplicados a todo un grupo de *actions*, por ejemplo en una aplicación que contiene seguridad cuando un cliente intenta acceder a ella fuera del *login* correspondiente muchos de los *actions* deben retornar a la página de autentificación.
- Elementos de resultado: Definen el tipo de resultado. En este trabajo se utilizaron los de tipo tile, JSON y uno personalizado para recuperar las imágenes guardadas en la base de datos.

La siguiente configuración detalla los puntos mencionados anteriormente:

```
<package name="default" extends="struts-default">
      <result-types>
          <result-type name="tiles"
          class="org.apache.struts2.views.tiles.TilesResult">
          </result-type>
      </result-types>
      <global-results>
           <result name="errorInside" type="tiles">
                 /errorInside.tiles
           /result>
      </global-results>
      <action name="login"
      class="com.restaurante.controller.LoginController"
      method="execute">
           <result name="success" type="tiles">
                 /inicio.tiles
           </result>
           <result name="error">/</result>
      </action>
</package>
```

6.3.2.7. Configuración de Log4j.

La configuración del registro de los eventos para la arquitectura se define por medio de un XML que contiene todas las características de los *logs*. Se consideran cuatro clases de eventos importantes para ser manejados en una bitácora, dos de estas clases tienen una categorización del nivel error y corresponden a los *frameworks* Hibernate y Spring. La definición de cada uno de estos dos *logs* se realiza creando un *appender* por cada uno de los *frameworks* mencionados en donde se le indica el nombre, el tipo de *log* (en este caso es un *log* que se rola periódicamente), el nombre del archivo, el formato de la fecha para rolar el archivo (se le indica que sea diariamente) y el formato que tendrá al momento de registrar los eventos. La siguiente definición es para el *log* de Spring.

```
<appender name="spring"
class="org.apache.log4j.DailyRollingFileAppender">
      <param name="File" value="../logs/spring.log" />
     <param name="DatePattern" value=".yyyy-MM-dd" />
     <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern" value="[%d{dd-MM-yy
HH:mm:ss}] [%5p] (%60.60C:%L) -> %m%n" />
     </layout>
</appender>
```

Para el *log* de Hibernate se cuenta con la misma estructura a excepción del nombre del archivo. Las siguientes clases de evento es la información de la aplicación que se requiere mantener en bitácora, para lo cual se necesita crear un *appender* con similares características a los dos anteriores pero con el nombre `restaurante.log`.

Finalmente se cuenta con el *appender* dedicado a la consola del servidor el cual es el *root* del log4j y funge como bitácora para los errores de la aplicación que son lanzados con el nivel fatal. A continuación se indica la configuración del *appender*.

```
<appender name="console" class="org.apache.log4j.ConsoleAppender">
      <param name="Target" value="System.out" />
     <param name="Threshold" value="fatal"/>
     <layout class="org.apache.log4j.PatternLayout">
           <param name="ConversionPattern" value="[%d{dd-MM-yy
           HH:mm:ss}] [%5p] (%60.60C -> %m%n" />
     </layout>
</appender>
```

El siguiente punto de la configuración del log4j es la definición de los elementos *loggers*. Cada elemento *logger* requiere definir las siguientes características: el paquete al cual se necesita registrar los eventos, el nivel a partir del cual se podrá registrar y la referencia de su *appender* correspondiente. La configuración que a continuación se muestra define el *logger* para Spring, sin embargo la definición del *logger* para Hibernate y para la aplicación contiene características similares donde el único cambio radica en el paquete, nivel y referencia de *appender*.

```
<logger name="org.springframework">
      <level value="error" />
      <appender-ref ref="spring" />
</logger>
```

El *logger* para la consola del servidor se define indicando que se trata de elemento *root* del *log* como se indica a continuación:

```
<root>
      <priority value="fatal" />
      <appender-ref ref="console" />
</root>
```

6.3.2.8. Clase de inicio.

La clase `SIRContextLoaderListener` se ejecuta una vez que se haya hecho el despliegue de la aplicación y contiene las características que a continuación se muestran:

- Hereda de la clase `ContextLoaderListener` el cual es un requisito debido a que esta clase servirá para la configuración de Spring.
- Sobrescribe dos métodos `contextInitialized` y `contextDestroyed`.
- Los procesos que debe realizar el método `contextInitialized` son tres: recuperar la variable de ambiente `SIR_ENV`, inicializar el `ApplicationContext` y configurar el log4j.
- El método `contextDestroyed` debe proporcionar los procesos de destrucción del `ApplicationContext` así como el apagado del log4j.

A continuación se indica la manera en la cual se realizan los procesos mencionados anteriormente.

Configuración.	Descripción.
System.getProperty("SIR_ENV")	Obtiene el valor de SIR_ENV.
Log4jWebConfigurer.initLogging(event.getServletContext());	Spring se encarga de inicializar el log4j.
super.contextInitialized(event);	Se inicializa el ApplicationContext de Spring.
super.contextDestroyed(event);	Destruye el ApplicationContext de Spring.
Log4jWebConfigurer.shutdownLogging(event.getServletContext());	Spring se encarga de apagar el log4j.

Tabla 6.2 Procesos realizados por la clase de inicio.

6.3.2.9. Ant.

El uso de Ant es debido a que se necesitan realizar ciertas tareas repetidamente para manejar el servidor Tomcat. Para este proyecto las tareas son:

a) Iniciar el servidor Tomcat.
b) Detener el servidor Tomcat.
c) Eliminar la publicación del proyecto restaurante.
d) Limpiar los archivos para crear una nueva publicación, lo cual implica borrar los archivos compilados del proyecto, borrar el WAR que se encuentra dentro de Tomcat y por último eliminar los directorios de publicación del proyecto.
e) Crear los directorios para la compilación dentro del servidor.
f) Compilar los recursos de la aplicación para posteriormente crear el WAR `restaurante.war`.
g) Desplegar el proyecto web `restaurante.war`

Para mayor referencia se recomienda consultar el archivo Ant para el proyecto en el apéndice E de este trabajo.

6.4. Sistema web para la administración de un restaurante.

6.4.1. Análisis.

Como es bien sabido, en la construcción de un sistema de software una parte imprescindible para que ésta sea exitosa se finca en el análisis previo del problema que se desea resolver de tal forma que antes de empezar a escribir el código fuente se tenga una plena comprensión de los requisitos que la solución informática propuesta deba solventar. El análisis de los requisitos permite a los desarrolladores especificar la función y rendimiento del sistema de cómputo y, al mismo tiempo, definir las restricciones que deban

cumplirse durante la actividad que se esté automatizando mediante el sistema informático desarrollado.

En el caso presente se desea elaborar una alternativa computacional para realizar el proceso de toma de órdenes en un restaurante. El objetivo buscado es que, a través de un sistema electrónico, la transacción de realizar una orden en un restaurante se agilice y se disminuya la posibilidad de cometer errores.

Con base en observaciones se identificó el comportamiento natural de la transacción ubicándose así los puntos susceptibles de ser automatizados de una manera sencilla y rápida; dichos puntos son los siguientes:

1. Recepción del menú por parte del comensal.
2. Selección y transmisión de los platillos deseados a la cocina.
3. Recepción y liquidación de la cuenta.

Dejando de lado la elaboración, transporte y consumo de los alimentos estos 3 pasos englobarían la totalidad de la transacción estudiada; además sus características naturales los posicionan como candidatos idóneos para un proceso de automatización sencilla ya que, como es fácil darse cuenta, los procesos de elaboración y transporte de alimentos, aunque son susceptibles de ser automatizados, presentan ciertas características intrínsecas que hacen que requieran de un proceso de automatización mucho más complejo pues involucran una alteración directa del mundo físico como es el mero hecho de transportar un alimento de un lugar a otro. Siendo el objetivo de este trabajo elaborar rápidamente un sistema funcional se optó por dejar de lado estos puntos.

Una vez que se han definido los procesos a automatizar se identificaron los principales requerimientos que esta tríada de puntos presentan.

6.4.1.1. Principales requerimientos funcionales.

- Identificación de comensales.

Se debe proporcionar una forma de identificación de comensales que permita reconocer qué orden le corresponde a cada comensal así como determinar qué comensales componen una sesión. Esto con el objetivo de poder calcular los totales tanto de forma individual como de forma global de la sesión. Cabe mencionar que la sesión se compone de todos los comensales que se encuentren asignados a la misma mesa y que, por ende, se infiere que desean consumir sus alimentos de una manera grupal.

- Asignación de sesión.

Se debe desarrollar un método de asignación de comensales a una sesión. Por razones de seguridad se le debe proporcionar al comensal una clave que identifique inequívocamente a la sesión a la que se quiera integrar. Para reducir las posibilidades de descubrir la clave a través de un ataque de fuerza bruta la sesión sólo debe permitir el registro de comensales durante 10 minutos acto seguido, aunque la sesión permanecerá activa, no permitirá registrar a ningún comensal más, siendo indispensable establecer un mecanismo para volver a permitir el registro de comensales durante otros 10 minutos más previo consentimiento de los comensales interesados. Sólo se debe permitir que un comensal se encuentre asignado a una sola sesión a la vez.

- Presentación del menú.

Se debe desarrollar una forma de desplegar el menú ante el comensal que sea ordenada permitiendo que éste sea capaz de seleccionar rápidamente los platillos que desee ordenar y su cantidad. Con este objetivo se definió una estructura jerárquica en el menú compuesta por tres elementos: secciones, subsecciones y platillos. Las subsecciones sólo pueden contener platillos y las secciones pueden contener tanto platillos como subsecciones. Los platillos necesariamente deben encontrarse contenidos en una sección o subsección. También se debe proporcionar una manera de controlar los platillos que se le muestren al cliente, es decir, que el restaurante tenga la posibilidad de ocultar algún platillo en caso de que por algún motivo éste no pueda ser preparado en un momento dado evitando así que el cliente pueda emitir una orden del platillo. El platillo debe contar con la suficiente información como para permitirle al cliente realizar su elección, entre esta información se incluye una descripción, el precio y una imagen del mismo.

- Compilación e identificación de órdenes.

Se debe procurar un mecanismo que permita centralizar todas las órdenes para posteriormente permitir su preparación. Esta información centralizada debe posibilitar la identificación del comensal que emitió la orden, su mesa y la cantidad pedida. Recopilar esta información facilitará implementar después un sistema para la distribución de los platillos y su elaboración.

- Liquidación de la cuenta.

Se deber suministrar una forma para que se dé por terminada la sesión de los comensales y puedan liquidar la cuenta por los servicios requeridos. La facultad de terminar una sesión sólo se le concede al personal del restaurante con la finalidad de que la sesión sólo sea acabada después de que los clientes hayan saldado la cuenta generada. Por parte de los clientes se les debe permitir en todo momento conocer el total de los servicios consumidos ya sea en su forma individual o en su forma global, así mismo se le debe permitir abandonar el sistema de gestión de órdenes sin que esto signifique la terminación de la sesión si el cliente no ha liquidado lo consumido.

6.4.1.2. Principales requerimientos no funcionales.

- Acceso universal.

Se debe garantizar que un alto espectro de usuarios tenga acceso a la herramienta, es decir, que sea lo bastante ligera como para no requerir condiciones estrafalarias para su implantación. Se recomienda que la herramienta esté enfocada a una implementación en dispositivos móviles de gama media o baja.

- Apariencia.

Se busca que la presentación del sistema tenga una forma elegante y ordenada evitando saturar al usuario de información y ayudándole a ejecutar las funciones del sistema de la forma más sencilla posible. Se debe procurar que la presentación ayude al cliente a hacer su elección proporcionándole toda la información necesaria para conocer el platillo de su interés.

- Concurrencia.

Se debe garantizar la funcionalidad del sistema en todo momento, aún ante la presencia de elevada concurrencia. El sistema debe soportar una alta demanda sin presentar retrasos, congelamientos o caídas totales. En caso de caída del sistema se debe de contar con un respaldo lo suficientemente robusto como para garantizar la nula perdida de información procesada.

- Costo.

Se pretende que el costo de la inversión para poner en marcha el sistema sea lo más bajo posible con el objetivo de hacerlo atractivo para restaurantes de gama media que representarían el nicho ideal para la utilización del sistema pues su público objetivo cuenta con la solvencia económica como para soportar la adquisición de dispositivos móviles que permitan el desplegado del sistema propuesto.

- Usabilidad.

Se debe procurar que el manejo de la aplicación sea sencillo y práctico. Una aplicación sencilla reducirá la curva de aprendizaje al usuario y permitirá una rápida implantación del sistema en un entorno comercial. El sistema debe ser fácilmente adaptable a cualquier dispositivo móvil con capacidad para ejecutarlo.

6.4.1.3. Sistema propuesto.

Partiendo de los requerimientos previamente comentados se puede proyectar una solución informática cimentada en un entorno web. El utilizar un entorno web proporciona la

flexibilidad para adaptar la solución propuesta a una extensa gama de infraestructuras de comunicación inalámbrica existentes evitando preocuparse por el establecimiento de la comunicación entre los dispositivos participantes.

Para garantizar cierta universalidad en los dispositivos clientes se ha decidido desplegar el sistema web mediante alguno de sus navegadores evitándose así la dificultad que presentaría el adaptar una versión de la herramienta por cada uno de los diferentes sistemas operativos que existen.

De lado del comensal se proyecta una aplicación web que permita identificar al usuario mediante un nombre de usuario y contraseña. El usuario debidamente identificado debe indicar la clave de la sesión a la que desea anexarse y su mesa asignada tras lo cual se le debe mostrar el menú y podrá empezar a emitir órdenes, ver los totales acumulados hasta el momento o cerrar su sesión de trabajo. La navegación se hará a través de pantallas de opciones o formularios. Las acciones que tiene permitido ejercer el comensal son: ver el menú, conocer algún total, emitir orden o cerrar sesión. Con la finalidad de no inundar de información la pantalla del dispositivo cliente la presentación del menú del restaurante se hará en forma jerarquizada mostrando primero las secciones del menú y posteriormente, según la sección que se haya seleccionado, los platillos y subsecciones; de igual forma se procederá con las subsecciones sólo desplegando los platillos de la subsección que se haya seleccionado. Durante la visualización del menú se debe procurar sólo desplegar la información indispensable (nombre y precio del platillo) pero siempre debe existir una forma de que el comensal pueda pedir ampliar la información con una descripción del platillo o su imagen. Antes de que el comensal envíe su orden al servidor central se le mostrará su orden permitiéndole verificarla y corregirla si no está conforme con ella. La información trasmitida por parte del dispositivo del comensal sólo será sus datos de

identificación y su orden. Las órdenes deberán ser ingresadas en la base de datos actualizando el mismo tiempo los totales a los que conciernan

Por parte del administrador del restaurante se prevé proveerle de una herramienta que le permita diseñar el menú del restaurante, administrar el menú del restaurante, crear nuevas sesiones, administrar las mesas disponibles y asignar o desasignar mesas a la sesiones. Por comodidad y facilidad de desarrollo se plantea elaborar la herramienta mediante una aplicación web que pueda ser desplegada en cualquier navegador disponible. Los usuarios de esta herramienta se dividen en dos grupos uno denominado administrador general que va a tener la capacidad de ejercer todas las actividades que se han comentado más la capacidad de crear nuevos usuarios de la herramienta y otro denominado administrador normal que carece de la capacidad de generar nuevos usuarios y crear o modificar menús. Esta distinción entre usuarios tiene el objetivo de que un administrador normal, por ejemplo un *hostess*, pueda asignar mesas a comensales sin tener el poder de administrar el menú pues dicha capacidad sólo le correspondería a un gerente debido a que conlleva mayor responsabilidad afectando directamente a los servicios ofrecidos por el restaurante. El acceso a la herramienta se hará por medio de una identificación de nombre de usuario y contraseña permitiendo así iniciar la aplicación con las opciones habilitadas según el tipo de usuario que haya ingresado. La aplicación constará de 5 pantallas: inicio, administración, menús, mesas y reportes. La pantalla de inicio será la pantalla principal y en ella el usuario será capaz de administrar las sesiones; a través de esta pantalla se genera la clave de una sesión, se activa la sesión y se asignan las mesas desocupadas a la misma. También es en esta pantalla donde el administrador puede reactivar el tiempo de registro de comensales en una sesión o finalizarla antes de proceder al cobro de los servicios proporcionados. La pantalla de administración sólo se mostrará cuando el usuario sea un

administrador general y contendrá las opciones que permitan la administración de usuarios, la creación o modificación de menús y la creación de nuevas mesas. La pantalla de menús contiene las opciones para activar o desactivar platillos del menú actual o alternar entre los menús disponibles del restaurante. La pantalla de mesas permite activar o desactivar mesas, esto con la finalidad de poder reservar mesas que no se muestren como mesas disponibles para su asignación a alguna sesión. Por último se provee una pantalla de reportes que permite la visualización gráfica de tendencias de consumo. La presentación de las pantallas se tiene pensada para que se despliegue en forma de pestañas y su edición se realice mediante formularios web.

La solución propuesta brinda una solución a todos los requisitos funcionales planteados anteriormente aparte de resolver los requisitos no funcionales de universalidad y costo pues se puede apreciar que la implementación de un sistema con estas características presenta un costo bajo y es sumamente accesible en su infraestructura usada. Del requisito de concurrencia se encarga la arquitectura utilizada ya que como se ha mencionado los *frameworks* utilizados proporcionan esta característica por si mismos. Los demás requisitos no funcionales se resolverán al realizar la implementación del sistema pues se refieren eminentemente a la presentación visual de la aplicación.

6.4.1.4. Modelado de BD.

Determinar la forma en que se implementará la base de datos es un proceso crucial en el desarrollo de cualquier sistema informático. Para realizar este modelado comúnmente se recurre a un diagrama entidad-relación (E-R) donde se muestran los objetos reales que componen los datos de la aplicación, los atributos de éstos y cómo se relacionan entre sí.

En nuestro caso el modelo se ha divido en dos secciones principales. La primera sección contiene toda la información del menú y en la segunda sección se concentran los datos referentes a la sesiones de los comensales. La región destinada al menú muestra cómo se organizan los platillos del menú en secciones y subsecciones así mismo se provee de una forma de catalogarlos con la finalidad de que una búsqueda entre todo el universo de platillos disponibles se agilice reduciendo la cantidad de platillos en los que se deba buscar. La región destina a la sesión muestra principalmente cómo se constituye la orden del comensal de tal forma que se pueda determinar rápidamente a que sesión pertenece la orden y qué comensal fue quien la emitió.

El diagrama E-R de la base de datos utilizada en la implementación de la solución computacional propuesta se puede observar en el apéndice D del presente trabajo.

6.4.1.5. Módulos funcionales. (Casos de uso)

A través del estudio de la aplicación propuesta se puede dividir la funcionalidad de la herramienta en 7 módulos de la siguiente forma:

- **Login.** Abarca todas las funciones que permiten que un usuario se identifique y acceda al sistema o a una funcionalidad específica que requiera de una identificación del usuario.
- **Administración de usuarios.** Este módulo se refiere a todas las operaciones que tienen relación con el registro de usuarios en el sistema. Este registro de usuarios es el que se usará para la identificación de los mismos.

- **Administración de recursos.** Este módulo abarca las funciones usadas para la administración de los recursos materiales del restaurante. En el diseño actual del sistema sólo se tienen contempladas las mesas como recursos materiales.
- **Administración de menús.** En este apartado se engloban todas las funciones que permiten la creación, modificación y activación de los menús y los platillos contenidos en estos.
- **Administración de órdenes.** Este módulo se refiere a los métodos que permiten la interacción del comensal con el sistema. Abarca desde la creación de la sesión hasta la recopilación de las órdenes en el servidor de base de datos.
- **Presentación.** En este módulo se encuentran contenidas la forma de desplegar el menú en el dispositivo del comensal y la forma en que éste selecciona los platillos que desea ordenar.
- **Reportes.** Las funciones relacionadas a la generación y presentación de los reportes pertenecen a este módulo.

Una vez descompuesto el sistema a desarrollar en sus módulos funcionales se pueden determinar los casos de uso ce componen cada módulo. Un caso de uso hace referencia a la forma en que se comporta una funcionalidad básica del sistema y la manera en que el desarrollador debe realizar su implementación.

A continuación se presentan los casos de uso identificados en el sistema propuesto y a qué módulos pertenecen:

Módulo de Login.

1. Login de administrador general.
2. Login de administrador.
3. Login de cliente.
4. Cierre de sesión
5. Activar registro de clientes.

Módulo de administración de usuarios.

1. Alta de administrador.
2. Eliminación de usuario.
3. Alta de cliente.

Módulo de administración de recursos.

1. Alta de nueva mesa.
2. Activar mesa.
3. Desactivar mesa.
4. Asignar mesa a sesión
5. Desasignar mesa a sesión

Módulo de administración de menús.

1. Alta de menú
2. Edición de menú.
3. Alta de sección.
4. Alta de subsección
5. Alta de platillo.
6. Edición de platillo.
7. Edición de sección.
8. Edición de subsección.
9. Activar menú.
10. Activar platillo.
11. Desactivar platillo.

Módulo de presentación.

1. Desplegado del menú.
2. Selección de platillos.

Módulo de administración de órdenes.

1. Creación de la sesión.
2. Creación de la orden.
3. Cálculo de total individual.
4. Cálculo de total de la sesión.
5. Procesado de orden.

Módulo de reportes.

1. Presentación de reporte de órdenes.
2. Presentación de reportes de ingresos

En el apéndice B y C del trabajo presente se proporcionan los casos de usos desarrollados para la aplicación y los diagramas que muestran cómo se relacionan.

6.4.2. Persistencia.

Una vez que se tiene el diseño y la base de datos de la aplicación se procede a comenzar con el desarrollo de la capa de persistencia. Primeramente se crea un proyecto nuevo para tener una separación de la aplicación del restaurante con las clases y archivos de configuración de la persistencia. El proyecto es de tipo genérico, es decir, un proyecto Java SE con la inclusión de las librerías hibernate3 y mysql-connector; la primera para poder utilizar el *framework* de Hibernate 3 y la segunda para poder lograr la conexión de Java con la base de datos MySQL.

También se incluye un archivo de configuración `hibernate.cfg.xml` con la información necesaria para conectar la aplicación con la base de datos en una sesión:

```
<session-factory>
     <property name="hibernate.connection.driver_class">
           org.gjt.mm.mysql.Driver
     </property>
     <property name="hibernate.connection.password">
           restaurante
     </property>
     <property name="hibernate.connection.username">
           restaurante
     </property>
     <property name="hibernate.connection.url">
           jdbc:mysql://localhost:3306/restaurante
     </property>
     <property name="hibernate.dialect">
           org.hibernate.dialect.MySQL5Dialect
     </property>
</session-factory>
```

En este archivo se describe el conector que se utilizará para realizar la conexión, el nombre de usuario, contraseña, dirección y dialecto que se empleará para poder realizar las consultas a la base de datos.

Hasta este momento, el proyecto tiene la siguiente estructura:

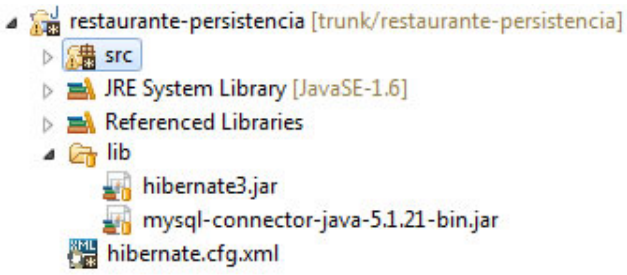


Imagen 6.4 Estructura del proyecto de persistencia.

El siguiente paso es la generación de los archivos de mapeo (`.hbm.xml`) y las clases que representarán a las tablas de las bases de datos (POJO). Como inicio se debe crear el archivo de configuración `hibernate-reveng.cfg.xml` que contendrá las tablas de la base de datos de las cuales se requiere realizar el mapeo y la conversión de los tipos de datos de MySQL a objetos de Java:

```
<hibernate-reverse-engineering>
      <type-mapping>
            <sql-type
                  jdbc-type="VARCHAR" hibernate-type="java.lang.String">
            </sql-type>
            <sql-type
                  jdbc-type="INTEGER" hibernate-type="java.lang.Integer">
            </sql-type>
            <sql-type
                  jdbc-type="TIMESTAMP" hibernate-type="java.util.Date">
            </sql-type>
            .
            .
            .
      </type-mapping>
      <table-filter match-catalog="restaurante" match-name="catalogos"/>
      <table-filter match-catalog="restaurante"
            match-name="catalogos_opciones"/>
```

```
      <table-filter match-catalog="restaurante" match-name="menu"/>
      <table-filter match-catalog="restaurante"
            match-name="menu_has_seccion"/>
      <table-filter match-catalog="restaurante" match-name="mesa"/>
      .
      .
      .
</hibernate-reverse-engineering>
```

Posteriormente con estos dos archivos de configuración se procede a generar los archivos de configuración de Hibernate para realizar los mapeos y los POJOs dentro de la carpeta `src` del proyecto mediante la herramienta de Hibernate Tools previamente instalada en el Eclipse como se indica en la imagen 6.5.

Los archivos hbm son archivos de configuración que describen las características de la tabla que se está mapeando y cómo serán representados en una clase de java, incluyendo sus relaciones con otras tablas. Por ejemplo, el archivo hbm de la tabla `Menu` es el siguiente:

```
<hibernate-mapping>
     <class name="com.restaurante.persistencia.Menu" table="menu"
     catalog="restaurante">
```

En la línea inicial se describe la tabla que se mapea y la clase en Java que representa a dicha tabla. El siguiente bloque indica las columnas de la tabla y la forma en que pasarán a ser un atributo de un objeto de java. En caso de ser una llave primaria o compuesta, también se indica:

```
<id name="id" type="java.lang.Integer">
     <column name="id" />
     <generator class="native" />
</id>
<property name="nombre" type="java.lang.String">
     <column name="nombre" length="100" not-null="true"/>
</property>
<property name="descripcion" type="java.lang.String">
     <column name="descripcion" length="65535" />
</property>
<property name="estado" type="java.lang.Boolean">
     <column name="estado" not-null="true" />
```

```
</property>
```

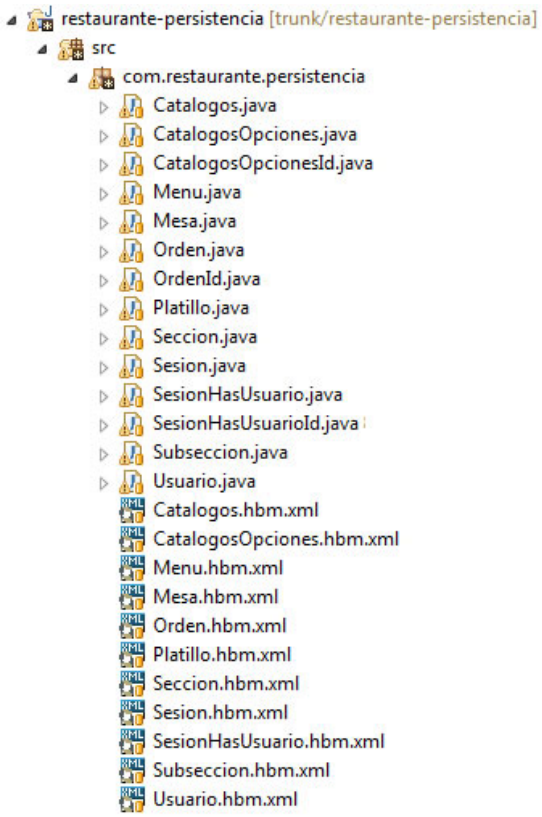


Imagen 6.5 Archivos .hbm.xml y POJOs

Por último se indican las relaciones que la tabla tiene con otras entidades, en el caso de `Menu`, tiene una relación de muchos a muchos con la tabla `Seccion` por lo que es necesaria la entidad `menu_has_seccion` para que se rompa esta relación. Generalmente cada tabla de la base de datos se mapea a un objeto en Java, sin embargo existen casos como éste donde la tabla intermedia no se mapea como un objeto, únicamente aparece indicada en los archivos hbm de las entidades afectadas (Menu y Seccion) y se le asigna un nombre de relación, en este caso, `secciones`. Dentro de la declaración de esta relación se establecen las llaves mediante las cuales se hace la relación y a qué entidad pertenecen.

```
        <set name="seccions" table="menu_has_seccion" lazy="true"
        fetch="select">
            <key>
                  <column name="id_menu" not-null="true" />
            </key>
            <many-to-many entity-
            name="com.restaurante.persistencia.Seccion">
                  <column name="id_seccion" not-null="true" />
            </many-to-many>
         </set>
      </class>
</hibernate-mapping>
```

Una de las opciones más interesantes que provee Hibernate es la de poder obtener los datos de aquellas entidades que tengan una relación con el registro que se está consultando. Para lograrlo se utilizan las opciones `fetch` y `lazy`. `Lazy` se refiere al "cuándo" y `fetch` al "cómo". Por defecto `lazy` tiene un valor de true, lo que significa que la colección no se recuperará de la base de datos hasta que se haga alguna operación sobre ella. Si establecemos `lazy` a "*false"*, cuando se recupere la información de una entidad también se traerá toda la información de los objetos relacionados. Por otro lado, `fetch` define qué tipo de sentencia SQL se utilizará para recuperar la información. Por defecto `fetch` tiene el valor de "*select*", lo que implica que para recuperar la información de la entidad relacionada se lanzará una nueva consulta a la base de datos. Si establecemos `fetch` a "*join*", en la misma consulta que se recupera la información de la entidad también se recuperará la información de la entidad relacionada mediante un *left outer join* en la sentencia SQL.

Los POJOs son clases que representan una tabla de datos teniendo como atributos las columnas de la tabla y las relaciones que tenga esa tabla con otras entidades representadas como una colección de tipo `set`. Todos los atributos van acompañados con sus respectivos métodos `get` y `set`. También incluyen un constructor vacío, otro que recibe todos los

atributos de la clase y, en caso de que la tabla tenga columnas que pueden ser nulas, otro constructor que recibe como parámetros únicamente aquellos atributos que no pueden ser nulos. Como ejemplo, tenemos el POJO de la clase Menu:

```
public class Menu implements java.io.Serializable {
     private Integer id;
     private String nombre;
     private String descripcion;
     private Boolean estado;
     private Set seccions = new HashSet(0);

     public Menu() {
     }

     public Menu(Integer id, String nombre, Boolean estado) {
           this.id = id;
           this.nombre = nombre;
           this.estado = estado;
     }

     public Menu(Integer id, String nombre, String descripcion,
Boolean estado,
                 Set seccions) {
           this.id = id;
           this.nombre = nombre;
           this.descripcion = descripcion;
           this.estado = estado;
           this.seccions = seccions;
     }

     public Integer getId() {
           return this.id;
     }
     public void setId(Integer id) {
           this.id = id;
     }
     public String getNombre() {
           return this.nombre;
     }
     public void setNombre(String nombre) {
           this.nombre = nombre;
     }
     public String getDescripcion() {
           return this.descripcion;
     }
     public void setDescripcion(String descripcion) {
           this.descripcion = descripcion;
     }
     public Boolean getEstado() {
           return this.estado;
     }
     public void setEstado(Boolean estado) {
           this.estado = estado;
     }
     public Set getSeccions() {
           return this.seccions;
     }
```

```
        public void setSeccions(Set seccions) {
              this.seccions = seccions;
        }
}
```

Dado que se está manejando esta parte de la capa de persistencia por separado, es necesario generar una librería de este proyecto e incluirla dentro del proyecto principal de la aplicación. Para realizar este proceso se recurre a la herramienta Ant, que como ya se ha mencionado, permite automatizar procedimientos en base a objetivos. En este caso, las tareas son:

a) Crear el directorio donde se guardará el jar o en caso de que exista limpiarlo para evitar redundancias.
b) Compilar las clases del proyecto restaurante-persistencia y generar un archivo jar.
c) Eliminar los archivos `class` que se crearon al compilar y el archivo `restaurante-persistencia.jar` existente en el proyecto restaurante.
d) Copiar el archivo jar generado en el proyecto restaurante.

Como consulta se incluye el archivo Ant de este proyecto en el apéndice F del presente trabajo.

6.4.3. Implementación del servidor.

El termino servidor hace referencia a los componentes que involucran a la administración del restaurante. Esto quiere decir que esta implementación es una parte del sistema para poder realizar tareas de gestión. El administrador debe ser capaz de crear/consultar/reactivar/terminar/desplegar una sesión de consumo, adicionar/desasociar una o más mesas a la sesión de consumo así como administrar usuarios, menús, secciones, platillos, mesas y generar reportes de consumo e ingresos ya sean diarios o mensuales.

En primer lugar se tiene la pantalla de acceso al sistema donde los elementos necesarios para ingresar son el nombre de usuario y contraseña del personal encargado de realizar las tareas de gestión.

Imagen 6.6 Pantalla inicial del administrador.

Una vez que el usuario se haya identificado en el sistema se procede a mostrar la pantalla principal, esta pantalla consta de un menú horizontal el cual se muestra a continuación.

Imagen 6.7 Menú de opciones de administración.

Como se puede observar en la imagen anterior el menú horizontal consta de 5 apartados:

- **Inicio**

Esta opción consta de dos pestañas en las cuales se puede realizar la creación de una sesión de consumo así como la modificación de sus mesas y el tiempo de activación. Para la creación de una sesión de consumo basta con generar un código único que fungirá como un código de acceso para el móvil del cliente. También es necesario seleccionar una o más mesas de la lista que no exista asociación alguna hacia otra sesión de consumo. Existe la posibilidad de que el código se repita para varias sesiones de consumo, sin embargo se tiene la regla de solo contar con una sesión con estado activa que pueda tener dicho código.

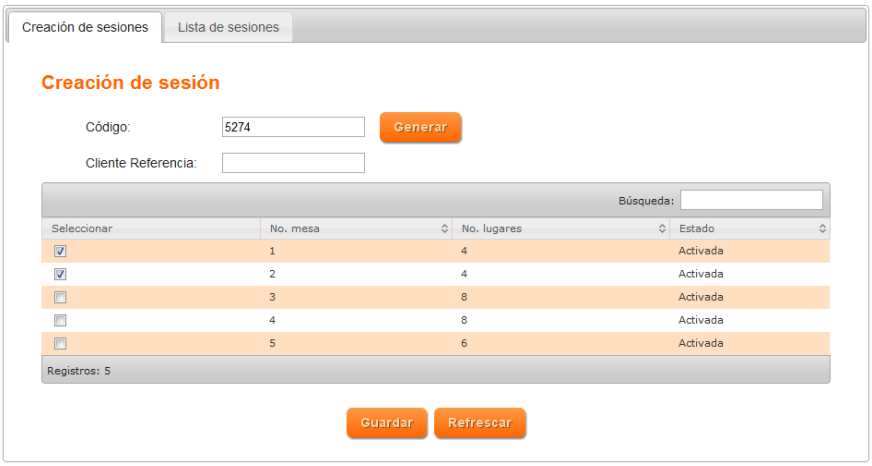


Imagen 6.8 Pantalla para la creación de una sesión.

La segunda pestaña despliega todas las sesiones de consumo que existan con un estado de valida, se muestra la información relevante como el código, referencia, inicio de sesión, mesas asignadas y el total de consumo.

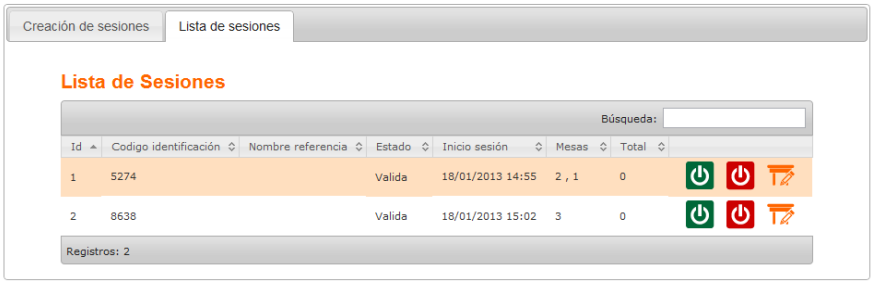


Imagen 6.9 Pantalla de información de sesiones.

Además cuenta con botones para poder activar/reactivar/visualizar una sesión de consumo. El primero de estos realiza una actualización del inicio de sesión haciendo que esta pueda ser utilizada nuevamente por los clientes. El segundo botón realiza la terminación de la sesión liberando las mesas que tenga asignadas y actualizando su estado a inválida lo cual implica que ya no se mostrara en el listado. Por último se encuentra el botón que proporciona la visualización del detalle de la sesión, en esta opción se puede agregar/desasignar una mesa a la sesión con la regla de negocio que no se puede quedar sin mesas, también se puede editar el número de lugares de alguna mesa que tenga asignada.

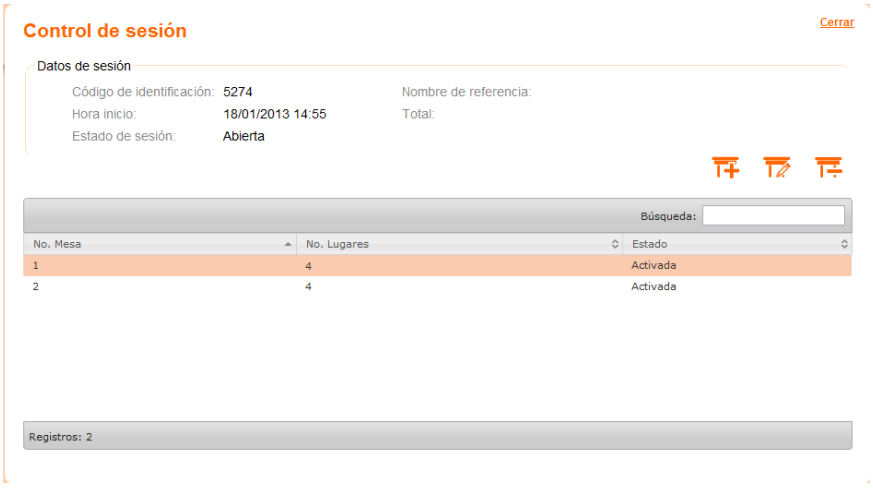


Imagen 6.10 Pantalla para la edición de una sesión.

- **Administración**

La opción de administración únicamente está disponible para los usuarios con un rol de administrador general. Los procesos que se pueden realizar están ligados a un nuevo menú visualmente más atractivo basado en fondos que cambian a medida que el usuario ubica el puntero del mouse en sus diferentes elementos, la elección de este tipo de menú se debe a que es importante proporcionarle al administrador una interfaz de usuario enriquecida. Las opciones que están disponibles son usuarios, menú y mesas.

Imagen 6.11 Pantalla de administración general.

El elemento usuarios contiene el desplegado de todos los usuarios del sistema donde también es posible registrar y eliminar usuarios con los botones localizados en la parte superior.

Imagen 6.12 Pantalla de administración de usuarios.

El registro de usuario requiere proporcionar el usuario, nombre, contraseña y rol al cual pertenecerá el usuario. En caso de que el usuario ya exista se informara lo que suceda.

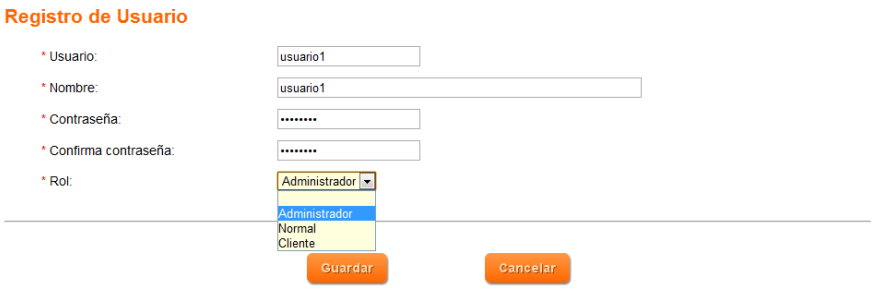


Imagen 6.13 Pantalla de registro de usuario.

En el siguiente apartado menú se encuentra la mayor parte de la administración. Existen cuatro subelementos que se encargan de crear o editar los platillos, las secciones, subsecciones así como visualizar la estructura de los menús.

El primer subelemento es la consulta de platillo, esta consulta proporciona en la parte superior la creación y edición de platillos.

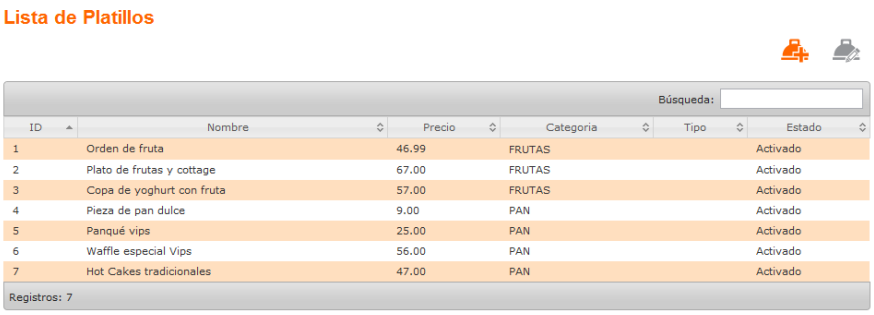


Imagen 6.14 Pantalla de administración de platillos.

El alta y edición de un platillo se realiza por medio de formularios de captura muy similares, la única diferencia radica en la lógica de negocio, es decir, si es alta significa que se trata de un nuevo registro y si es edición solo se actualizan los datos de dicho platillo.

Imagen 6.15 Pantalla de registro de platillo.

El subelemento de secciones, visualiza las secciones que conforman los menús. Existen botones que se encuentran en la parte superior derecha de la consulta para dar de alta y editar una sección.

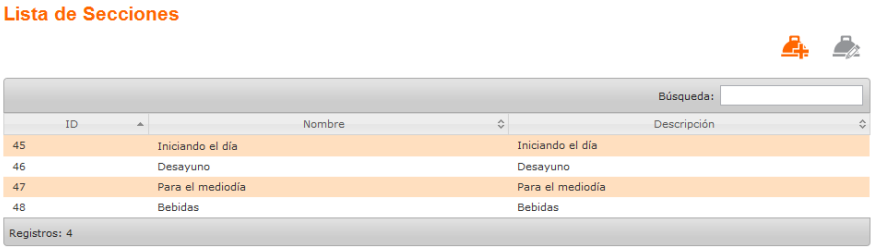


Imagen 6.16 Pantalla de administración de secciones.

La pantalla de alta de una sección cuenta con dos paneles donde el primero de estos se localiza en la parte izquierda y contiene todas las subsecciones excepto las que ya se hayan agregado. El segundo panel contiene las subsecciones agregadas.

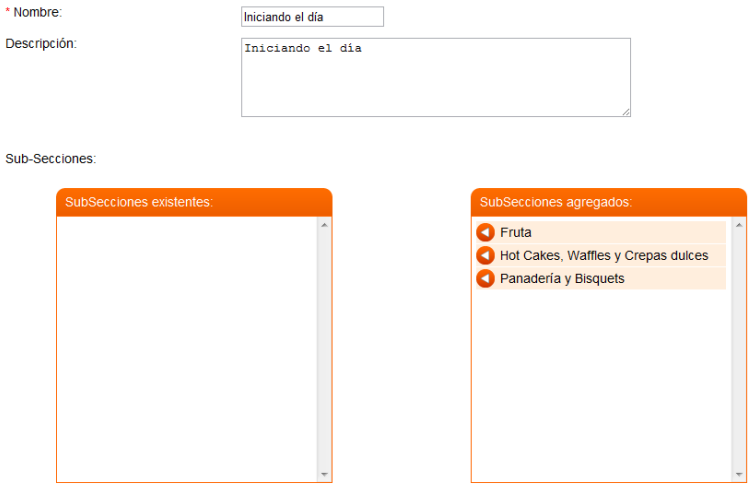


Imagen 6.17 Pantalla de registro de subsecciones en una sección.

La asociación de platillos se lleva a cabo del mismo modo que las subsecciones a excepción que se muestran los platillos por las diferentes categorías existentes. Cabe mencionar que la edición de una sección cuenta con la lógica requerida para agregar/quitar un platillo o subsección que ya se haya realizado, también cuenta con la opción de permitir realizar una

copia de dicha sección. La funcionalidad entre subsecciones y secciones es muy parecida, la única diferencia radica en el hecho de que una subsección solo contiene platillos.

Imagen 6.18 Pantalla de registro de platillos en una sección.

En el subelemento menú se proporciona la consulta de los menús y los procesos de alta, edición y visualización de un menú. En el caso de alta y edición se maneja de la misma forma en la que se da de alta una sección, se cuenta con un panel en el cual se encuentran las secciones existentes y del otro lado se encuentran las secciones que ya están agregadas. La edición cuenta con la opción de poder guardar una copia de un menú.

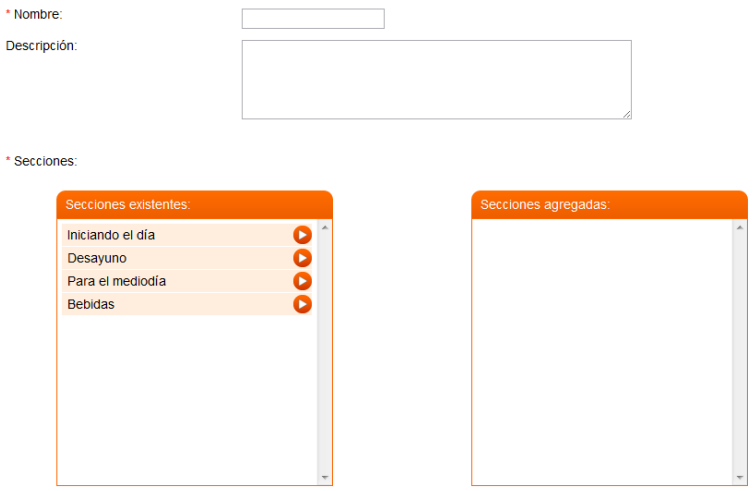


Imagen 6.19 Pantalla de registro de un menú.

La visualización del menú es muy importante debido a que es la vista previa del menú que será mostrado en el teléfono del cliente. Las secciones que conforman el menú se encuentran en la barra de navegación superior y las subsecciones son agrupadas en una configuración parecida a un acordeón, la apertura de una subsección se realiza dando *click* sobre esta.

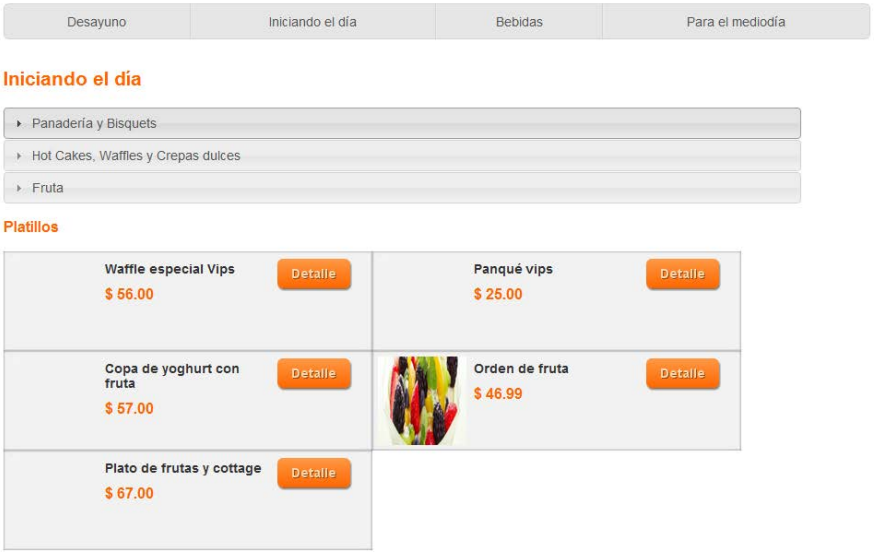


Imagen 6.20 Pantalla de pre-visualización de un menú.

También es posible realizar una visualización de un platillo en específico detallando toda su información haciendo *click* sobre el botón detalle que se encuentra en cada platillo.

Regresando a los elementos del menú de administración la última opción mesas sirve para desplegar, dar de alta y editar las mesas existentes, se cuenta con botones para poder realizar estos procesos.

Imagen 6.21 Pantalla de administración de mesas.

- **Menú**

En el apartado de menú se proporcionan las actividades de activación y desactivación de menús y platillos. En el caso de los platillos, es una forma rápida de eliminar el platillo del menú cuando ya no puede ser preparado por diversas razones, como la falta de ingredientes por ejemplo. Es suficiente con hacer *click* en los botones de activación/desactivación que se encuentran en la última columna de la lista para que ya no sea desplegado o para que se vuelva a mostrar, según sea el caso. También permite realizar una búsqueda para la pronta localización del platillo en el recuadro localizado en la parte superior derecha de la tabla.

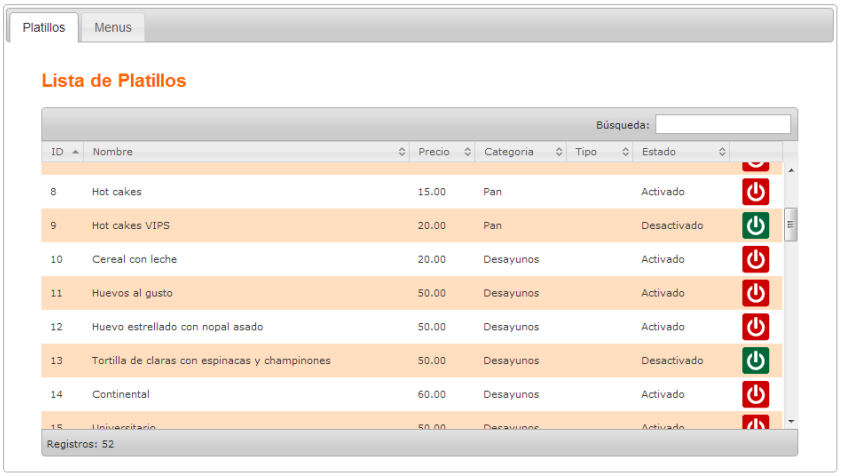


Imagen 6.22 Pantalla de activación de platillos.

Para el menú, a pesar de ser también una pantalla de activación y desactivación, tiene una lógica de negocio distinta por el hecho de que únicamente puede haber un menú activo para que éste sea proporcionado al cliente. En la parte superior de la pestaña se muestran los datos del menú actual y en la parte inferior una tabla con los demás menús disponibles. Al momento en que se haga la activación de otro menú, el actual se desactivará automáticamente. En esta tabla también es posible realizar una búsqueda de los menús existentes para un manejo más ágil de la información.

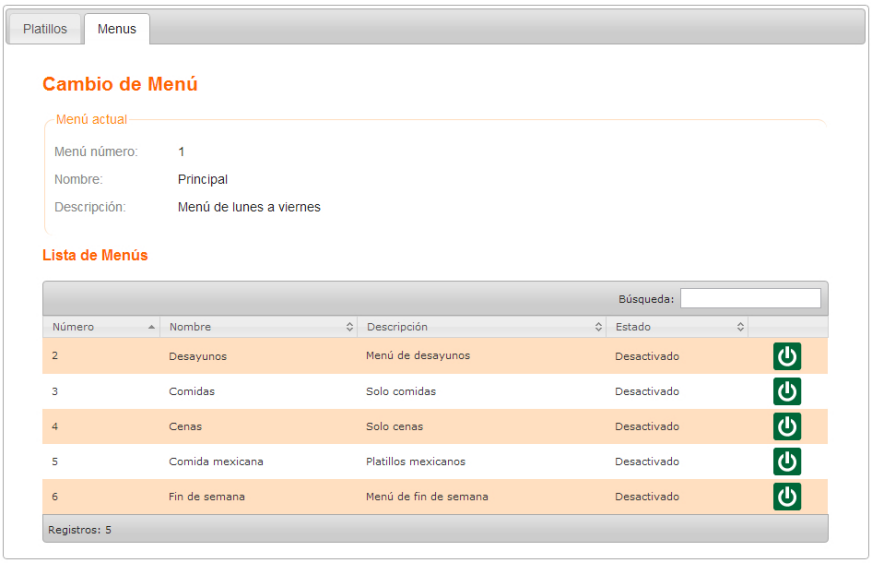


Imagen 6.23 Pantalla de activación de menús.

- **Mesas**

En la opción de mesas únicamente proporciona los procesos de activar o desactivar las mesas que existan en el sistema.

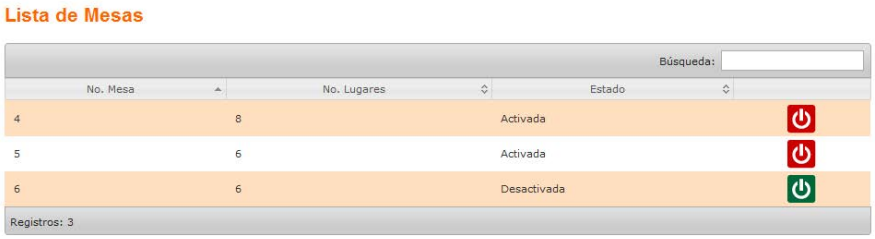


Imagen 6.24 Pantalla de activación de mesas.

- **Reportes**

En la sección de reportes se puede obtener una gráfica de los ingresos diarios o mensuales del consumo por platillo y de los ingresos obtenidos en un periodo de tiempo especificado.

Para los reportes diarios únicamente se requiere especificar el rango de fechas de los cuales se desea obtener el reporte, como se muestra en la siguiente figura:

Imagen 6.25 Pantalla de selección de fechas para un reporte diario.

Cabe destacar que únicamente se mostrarán aquellas fechas en las que se encontraron ingresos, es decir, aquellas fechas en los que el ingreso tiene un valor de 0 no se mostrarán.

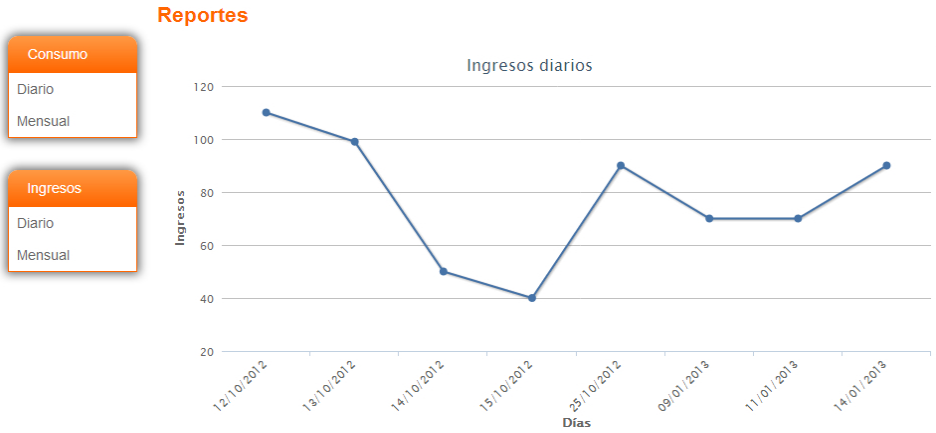


Imagen 6.26 Pantalla de un reporte diario de ingresos.

Así mismo, para obtener un reporte mensual se requiere especificar un periodo de tiempo medido en meses y, en caso de ser un reporte de consumo, es necesario especificar aquellos

platillos de los que se desea obtener el informe. En este tipo de reporte los platillos están agrupados por categorías de acuerdo a como se registraron en la aplicación para tener una mayor facilidad de selección.

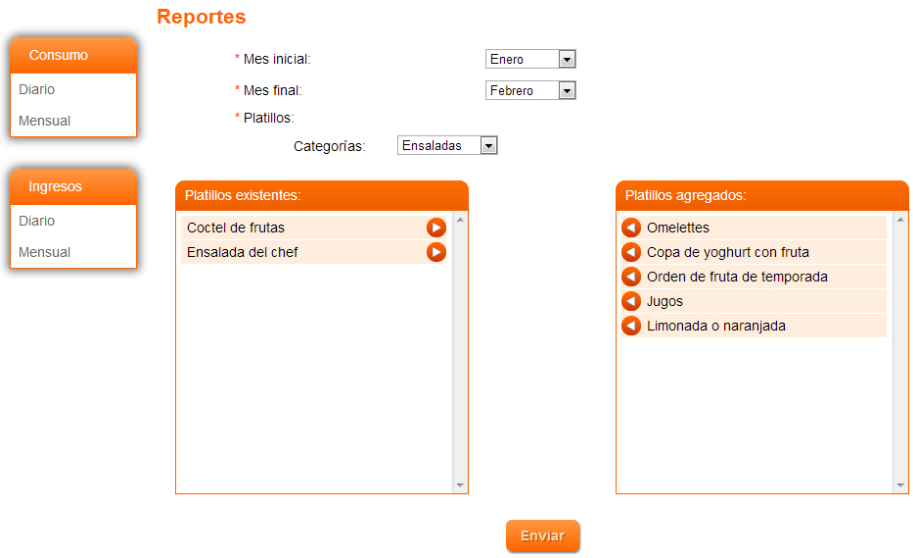


Imagen 6.27 Pantalla de selección de fecha para un reporte de consumo mensual.

Siguiendo con el mismo formato, en un reporte de consumo no se mostrarán aquellos platillos que fueron seleccionados en la petición del reporte que no hayan tenido un consumo en el periodo. Adicionalmente, en este reporte es posible activar y desactivar los platillos para una mejor visualización del mismo únicamente haciendo *click* en la lista de platillos localizada en la parte inferior de la gráfica.

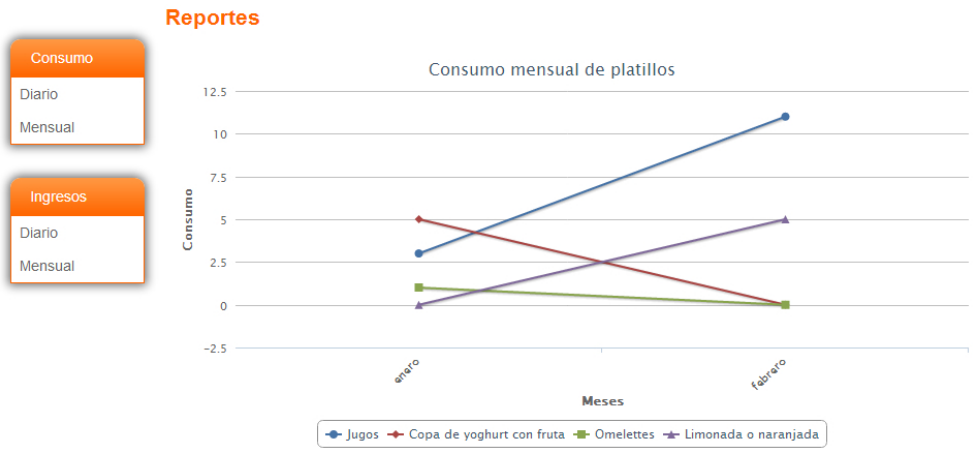


Imagen 6.28 Pantalla de un reporte de consumo mensual.

Por último, la opción "Cerrar Sesión" que se encuentra en la parte superior derecha proporciona la funcionalidad de terminar la sesión del usuario haciendo que la navegación de la administración finalice.

6.4.4. Implementación del cliente.

En esta aplicación, cliente hace referencia a aquella parte del sistema que hace partícipe a los comensales del restaurante. En este sentido, el cliente debe ser capaz de desplegar el menú del restaurante, permitir la selección de platillos, realizar la confirmación de la orden y mostrar el consumo realizado en el mismo dispositivo (consumo individual) y en el de la mesa (consumo global).

Como punto de partida se muestra la pantalla de ingreso al sistema donde se requiere introducir el nombre de usuario y la contraseña con la que el comensal está registrado en el sistema.

Imagen 6.29 Pantalla de inicio para un comensal.

En caso de no contar con una cuenta, éste puede realizar su registro en la opción "Registrarse" localizada en la parte inferior derecha de la pantalla.

Imagen 6.30 Pantalla de registro para un comensal.

Para llevar a cabo dicho proceso, el usuario debe proporcionar información básica de identificación y de acceso al sistema. Una vez concluido el registro, se mostrará de nuevo la pantalla de log-in.

Después de que el usuario se haya identificado en el sistema, éste deberá suscribirse a la mesa que se le asigne seleccionando el número de dicha mesa e introduciendo un código de acceso; esta información debe ser proporcionada por el personal del restaurante al crear la sesión de consumo.

Imagen 6.31 Pantalla de ingreso a una sesión por parte de un comensal.

Con este mecanismo, si el cliente cierra su sesión en su navegador, al volver a ingresar a la aplicación ya no tendrá que identificarse a una mesa ni ingresar otro código, pasará automáticamente a la siguiente pantalla que consiste en una serie de opciones disponibles en las que podrá desplegar el menú y realizar su orden, consultar la cuenta individual o la global y cerrar su sesión.

Imagen 6.32 Pantalla principal de la aplicación para un comensal.

En el caso de la opción “Menú”, la siguiente pantalla mostrará una lista de las secciones en las que se agrupan los platillos disponibles en el menú y en la parte inferior un botón que permite consultar los platillos que se han seleccionado. En caso de que aún no existan platillos seleccionados, la aplicación mostrará un aviso de dicho estado y a continuación se mostrará esta misma pantalla.

Imagen 6.33 Pantalla de visualización de un menú por parte del comensal.

Cuando se selecciona una de las opciones del menú, la siguiente pantalla desplegará una lista de platillos pertenecientes a la categoría seleccionada. En este punto puede que se muestre otro grupo de secciones, una lista de platillos o ambos, todo depende de cómo el restaurante haya organizado el menú disponible.

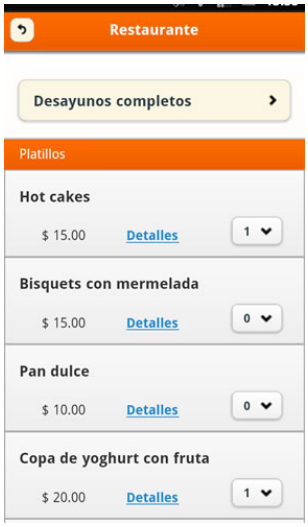


Imagen 6.34 Despliegue de una sección para un comensal.

Cada elemento de la lista muestra el nombre del platillo, su precio, un vínculo con la etiqueta “Detalles” la cual llevará a otra pantalla donde se presenta información concisa y una imagen representativa del platillo seleccionado; y una lista de números consecutivos que indican el número de platillos del mismo tipo que se desea ordenar.

La navegación entre las opciones del menú se realiza mediante los botones localizados en los extremos de la parte superior de la aplicación, teniendo la acción de llevar a la pantalla anterior o bien, a las categorías generales del menú.

Una vez que se haya terminado de seleccionar los platillos deseados se debe proceder a la confirmación de la orden. Este siguiente paso se realiza con el botón “Ver orden” localizado en cualquiera de las pantallas del menú, en la parte inferior.

Imagen 6.35 Pantalla de confirmación de orden.

La pantalla de confirmación de orden muestra un resumen de los platillos seleccionados presentando el nombre del platillo, su precio y la cantidad indicada. En caso de que se quiera seguir seleccionando platillos, o bien, modificar la cantidad de uno de ellos, se puede regresar a la pantalla de categorías generales del menú mediante el botón localizado en la parte superior izquierda de la aplicación. Caso contrario, si la orden está completa, se deberá presionar el botón de “Enviar” para que la orden sea canalizada a la instancia correspondiente del restaurante para proceder con la preparación. La aplicación del cliente sólo mostrará una pantalla de recepción de su orden como dato informativo.

Imagen 6.36 Pantalla que informa la transmisión de una orden.

Si en la pantalla de opciones principales de la aplicación se selecciona la etiquetada con la leyenda “Cuenta individual”, el sistema mostrará una lista de los platillos ya ordenados con la información de: nombre del platillo, precio unitario, cantidad y total. Al final se muestra la suma de los totales individuales, es decir, la suma a pagar. Cabe mencionar que en esta pantalla se muestra la lista de platillos ordenados mediante el dispositivo, no por persona, por lo que para un eficiente funcionamiento del sistema lo ideal es que cada persona ordene con su propio teléfono.

Imagen 6.37 Pantalla de total individual.

En el caso de la opción “Cuenta global”, se muestra una pantalla de las mismas características que la pantalla de “Cuenta individual” con la diferencia de que en esta pantalla se incluyen todos los platillos ordenados por los clientes pertenecientes a la sesión de consumo.

Imagen 6.38 Pantalla de total global.

Por último, la opción de “Cerrar sesión” termina la sesión del usuario con la aplicación y muestra la pantalla inicial del sistema. Esto no implica que la sesión de consumo sea finalizada ya que esta es controlada por el personal del restaurante y únicamente puede ser terminada por el restaurante.

6.4.5. Puesta en marcha del sistema.

La ventaja que una aplicación web ofrece frente a una de escritorio, aún desarrollada con la misma tecnología Java, es que la inversión necesaria en hardware y software desciende considerablemente. Por ejemplo los requerimientos de hardware de las computadoras

personales para administrar el sistema no exigen componentes especiales o de un costo elevado para un buen desempeño en el manejo de la aplicación, únicamente requieren tener soporte para un navegador web. La fluidez del sistema más bien radica en dos aspectos fundamentalmente: la red inalámbrica (configuración y ancho de banda) y la capacidad del servidor.

Respecto al servidor, sus características dependen de la capacidad física que posea el restaurante para colocar a sus clientes ya que se debe considerar el soporte a un acceso simultáneo. El componente que demanda cierta atención es el tamaño del disco duro debido al número de clientes y registros que se generarían, sobre todo si se quiere almacenar la información capturada por periodos largos de tiempo para un análisis profundo en distintos aspectos como el consumo de platillos en las distintas épocas del año y la proliferación o reducción de ganancias, según sea el caso. En el presente trabajo se utilizó como servidor una computadora con procesador Intel Atom a 1.6 GHz con 1GB en memoria RAM y disco duro de 60 GB. En cuanto a software, un sistema operativo Linux en su distribución Ubuntu versión 12.10, JDK 1.6 y Apache Tomcat como contenedor web.

Sobre la red inalámbrica, se plantea la instalación de una red local sin acceso a internet facilitada por el restaurante. El número de puntos de acceso inalámbricos es proporcional al tamaño y tipo de construcción del establecimiento ya que tanto paredes como techo reducen el alcance de los dispositivos. Es importante mencionar que la restricción del sistema para ser visto por internet radica en que la aplicación desarrollada no cuenta con una funcionalidad de esta índole como podría ser la reservación o la orden de platillos por lo que no tiene sentido proporcionar dicho acceso y además contribuye a tener un mejor control sobre los comensales que se encuentren presentes.

La terminal de administración, como se mencionó antes, no requiere de características especiales más que el soporte de un navegador web. La aplicación propuesta se desarrolló para una correcta visualización en los navegadores Google Chrome versión 24 y Mozilla Firefox versión 18.

Finalmente, el último eslabón en esta cadena de dispositivos es el teléfono inteligente proporcionado por el cliente. Dicho dispositivo debe contar con la capacidad para conectarse a una red inalámbrica mediante WiFi y tener un navegador web instalado. En este rubro, las pruebas se realizaron con los teléfonos Sony Xperia Neo, Sony Xperia S, Iphone 4 y Nokia 5530, todos ellos con los navegadores nativos, obteniendo resultados satisfactorios.

6.4.6. Proyección a futuro.

Dentro del ciclo de vida del software se contempla que una vez implementado un sistema se detecten áreas de mejoras para acrecentar la funcionalidad del mismo y solucionar las deficiencias que presente. En este tenor se puede mencionar que el desarrollo realizado en el presente trabajo constituye sólo la punta del iceberg de un sistema de gestión para restaurantes avanzado, esto es debido a que la prioridad del trabajo fue el diseño e implementación de una arquitectura que permitiera el desarrollo eficaz de sistemas de software, razón por la cual se decidió dejar al sistema propuesto con una funcionalidad muy limitada en lo referente a una completa administración de un restaurante. Esta situación hace que se tenga un amplio abanico de posibilidades para incrementar el funcionamiento del sistema propuesto.

Entre las oportunidades de mejora detectadas se pueden enunciar las siguientes:

- Desarrollo de aplicaciones nativas de dispositivos móviles.

En el actual estado del trabajo se ha propuesto un modelo que funciona utilizando los navegadores web para el manejo del sistema. Un paso natural sería el diseño e implementación de aplicaciones nativas de cada plataforma móvil capaces de interactuar con el sistema lo que conllevaría a la ventaja inmediata de poder interactuar directamente con los recursos del dispositivo permitiéndonos transferir parte del procesamiento del lado del servidor al dispositivo mismo.

- Modulo pago electrónico

El sistema se puede ver ampliamente beneficiado de la implementación de formas que permitan al comensal realizar el pago del servicio sin tener necesidad de recurrir a saldar la cuenta en la caja del establecimiento. Esta situación agilizaría la cobranza y permitirá un férreo control de los ingresos del establecimiento.

- Gestión de inventario.

El sistema actualmente puede contener la información de los platillos consumidos y los ingresos recibidos por lo tanto es factible desarrollar un módulo encargado de gestionar los inventarios que sea capaz de determinar cuándo se requiere hacer la compra de ingredientes o incluso realizar el pedido automáticamente. Este módulo también podría ser capaz de dar un seguimiento permanente a la caducidad de los alimentos permitiendo así que el restaurante fuera capaz de asegurar la frescura de sus platillos.

- Análisis de tendencias de consumo.

Un módulo de análisis de tendencias de consumo permitiría al establecimiento diseñar estrategias comerciales que le permitieran mejorar su posicionamiento en el mercado por medio de promociones relacionadas con sus platillos más populares o incluso le permitiría identificar problemas en los casos específicos de los platillos con más baja demanda.

- Personalización de platillos.

El hecho de cambiar el trato con los clientes de un mesero por un dispositivo electrónico no es motivo para perder la personalización en diversos aspectos como lo puede ser la adición o eliminación de ciertos ingredientes en la preparación de los platillos. En este sentido complementar las pantallas con los elementos necesarios para que los comensales personalicen en la medida de lo posible su experiencia en el restaurante ayudaría a conservar esa relación estrecha proporcionada anteriormente por el mesero.

- Publicidad dirigida y recompensa de fidelidad de clientes.

El tener un acceso tan personal con el cliente le permite al restaurante diseñar promociones dirigidas a sectores específicos de sus clientes así como beneficiarse de sistemas de cupones externos tan en boga hoy en día. El sistema incluso permitiría identificar hábitos de clientes para ofrecerle experiencias gastronómicas únicas elaboradas a su medida, por ejemplo si un cliente suele evitar el picante se le puede advertir automáticamente cuando está a punto de pedir un platillo extremadamente picante para que lo tenga en consideración.

- Sistema de reservaciones.

Otro paso natural en un sistema de gestión de restaurantes es el relativo al módulo de reservaciones ya que al tener este proceso automatizado se evitarían errores en las reservaciones que pudieran conducir a una mala imagen del establecimiento por parte del público en general. Este módulo resultaría crucial en restaurantes muy populares que suelen contar con una demanda exagerada.

7. Resultados y conclusiones.

A través de la elaboración del presente trabajo se han podido constatar los beneficios de tener una arquitectura estable, robusta y segura que permita realizar el desarrollo de software de una manera sencilla y homogénea. Como puede apreciarse la arquitectura que se ha propuesto elimina en gran medida la carga que le significa al desarrollador el considerar aspectos como la seguridad, concurrencia, acceso a base de datos, sobrecarga de operaciones en el sistema, etc. Tareas que recaen, ahora, en la implementación de cada uno de los *frameworks* quienes, coordinados, coadyuvan en la consecución de estas finalidades reduciendo el papel del desarrollador a la simple configuración de éstos. El uso de *frameworks* comerciales también trae consigo una ganancia en el soporte pues se encuentran respaldados por empresas serias y son apoyados por amplias comunidades informáticas que se encargan de mantenerlos en constante evolución confiriéndole a la arquitectura, y por ende a los desarrollos elaborados en ella, una gran adaptabilidad ante posibles contingencias imprevistas en el estado actual de la tecnología utilizada. De la misma forma la estandarización de la arquitectura posibilita un desarrollo meticuloso en donde los módulos estén claramente diferenciados proporcionando así un fácil mantenimiento, una rápida detección y corrección de errores e incrementando la capacidad de agregar nuevas funcionalidades; todo ello gracias a que los nombres de las clases y métodos usados son lo bastantemente descriptivos como para focalizar su función y comportamiento en caso de requerir alguna alteración en éstos.

Con lo que respecta al sistema desarrollado, en la parte de la administración de los activos de un restaurante se ha provisto de una herramienta capaz de permitirle a un administrador crear y gestionar los menús, seleccionar los platillos que se mostrarán en el menú,

administrar las mesas disponibles y monitorear de una forma sencilla la asignación de mesas a los clientes así como cerrar la sesión del comensal para proceder al cobro del servicio impartido. También se ha añadido una sección donde se pueden generar reportes gráficos de ingresos y platillos elaborados, con la finalidad principal de proporcionarle al administrador una forma de observar el comportamiento medular del negocio y, con base en ello, poder tomar las determinaciones que considere necesarias en pos del bien del negocio.

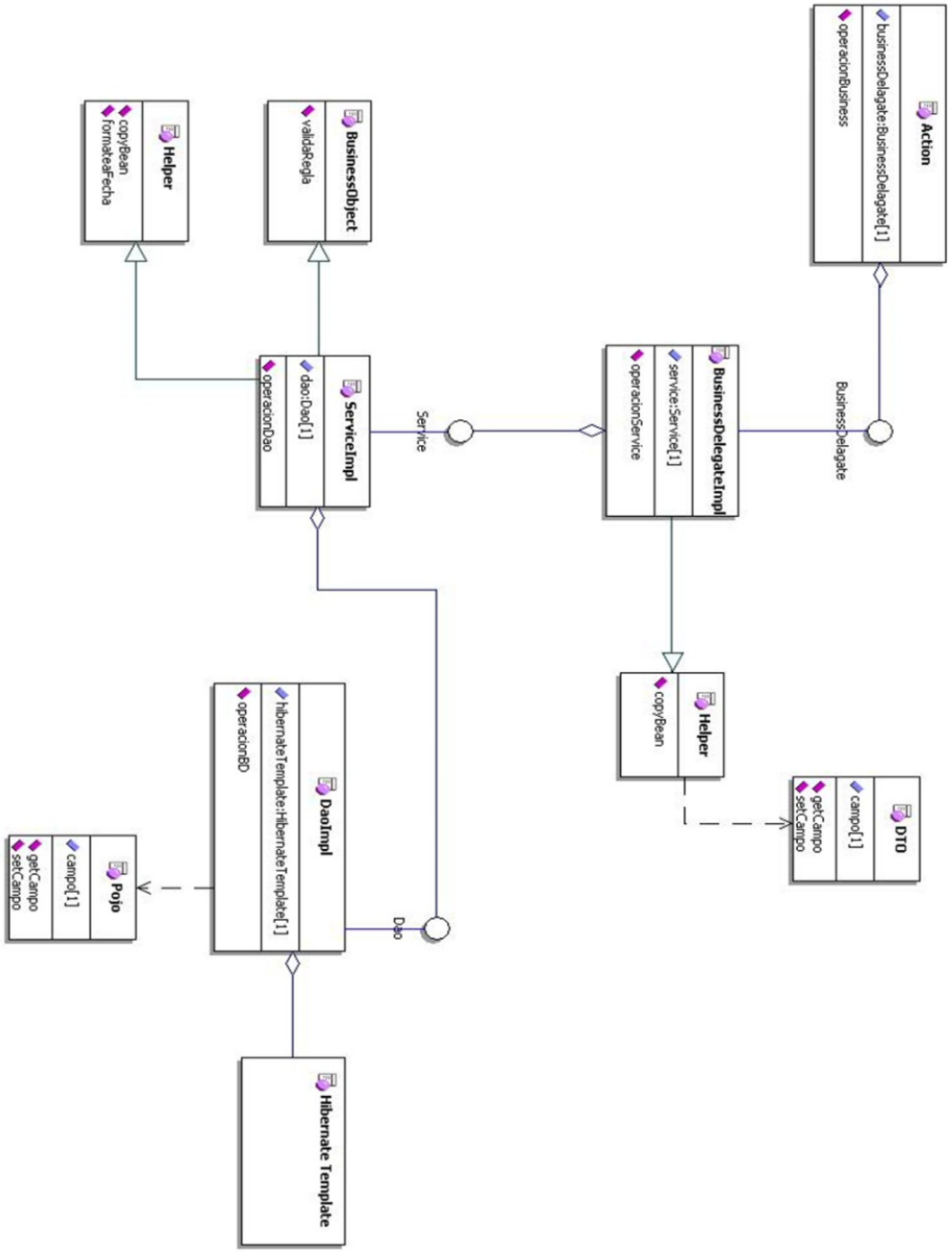
En lo concerniente al comensal se le ha dotado de una interfaz bastante simple que logra desplegar de forma clara los platillos del menú así como sus características más transcendentales, además le permite visualizar en todo momento el total de su cuenta ya sea en forma individual o en forma global, es decir, de todos los comensales asignados a una mesa en particular. Esta interfaz resulta sumamente ligera por lo que su descarga en un dispositivo móvil es lo suficientemente veloz como para no incomodar al usuario además de ayudar a una navegación óptima que no dificulta al usuario la transición entre pantallas. Un aspecto importante de esta interfaz móvil es que es capaz de funcionar sin problemas en la mayoría de los navegadores web de las plataformas móviles más importantes actualmente.

Dados estos resultados se puede concluir que la arquitectura propuesta resuelve con creces muchos de los problemas clásicos de la construcción de software ya que además de agilizar la programación del código fuente permite una separación eficiente entre capas de negocio y una modularización eficaz de los sistemas desarrollados bajo ella. También ha demostrado ser lo suficientemente flexible como para poder desplegarse en la mayoría de las infraestructuras disponibles en el mercado hoy en día. Por su parte el sistema desarrollado demuestra constituir una alternativa viable de bajo costo para la

automatización de la toma de órdenes de comensales en un restaurante de un tamaño considerable beneficiando tanto al lado del restaurante como al de los comensales.

8. Apéndices.

8.1. Apéndice A. Diagrama de clases de la arquitectura propuesta.



8.2. Apéndice B. Casos de usos de la aplicación propuesta.

CASO DE USO: LOGIN DE ADMINISTRADOR GENERAL

1. Descripción

El caso de uso explica el procedimiento a través del cual el administrador general del sistema se identifica para ingresar a él.

1.1. Actores.

- Administrador general.

1.2. Referencias

- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

El caso de uso no cuenta con precondiciones del sistema.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema requiere la introducción del usuario y contraseña del usuario que desea ingresar a él.
2) El usuario introduce su nombre de usuario y su contraseña.
3) El sistema verifica que los datos accedidos sean correctos. **[FA-1][FA-2][FE-1]**
4) El sistema presenta la pantalla de administración con las opciones de administrador general activadas.

3.2. Flujo alternativo.

3.2.1. FA-1 Usuario incorrecto

1) El sistema informa que el usuario ingresado no es correcto.
2) El sistema vuelve a pedir la introducción del usuario y la contraseña.

3.2.2. FA-2 Contraseña incorrecta

1) El sistema informa que la contraseña ingresada no es correcta.
2) El sistema vuelve a pedir la introducción del usuario y contraseña.

3.3. Flujo de excepción.

3.3.1. FE-1 Excepción en validación de datos

1) El sistema informa del error ocurrido durante la validación de datos.
2) El sistema vuelve a pedir la introducción del usuario y la contraseña.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El administrador general tiene una sesión iniciada del sistema donde se le permite ejercer todas las actividades propias de su rol

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: LOGIN DE ADMINISTRADOR

1. Descripción

El caso de uso explica el procedimiento a través del cual el administrador normal del sistema se identifica para ingresar a él.

1.1. Actores.

- Administrador

1.2. Referencias

- CU_Login_De_Admin

2. Precondiciones

2.1. Del sistema.

El caso de uso no cuenta con precondiciones del sistema.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema requiere la introducción del usuario y contraseña del usuario que desea ingresar a él.
2) El usuario introduce su nombre de usuario y su contraseña.
3) El sistema verifica que los datos accedidos sean correctos. [**FA-1][FA-2][FE-1]**
4) El sistema presenta la pantalla de administración.

3.2. Flujo alternativo.

3.2.1. FA-1 Usuario incorrecto

1) El sistema informa que el usuario ingresado no es correcto.
2) El sistema vuelve a pedir la introducción del usuario y la contraseña.

3.2.2. FA-2 Contraseña incorrecta

1) El sistema informa que la contraseña ingresada no es correcta.
2) El sistema vuelve a pedir la introducción del usuario y contraseña.

3.3. Flujo de excepción.

3.3.1. FE-1 Excepción en validación de datos

1) El sistema informa del error ocurrido durante la validación de datos.
2) El sistema vuelve a pedir la introducción del usuario y la contraseña.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El administrador tiene una sesión iniciada del sistema donde se le permite ejercer todas las actividades propias de su rol.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: LOGIN DE CLIENTE

1. Descripción

El caso de uso permite el registro de un cliente un una sesión y su ingreso al sistema para visualizar el menú y hacer órdenes.

1.1. Actores.

- Cliente

1.2. Referencias

- CU_Login_De_Cliente
- CU_Alta_De_Cliente.

2. Precondiciones

2.1. Del sistema.

El caso de uso no cuenta con precondiciones del sistema.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) Se solicita el ingreso del nombre de usuario y contraseña del cliente. **[FA-1] [FA-2] [FA-3] [FE-1]**
2) Se le pide al cliente ingresar su código de seguridad y una de las mesas que tenga asignadas.
3) El cliente ingresa su mesa y su código de seguridad.
4) El sistema verifica que el código de seguridad este activo actualmente. **[FA-4] [FE-1]**
5) El sistema verifica que la mesa ingresada este activada. **[FA-5] [FE-1]**
6) El sistema verifica que la mesa ingresada corresponda a la sesión activa con ese código de seguridad. **[FA-5][FE-1]**
7) El sistema registra al cliente en la sesión indicada.
8) El sistema muestra la interfaz de operación del cliente.

3.2. Flujo alternativo.

3.2.1. FA-1 Registro del cliente como usuario.

1) El cliente selecciona la opción de registrarse.
2) Se ejecuta el punto de extensión [PE-1].

3.2.2. FA-2 Usuario incorrecto

1) El sistema informa que el usuario ingresado no es correcto.
2) El sistema vuelve a pedir la introducción del usuario y la contraseña.

3.2.3. FA-3 Contraseña incorrecta

1) El sistema informa que la contraseña ingresada no es correcta.
2) El sistema vuelve a pedir la introducción del usuario y contraseña.

3.2.4. FA-4 Código de seguridad incorrecto

1) El sistema informa que el código de seguridad es incorrecto.
2) El sistema vuelve a pedir la introducción del código de seguridad y la mesa.

3.2.5. FA-5 Mesa incorrecta

1) El sistema informa que la mesa es incorrecta.
2) El sistema vuelve a pedir la introducción del código de seguridad y la mesa.

3.3. Flujo de excepción.

3.3.1. FE-1 Excepción en el login

1) El sistema informa que se ha producido alguna excepción.
2) El sistema vuelve a requerir el ingreso del código de seguridad y mesa.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente se encuentra asociado a una sesión activa y está posibilitado para ver el menú, hacer órdenes y pedir los totales de la cuenta.

6. Puntos de Extensión

6.1. PE-1 Registro del cliente en el sistema.

- Se ejecuta el caso de uso de alta de cliente desde su flujo alternativo referente al lata de cliente desde un dispositivo móvil.

CASO DE USO: CIERRE DE SESIÓN

1. Descripción

El presente caso de uso describe el procedimiento mediante el cual el usuario (administrador general, administrador o cliente) termina la sesión del cliente.

1.1. Actores.

- Administrador general
- Administrador
- Cliente

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General
- CU_Login_De_Cliente
- CU_Desasignar_Mesa

2. Precondiciones

2.1. Del sistema.

- El cliente debe estar registrado en una sesión activa.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción para ver la lista de sesiones activas. **[FA-1]**
2) El sistema gestor de usuarios muestra los datos de identificación de las sesiones activas actualmente.
3) El administrador selecciona la sesión que quiere dar por terminada.
4) El sistema gestor de usuarios muestra los datos de la sesión seleccionada.
5) El administrador indica que se desea cerrar la sesión.
6) El sistema gestor de usuarios desactiva la sesión. **[E-1]**
7) El sistema gestor de usuarios desasigna las mesas actuales de esa sesión. **[PE-1] [E-1]**
8) El sistema gestor de usuarios informa del éxito de la operación.
9) El sistema retorna a la interfaz de administrador de sesiones.

3.2. Flujo alternativo.

3.2.1. FA-1 Cierre de sesión por parte del cliente

Este flujo se ejecutará si el que requirió el cierre de la sesión fue el cliente en su dispositivo móvil.

1) El cliente selecciona terminar sesión desde su dispositivo.
2) El sistema elimina la información que identifica la sesión en el dispositivo móvil del cliente. **[E-1]**
3) El sistema retorna a la pantalla de registro de cliente dando por terminada la sesión.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que se produjo una excepción
2) El sistema retorna a la pantalla principal del usuario.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

Las mesas utilizadas en la sesión quedan disponibles para ser usadas en una nueva sesión y la sesión original queda desactivada no pudiéndose hacer uso nuevamente de ella.

6. Puntos de Extensión

6.1. PE-1 Desasignar mesa a sesión.

- Se ejecuta un procedimiento similar al descrito en el caso de uso de des asignación de mesas pero pasándole las mesas en forma automática.

CASO DE USO: ACTIVAR REGISTRO DE CLIENTES

1. Descripción

El caso de uso establece el procedimiento a seguir para activar el registro de clientes en una sesión. Este registro sólo estará disponible durante 15 minutos cuando la sesión es creada y 10 minutos cuando es activada manualmente.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador debe estar *logueado* en el sistema.
- La sesión de cliente debe estar creada y activada.
- El registro de clientes de la sesión debe estar desactivado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema de gestión de órdenes muestra una lista con las sesiones activas actualmente.
2) El administrador selecciona la sesión a la que le desea activar su registro de clientes.
3) El sistema de gestión de órdenes despliega la información de la sesión seleccionada.
4) El administrador selecciona la opción de re-activar registro.
5) El sistema de gestión de órdenes activa el registro en la sesión seleccionada por 10 minutos. **[E-1]**
6) El sistema retorna a la interfaz principal de administración.
7) Después de 10 minutos el sistema desactiva automáticamente el registro de usuarios en esa sesión

3.2. Flujo de excepción.

3.2.1. E-1 Excepción

1) El sistema informa de la excepción ocurrida.
2) El sistema muestra la interfaz principal de administración.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La sesión seleccionada permite el registro de clientes durante los próximos 10 minutos después de la activación del registro de clientes.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ALTA DE ADMINISTRADOR

1. Descripción

El caso de uso explica el procedimiento para dar de alta a los administradores del sistema.

1.1. Actores.

- Administrador general.

1.2. Referencias

- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: administración de la interfaz principal de administración.
2) El administrador selecciona la opción de alta de usuario.
3) El sistema de gestión de usuarios pide el ingreso del nombre, usuario, contraseña y tipo de administrador.
4) El administrador introduce la información requerida.
5) El sistema de gestión de usuarios requiere confirmación de los datos ingresados por el usuario.
6) El usuario confirma los datos.
7) El sistema de gestión de usuarios registra la información ingresada en la base de datos. **[E-1]**
8) El sistema de gestión de usuarios informa del éxito de la operación.
9) Se regresa a la pantalla de alta de usuario.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) Se vuelven a pedir los datos del usuario a dar de alta.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El nuevo administrador ha sido registrado en el sistema y puede hacer uso de él mediante su nombre de usuario y su contraseña asignados.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ELIMINACIÓN DE USUARIO

1. Descripción

El caso de uso explica le procedimiento para dar de baja del sistema a un usuarior. Es importante mencionar que debe permanecer el registro de al menos un administrador general después de ejecutarse este proceso.

1.1. Actores.

- Administrador general.

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Administrador

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Debe haber al menos un administrador dado de alta en el sistema a través del proceso de registro de administradores.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: administración de la interfaz principal de administración.
2) El administrador selecciona la opción: baja de usuario.
3) El sistema de gestión de usuarios muestra una lista con los nombres de usuario de los administradores del sistema y su tipo.
4) El administrador selecciona al usuario que desea dar de baja. **[FA-1]**
5) El sistema muestra la información del administrador seleccionado.
6) El administrador selecciona la opción dar de baja.
7) El sistema requiere confirmación de la acción a realizar.
8) El administrador confirma la acción.
9) El sistema borra al usuario de la base de datos. **[E-1]**
10) El sistema informa del éxito de la operación.
11) El sistema muestra la interfaz de administración de usuarios.

3.2. Flujo alternativo.

3.2.1. *FA-1 El administrador general que se quiere dar de baja es el único con privilegios de administración que queda en el sistema.*

1) El sistema de gestión de usuarios indica que antes de dar de baja a este administrador se tiene que crear un nuevo administrador general que se haga responsable del sistema.
2) El sistema vuelve a interfaz de administración de usuarios.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema vuelve a interfaz de baja de usuario.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

Al administrador indicado es dado de baja del sistema y ya no podrá ingresar él.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ALTA DE CLIENTE

1. Descripción

El caso de uso explica el procedimiento para dar de alta a los clientes del restaurante en el sistema.

1.1. Actores.

- Administrador general.
- Cliente.

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Login_De_Cliente

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: administración de la interfaz principal de administración. **[FA-1]**
2) El administrador selecciona la opción de alta de usuario.
3) El sistema de gestión de usuarios pide el ingreso del nombre, usuario, contraseña y tipo de usuario.
4) El administrador introduce la información requerida.
5) El sistema de gestión de usuarios registra la información ingresada en la base de datos. **[E-1]**
6) El sistema de gestión de usuarios informa del éxito de la operación.
7) Se regresa a la pantalla de alta de usuario.

3.2. Flujo alternativo.

3.2.1. <u>FA-1 Registro desde dispositivo del cliente.</u>

1) El sistema solicita el nombre de usuario, la contraseña, el nombre y apellido con los que el cliente desea ser reconocido por el sistema.
2) El cliente introduce la información requerida.
3) El sistema automáticamente pone el tipo de usuario como cliente.
4) El sistema de gestión de usuarios registra la información ingresada en la base de datos. **[E-1]**
5) El sistema retorna a la pantalla de ingreso del sistema donde se le solicitan el nombre de usuario y su contraseña.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) Se vuelven a pedir los datos del usuario a dar de alta.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente ha sido registrado en el sistema y puede hacer uso de él mediante su nombre de usuario y su contraseña asignados.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ALTA DE MESA NUEVA

1. Descripción

El caso de uso describe la forma de dar de alta una mesa nueva en la base de datos del restaurante, el hecho de que una mesa exista no quiere decir que pueda ser asignada a una sesión sino que, al mismo tiempo, la mesa debe estar activada. Esto es así para poder tener mesas de repuesto en caso de ser necesarias.

1.1. Actores.

- Administrador general.

1.2. Referencias

- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: administración de la interfaz principal de administración.
2) El administrador selecciona: nueva mesa.
3) Se muestra el número asignado a la mesa y se pide el ingreso del número de lugares disponibles para ella.
4) Se pregunta si la mesa se dará de alta como activa o como inactiva.
5) El administrador ingresa la información requerida.
6) El sistema de gestión de recursos registra la nueva mesa en la base de datos. **[E-1]**
7) Se regresa a la pantalla de alta de mesas.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) Se informa que ocurrió una excepción.
2) Se vuelve a pedir la introducción de los datos de la mesa nueva retomando el flujo básico desde el punto 3.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La mesa nueva queda registrada para su disposición en el sistema.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ACTIVAR MESA

1. Descripción

El caso de uso describe la forma de activar una mesa. Una mesa activada estará disponible para ser asignada a una sesión activa.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General
- CU_Alta_De_Mesa

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea general o normal, debe estar registrado.

2.2. Del proceso.

- Por lo menos debe haber una mesa registrada en el sistema mediante el procedimiento de alta de mesa.
- Por lo menos debe haber una mesa desactivada.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: mesa de la interfaz principal de administración.
2) El administrador selecciona: activar mesa.
3) El sistema de gestión de recursos muestra una lista con las mesas desactivadas en el momento actual.
4) El administrador selecciona la mesa que desea activar.
5) El sistema de gestión de recursos pide confirmación para ejecutar la operación.
6) El administrador confirma la operación, en caso contrario el sistema regresa a la pantalla de activación de mesas.
7) El sistema de gestión de recursos pone la mesa en estado activado. **[E-1]**
8) El sistema de gestión de recursos informa del éxito de la operación.
9) Se regresa a la pantalla de activación de mesas.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El retorna a la pantalla de activación de mesas.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La mesa seleccionada se encuentra activada y esta disponible para ser asignada a alguna sesión.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: DESACTIVAR MESA

1. Descripción

El caso de uso describe la forma de desactivar una mesa. Una mesa desactivada no estará disponible para ser asignada a una sesión activa.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General
- CU_Alta_De_Mesa

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea general o normal, debe estar registrado.

2.2. Del proceso.

- Por lo menos debe haber una mesa registrada en el sistema mediante el procedimiento de alta de mesa.
- Por lo menos debe haber una mesa activada.
- La mesa que se desee desactivar no puede estar asignada a alguna sesión activa.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: mesa de la interfaz principal de administración.
2) El administrador selecciona: desactivar mesa.
3) El sistema de gestión de recursos muestra una lista con las mesas activadas en el momento actual y que no se encuentras asignadas a alguna sesión.
4) El administrador selecciona la mesa que desea desactivar.
5) El sistema de gestión de recursos pide confirmación para ejecutar la operación.
6) El administrador confirma la operación, en caso contrario el sistema regresa a la pantalla de desactivación de mesas.
7) El sistema de gestión de recursos pone la mesa en estado desactivado. **[E-1]**
8) El sistema de gestión de recursos informa del éxito de la operación.
9) Se regresa a la pantalla principal de desactivación de mesas.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) Se regresa a la pantalla principal de desactivación de mesas.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La mesa seleccionada se encuentra desactivada y no será posible asignarla a alguna sesión.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ASIGNAR MESA A SESIÓN

1. Descripción

El caso de uso describe la forma de asignar una mesa a una sesión ya creada previamente.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General
- CU_Alta_De_Mesa
- CU_Activar_Mesa

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea general o normal, debe estar registrado.

2.2. Del proceso.

- Por lo menos debe haber una mesa registrada en el sistema mediante el procedimiento de alta de mesa.
- Por lo menos debe haber una mesa activada que no este asignada a alguna sesión.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema de gestión de órdenes muestra una lista de las sesiones activas.
2) El administrador elige la sesión a la que quiere asignar la mesa.
3) El sistema muestra la información de la sesión.
4) El administrador elige la opción de agregar mesa.
5) El sistema de gestión de recursos muestra las mesas activas que no están asignadas a ninguna sesión.
6) El sistema de gestión de órdenes solicita el ingreso de las mesas que se desean asignar a la sesión.
7) El administrador selecciona las mesas que va asignar a esta sesión.
8) El sistema de gestión de órdenes realiza la operación solicitada. **[E-1]**
9) El sistema de gestión de órdenes informa del éxito de la operación.
10) El sistema retorna a la interfaz principal de administración de sesiones.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema de gestión de órdenes informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de administración de sesiones sin hacer cambio alguno.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

Las mesas quedan registradas en la sesión elegida y dejan de estar disponibles mientras dure ésta.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: DESASIGNAR MESA DE SESIÓN

1. Descripción

El caso de uso describe la forma de quitar una mesa a una sesión ya creada previamente.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General
- CU_Alta_De_Mesa
- CU_Activar_Mesa
- CU_Asignar_Mesa

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea general o normal, debe estar registrado.

2.2. Del proceso.

- Por lo menos debe haber una mesa registrada en el sistema mediante el procedimiento de alta de mesa que esté asignada a una sesión activa.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema de gestión de órdenes muestra una lista de las sesiones activas.
2) El administrador elige la sesión a la que le quiere quitar la mesa.
3) El sistema muestra la información de la sesión.
4) El sistema de gestión de órdenes muestra las mesas asignadas a esa sesión. **[FA-1]**
5) El administrador elige la mesa que a va a quitar.
6) El sistema de gestión de órdenes realiza la operación solicitada**. [E-1]**
7) El sistema informa del éxito de la operación.
8) El sistema retorna a la interfaz principal de administración.

3.2. Flujo alternativo.

3.2.1. FA-1 La mesa sólo tiene asignada una mesa

1) El sistema de gestión de órdenes informa que toda sesión debe tener al menos una mesa y que no se permite la operación.
2) El sistema vuelve a interfaz principal de administración de sesiones.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de administración de sesiones.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

Las mesa seleccionada es dada de baja de la sesión y esta disponible para usarse en una nueva sesión.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ALTA DE MENU

1. Descripción

El caso de uso explica la forma de dar de alta un nuevo menú en el sistema del restaurante.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General.
- CU_Visualizacion_De_Menu.
- CU_Alta_De_Seccion.
- CU_Activar_Menu.
- CU_Edicion_De_Menu.

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien dé de alta un menú.
- Debe haber al menos una sección registrada en el sistema mediante el proceso de alta de sección.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: nuevo menú. **[PE-1][PE-2]**
4) El sistema requiere el nombre del menú y su descripción.
5) El sistema muestra una lista con las secciones disponibles para agregar al menú.
6) El usuario ingresa los datos solicitados y selecciona las secciones que desea agregar al menú. **[FA-1]**
7) El sistema ingresa el menú al sistema en estado desactivado. **[PE-3][E-1]**
8) El sistema muestra la interfaz principal de administración de menús.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar

1) El usuario selecciona cancelar la operación.
2) El sistema elimina la información sobre el menú de su memoria temporal.
3) El sistema muestra la interfaz principal de gestión de menú.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de gestión de menú.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El menú creado se encuentra listo para su activación y uso.

6. Puntos de Extensión

6.1. PE-1 Visualización de menú

- Se ejecuta el caso de uso de visualización de menú.

6.2. PE-2 Edición de menú

- Se ejecuta el caso de uso de edición de menú.

6.3. PE-3 Activación de menú

- Se ejecuta el caso de uso de activación de menú en sus pasos.

CASO DE USO: EDICIÓN DE MENÚ

1. Descripción

El caso de uso explica la forma de editar un menú creado anteriormente.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Menu
- CU_Activar_Menu
- CU_Visualizacion_De_Menu

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien edite un menú.
- Debe haber por lo menos un menú registrado en el sistema mediante el proceso de alta de menú.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: menú.
4) El sistema muestra una lista de los menús registrados en él.
5) El usuario selecciona el menú que desea editar.
6) El usuario selecciona la opción editar. **[PE-1][PE-2]**
7) El sistema muestra las características del menú en un estado editable.
8) El usuario edita los datos generales del menú y selecciona las secciones que desea que tenga el menú.
9) El sistema pregunta si se desea guardar como un nuevo menú o los datos se guardan en el mismo menú.
10) El usuario indica que se van a guardar los cambios en el mismo menú. **[FA-1]**
11) El usuario presiona guardar. **[FA-2]**
12) El sistema ingresa las modificaciones del menú al sistema. **[PE-1] [E-1]**
13) El sistema muestra la interfaz principal de gestión de menús.

3.2. Flujo alternativo.

3.2.1. <u>Flujo alternativo de guardar como copia</u>

Esta opción se utiliza para generar un nuevo menú usando como base las características de un menú creado anteriormente. Este flujo se lanza cuando el usuario ha indicado que los cambios de edición se guardaran en un nuevo menú.

1) El usuario indica que los cambios se guardarán como un nuevo menú.
2) El usuario presiona guardar. **[FA-2]**
3) El sistema ingresa el nuevo menú al sistema en estado desactivado. **[PE-3] [E-1]**
4) El sistema muestra la interfaz principal de gestión de menú.

3.2.2. <u>FA-2 Cancelar.</u>

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de gestión de menú sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de gestión de menú.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El menú editado se encuentra listo para su activación y uso.

6. Puntos de Extensión

6.1. PE-1 Visualización de menú

- Se ejecuta el caso de uso de visualización de menú.

6.2. PE-2 Alta de menú

- Se ejecuta el caso de uso de alta de menú.

6.3. PE-3 Activación de menú

- Se ejecuta el caso de uso de activación de menú en sus pasos.

CASO DE USO: ALTA DE SECCIÓN

1. Descripción

El caso de uso explica la forma de dar de alta una sección. Los menús del restaurante se compondrán de varias secciones. Las secciones pueden tener tanto subsecciones como platillos que sólo pertenezcan a la sección sin estar en una subsección.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Platillo
- CU_Alta_De_Subseccion
- CU_Edicion_De_Seccion

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien dé de alta una sección.
- Debe haber al menos un platillo registrado en el sistema mediante el proceso de alta de platillo.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: sección.
4) El sistema muestra las secciones registradas en el sistema.
5) El usuario selecciona la opción de alta de sección. **[PE-1]**
6) El sistema solicita la introducción de los datos de nombre y descripción de la sección.
7) El sistema muestra las subsecciones que se pueden agregar a la sección.
8) El sistema muestra los platillos que se pueden agregar directamente a la sección.
9) El usuario introduce los datos requeridos y selecciona los platillos y subsecciones que desea.
10) El usuario presiona guardar. **[FA-1]**
11) El sistema da de alta la sección en la base de datos. **[E-1]**
12) El sistema muestra la interfaz principal de gestión de sección.

3.2. Flujo alternativo.

3.2.1. <u>FA-1 Cancelar</u>

1) El usuario selecciona cancelar la operación.
2) El sistema elimina la información sobre la sección de su memoria temporal.
3) El sistema muestra la interfaz principal de gestión de sección.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de gestión de sección.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La sección creada esta disponible para su utilización en el sistema.

6. Puntos de Extensión

6.1. PE-1 Edición de sección

- Se ejecuta el caso de uso de edición de sección.

CASO DE USO: ALTA DE SUBSECCIÓN

1. Descripción

El caso de uso explica la forma de dar de alta una subsección. Las secciones de los menús se compondrán de varios platillos.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Platillo
- CU_Edicion_De_Subseccion

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien dé de alta una subsección.
- Debe haber al menos un platillo registrado en el sistema mediante el procedimiento de alta de platillo.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: subsección.
4) El sistema muestra las subsecciones registradas en él.
5) El usuario selecciona alta de subsección. **[PE-1]**
6) El sistema solicita la introducción de los datos de nombre y descripción de la subsección.
7) El sistema muestra los platillos que se pueden agregar.
8) El usuario introduce los datos requeridos y selecciona los platillos que desea.
9) El usuario presiona guardar. **[FA-1]**
10) El sistema da de alta la subsección en la base de datos. **[E-1]**
11) El sistema muestra la interfaz de subsección.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar

1) El usuario selecciona cancelar la operación.
2) El sistema elimina la información sobre la sección de su memoria temporal.
3) El sistema muestra la interfaz principal de gestión de subsección.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de gestión de subsección.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La subsección creada esta disponible para su utilización en el sistema.

6. Puntos de Extensión

6.1. PE-1 Edición de subsección

- Se ejecuta el caso de uso de edición de subsección.

CASO DE USO: ALTA DE PLATILLO

1. Descripción

El caso de uso explica la forma de dar de alta un platillo para que pueda ser añadido a cualquier sección o subsección.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General.
- CU_Edicion_De_Platillo.

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien dé de alta un platillo.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: platillo.
4) El sistema despliega una lista con los platillos registrados en él.
5) El usuario selecciona alta de platillo. **[PE-1]**
6) El sistema solicita la introducción del nombre del platillo, precio, su descripción, su imagen y si estará activado en el momento de su creación.
7) El sistema muestra las categorías a las que puede pertenecer el platillo.
8) El sistema muestra los tipos a los que puede pertenecer el platillo.
9) El usuario introduce los datos requeridos y selecciona su categoría y tipo.
10) El usuario presiona guardar. **[FA-1]**
11) El sistema da de alta el platillo en la base de datos. **[E-1]**
12) El sistema muestra la interfaz de alta de platillo.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar

1) El usuario selecciona cancelar la operación.
2) El sistema elimina la información sobre el platillo de su memoria temporal.
3) El sistema muestra la interfaz principal de gestión de platillo.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de gestión de platillo.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El platillo creado está disponible para su utilización en el sistema.

6. Puntos de Extensión

6.1. PE-1 Edición de platillo

- Se ejecuta el caso de uso de edición de platillo.

CASO DE USO: EDICIÓN DE PLATILLO

1. Descripción

El caso de uso explica la forma de editar un platillo registrado previamente en el sistema.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Platillo

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien edite un platillo.
- Debe haber por lo menos un platillo registrado en el sistema mediante el proceso de alta de platillo.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: platillo.
4) El sistema presenta los platillos disponibles.
5) El usuario selecciona el platillo que desea editar.
6) El usuario selecciona la opción de editar. **[PE-1]**
7) El sistema muestra los datos registrados del platillo en campos editables.
8) El usuario edita la información deseada.
9) El usuario presiona guardar. **[FA-1]**
10) El sistema registra la nueva información en la base de datos. [**E-1]**
11) El sistema muestra la interfaz principal de gestión de platillo.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de baja de platillo sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.2.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de baja de platillo.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El platillo dado de baja ya no esta disponible para su utilización en el sistema.

6. Puntos de Extensión

6.1. PE-1 Alta de platillo

- Se ejecuta el caso de uso de alta de platillo.

CASO DE USO: EDICIÓN DE SECCIÓN

1. Descripción

El caso de uso explica la forma de editar una sección creada anteriormente.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Seccion

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien edite una sección.
- Debe haber por lo menos una sección registrada en el sistema mediante el proceso de alta de sección.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: sección.
4) El sistema muestra una lista de las secciones registradas en él.
5) El usuario selecciona la sección que desea editar.
6) El usuario selecciona la opción: editar**. [PE-1]**
7) El sistema muestra las características de la sección en un estado editable.
8) El usuario edita los datos generales de la sección y selecciona las subsecciones y platillos que desea que tenga el menú.
9) El usuario presiona guardar. **[FA-1]**
10) El sistema ingresa los cambios al sistema. **[E-1]**
11) El sistema muestra la interfaz principal de gestión de sección.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de gestión de sección sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz de gestión de sección.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La sección editada se encuentra lista para su uso.

6. Puntos de Extensión

6.1. PE-1 Alta de sección

- Se ejecuta el caso de uso de alta de sección.

CASO DE USO: EDICIÓN DE SUBSECCIÓN

1. Descripción

El caso de uso explica la forma de editar una subsección creada anteriormente.

1.1. Actores.

- Administrador general

1.2. Referencias

- CU_Login_De_Admin_General
- CU_Alta_De_Subseccion

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

- Únicamente el administrador general puede ser quien edite una subsección.
- Debe haber por lo menos una subsección registrada en el sistema mediante el proceso de alta de subsección.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: administración de la interfaz principal de administración.
2) El sistema presenta las opciones de administración.
3) El usuario selecciona la opción: subsección.
4) El sistema muestra una lista de las subsecciones registradas en él.
5) El usuario selecciona la subsección que desea editar.
6) El usuario selecciona la opción de editar. **[PE-1]**
7) El sistema muestra las características de la subsección en un estado editable.
8) El usuario edita los datos generales de la subsección y platillos que desea que tenga la subsección.
9) El usuario presiona guardar. **[FA-1]**
10) El sistema ingresa los cambios al sistema. **[E-1]**
11) El sistema muestra la interfaz principal de gestión de subsección.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de gestión de subsección sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz de gestión de subsección.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La subsección editada se encuentra lista para su uso.

5. Puntos de Extensión

6.1. PE-1 Alta de subsección

- Se ejecuta el caso de uso de alta de subsección.

CASO DE USO: ACTIVAR MENÚ

1. Descripción

El caso de uso describe el procedimiento para activar un menú manualmente. El sistema no debe permitir que pueda haber un momento en que no haya algún menú activado por lo que para desactivar un menú se debe de activar otro forzosamente.

1.1. Actores.

- Administrador general
- Administrador

1.2. Referencias

- CU_Login_De_Admin_General.
- CU_Login_De_Admin.
- CU_Alta_De_Menu

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea normal o general, debe estar registrado.

2.2. Del proceso.

- Se debe de tener por lo menos un menú registrado.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción menú de la interfaz principal de administración.
2) El usuario muestra la interfaz principal de administración de menús.
3) El usuario selecciona la opción de activar menú.
4) El sistema muestra el menú actualmente activo.
5) El sistema muestra una lista con los menús disponibles para su activación.
6) El usuario selecciona el menú que desea activar.
7) El sistema pide confirmación de la operación.
8) El usuario confirma la operación. **[FA-1]**
9) El sistema desactiva el menú activado actualmente. **[E-1]**
10) El sistema activa el menú seleccionado. **[E-1]**
11) El sistema muestra la interfaz principal de administración de menús.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de activar menú sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz de activación de menú.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El menú seleccionado queda activado.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: ACTIVAR PLATILLO

1. Descripción

El caso de uso describe el procedimiento para activar un platillo manualmente. Se permite desactivar platillos para que estos no aparezcan en el menú aunque éste los contenga.

1.1. Actores.

- Administrador general
- Administrador

1.2. Referencias

- CU_Login_De_Admin_General.
- CU_Login_De_Admin.
- CU_Alta_De_Menu

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea normal o general, debe estar registrado.

2.2. Del proceso.

- Se debe de tener por lo menos un menú registrado.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción menú de la interfaz principal de administración.
2) El usuario muestra la interfaz principal de administración de menús.
3) El usuario selecciona la opción de platillos.
4) El sistema muestra los platillos registrados y su estado.
5) El usuario selecciona el platillo que desea activar.
6) El sistema pide confirmación de la operación.
7) El usuario confirma la operación. **[FA-1]**
8) El sistema activa el platillo seleccionado. **[E-1]**
9) El sistema muestra la interfaz principal de administración de menús.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de activar menú sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz de activación de menú.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El menú seleccionado queda activado.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: DESACTIVAR PLATILLO

1. Descripción

El caso de uso describe el procedimiento para activar un platillo manualmente. Se permite desactivar platillos para que estos no aparezcan en el menú aunque éste los contenga.

1.1. Actores.

- Administrador general
- Administrador

1.2. Referencias

- CU_Login_De_Admin_General.
- CU_Login_De_Admin.
- CU_Alta_De_Menu

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea normal o general, debe estar registrado.

2.2. Del proceso.

- Se debe de tener por lo menos un menú registrado.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción menú de la interfaz principal de administración.
2) El usuario muestra la interfaz principal de administración de menús.
3) El usuario selecciona la opción de platillos.
4) El sistema muestra los platillos registrados y su estado.
5) El usuario selecciona el platillo que desea desactivar.
6) El sistema pide confirmación de la operación.
7) El usuario confirma la operación. **[FA-1]**
8) El sistema desactiva el platillo seleccionado. **[E-1]**
9) El sistema muestra la interfaz principal de administración de menús.

3.2. Flujo alternativo.

3.2.1. FA-1 Cancelar.

1) El usuario selecciona cancelar la operación.
2) El sistema muestra la interfaz principal de activar menú sin haber realizado cambio alguno.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz de activación de menú.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El menú seleccionado queda activado.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: DESPLEGADO DE MENÚ

1. Descripción

El caso de uso describe la forma en que el cliente puede interactuar con el sistema para observar el menú del restaurante actualmente activo.

1.1. Actores.

- Cliente
- Administrador General

1.2. Referencias

- CU_Login_De_Cliente
- CU_Login_De_Administrador_General
- CU_Seleccion_De_Platillo

2. Precondiciones

2.1. Del sistema.

- El usuario debe estar logueado en el sistema.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona menú en la interfaz principal de cliente. **[FA-1]**
2) El sistema despliega el nombre del menú actualmente activo y una lista de sus secciones.
3) El usuario selecciona la sección que quiere ver.
4) El sistema muestra los platillos y las subsecciones que conforman la sección.
5) El usuario selecciona la subsección que desea visualizar.
6) El sistema muestra los platillos pertenecientes a la subsección seleccionada y permite ingresar la cantidad de ello que se desee. **[PE-1]**
7) El usuario selecciona un platillo para ver su s datos.
8) El sistema presenta las características del platillo.

3.2. Flujo alternativo.

3.2.1. FA-1 Visualización de menú en pantalla de administración

1) El administrador general ingresa a la sección de gestión de menús.
2) El sistema de gestión de menú muestra una lista con los menús registrados en el sistema.
3) El administrador selecciona el menú que desea pre visualizar.
4) El administrador selecciona la opción de pre visualizar el menú
5) Sistema de gestión de menú genera la vista del menú y muestra sus platillos jerarquizados según su sección y subsección.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente visualiza el menú en su dispositivo móvil.

6. Puntos de Extensión

6.1. PE-1 Selección de platillos.

- Se ejecuta el caso de uso de selección de platillos.

CASO DE USO: SELECCIÓN DE PLATILLO

1. Descripción

El caso de uso describe la forma en que el cliente puede seleccionar un platillo para conformar su orden.

1.1. Actores.

- Cliente

1.2. Referencias

- CU_Login_De_Cliente

2. Precondiciones

2.1. Del sistema.

- El usuario debe estar logueado en el sistema.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario se encuentra visualizando una lista de platillos.
2) El usuario ingresa la cantidad del platillo que desea.
3) El sistema muestra que el platillo fue seleccionado.
4) El sistema ingresa el identificador del platillo y la cantidad en su cola de orden.
5) El usuario selecciona ordenar.
6) El sistema muestra los platillos seleccionados por el usuario, su cantidad y su total.
7) El sistema pide confirmación.
8) El usuario confirma que es l que desea ordenar.
9) El sistema genera la orden y la transmite para su procesamiento. **[E-1]**
10) El sistema espera mensaje de orden exitosa.
11) El sistema indica el éxito de la operación.
12) El sistema presenta la interfaz principal del cliente.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente ha seleccionado los platillos que desea y estos han sido transmitidos al sistema central.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: CREACIÓN DE LA SESIÓN

1. Descripción

El caso de uso describe la forma en que un administrador puede crear una sesión para que los clientes se registren en ella y sean capaces de generar órdenes de platillos.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin
- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador, ya sea general o normal, debe estar registrado..

2.2. Del proceso.

El caso de uso no tiene precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El administrador selecciona la opción: inicio de la interfaz principal de administración.
2) El sistema genera el código de seguridad que ocupará la nueva sesión. **[E-1]**
3) El sistema requiere las mesas que tendrá asignadas la sesión.
4) El administrador asigna las mesas que tendrá la sesión.
5) El sistema requiere el ingreso de la referencia del cliente.
6) El administrador ingresa la referencia del cliente.
7) El sistema muestra el código de seguridad, las mesas y la referencia y requiere confirmar la operación.
8) El administrador confirma la operación.
9) El sistema introduce los datos de la sesión en la base de datos y la activa. **[E-1]**
10) El sistema informa el éxito de la operación.
11) El administrador proporciona al cliente el código de seguridad y le informa de sus mesas asignadas.

3.2. Flujo alternativo.

El sistema no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de administración.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

Se ha creado una nueva sesión activa y esta disponible para que cualquier cliente se registre en ella.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: CREACIÓN DE LA ORDEN

1. Descripción

El caso de uso describe la forma en que el cliente formará la orden que será enviada al sistema para su procesamiento.

1.1. Actores.

- Cliente

1.2. Referencias

- CU_Login_De_Cliente

2. Precondiciones

2.1. Del sistema.

- El cliente debe de estar registrado en una sesión activa.

2.2. Del proceso.

- El cliente no debe de haber cerrado su sesión.

3. Flujo de eventos

3.1. Flujo básico.

1) El cliente selecciona la opción de hacer orden en la interfaz principal del cliente.
2) El sistema le muestra los platillos que han seleccionado.
3) El sistema solicita confirmación de la operación.
4) El cliente confirma la operación**. [FA-1]**
5) El sistema arma el mensaje de orden. **[E-1]**
6) El sistema transmite la orden al sistema de gestión de órdenes para su procesado. **[E-1]**
7) El sistema de gestión de órdenes informa de la transmisión correcta de los datos.
8) El sistema recibe la indicación de que fue correctamente transmitida la orden y le informa al usuario que su pedido se esta procesando.
9) El sistema presenta nuevamente la interfaz principal del cliente.

3.2. Flujo alternativo.

3.2.1. FA- 1 Cancelar.

1) El cliente pulsa retroceder.
2) El sistema presenta la interfaz del menú permitiéndole al cliente alterar su orden.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal del cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La orden generada esta registrada en el sistema principal para su posterior procesamiento.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: CALCULO DEL TOTAL INDIVIDUAL

1. Descripción

El caso de uso describe la forma en que el cliente visualiza su total individual.

1.1. Actores.

- Cliente

1.2. Referencias

- CU_Login_De_Cliente

2. Precondiciones

2.1. Del sistema.

- El cliente debe de estar registrado en una sesión activa.

2.2. Del proceso.

- El cliente no debe de haber cerrado su sesión.

3. Flujo de eventos

3.1. Flujo básico.

1) El cliente selecciona la opción de ver total individual en la interfaz principal del cliente.
2) El sistema genera una petición de total individual.
3) El sistema transmite el mensaje de petición de total al sistema principal. **[E-1]**
4) El sistema principal calcula el total pedido y transmite la respuesta al cliente. **[E-1]**
5) El sistema recibe la respuesta generada. [**E-1]**
6) El sistema despliega el total pedido en la pantalla del cliente.
7) El usuario selecciona retornar y se vuelve a la pantalla principal de cliente.

3.2. Flujo alternativo.

El sistema no cuenta con flujos alternativos

3.3. Flujo de excepción.

3.3.1. <u>E-1 Excepción</u>

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal del cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente visualiza el total pedido.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: CALCULO DEL TOTAL DE LA SESIÓN

1. Descripción

El caso de uso describe la forma en que el cliente visualiza su total de la sesión activa en donde esta registrado.

1.1. Actores.

- Cliente

1.2. Referencias

- CU_Login_De_Cliente

2. Precondiciones

2.1. Del sistema.

- El cliente debe de estar registrado en una sesión activa.

2.2. Del proceso.

- El cliente no debe de haber cerrado su sesión.

3. Flujo de eventos

3.1. Flujo básico.

1) El cliente selecciona la opción de ver total de la sesión en la interfaz principal del cliente.
2) El sistema presenta las opciones de cuenta individual y cuenta total.
3) El usuario selecciona cuenta total.
4) El sistema genera una petición de cuenta total.
5) El sistema transmite el mensaje de petición de total al sistema principal. **[E-1]**
6) El sistema principal calcula el total pedido y transmite la respuesta al cliente. **[E-1]**
7) El sistema recibe la respuesta generada. [E-1]
8) El sistema despliega el total pedido en la pantalla del cliente.
9) El usuario selecciona retornar y se vuelve a la pantalla principal de cliente.

3.2. Flujo alternativo.

El sistema no cuenta con flujos alternativos

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal del cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El cliente visualiza el total pedido.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: PROCESADO DE ORDEN

1. Descripción

El presente caso de uso describe la forma en que un mensaje de orden es procesado por el sistema principal.

1.1. Actores.

- Sistema de gestión de órdenes.

2. Precondiciones

2.1. Del sistema.

- La sesión que quiere registrar la orden debe estar activada.
- El cliente que efectúa la orden debe estar asignado a la sesión que la va a registrar.

2.2. Del proceso.

- El cliente no debe de haber cerrado su sesión.

3. Flujo de eventos

3.1. Flujo básico.

1) El sistema de gestión de órdenes recibe una petición de procesado de orden junto con el mensaje que contiene la orden.
2) El sistema de gestión de órdenes revisa que el cliente que emite la orden tenga una sesión activa. **[E-1]**
3) El sistema de gestión de órdenes registra las órdenes en la base de datos. **[E-1]**
4) El sistema de gestión de órdenes actualiza el total individual del cliente que hizo la orden. **[E-1]**
5) El sistema de gestión de órdenes actualiza el total de la sesión. **[E-1]**
6) La orden se coloca como activa para su procesamiento en la cocina. **[E-1]**
7) El sistema de gestión de órdenes genera la respuesta de orden procesada que será enviada al cliente.
8) El sistema transmite la respuesta generada al cliente.

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema de gestión de órdenes deshace los cambios que haya efectuado en la base de datos.
2) El sistema de gestión de órdenes genera un mensaje de error.
3) El sistema de gestión de órdenes transmite el mensaje de error al cliente.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

La orden recibida ha sido procesada y los totales individual y de sesión actualizados.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: PRESENTACIÓN DE REPORTES DE ÓRDENES

1. Descripción

El caso de uso explica la forma de visualizar un reporte de cómo han sido ordenados los platillos a través del tiempo.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: reporte de la interfaz principal de administración.
2) El sistema muestra una lista de los platillos.
3) El usuario selecciona los platillos de su interés.
4) El sistema despliega una lista con los tiempos disponibles para generar un reporte: diario o mensual.
5) El usuario selecciona el tiempo.
6) El sistema pide el intervalo de tiempo.
7) El usuario ingresa intervalo.
8) El sistema genera la información pedida y la muestra en forma de una gráfica de poligonal donde el eje y es la cantidad del platillo ordenado y el eje x el tiempo. **[E-1]**

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de administración.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

El usuario puede ver un reporte con los datos de su interés.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

CASO DE USO: PRESENTACIÓN DE REPORTES DE INGRESOS

1. Descripción

El caso de uso explica la forma de visualizar un reporte de cómo han sido los ingresos a través del tiempo.

1.1. Actores.

- Administrador general.
- Administrador

1.2. Referencias

- CU_Login_De_Admin_General

2. Precondiciones

2.1. Del sistema.

- El administrador general debe estar registrado.

2.2. Del proceso.

El caso de uso no cuenta con precondiciones del proceso.

3. Flujo de eventos

3.1. Flujo básico.

1) El usuario selecciona la opción: reporte de la interfaz principal de administración.
2) El usuario selecciona la opción: ingresos.
3) El sistema despliega una lista con los tiempos disponibles para generar un reporte: diario, o mensual.
4) El usuario selecciona el tiempo.
5) El sistema pide el intervalo de tiempo.
6) El usuario lo ingresa.
7) El sistema genera la información pedida y la muestra en forma de una gráfica poligonal donde el eje y es el ingreso y el eje x el tiempo (días o meses). **[E-1]**

3.2. Flujo alternativo.

El caso de uso no cuenta con flujos alternativos.

3.3. Flujo de excepción.

3.3.1. E-1 Excepción

1) El sistema informa que ocurrió una excepción.
2) El sistema retorna a la interfaz principal de administración.

4. Requerimientos especiales

El caso de uso no cuenta con requerimientos especiales.

5. Postcondiciones

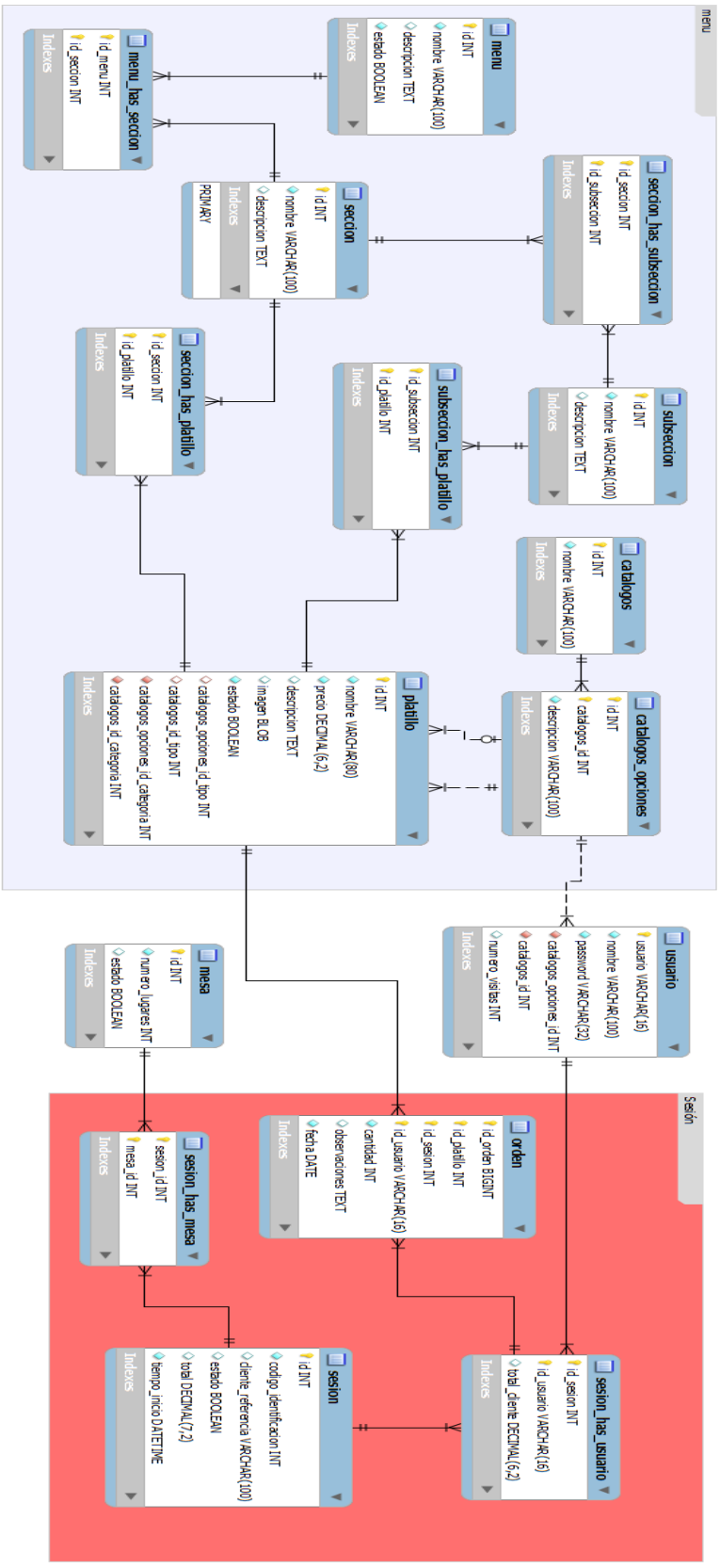
El usuario puede ver un reporte con los datos de su interés.

6. Puntos de Extensión

El caso de uso no cuenta con puntos de extensión.

8.3. Apéndice C. Diagramas de caso de uso de la aplicación propuesta.

8.4. Apéndice D. Diagrama E-R de la aplicación propuesta.



8.5. Apéndice E. Archivo Ant de proyecto restaurante.

```
<project name="restaurante" default="build" basedir="../">
      <property file="${basedir}/ant/build.properties" />

      <property name="src.dir" value="${basedir}/src" />
      <property name="webroot.dir" value="${basedir}/WebContent" />
      <property name="webinf.dir" value="${webroot.dir}/WEB-INF" />
      <property name="build.dir" value="${basedir}/build" />
      <property name="webxml" value="${webinf.dir}/web.xml" />

      <!-- Classpath para utilerias tomcat -->
      <path id="ant.classpath">
                  <fileset dir="${tomcat.home}/lib">
                  <include name="catalina-ant.jar"/>
                  <include name="tomcat-coyote.jar"/>
                  <include name="tomcat-util.jar"/>
                  </fileset>
                  <fileset dir="${tomcat.home}/bin">
                  <include name="tomcat-juli.jar"/>
                  </fileset>
            </path>

      <!-- Classpath para el proyecto -->
      <path id="compile.classpath">
            <!-- Libs del proyecto -->
            <fileset dir="${webinf.dir}/lib">
                  <include name="*.jar"/>
                  <include name="*.zip"/>
            </fileset>

            <!-- Libs de tomcat -->
            <fileset dir="${tomcat.home}/lib">
                  <include name="*.jar"/>
            </fileset>

            <pathelement path ="${classpath}"/>
            <pathelement path ="${webinf.dir}/classes"/>
      </path>

      <!-- Define tareas de inicio, deploy, undeploy y paro de la
      aplicacion en servidor -->
      <taskdef name="start" classname="org.apache.catalina.ant.StartTask"
      classpathref="ant.classpath" />
      <taskdef name="stop" classname="org.apache.catalina.ant.StopTask"
      classpathref="ant.classpath" />
      <taskdef name="deploy"
      classname="org.apache.catalina.ant.DeployTask"
      classpathref="ant.classpath" />
      <taskdef name="undeploy"
      classname="org.apache.catalina.ant.UndeployTask"
      classpathref="ant.classpath" />

      <!-- Inicia aplicacion -->
      <target name="start" description="Inicia aplicacion">
            <start url="${tomcat.manager.url}"
                        username="${tomcat.manager.username}"
                        password="${tomcat.manager.password}"
                        path="/${project.name}" />
      </target>

      <!-- Detiene aplicacion -->
      <target name="stop" description="Detiene aplicacion">
```

```
      <stop
            failonerror="false"
            url="${tomcat.manager.url}"
            username="${tomcat.manager.username}"
            password="${tomcat.manager.password}"
            path="/${project.name}" />
</target>

<target name="undeploy" description="Quita la publicacion"
depends="stop">
      <undeploy
            failonerror="false"
            url="${tomcat.manager.url}"
            username="${tomcat.manager.username}"
            password="${tomcat.manager.password}"
            path="/${project.name}"
      />
</target>

<!-- Limpia total -->
<target name="clean-all" depends="undeploy">
      <delete dir="${build.dir}"/>
      <delete dir="${webinf.dir}/classes"/>

      <!-- Borra el war de tomcat -->
      <delete file="${webapps.dir}/${project.name}.war" />

      <!-- Borra directorio de tomcat -->
      <delete dir="${webapps.dir}/${project.name}"
      failonerror="false" />
</target>

<!-- Crea directorios -->
<target name="prepare">
      <mkdir dir="${build.dir}"/>
      <mkdir dir="${webinf.dir}/classes" />
</target>

<!-- copia recursos -->
<target name="resources" depends="prepare">
      <copy todir="${webinf.dir}/classes" includeEmptyDirs="no">
            <fileset dir="src">
                  <patternset>
                        <include name="**/*.conf" />
                        <include name="**/*.properties" />
                        <include name="**/*.xml" />
                  </patternset>
            </fileset>
      </copy>
</target>

<!-- Compila -->
<target name="compile" depends="resources">
<javac srcdir="${src.dir}" destdir="${webinf.dir}/classes"
debug="on" debuglevel="source,lines,vars"
includeantruntime="false">
      <classpath refid="compile.classpath"/>
</javac>
</target>

<!-- Hace el war -->
<target name="war" depends="compile">
      <war warfile="${build.dir}/${project.name}.war"
      webxml="${webxml}">
            <classes dir="${webinf.dir}/classes" />
```

```
            <fileset dir="${webroot.dir}">
                  <exclude name="WEB-INF/web.xml" />
            </fileset>
      </war>
</target>

<!-- Publica -->
<target name="deploy" depends="war">
      <deploy
            url="${tomcat.manager.url}"
            username="${tomcat.manager.username}"
            password="${tomcat.manager.password}"
            path="/${project.name}"
            war="file:${build.dir}/${project.name}.war"
      />
</target>

<target name="tomcat-start">
      <echo>---------------------------------------</echo>
      <echo>- target - tomcat-start</echo>
      <echo>-</echo>
      <echo>- starting tomcat</echo>
      <echo>---------------------------------------</echo>
      <java classname="org.apache.catalina.startup.Bootstrap"
      failonerror="true" fork="true">
            <classpath
            path="${tomcat.home}/bin/bootstrap.jar:${tomcat.home}/b
            in/tomcat-juli.jar" />
            <jvmarg value="-
            Djava.util.logging.manager=org.apache.juli.ClassLoaderL
            ogManager" />
            <jvmarg value="-
            Djava.util.logging.config.file=${tomcat.home}/conf/logg
            ing.properties" />
            <jvmarg value="-Dcatalina.home=${tomcat.home}" />
            <jvmarg value="-Dcatalina.base=${tomcat.home}" />
            <jvmarg value="-Djava.io.tmpdir=${tomcat.home}/temp" />
            <jvmarg value="-DSIR_ENV=tomcat/qa" />
            <!-- Esta propiedad define el puerto debug remoto -->
            <!-- <jvmarg value="-
            Xrunjdwp:transport=dt_socket,address=8099,server=y,susp
            end=n"/> -->
            <arg line="start" />
      </java>
</target>

<target name="tomcat-stop">
      <echo>---------------------------------------</echo>
      <echo>- target - tomcat-stop</echo>
      <echo>-</echo>
      <echo>- stopping tomcat</echo>
      <echo>---------------------------------------</echo>
      <java classname="org.apache.catalina.startup.Bootstrap"
      fork="true">
            <classpath
            path="${tomcat.home}/bin/bootstrap.jar:${tomcat.home}/b
            in/tomcat-juli.jar" />
            <jvmarg value="-Dcatalina.home=${tomcat.home}" />
            <arg line="stop" />
      </java>
</target>

<!-- Por defecto -->
<target name="build" depends="clean-all,deploy,start" />
</project>
```

8.6. Apéndice F. Archivo Ant de proyecto persistencia.

```
<project name="restaurante-persistencia" basedir="../" default="build">
      <!-- Project settings -->
      <property name="project.name" value="restaurante-persistencia"/>
      <property name="project.jar" value="${basedir}/restaurante-
      persistencia" />
      <property name="project.war"
      value="${basedir}/restaurante/WebContent/WEB-INF/lib" />

      <!-- global properties -->
      <property name="source.dir" value="${project.jar}/src" />
      <property name="lib.dir" value="${project.jar}/lib" />

      <property name="build.dir" value="${project.jar}/jar" />
      <property name="build.classes" value="${build.dir}/classes" />

      <property name="component-jar" value="restaurante-persistencia.jar"
      />

      <!-- Create directory -->
      <target name="directorio">
            <mkdir  dir="${build.dir}" />
      </target>

      <!-- Remove classes directory for clean build -->
      <target name="clean" depends="directorio">
            <delete dir="${build.dir}" />
      </target>

      <!-- Normal build of application -->
    <target name="compile" depends="clean">
        <mkdir  dir="${build.classes}" />
        <javac srcdir="${source.dir}" destdir="${build.classes}"
includeantruntime="false">
            <classpath refid="compile.classpath"/>
        </javac>
      <copy todir="${build.classes}">
                  <fileset dir="${source.dir}">
                        <exclude name="**/*.java" />
                  </fileset>
      </copy>
        <jar jarfile="${build.dir}/${component-jar}"
        basedir="${build.classes}" update="no" ></jar>
    </target>


      <!-- Build entire project -->
      <target name="build" depends="directorio,clean,compile,limpiar-
      compile,copiar-jar-proyecto"/>

      <!-- Clean generated elements after compile -->
      <target name="limpiar-compile" depends="compile">
            <delete dir="${build.classes}"/>
            <delete file="${project.war}/${component-jar}"/>
      </target>

      <!-- Copy Jar Generated into project war library -->
      <target name="copiar-jar-proyecto">
            <copy file="${build.dir}/${component-jar}"
            todir="${project.war}"/>
      </target>
```

```
      <!-- classpath for project -->
      <path id="compile.classpath">
            <fileset dir="${project.jar}/lib">
                  <include name="**/*.jar"/>
            </fileset>
            <pathelement path ="${project.jar}/build/classes"/>
            <pathelement path ="${classpath}"/>
      </path>
</project>
```

9. Bibliografía.

- Hall, Marty; Brown, Larry. *Core Servlets and Java Server Pages: Volume 1: Core Technologies*. 2a edición. California: Sun Microsystems Press, 2004.
- Gamma, Erich; Helm, Richard; Johnson, Ralph; Vissides, John. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1a edición. Addison-Wesley Professional, 1994.
- Johnson, Rod. *Expert One-on-one J2EE Design and Development*. 1a edición. Hungry Minds Inc, 2002.
- Kayal, Dhrubojyoti, *Pro Java EE Spring Patterns: Best Practices and Design Strategies Implementing Java EE Patterns with the Spring Framework*. 1a edición. Nueva York: Apress, 2008.
- Fowler, Martin. *Patterns of Enterprise Application Architecture*. 1a edición. Addison-Wesley Educational Publishers Inc, 2002.
- Evans, Eric. *Domain-driven Design: Tackling Complexity in the Heart of Software*. 1a edición. Addison-Wesley Educational Publishers Inc, 2003.
- Crupi, John; Malks, Dan; Alur, Deepak. *Core J2EE Patterns*. 1a edición. California: Sun Microsystems Press, 2001.
- Newton, Dave. *Apache Struts 2: Web Application Development*. 1a edición. Birmingham: Packtpub, 2009.
- Roughley, Ian. *Practical Apache Struts2 Web 2.0 Projects*. 1a edición. Nueva York: Apress, 2007.
- Clarence, Ho; Rob, Harrop. *Pro Spring 3*. 1a edición. Nueva York: Apress, 2012.

- Walls, Craig. *Spring in Action*. 3a edición. Nueva York: Manning Publications Co., 2011.
- Vokotic, Aleksa; Goodwill, James. *Apache Tomcat 7*. 1a edición. Nueva York: Apress, 2011.
- Reza Seddighi, Ahmad. *Spring Persistance with Hibernate*. 1a edición. Birmingham: Packtpub, 2009.
- Minter, Dave; Linwood, Jeff. *Pro Hibernate 3*. 1a edición. Nueva York: Apress, 2005.
- Den Haan, Peter; Lavandowksa, Lance; Perrumal, Krishnaraj. *Beginning JSP: From Novice to Professional*. 1a edición. Nueva York: Apress, 2004.
- Zambon, Guilio; Sekler, Michael. *Beginning JSP, JSF and Tomcat Web Development: From Novice to Professional*. 1a edición. Nueva York: Apress, 2007.
- Freeman, Adam. *Pro Javascript for Web Apps*. 1a edición. Nueva York: Apress, 2012.
- Java EE: http://www.oracle.com/technetwork/java/javaee/downloads/index.html, última revisión en Noviembre 2012.
- Java EE documentación: http://docs.oracle.com/javaee/, última revisión en Noviembre 2012.
- Java EE tutorial: http://docs.oracle.com/javaee/6/tutorial/doc/, última revisión en Noviembre 2012.
- Struts 2: http://struts.apache.org/2.3.8/index.html, última revisión en Noviembre 2012.

- Struts 2, referencia de tags: http://struts.apache.org/2.0.12/docs/tag-reference.html, última revisión en Diciembre 2012.
- Hibernate 3: http://www.hibernate.org/, última revisión en Noviembre 2012.
- Hibernate 3, documentación: http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/, última revisión en Diciembre 2012.
- HTML, documentación: http://www.w3schools.com/html/default.asp, última revisión en Diciembre 2012.
- CSS, documentación: http://www.w3schools.com/css/default.asp, última revisión en Diciembre 2012.
- Javascript, documentación: http://www.w3schools.com/js/default.asp, última revisión en Diciembre 2012.
- JQuery: http://jquery.com/, última revisión en Diciembre 2012.
- JQuery mobile: http://jquerymobile.com/, última revisión en Diciembre 2012.
- Apache Ant: http://ant.apache.org/, última revisión en Noviembre 2012.
- Subversion: http://subversion.tigris.org/, última revisión en Octubre 2012.
- Apache Log4j: http://logging.apache.org/log4j/1.2/, última revisión en Octubre 2012.
- Eclipse Indigo: http://www.eclipse.org/indigo/, última revisión en Octubre 2012.
- JBoss Tools Indigo: http://marketplace.eclipse.org/node/121986#.UP2oTScsB-M, última revisión en Septiembre 2012.
- Apache Tiles: http://tiles.apache.org/, última revisión en Noviembre 2012.
- Mysql: http://www.mysql.com/, última revisión en Noviembre 2012.
- Apache Tomcat: http://tomcat.apache.org/, última revisión en Noviembre 2012.