

**DIPLOMADO EN GERENCIA DE PROYECTOS**

**EN DESARROLLO DE SOFTWARE**

**MÓDULO: GESTION DE LA CONFIGURACION DEL SOFTWARE**



**Grupo**

**CASO DE ESTUDIO: SISTEMA DE INFORMACIÓN PARA GESTIONAR ÓRDENES DE PEDIDO PARA UNA HELADERIA**

**INTEGRANTES**

Andrea Ysabel Carranza Daza

Ely Condori Guevara

Alejandra Soca Alaca

Fatima Condori Guevara

Edson Gabriel Valdez

**DOCENTE:** Dra. Karem Esther Infantas Soto.

**SANTA CRUZ – BOLIVIA**

INDICE GENERAL

[CAPÍTULO I 6](#_Toc469924213)

[ASPECTOS METODOLÓGICOS 6](#_Toc469924214)

[INTRODUCCIÓN 7](#_Toc469924215)

[1.1. PLANTEAMIENTO DEL PROBLEMA 7](#_Toc469924216)

[1.1.1 Situación problemática 7](#_Toc469924217)

[1.1.2 Situación deseada 7](#_Toc469924218)

[1.1.3 Delimitación del contenido 7](#_Toc469924219)

[1.2 ACTORES 8](#_Toc469924220)

[METODOLOGÍA 9](#_Toc469924221)

[1.3. PROCESO METODOLÓGICO. 9](#_Toc469924224)

[1.3.1. Fase de exploración 9](#_Toc469924229)

[1.3.2. Fase de planificación 9](#_Toc469924230)

[1.3.3. Fase de iteraciones 10](#_Toc469924231)

[1.3.4. Fase de puesta en producción 10](#_Toc469924232)

[1.3.5. Planificación temporal 11](#_Toc469924233)

[1.4. SCRUM 12](#_Toc469924234)

[1.4.1. Procesos 12](#_Toc469924236)

[1.4.2. Roles y Responsabilidades 14](#_Toc469924237)

[1.4.3. Prácticas 14](#_Toc469924238)

[1.5. XTREME PROGRAMMING 15](#_Toc469924239)

[1.5.1. Procesos 16](#_Toc469924241)

[1.5.2. Roles y responsabilidades. 17](#_Toc469924242)

[1.5.3. Prácticas 18](#_Toc469924243)

[1.6. COMBINACIÓN DE XTREME PROGRAMMING Y SCRUM 20](#_Toc469924244)

[1.7. MÉTODOS Y TÉCNICAS DE SCRUM 20](#_Toc469924245)

[1.8. MÉTODOS Y TÉCNICAS DE XTREME PROGRAMMING. 21](#_Toc469924246)

[CAPÍTULO II 22](#_Toc469924247)

[REQUERIMIENTOS 22](#_Toc469924248)

[2.1. REQUERIMIENTOS FUNCIONALES (PRODUCT BACKLOG) 23](#_Toc469924251)

[2.1.2. REQUERIMIENTOS FUNCIONALES 24](#_Toc469924252)

[2.2. REQUERIMIENTOS NO FUNCIONALES 25](#_Toc469924253)

[2.1.1. Tiempo de aprendizaje 25](#_Toc469924254)

[2.1.2. Confiabilidad 25](#_Toc469924255)

[2.1.3. Performance 25](#_Toc469924256)

[2.1.4. Facilidad de Mantenimiento 26](#_Toc469924257)

[2.1.5. Estándares de arquitectura 26](#_Toc469924258)

[2.1.5. Motor de base de datos 26](#_Toc469924259)

[2.1.7. Cliente del navegador 26](#_Toc469924260)

[2.1.8. Servidor Web 26](#_Toc469924261)

[2.1.9. Interfaces 26](#_Toc469924262)

[CAPÍTULO III 28](#_Toc469924263)

[ANÁLISIS 29](#_Toc469924265)

[ESPECIFICACIÓN DE REQUERMIENTOS DE SOFTWARE 29](#_Toc469924267)

[3.1 Historias de usuario 29](#_Toc469924268)

[CAPÍTULO IV 37](#_Toc469924269)

[DISEÑO 38](#_Toc469924271)

[4.1. Arquitectura de la solución para modelo en capas 38](#_Toc469924273)

[4.1.1. Descripción de la arquitectura de la solución 39](#_Toc469924274)

[4.2. Arquitectura de la solución usando modelo n capas 40](#_Toc469924275)

[4.3. DISEÑO DE BASE DE DATOS 41](#_Toc469924276)

[4.3.1. Diagrama conceptual de la base de datos 41](#_Toc469924284)

[4.3.2. Modelo Lógico 42](#_Toc469924286)

[CAPÍTULO V 47](#_Toc469924289)

[IMPLEMENTACIÓN 47](#_Toc469924290)

[5.1. APLICACIÓN WEB 48](#_Toc469924292)

[5.1.1. Implementación de presentación Categoria 48](#_Toc469924293)

[5.1.2. Implementación de negocio Categoria 54](#_Toc469924294)

[5.1.4. Implementación de dato 56](#_Toc469924300)

[5.1.5. Modelo de despliegue para la solución 63](#_Toc469924301)

[CAPÍTULO VI 64](#_Toc469924303)

[PRUEBA 64](#_Toc469924304)

[6.1. PRUEBAS DE SOFTWARE 65](#_Toc469924305)

[7. REFERENCIAS 69](#_Toc469924306)

[7.1. BIBLIOGRAFÍA 69](#_Toc469924307)

[7.2. ENLACES 69](#_Toc469924308)

***INDICE DE TABLAS***

[Tabla 1 Actores del sistema 8](#_Toc468719339)

[Tabla 2 Fases de SCRUM 13](#_Toc468719340)

[Tabla 3 Roles y Responsabilidades de SCRUM 14](#_Toc468719341)

[Tabla 4 Prácticas y Herramientas de SCRUM 15](#_Toc468719342)

[Tabla 5 Fases de Xtreme Programming 17](#_Toc468719343)

[Tabla 6 Roles y Responsabilidades de XP 18](#_Toc468719344)

[Tabla 7 Técnicas y Prácticas de XP 19](#_Toc468719345)

***INDICE DE ILUSTRACIONES***

[Ilustración 1Integración XP y SCRUM 9](#_Toc469924380)

[Ilustración 2 Diagrama de Gantt 11](#_Toc469924381)

[Ilustración 3 Proceso SCRUM 12](#_Toc469924382)

[Ilustración 4 Ciclo de Vida XP 16](#_Toc469924383)

[Ilustración 5 Arquitectura de la Solución 38](#_Toc469924384)

[Ilustración 6 Arquitectura de Software 40](#_Toc469924385)

[Ilustración 7 Diseño Conceptual de la Base de Datos 41](#_Toc469924386)

[Ilustración 8 Diseño Lógico Relacional de la Base de Datos 42](#_Toc469924387)

[Ilustración 10 Modelo de Despliegue 63](#_Toc469924388)

# CAPÍTULO I

# ASPECTOS METODOLÓGICOS

# INTRODUCCIÓN

## 1.1. PLANTEAMIENTO DEL PROBLEMA

### 1.1.1 Situación problemática

En las heladerías el proceso de pedido se realiza de forma que no deja satisfecho al cliente: la carta tiene muchos platillos para escoger; lo cual el cliente demora demasiado y cuando elige dicho plato resulta que ya no está disponible porque sobrepaso el stock. ¿Cuántas veces nos quejamos porque se tardan demasiado tiempo en atendernos?, o por el contrario, ¿aún no hemos decidido qué elegir y el encargado de registrar los pedidos ya está dispuesto a tomar nota?

### 1.1.2 Situación deseada

Se proveerá a la heladería y pastelería Bambi una aplicación móvil y un sistema web, mediante la aplicación móvil el cliente podrá ingresar datos de facturación (razón social y nit[[1]](#footnote-1)) además de escoger y seleccionar los productos que desea consumir, estos pedidos se mostraran encolados en una interfaz web para que los meseros atiendan las solicitudes y posteriormente imprimir la factura generada por consumo.

### 1.1.3 Delimitación del contenido

**Productos**

La empresa ofrece una gran variedad de helados y pasteles, los cuales fueron clasificados por categorías. Los helados y pasteles son los productos de los cuales se necesita poder ingresar uno nuevo al menú, poder modificar algunos atributos como el precio y tener un listado de sus productos con los siguientes datos: nombre, precio, stock, descripción e imagen.

**Categoría**

En la aplicación web se realizara la parametrización de los productos disponibles. Se los clasificara por categoría, lo cual tendrá los siguientes datos: nombre de la categoría y una descripción acerca de ella. Algunos ejemplos de las categorías que maneja en la Heladería y Pastelería Bambi son: Horneados, Tortas, Cafés/Té.

**Registro de las Mesa**

Se configurará la cantidad de mesas que tienen disponibles.

Se podrá cargar las fotos de los productos.

Se podrá visualizar los pedidos que realice un cliente.

**Pedido**

Los pedidos realizados por el cliente necesita los siguientes datos para su registro: la fecha, el número de mesa y los productos que fueron elegidos para consumir, se le mostrara un detalle de los productos elegidos y el monto total del pedido.

## 1.2 ACTORES

|  |  |  |
| --- | --- | --- |
| **Código** | **Actor** | **Descripción** |
| Admin | Administrador | Se encarga de la administración de la heladería y pastelería. Es el encargado de subir todos los datos en el sistema web, encargado de la gestión de mesas, categorías, productos. |
| Cliente | Cliente | Persona que va a realizar el pedido. |
| Mesero | Mesero | Persona que atiende (lleva su pedido) al cliente. |

Tabla 1 Actores del sistema

# METODOLOGÍA



## 1.3. PROCESO METODOLÓGICO.

Para la planificación y gestión del proyecto se seguirá las fases que plantea la metodología Programación Extrema (XP), para el desarrollo del producto se utilizara el Framework SCRUM y UML para documentar los artefactos del sistema.

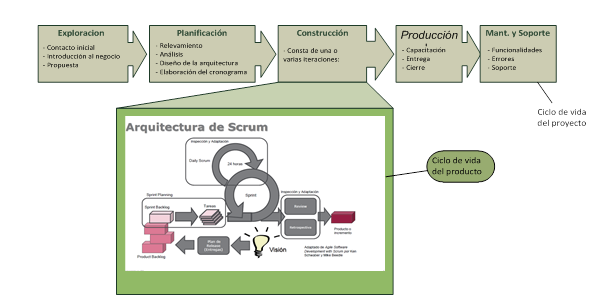


Ilustración 1Integración XP y SCRUM



### 1.3.1. Fase de exploración

Es la fase en la que se define el alcance general del proyecto. En esta fase, el cliente define lo que necesita mediante la redacción de “historias de usuarios”. (Schwaber & Sutherland, 2012)

### 1.3.2. Fase de planificación

La planificación es una fase corta, en la que el cliente, los gerentes y el grupo de desarrolladores acuerdan el orden en que deberán implementarse las historias de usuario, y, asociadas a éstas, las entregas.

### 1.3.3. Fase de iteraciones

Esta es la fase principal en el ciclo de desarrollo de XP. Las funcionalidades son desarrolladas en esta fase, generando al final de cada una un entregable funcional que implementa las historias de usuario asignadas a la iteración.

En esta fase se empleara el Frameworks planteado por SCRUM. Para gestionar el desarrollo del producto.

### 1.3.4. Fase de puesta en producción

Al final de cada Sprint se entregan módulos funcionales, potencialmente usables, puede ser deseable por parte del cliente poner el sistema en producción.

En esta fase no se realizan más desarrollos funcionales, pero pueden ser necesarias tareas de ajuste. (Schwaber & Sutherland, 2012).

### 1.3.5. Planificación temporal

|  |  |
| --- | --- |
| **Proyecto** | Sistema de información para la gestión de órdenes de pedido |
| **Metodología** | Programación Extrema y Framework SCRUM |
| **Duración** | 30 días |
| **Costo** | 10000 $us |
| **Contratista** | Cloud Solutions |

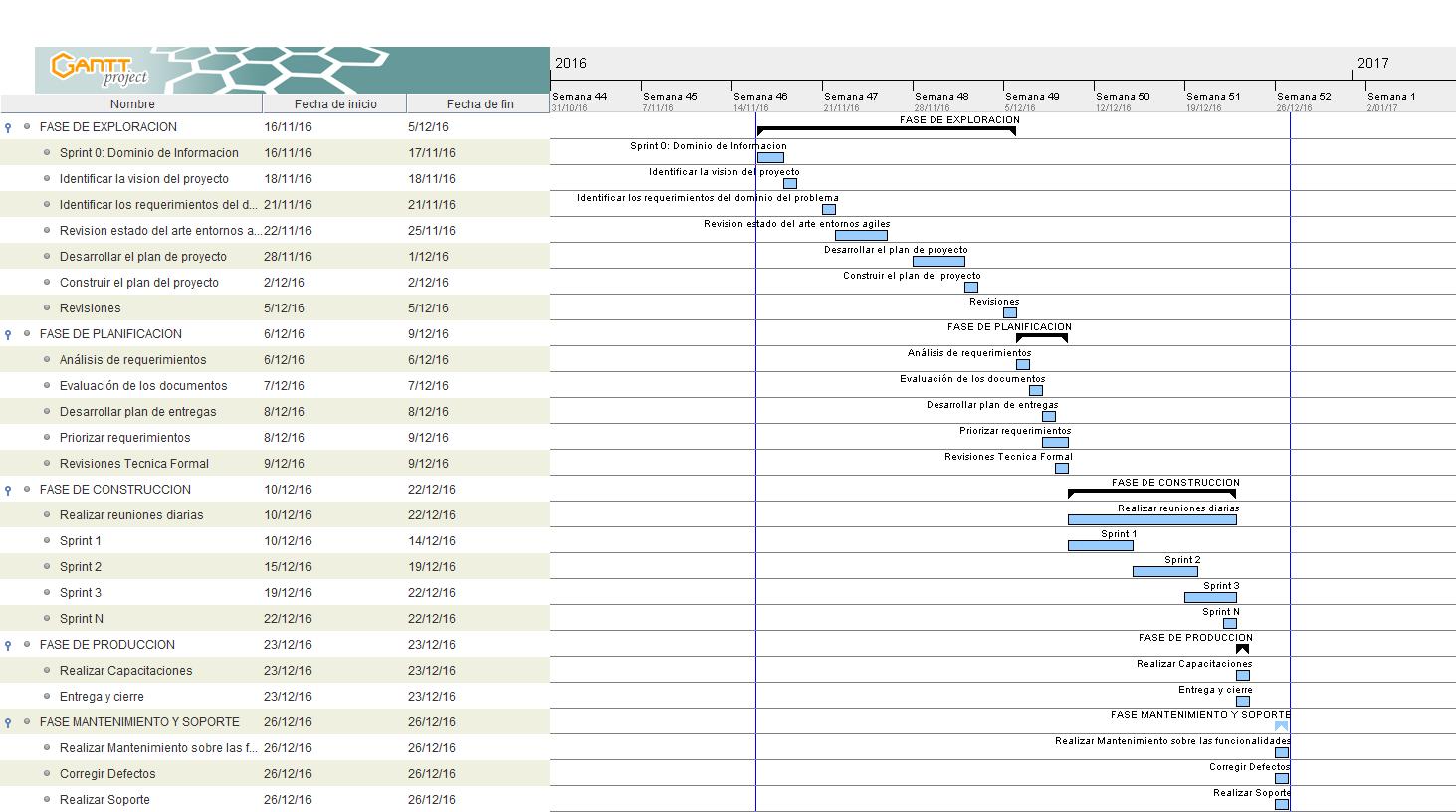


Ilustración 2 Diagrama de Gantt

## 1.4. SCRUM

La primera vez que se asoció el término SCRUM a los procesos de desarrollo fue en 1986, cuando Nonaka y Takeuchi presentaron su artículo The New Product Development Game. Nonaka y Takeuchi presentaban en este artículo un proceso adaptativo, rápido y auto-organizado de desarrollo de productos. El término Scrum deriva del mismo término en rugby, que hace referencia a como se devuelve un balón que ha salido fuera del campo, al terreno de juego de una manera colectiva, la traducción al castellano sería melé.

Scrum surgió como práctica en el desarrollo de productos tecnológicos y no sería hasta 1993 que Jeff Sutherland aplicara el modelo al desarrollo de software en la Easel Corporation. En 1996 Sutherland presentó junto con Ken Schwaber las prácticas que empleaba como proceso formal para la gestión del desarrollo de software en OOPSLA 96. Estas prácticas de gestión pasarían a incluirse junto con otras muchas en la lista de modelos ágiles de Agile Alliance en el año 2001.



### 1.4.1. Procesos

En SCRUM se puede identificar tres fases: planificación del sprint, seguimiento del Sprint y revisión del sprint, como se puede apreciar en la Ilustración 2.

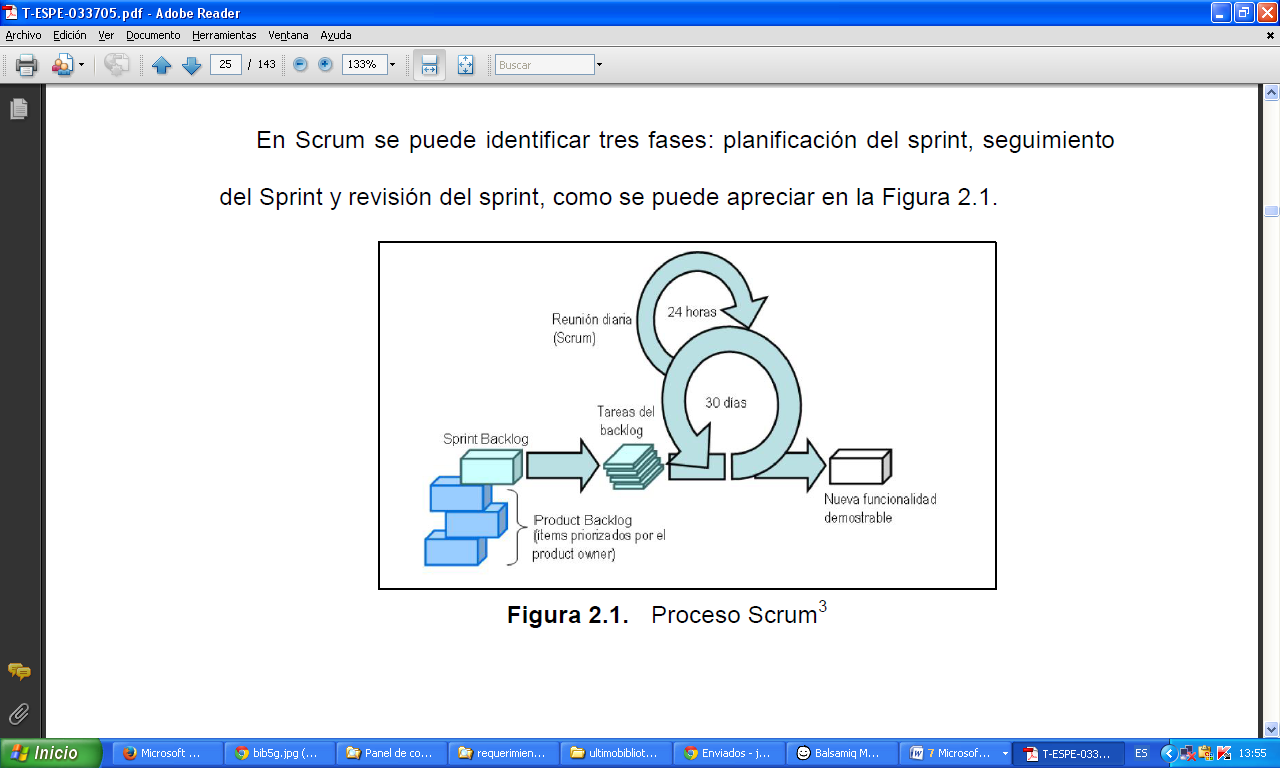


Ilustración 3 Proceso SCRUM

En la Tabla 2 se puede observar las principales tareas que se llevan a cabo en cada fase.

|  |  |
| --- | --- |
| **Fase** | **Descripción** |
| **Planificación del Sprint** | En esta fase se define el Product Backlog. Si todavía no ha sido definido, consiste en una lista priorizada de requisitos del sistema y es un documento vivo, que puede ser continuamente actualizado. En cada iteración el Product Backlog es revisado por el equipo. También se lleva a cabo la planificación del primer Sprint. La planificación de cualquier sprint es la jornada de trabajo previa al inicio de cualquier sprint y en la cual se determinan cuáles son los objetivos y el trabajo que se deben cubrir en esa iteración. En esta reunión se obtiene una lista de tareas que se denomina Sprint Backlog, y el lema u objetivo principal del sprint. |
| **Seguimiento del Sprint** | A lo largo de esta fase se llevan a cabo breves reuniones diarias, para ver el avance de las tareas y el trabajo que está previsto para la jornada. En estas reuniones solo están presentes el SCRUM Master y el equipo, las preguntas que se realizan suelen ser tres:  1. Qué trabajo se ha realizado desde la reunión anterior.  2. Qué trabajo que se va a hacer hasta la próxima reunión.  3. Qué impedimentos que deben solventarse para proseguir con el trabajo. |
| **Revisión del Sprint** | Una vez finalizado el Sprint, se realiza un análisis y revisión del incremento generado. En esta reunión se presentan los resultados finales y se recomienda siempre tener preparada una demo. Existen múltiples razones para recomendar tener una demo al final de cada sprint, entre ellas la mejora del feedback con los interesados, reconocimiento del trabajo, un esfuerzo por finalizar las cosas o un correctivo en caso de tener una demo mal desarrollada. |

Tabla 2 Fases de SCRUM

### 1.4.2. Roles y Responsabilidades

En la Tabla 3 se describe los roles y responsabilidades de cada persona en SCRUM.

|  |  |
| --- | --- |
| **Rol** | **Descripción** |
| **SCRUM Master (Líder del Proyecto)** | Es el encargado de garantizar el funcionamiento de los procesos y de la metodología. Es importante darse cuenta que SCRUM Master es más que un rol, es la responsabilidad de funcionamiento de modelo, por tanto muchas veces es aconsejable utilizar a personas y puestos más adecuados según la organización. Un SCRUM master debe interactuar tanto con el equipo como con el cliente y con los gestores. |
| **Propietario del producto (Product Owner):** | Es la única persona del proyecto conocedora del entorno de negocio del cliente y de la visión del producto y es el responsable de obtener el resultado de mayor valor posible para el cliente. También es el responsable de la financiación necesaria para el proyecto, de tomar las decisiones que afecten a cómo va a ser el resultado final, fechas de lanzamiento y el retorno de inversión. Por regla general y si no se trata de proyectos internos, el propietario del producto suele ser el responsable del proceso de adquisición del cliente.  El equipo está involucrado en la estimación del esfuerzo de las tares del Product Backlog, en la creación del Sprint Backlog, etc. |
| **Equipo de desarrollo (SCRUM Team)** | Es el equipo del proyecto y tiene la autoridad para decidir en las acciones necesarias y para auto-organizarse con la finalidad de alcanzar los objetivos del Sprint. |

Tabla 3 Roles y Responsabilidades de SCRUM

### 1.4.3. Prácticas

SCRUM no requiere y/o provee de ninguna práctica concreta para el desarrollo del software, sin embargo sí que dispone de prácticas y herramientas para la gestión de las diferentes fases de SCRUM. En la Tabla 4 se muestra las principales prácticas y herramientas de la metodología Scrum.

|  |  |
| --- | --- |
| **Prácticas** | **Descripción** |
| **Product Backlog (Pila del producto)** | Define los requisitos del sistema o el trabajo a hacer a lo largo del proyecto. Está compuesto por una lista de requisitos de negocios y técnicos, actualizados y priorizados. El responsable de mantener el product backlog es el propietario del producto. |
| **Sprint Backlog (Pila de tareas)** | Es una lista de trabajos que el equipo se compromete a realizar para generar el incremento previsto. Las tareas están asignadas a personas y tienen estimados el tiempo y los recursos necesarios. |
| **Estimación de esfuerzo** | Es un proceso iterativo en el cual las estimaciones de los ítems del product backlog son reajustadas acorde a la información obtenida en la última iteración. Este reajuste lo llevan a cabo el equipo de desarrollo y el propietario del producto. |
| **Gráfico Burn-down.** | Es una herramienta para gestionar y seguir el trabajo de cada sprint y representa gráficamente el avance del sprint. |
| **Gráfico Burn-up.** | Herramienta de gestión y seguimiento que sirve al propietario del producto para controlar las versiones de producto previstas, las funcionalidades de cada una, la velocidad estimada, fechas probables de cada versión, margen de error previsto en las estimaciones y avance real. |

Tabla 4 Prácticas y Herramientas de SCRUM

## 1.5. XTREME PROGRAMMING

Xtreme Programming es sin duda es el abanderado de las metodologías ágiles. Nació como un intento, bastante exitoso, de establecer un conjunto de prácticas que facilitasen la finalización de los proyectos. Después de unas cuantas exitosas pruebas, estas prácticas se plasmaron de forma teórica, dando lugar a una metodología, que mantenía sus principales principios y prácticas. El término Xtreme es debido a que las prácticas que se utilizaron, fueron llevadas hasta el extremo.



### 1.5.1. Procesos

El ciclo de vida de Xtreme Programming está compuesto de seis fases, ver Ilustración 3.

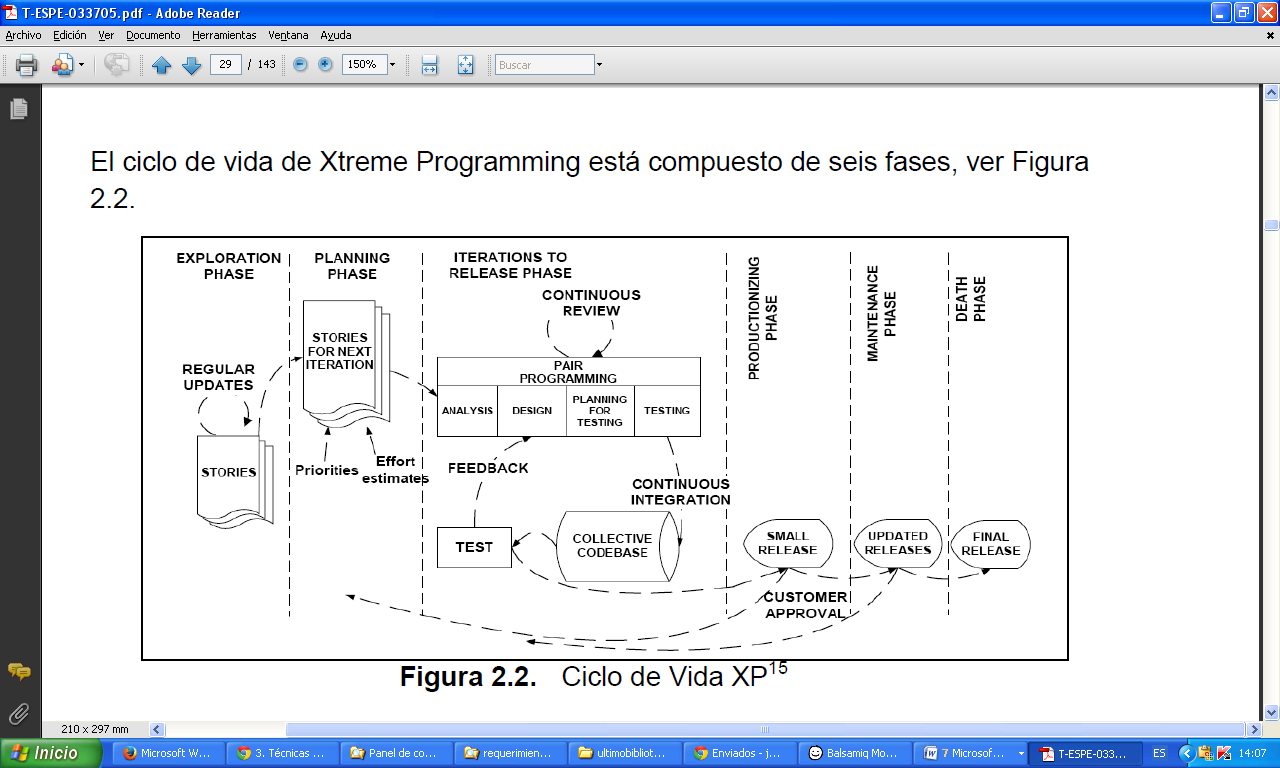


Ilustración 4 Ciclo de Vida XP

En la Tabla 5 se explica de una manera más detallada las fases de Xtreme Programming.

|  |  |
| --- | --- |
| **Fases** | **Descripción** |
| **Fase de exploración** | Los usuarios escriben las tarjetas de historia que ellos quieren que sean incluidas en la primera versión. Cada una de las tarjetas de historia describe una funcionalidad que será añadida al programa. El equipo de proyecto durante este tiempo se dedica a familiarizarse con las tecnologías y herramientas que utilizará a lo largo del proyecto, probando las herramientas y construyendo un prototipo simple para comprobar la arquitectura. El periodo de tiempo de esta fase puede variar desde unas pocas semanas hasta unos pocos meses, dependiendo de la familiaridad del equipo con las tecnologías. |
| **Fase de planificación** | Se establece la prioridad de las diferentes historias y se acuerda el contenido de la primera entrega del proyecto.  La estimación temporal se basa en un cálculo por parte de los desarrolladores de cada una de las historias, la primera entrega no suele tardar más de dos meses en realizarse.  La duración de esta fase no suele exceder el plazo de unos pocos días. |
| **Fase de iteraciones** | Esta fase incluye la realización de diferentes fases antes de liberar la primera versión del producto. La planificación realizada en la etapa anterior se divide en diferentes iteraciones, de una duración variable entre una semana y cuatro. Los usuarios son los que deciden que historias se van a realizar en cada iteración, sabiendo que en la primera se suele realizar un sistema con la arquitectura de todo el sistema, seleccionando aquellas historias que ayuden a construirla. Las pruebas funcionales creadas por el cliente son ejecutadas al final de cada iteración, de tal manera que al final de esta fase se obtiene una versión lista para producción. |
| **Fase de mantenimiento** | Una vez que se ha liberado la primera versión a los usuarios, el proyecto se debe mantener en el entorno de producción siempre y cuando aún haya iteraciones en fase de producción. Esto supone un esfuerzo considerable en la fase de mantenimiento e incluso se sugiere la contratación de nuevo personal para dar soporte a los clientes hasta puede cambiar la estructura del equipo.19 |
| **Fase de cierre del proyecto.** | Los clientes ya no tienen más historias que deban ser implementadas. Es necesario para que se considere que en esta fase se satisfagan todas las necesidades de los clientes y otros aspectos como fiabilidad, rendimiento, etc. La documentación del proyecto se realiza en esta fase, ya que ni la arquitectura, ni el diseño, ni el código sufrirán cambio alguno. Cuando el coste del producto es demasiado elevado para su desarrollo también se puede entrar en esta fase. |

Tabla 5 Fases de Xtreme Programming

### 1.5.2. Roles y responsabilidades.

En la Tabla 6 se nombra los diferentes roles que se puede observar en un proyecto XP, según las tareas que desarrollan y los propósitos que persiguen.

|  |  |
| --- | --- |
| **Rol** | **Descripción** |
| **Cliente** | El cliente es el encargado de escribir las historias y las pruebas funcionales, y es el que decide cuando un requisito es satisfecho. También es el encargado de establecer la prioridad de las funcionalidades a implementar. |
| **Programador** | Escribe tanto las pruebas como el código de la aplicación y se debe comunicar fluidamente y frecuentemente con sus compañeros. |
| **Probador** | Ayuda a escribir las pruebas funcionales al cliente, las ejecuta regularmente, transmite los resultados de las pruebas y mantiene las herramientas. |

|  |  |
| --- | --- |
| **Probador** | Ayuda a escribir las pruebas funcionales al cliente, las ejecuta regularmente, transmite los resultados de las pruebas y mantiene las herramientas. |
| **Rastreador** | Es el encargado dar el “feedback” en XP. Se encarga de seguir las estimaciones hechas por los clientes y de ir avisando de las desviaciones posibles y como cuanto se ajustan a la realidad, con tal de mejorar las futuras estimaciones. También se encarga de realizar el seguimiento de las iteraciones y valorar si los objetivos se pueden alcanzar con los recursos disponibles (tiempo, personal) o si es necesario hacer algún cambio. |
| **Coach o tutor** | Es la persona responsable de todo el proceso. Es importante que tenga conocimientos y experiencia en otros proyectos de XP y de este modo guiar y ayudar al equipo a adaptarse a XP. |
| **Consultor** | Es un miembro externo que tiene los conocimientos técnicos necesarios. El consultor guía al equipo para solucionar los problemas específicos. |
| **Gestor o Manager** | Más conocido como el jefe. Es el que toma las decisiones críticas y está en permanente contacto con el equipo para poder discernir las diferentes situaciones críticas. |

Tabla 6 Roles y Responsabilidades de XP

### 1.5.3. Prácticas

Xtreme Programming dispone de un gran abanico de técnicas y prácticas, muchas de ellas que han sido escogidas de diferentes metodologías existentes que tenían suficientemente probada su eficiencia.

En la Tabla 7 se lista las principales técnicas y prácticas de XP.

|  |  |
| --- | --- |
| **PRÁCTICAS** | **DESCRIPCIÓN** |
| **El juego de la planificación** | El equipo de desarrolladores estima el esfuerzo necesario para implementar las historias y los clientes determinan los objetivos y tiempos de entrega. |
| **Historias de usuario** | Son los requisitos del sistema formulados en una o dos sentencias, en el lenguaje común del cliente. |
| **Cortas y pequeñas iteraciones** | Un sistema simple se libera cada dos o tres meses y las diferentes versiones del mismo se suceden en periodos no superiores al mes. |
| **Metáforas** | El sistema se define utilizando un conjunto de metáforas acordadas entre el cliente y los programadores. Esta historia compartida guiará todo el proceso describiendo cómo funciona el sistema. |
| **Diseño simple** | Se da gran importancia a la obtención de diseños simples que se puedan implementar rápidamente, evitando diseños complejos y código extra. |
| **Pruebas** | El desarrollo del software es orientado a pruebas (Test Driven Development). Las pruebas unitarias se escriben antes que el código y están funcionando continuamente. Las pruebas funcionales las escriben los clientes.25 |
| **Refactorizar** | Restructurar el sistema eliminando información duplicada, mejorando la comunicación, simplificando y añadiendo flexibilidad. |
| **Programación por pares:** | Es la técnica que promulga que dos personas escriban código en el mismo ordenador. |
| **Propiedad colectiva** | Cualquiera puede compartir cualquier parte de código con cualquier otro componente del equipo. |
| **Integración continúa** | Una nueva porción de código es integrada en el código fuente, tan pronto como este lista. El sistema es integrado y construido muchas veces a lo largo del día, todas las pruebas son ejecutadas y deben ser pasadas para aceptar la nueva porción de código. |
| **40 horas a la semana** | Se establece un máximo de 40 horas de trabajo  semanales. |
| **Prácticas** | Descripción |
| **Disponibilidad del cliente** | Los clientes deben estar disponibles y presentes cuando sean requeridos por el equipo de desarrollo. |
| **Estándares de codificación** | Las reglas de codificación son establecidas y seguidas por los programadores. Se enfatiza la comunicación a través del código. |
| **Espacios de trabajo abiertos** | Un gran espacio, con cubículos es lo recomendable. 27 |

Tabla 7 Técnicas y Prácticas de XP

## 1.6. COMBINACIÓN DE XTREME PROGRAMMING Y SCRUM

La elección de las metodologías ágiles a utilizar en este proyecto se ha sustentado fundamental y principalmente, en la popularidad y la documentación, siendo SCRUM y Xtreme Programming las dos metodologías que cumplen estos requisitos.

## 1.7. MÉTODOS Y TÉCNICAS DE SCRUM

Se ha seguido con las indicaciones de Henrik Kniberg para la realización de proyectos con SCRUM y Xtreme Programming, que presenta en su libro Scrum y Xp desde las Trincheras. De tal manera que los métodos y técnicas que se utilizará para SCRUM son:

• **La pila del producto o product backlog:** Es el documento a través del cual se recogen los requisitos de los clientes.

• **La pila de tareas o sprint backlog:** Es el conjunto de historias, que pertenecen a la pila del producto, que se va a realizar en el sprint.

• **Estimación del esfuerzo:** Cada historia de la pila del producto es estimada con lo que se llama puntos de historia y que se corresponde a días-persona ideales.

• **Gráfico Burn-down:** Este gráfico ayuda a medir la productividad y detectar riesgos de mala distribución del trabajo y desviaciones temporales.

* + - Reuniones para cada sprint.
    - Reuniones de seguimiento diarias.

• **Planning Poker:** Esta técnica se utiliza a la hora de realizar la estimación de las historias. Simula el juego de Poker para obtener una estimación colectiva y cooperativa de cada historia.

## 1.8. MÉTODOS Y TÉCNICAS DE XTREME PROGRAMMING.

Los métodos y técnicas que se tomará de Xtreme Programming son las siguientes:

• **Test Driven Development:** Diseñar las pruebas antes que implementar las funcionalidades. Todo código que pasa a producción tiene sus pruebas asociadas.

• **Refactorizar:** Rescribir aquellos métodos y clases, para mejorar la legibilidad del código. Esta es una técnica fundamental cuando se genera software de manera incremental y a partir de las pruebas.

• **Diseño incremental:** Mantener un diseño simple desde el principio e ir mejorándolo continuamente.

• **Integración continúa:** Esta técnica permite compilar el código cada vez que se sube al repositorio de código, ejecutar las pruebas y desplegarlo en un entorno de producción. A su vez, puede realizar este proceso de forma automática y cada cierto tiempo, de tal manera que se puede observar los diferentes resultados de las pruebas en diferentes condiciones y horas.

.

# CAPÍTULO II

# REQUERIMIENTOS

# REQUERIMIENTOS DE SOFTWARE



## 2.1. REQUERIMIENTOS FUNCIONALES (PRODUCT BACKLOG)

|  |  |
| --- | --- |
| Identificación del documento | |
| Código | MTRB-SCRUM-REQ-SRS-001 |
| Título | Sistema de Información para gestionar órdenes de pedido para la heladería y Pastelería Bambi |
| Fecha creación | 12/12/2016 |
| Elaborado por | Dra. Karem Esther Infantas Soto |

|  |  |
| --- | --- |
| Lista de distribución | |
| DESARROLLO | Cloud Solutions:  Ely Condori Guevara  Edson Gabriel Valdez  Alejandra Soca Alaca  Andrea Ysabel Carranza Daza  Fátima Condori Guevara |
| {CLIENTE} | Heladería y Pastelería Bambi |

|  |  |
| --- | --- |
| Revisión del documento | |
| Revisado por | Dra. Karem Esther Infantas Soto |
| En fecha | 19/12/2016 |

|  |  |
| --- | --- |
| Aprobación del documento | |
| Aprobado por | Dra. Karem Esther Infantas Soto |
| En fecha | 20/12/2016 |

|  |  |  |  |
| --- | --- | --- | --- |
| Control de versiones | | | |
| Versión | **Causa del Cambio** | **Responsable** | Fecha |
| 1.0 | Revisión | Cloud Solutions | 15/12/2016 |

### 2.1.2. REQUERIMIENTOS FUNCIONALES

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Nombre del requerimiento** | **Como Probarlo**  **(Atributos de calidad)** | **Departamento seccion** | **Estimación**  **(History point)** |
| **RF000** | *Página de inicio* | * *Al ingresar que la pagina muestre el logo de la empresa, con una pequeña descripción de ella.* * *Un menú con las siguientes opciones:*   *Administrador:*   * *Inicio* * *Mesas* * *Salir.*   *Cliente:*   * *Categoría de productos* * *Productos* * *Pedido* * *Salir* | *Depto. administración* | *24* |
| **RF001** | *Autenticación del usuario en el sistema* | * *Permitir el ingreso de usuarios al sistema web, mediante un nombre de usuario y contraseña.* * *Los permisos dependerán del rol que tenga.* | *Administración* | *16* |
| **RF002** | *Gestionar categorías* | * *Permitir registrar, modificar y eliminar una categoría.* * *El código es autogenerado del sistema.* | *Depto. Inventario y administración* | *8* |
| **RF003** | *Gestionar productos* | * *Permitir registrar, modificar y eliminar a un producto y la búsqueda del mismo.* * *El código es autogenerado del sistema.* | *Depto. Inventario y administración* | *12* |
| **RF004** | *Gestionar empresa* | * *Permitir registrar, modificar y eliminar a una empresa y la búsqueda del mismo.* * *El código es autogenerado del sistema.* | *Depto. administración* | *16* |
| **RF005** | *Registrar pedido* | * *Permitir al cliente añadir los productos en el pedido, mientras navega por las categorías.* * *En el momento de confirmación se creará en pedido.* | *Depto. de comercialización* | *24* |
| **RF006** | *Gestionar mesa* | * *Permitir registrar, modificar y eliminar a una mesa y la búsqueda del mismo.* * *El código es autogenerado del sistema.* | *Depto. Inventario y administración* | *8* |
| **RF007** | Registrar Datos de Facturación | * *Permite registrar el nombre completo de la persona y su carnet de identidad* | *Depto. de administración* | *8* |
| **RF008** | Generar Factura | * *Permite generar la factura con los datos enviados desde la aplicación móvil al servidor* | *Depto. de administración* | *8* |
| **RF009** | Monitoreo de Pedido | * *Permite ver los pedidos si fueron atendidos o están pendientes* | *Depto. Administración* | *8* |
| **RF010** | *Reporte: Listado de ventas.* | * *Permite al usuario ver un listado de las ventas.* | *Depto. administración* | *8* |

## 2.2. REQUERIMIENTOS NO FUNCIONALES

### 2.1.1. Tiempo de aprendizaje

La capacitación del personal, en el manejo de software y mantenimiento del sistema deberá considerar un tiempo considerable de por lo menos 40 horas laborales de capacitación (por día mínimo 2 horas.), hasta que los lineamientos de manejo del sistema y otros hayan sido debidamente adquiridos por el personal previamente asignado.

### 2.1.2. Confiabilidad

#### 2.1.2.1. Tiempo de disponibilidad del sistema

La aplicación debe estar disponible las 24 horas todos los días del año, para que los clientes que hagan su pedido y administradores del restaurante.

### 2.1.3. Performance

#### 2.1.3.1. Acceso de clientes en línea

Los usuarios clientes deben poder acceder a los datos en línea, en tiempo real.

#### 2.1.3.2. Tiempo de respuesta

El tiempo de respuesta al acceso del usuario debe ir de 7 segundos, la primera vez que ingresa al sistema, luego menos de 5 segundos.

El tiempo de respuesta para una transacción promedio también debe ser de 10 segundos.

#### 2.1.3.3. Cantidad de atención a usuarios

El sistema debe poder atender normalmente a 40 o más usuarios a la vez.

### 2.1.4. Facilidad de Mantenimiento

Debe aprovecharse la característica de la tecnología Web, en este caso estamos utilizando el servidor JBoss, además estamos ocupando Eclipse IDE en el lenguaje de programación java.

### 2.1.5. Estándares de arquitectura

* Se debe usar para el desarrollo del software la arquitectura en 3 capas.
* Para la infraestructura de red, se debe implementar la arquitectura Cliente/Servidor.

### 2.1.5. Motor de base de datos

Se debe utilizar el motor de base de datos PostgreSQL 9.5

### 2.1.7. Cliente del navegador

La aplicación deberá ser accesible utilizando el siguiente tipo de navegador: Chrome.

### 2.1.8. Servidor Web

El servidor web que se utiliza es JBoss 6.2 es una plataforma middleware poderosa, rápida y segura construida con base en estándares libres y cumple con los requerimientos de la especificación Java.

### 2.1.9. Interfaces

#### 2.1.9.1 Interfaz de usuario

* EL sistema debe poseer una interfaz gráfica basada en el patrón “F” de distribución de componentes de manera intuitiva que permita la fácil interacción con el usuario.
* Mantener una combinación de colores de la interfaz que no afecte la utilización del software con el color Rojo como color corporativo y la gama de colores magenta gray.
* Colocar los botones ó íconos de forma ergonómica que no se le haga difícil al usuario utilizarlos.
* No deben existir presentación de imágenes no distorsionadas o difíciles de entender.
* La presentación de mensajes de error o de información al usuario deberá ser lo más específico posible y comunicarse con el administrador del sistema.

#### 2.1.9.2. Interfaz de Hardware

* Los usuarios necesitarán un dispositivo que mediante una interfaz de red (Ethernet, modem ADLS, etc.) les permita acceder al sistema, vía internet o red interna.
* Este dispositivo puede estar basado en cualquier arquitectura pero deberá disponer al menos de un procesador con capacidad suficiente para ejecutar un navegador Web moderno.

#### 2.1.9.3. Interfaz de Comunicaciones

* Existen dos tipos de conexión entre los usuarios y el servidor dónde está alojado el sistema web:
* Los usuarios clientes se conectarán al sistema web mediante Internet. Esta conexión la realizaran desde la heladería y pastelería.
* Para el **Entorno de Red**: la aplicación deberá tener la capacidad de funcionar en un entorno de red LAN y WAN.

#### 2.1.9.4. Interfaces de Software

* Cualquier usuario que desee conectarse al sistema web no necesitará ningún sistema operativo en concreto para poder acceder a ella. Únicamente bastará con un navegador moderno (Chrome para poder visualizarla.
* Por otra parte el servidor de base de datos (Postgres) y el servidor web (JBoss), se implementará bajo el sistema operativo Windows 8 o superior.
* En la parte móvil el usuario podrá realizar su pedido.

# CAPÍTULO III

# ANÁLISIS

# ANÁLISIS



## ESPECIFICACIÓN DE REQUERMIENTOS DE SOFTWARE

A diferencia de la metodología tradicional, XP utiliza las historias de usuario para la especificación de requisitos, permitiendo disminuir la documentación. XP presenta 4 valores que al seguirlos y utilizarlos facilita la especificación de requerimientos:

* **La comunicación:** permite que el cliente y el programador lleguen a un acuerdo en la especificación de requerimientos evitando los malos entendidos.
* **La sencillez**: es lo que diferencia a XP con las demás metodologías tradicionales las cuales utilizan estándares para la especificación de requerimientos que hacen del sistema muy complejo. La sencillez evita la documentación extensa centrándose en lo básico, en lo que se utiliza en este momento y no en lo que se podrá utilizar.
* **La realimentación**: permite que la especificación de requerimientos se comprenda mejor con el pasar del tiempo, permitiendo que los usuarios aprendan a describir mejor las Historias.
* **Las Historias de usuario**: es una pequeña descripción del programa con el fin de estimar tiempos y costos. Para obtener mayor detalle de las historias de usuario en el momento de la implementación, el programador preguntará al cliente, aumentando el detalle de cada historia.

### 3.1 Historias de usuario

Las historias de usuario permiten obtener los requerimientos del sistema a implementar los primeros requerimientos por parte del usuario, es importante no detallar las historias de usuario porque son utilizadas solo para dar una pequeña visión de lo que se quiere obtener. Cuando se inicie la fase de desarrollo el investigador con la ayuda del usuario detallarán las historias de usuario.

|  |  |  |
| --- | --- | --- |
| **ID** | 001 | |
| **Titulo** | Autenticación de Usuario | |
| **Descripción** | | |
| Permite el ingreso de usuarios al sistema web, mediante un nombre de usuario y contraseña | | |
| **Estimación (HP)** | **Prioridad** | **Dependiente de** |
| 16 | alta |  |
| **Criterios de aceptación** | | |
| * Se podrá iniciar sesión * El Sistema deberá encriptar la contraseña | | |
| **Responsable** | Edson Valdez | |
| **Adjunto** | | |
| Mockups | | |

#### 3.1.1. Historia de usuario Autenticación de Usuario

#### 3.1.2. Historia de usuario para registro de categoría

|  |  |  |
| --- | --- | --- |
| **ID** | 002 | |
| **Nombre** | Gestionar categoría | |
| **Descripción** | | |
| Permite el registro del nombre de la categoría a la que pertenece un determinado producto. | | |
| **Estimación (HP) puntos de historia** | **Prioridad** | **Dependiente de** |
| 8 | media |  |
| **Criterios de aceptación** | | |
| 1. La interfaz de usuario debe seguir el patrón F para la distribución de controles y entendible para usar por el cliente. 2. El código de cada categoría se debe generar de manera automática y debe ser único. 3. El nombre y descripción de la categoría podrá ser modificada en cualquier momento. 4. Es obligatorio colocar el nombre y descripción de una categoría. No se podrá guardar en la base de datos si el campo está vacío. | | |
|  | | |
| **Responsable** | Andrea Carranza | |
| **Adjunto** | | |
| G:\Taller\Prototipos\2\Categoria.png | | |

#### 3.1.3. Historia de usuario para registro de productos

|  |  |  |
| --- | --- | --- |
| **ID** | 003 | |
| **Nombre** | Gestionar producto | |
| **Descripción** | | |
| Permite el registro de datos de un producto. Cada producto deberá contener nombre, descripción, precio, stock, categoría. | | |
| **Estimación (HP) puntos de historia** | **Prioridad** | **Dependiente de** |
| 12 | Media | Gestionar categoría |
| **Criterios de aceptación** | | |
| 1. La interfaz de usuario debe seguir el patrón F para la distribución de controles y entendible para el uso del cliente. 2. El código del producto se debe generar de manera automática y debe ser único. 3. El nombre, la descripción, precio, stock del producto y categoría podrá ser modificada en cualquier momento. 4. Es obligatorio colocar el nombre y descripción de un producto. No se podrá guardar en la base de datos si el campo está vacío. 5. En el precio solo se debe colocar valores numéricos, al colocar letras le aparecerá un mensaje debe ser un número. 6. En el stock solo se debe colocar valores numéricos, al colocar letras le aparecerá un mensaje debe ser un número. | | |
|  | | |
| **Responsable** | Alejandra Soca | |
| **Adjunto** | | |
| G:\Taller\Prototipos\2\Producto.png | | |

|  |  |  |
| --- | --- | --- |
| **ID** | 004 | |
| **Nombre** | Gestionar mesa | |
| **Descripción** | | |
| Permite el registro de datos de una mesa. | | |
| **Estimación (HP) puntos de historia** | **Prioridad** | **Dependiente de** |
| 8 | Media |  |
| **Criterios de aceptación** | | |
| 1. La interfaz de usuario debe seguir el patrón F para la distribución de controles y entendible para usar por el cliente. 2. Al registrar una nueva mesa el código se debe generar de manera automática y debe ser único. 3. El número, la descripción de la mesa podrá ser modificada en cualquier momento. | | |
|  | | |
| **Responsable** | Fátima Condori | |
| **Adjunto** | | |
| G:\Taller\Prototipos\2\Mesa.png | | |

#### 3.1.4. Historia de usuario para registro de mesa

#### 3.1.5. Historia de usuario para registro de pedido

|  |  |  |
| --- | --- | --- |
| **ID** | 005 | |
| **Nombre** | Gestionar pedido | |
| **Descripción** | | |
| **Como** Usuario **quiero** poder realizar mi pedido desde la aplicación móvil en el que pueda detallar los productos y la cantidad **para** poder agilizar mi orden. | | |
| **Estimación (HP)** | **Prioridad** | **Dependiente de** |
| 24 horas | Alta |  |
| **Criterios de aceptación** | | |
| 1. El código del pedido de venta se debe generar de manera automática, y debe ser único para cada pedido de venta.  2. El sistema deberá informar si el artículo que se va a añadir al pedido de venta aún tiene stock disponible. No se puede vender un artículo que no tenga stock.  3. Se debe verificar que el stock del artículo satisfaga la cantidad solicitada del pedido.  4. El cliente que solicita el pedido de venta, puede o no estar registrado en el sistema.  5. Debe permitir buscar un producto.  6. La fecha y hora del pedido de venta por defecto debe ser la fecha actual del sistema.  7. Un pedido de venta no se podrá modificar, si esta anulado, entregado o facturado.  8. Debe permitir consultar las categorías. | | |
| **Responsable** | Edson Valdez | |
| **Adjunto** | | |
| G:\Taller\Prototipos\2\Pedido.png | | |

|  |  |  |
| --- | --- | --- |
| **ID** | 006 | |
| **Nombre** | Gestionar empresa | |
| **Descripción** | | |
| Permite el registro de los datos de la empresa. | | |
| **Estimación (HP) puntos de historia** | **Prioridad** | **Dependiente de** |
| 16 | baja |  |
| **Criterios de aceptación** | | |
| 1. La interfaz de usuario debe seguir el patrón F para la distribución de controles y entendible para usar por el cliente. 2. El código de cada empresa se debe generar de manera automática y debe ser único. 3. El nombre, sucursal, dirección, nit, nro autorización, comentario factura y correo podrán ser actualizados en cualquier momento | | |
|  | | |
| **Responsable** | Ely Condori | |
| **Adjunto** | | |
| G:\Taller\Prototipos\2\Empresa.png | | |

#### 3.1.6. Historia de usuario para registro de empresa

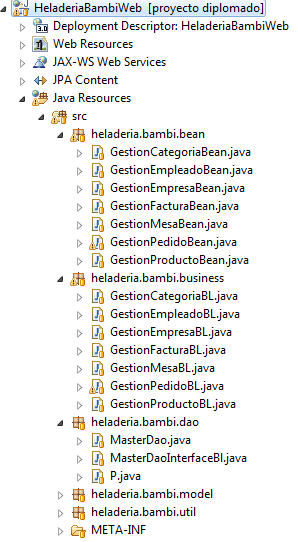
# CAPÍTULO IV

# DISEÑO

# DISEÑO



## 4.1. Arquitectura de la solución para modelo en capas



**Util**

**Model**

**Dato**

**Negocio**

**Presentación**

Ilustración 5 Arquitectura de la Solución

### 4.1.1. Descripción de la arquitectura de la solución

A continuación se hace una breve descripción de los distintos componentes que forman parte de la solución en general.

* **Paquete Presentación**

Esta capa es la que ve el usuario, presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso.

Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser “amigable” para el usuario generalmente se presentan como formularios.

* **Paquete Negocio**

Aquí es donde, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio porque es aquí donde se establecen todas las reglas que deben cumplirse.

Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él.

* **Paquete Dato**

Esta capa se encarga de guardar los datos. Será donde se gestione todo lo relativo a la base de datos y a la creación, edición y borrado de datos de ésta.

* **Model**

Clases de hibernate

* **Util**

Clases auxiliares

## 4.2. Arquitectura de la solución usando modelo n capas

La arquitectura base para las soluciones desarrolladas en la empresa está reflejada en el siguiente gráfico.

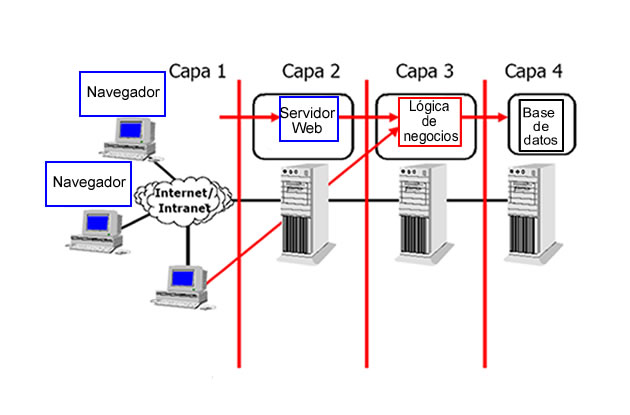


Ilustración 6 Arquitectura de Software

## 4.3. DISEÑO DE BASE DE DATOS



### 4.3.1. Diagrama conceptual de la base de datos

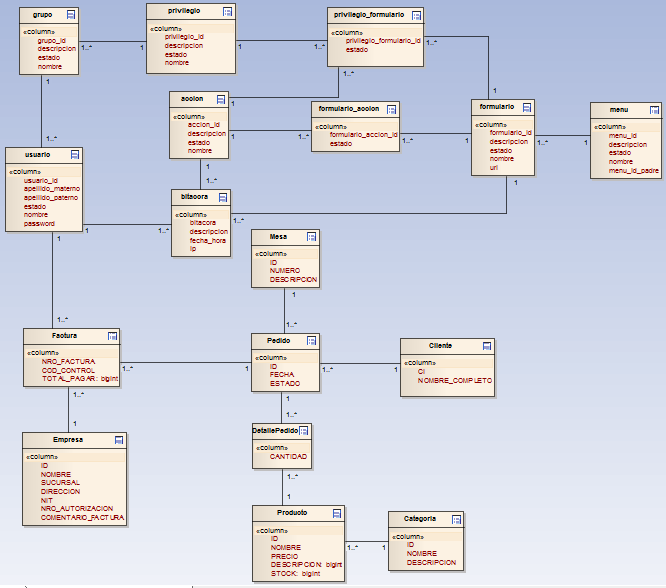


Ilustración 7 Diseño Conceptual de la Base de Datos



### 4.3.2. Modelo Lógico



#### 4.3.2.1. Modelo Lógico relacional de la base de datos



Ilustración 8 Diseño Lógico Relacional de la Base de Datos



#### 4.3.2.2. Diseño lógico específico de la base de datos

**CREATE TABLE accion (**

**accion\_id bigint NOT NULL,**

**descripcion varchar(255),**

**estado boolean,**

**nombre varchar(255)**

**);**

**CREATE TABLE bitacora (**

**bitacora\_id bigint NOT NULL,**

**descripcion varchar(255),**

**fecha\_hora timestamp,**

**ip varchar(255),**

**accion\_id bigint,**

**formulario\_id bigint,**

**usuario\_id bigint**

**);**

**CREATE TABLE Categoria (**

**ID bigint NOT NULL,**

**NOMBRE varchar(100),**

**DESCRIPCION varchar(500)**

**);**

**CREATE TABLE Cliente (**

**CI bigint NOT NULL,**

**NOMBRE\_COMPLETO varchar(500)**

**);**

**CREATE TABLE DetallePedido (**

**CANTIDAD bigint,**

**ID\_PEDIDO bigint,**

**ID\_PRODUCTO bigint**

**);**

**CREATE TABLE Empleado (**

**CI bigint NOT NULL,**

**NOMBRE\_COMPLETO varchar(150),**

**CELULAR varchar(50),**

**CORREO varchar(150)**

**);**

**CREATE TABLE Empresa (**

**ID bigint NOT NULL,**

**NOMBRE varchar(150),**

**SUCURSAL bigint,**

**DIRECCION varchar(150),**

**NIT varchar(50),**

**NRO\_AUTORIZACION varchar(50),**

**COMENTARIO\_FACTURA varchar(500)**

**);**

**CREATE TABLE Factura (**

**NRO\_FACTURA varchar(150),**

**COD\_CONTROL varchar(50),**

**TOTAL\_PAGAR decimal(10,2),**

**ID\_EMPRESA bigint,**

**ID\_PEDIDO bigint,**

**ID\_EMPLEADO bigint,**

**usuario\_id bigint**

**);**

**CREATE TABLE formulario (**

**formulario\_id bigint NOT NULL,**

**descripcion varchar(255),**

**estado boolean,**

**nombre varchar(255),**

**url varchar(255),**

**menu\_id bigint**

**);**

**CREATE TABLE formulario\_accion (**

**formulario\_accion\_id bigint NOT NULL,**

**estado boolean,**

**accion\_id bigint,**

**formulario\_id bigint**

**);**

**CREATE TABLE grupo (**

**grupo\_id bigint NOT NULL,**

**descripcion varchar(255),**

**estado boolean,**

**nombre varchar(255),**

**privilegio\_id bigint**

**);**

**CREATE TABLE menu (**

**menu\_id bigint NOT NULL,**

**descripcion char(100),**

**estado boolean,**

**nombre char(50),**

**menu\_id\_padre bigint**

**);**

**CREATE TABLE Mesa (**

**ID bigint NOT NULL,**

**NUMERO varchar(50),**

**DESCRIPCION varchar(150)**

**);**

**CREATE TABLE Pedido (**

**ID bigint NOT NULL,**

**FECHA\_HORA timestamp,**

**ESTADO varchar(50),**

**CI\_CLIENTE bigint,**

**ID\_MESA bigint**

**);**

**CREATE TABLE privilegio (**

**privilegio\_id bigint NOT NULL,**

**descripcion varchar(255),**

**estado boolean,**

**nombre varchar(255)**

**);**

**CREATE TABLE privilegio\_formulario (**

**privilegio\_formulario\_id bigint NOT NULL,**

**estado boolean,**

**accion\_id bigint,**

**formulario\_id bigint,**

**privilegio\_id bigint**

**);**

**CREATE TABLE Producto (**

**ID bigint NOT NULL,**

**NOMBRE varchar(500),**

**PRECIO decimal(10,2),**

**DESCRIPCION varchar(500),**

**STOCK bigint,**

**ID\_CATEGORIA bigint**

**);**

**CREATE TABLE usuario (**

**usuario\_id bigint NOT NULL,**

**apellido\_materno varchar(255),**

**apellido\_paterno varchar(255),**

**estado boolean,**

**login varchar(255),**

**password varchar(255),**

**grupo\_id bigint**

**);**

**ALTER TABLE accion ADD CONSTRAINT PK\_accion**

**PRIMARY KEY (accion\_id)**

**;**

**ALTER TABLE bitacora ADD CONSTRAINT PK\_bitacora**

**PRIMARY KEY (bitacora\_id)**

**;**

**ALTER TABLE Categoria ADD CONSTRAINT PK\_Categoria**

**PRIMARY KEY (ID)**

**;**

**ALTER TABLE Cliente ADD CONSTRAINT PK\_Cliente**

**PRIMARY KEY (CI)**

**;**

**ALTER TABLE Empleado ADD CONSTRAINT PK\_Empleado**

**PRIMARY KEY (CI)**

**;**

**ALTER TABLE Empresa ADD CONSTRAINT PK\_Empresa**

**PRIMARY KEY (ID)**

**;**

**ALTER TABLE formulario ADD CONSTRAINT PK\_formulario**

**PRIMARY KEY (formulario\_id)**

**;**

**ALTER TABLE formulario\_accion ADD CONSTRAINT PK\_formulario\_accion**

**PRIMARY KEY (formulario\_accion\_id)**

**;**

**ALTER TABLE grupo ADD CONSTRAINT PK\_grupo**

**PRIMARY KEY (grupo\_id)**

**;**

**ALTER TABLE menu ADD CONSTRAINT PK\_menu**

**PRIMARY KEY (menu\_id)**

**;**

**ALTER TABLE Mesa ADD CONSTRAINT PK\_Mesa**

**PRIMARY KEY (ID);**

**ALTER TABLE Pedido ADD CONSTRAINT PK\_Pedido**

**PRIMARY KEY (ID)**

**;**

**ALTER TABLE privilegio ADD CONSTRAINT PK\_privilegio**

**PRIMARY KEY (privilegio\_id)**

**;**

**ALTER TABLE privilegio\_formulario ADD CONSTRAINT PK\_privilegio\_formulario**

**PRIMARY KEY (privilegio\_formulario\_id)**

**;**

**ALTER TABLE Producto ADD CONSTRAINT PK\_Producto**

**PRIMARY KEY (ID)**

**;**

**ALTER TABLE usuario ADD CONSTRAINT PK\_usuario**

**PRIMARY KEY (usuario\_id)**

**;**

**ALTER TABLE bitacora ADD CONSTRAINT FK\_bitacora\_accion**

**FOREIGN KEY (accion\_id) REFERENCES accion (accion\_id)**

**;**

**ALTER TABLE bitacora ADD CONSTRAINT FK\_bitacora\_formulario**

**FOREIGN KEY (formulario\_id) REFERENCES formulario (formulario\_id)**

**;**

**ALTER TABLE bitacora ADD CONSTRAINT FK\_bitacora\_usuario**

**FOREIGN KEY (usuario\_id) REFERENCES usuario (usuario\_id)**

**;**

**ALTER TABLE DetallePedido ADD CONSTRAINT FK\_DetallePedido\_Pedido**

**FOREIGN KEY (ID\_PEDIDO) REFERENCES Pedido (ID)**

**;**

**ALTER TABLE DetallePedido ADD CONSTRAINT FK\_DetallePedido\_Producto**

**FOREIGN KEY (ID\_PRODUCTO) REFERENCES Producto (ID)**

**;**

**ALTER TABLE Factura ADD CONSTRAINT FK\_Factura\_Empleado**

**FOREIGN KEY (ID\_EMPLEADO) REFERENCES Empleado (CI);**

**ALTER TABLE Factura ADD CONSTRAINT FK\_Factura\_Empresa**

**FOREIGN KEY (ID\_EMPRESA) REFERENCES Empresa (ID)**

**;**

**ALTER TABLE Factura ADD CONSTRAINT FK\_Factura\_Pedido**

**FOREIGN KEY (ID\_PEDIDO ) REFERENCES Pedido (ID)**

**;**

**ALTER TABLE Factura ADD CONSTRAINT FK\_Factura\_usuario**

**FOREIGN KEY (usuario\_id) REFERENCES usuario (usuario\_id)**

**;**

**ALTER TABLE formulario ADD CONSTRAINT FK\_formulario\_menu**

**FOREIGN KEY (menu\_id) REFERENCES menu (menu\_id)**

**;**

**ALTER TABLE formulario\_accion ADD CONSTRAINT FK\_formulario\_accion\_accion**

**FOREIGN KEY (accion\_id) REFERENCES accion (accion\_id)**

**;**

**ALTER TABLE formulario\_accion ADD CONSTRAINT FK\_formulario\_accion\_formulario**

**FOREIGN KEY (formulario\_id) REFERENCES formulario (formulario\_id)**

**;**

**ALTER TABLE grupo ADD CONSTRAINT FK\_grupo\_privilegio**

**FOREIGN KEY (privilegio\_id) REFERENCES privilegio (privilegio\_id)**

**;**

**ALTER TABLE Pedido ADD CONSTRAINT FK\_Pedido\_Cliente**

**FOREIGN KEY (CI\_CLIENTE ) REFERENCES Cliente (CI)**

**;**

**ALTER TABLE Pedido ADD CONSTRAINT FK\_Pedido\_Mesa**

**FOREIGN KEY (ID\_MESA ) REFERENCES Mesa (ID)**

**;**

**ALTER TABLE privilegio\_formulario ADD CONSTRAINT FK\_privilegio\_formulario\_accion**

**FOREIGN KEY (accion\_id) REFERENCES accion (accion\_id)**

**;**

**ALTER TABLE privilegio\_formulario ADD CONSTRAINT FK\_privilegio\_formulario\_formulario**

**FOREIGN KEY (formulario\_id) REFERENCES formulario (formulario\_id)**

**;**

**ALTER TABLE privilegio\_formulario ADD CONSTRAINT FK\_privilegio\_formulario\_privilegio**

**FOREIGN KEY (privilegio\_id) REFERENCES privilegio (privilegio\_id)**

**;**

**ALTER TABLE Producto ADD CONSTRAINT FK\_Producto\_Categoria**

**FOREIGN KEY (ID\_CATEGORIA) REFERENCES Categoria (ID)**

**;**

**ALTER TABLE usuario ADD CONSTRAINT FK\_usuario\_grupo**

**FOREIGN KEY (grupo\_id) REFERENCES grupo (grupo\_id)**

**;**

# CAPÍTULO V

# IMPLEMENTACIÓN



## 5.1. APLICACIÓN WEB

### 5.1.1. Implementación de presentación Categoria

package heladeria.bambi.bean;

import heladeria.bambi.business.GestionCategoriaBL;

import heladeria.bambi.model.Categoria;

import heladeria.bambi.util.SysMessage;

import java.io.Serializable;

import java.util.List;

import javax.annotation.PostConstruct;

import javax.faces.bean.ManagedBean;

import javax.faces.bean.ViewScoped;

import javax.inject.Inject;

@ManagedBean

@ViewScoped

public class GestionCategoriaBean implements Serializable {

private static final long serialVersionUID = 1L;

@Inject

private GestionCategoriaBL bl;

private List<Categoria> lst;

private Categoria selected;

private Categoria entidad;

private boolean nuevo;

private boolean visibleNuevoEditar;

@PostConstruct

public void init() {

cargarLst();

nuevo();

}

public void nuevo() {

entidad = new Categoria();

nuevo = true;

visibleNuevoEditar = false;

}

public void nuevaEntidad() {

nuevo();

visibleNuevoEditar = true;

}

@SuppressWarnings("unchecked")

private void cargarLst() {

try {

lst = bl.findAll();

} catch (Exception e) {

e.printStackTrace();

}

}

public void guardar() {

if (entidad != null) {

String validacion = bl.validar(entidad, nuevo);

if (validacion.trim().isEmpty()) {

if (nuevo) {

try {

bl.save(entidad);

SysMessage.info("Guardado correctamente.");

nuevo();

cargarLst();

visibleNuevoEditar = true;

} catch (Exception e) {

SysMessage.error("Fallo al guardar.");

}

} else {

try {

bl.update(entidad);

SysMessage.info("Guardado correctamente.");

nuevo();

cargarLst();

visibleNuevoEditar = false;

} catch (Exception e) {

SysMessage.error("Fallo al guardar.");

}

}

} else {

SysMessage.warn("Se encontro observaciones al intentar guardar: " + validacion);

}

}

}

public void editar() {

if (selected != null) {

entidad = selected;

nuevo = false;

visibleNuevoEditar = true;

} else {

SysMessage.warn("No se encontro ningun registro seleccionado.");

}

}

public void editar(Categoria obj) {

selected = obj;

editar();

}

public void eliminar() {

if (selected != null) {

try {

bl.remove(selected);

SysMessage.info("Eliminado correctamente.");

cargarLst();

} catch (Exception e) {

SysMessage.info("Fallo al eliminar.");

}

} else {

SysMessage.warn("No se encontro ningun registro seleccionado.");

}

}

public void eliminar(Categoria obj) {

selected = obj;

eliminar();

}

public List<Categoria> getLst() {

return lst;

}

public void setLst(List<Categoria> lst) {

this.lst = lst;

}

public Categoria getSelected() {

return selected;

}

public void setSelected(Categoria selected) {

this.selected = selected;

}

public Categoria getEntidad() {

return entidad;

}

public void setEntidad(Categoria entidad) {

this.entidad = entidad;

}

public boolean isNuevo() {

return nuevo;

}

public void setNuevo(boolean nuevo) {

this.nuevo = nuevo;

}

public boolean isVisibleNuevoEditar() {

return visibleNuevoEditar;

}

public void setVisibleNuevoEditar(boolean visibleNuevoEditar) {

this.visibleNuevoEditar = visibleNuevoEditar;

}

}

### 5.1.2. Implementación de negocio Categoria

package heladeria.bambi.business;

import heladeria.bambi.dao.MasterDao;

import heladeria.bambi.dao.MasterDaoInterfaceBl;

import heladeria.bambi.dao.P;

import heladeria.bambi.model.Categoria;

import java.io.Serializable;

import java.math.BigDecimal;

import java.util.List;

import javax.inject.Inject;

import javax.inject.Named;

@Named

public class GestionCategoriaBL implements Serializable, MasterDaoInterfaceBl {

private static final long serialVersionUID = 1L;

@Inject

private MasterDao masterDao;

public String validar(Object entidad, boolean nuevo) {

if (entidad != null && entidad instanceof Categoria) {

StringBuilder sb = new StringBuilder();

return sb.toString();

} else {

return "No se pudo validar el objeto.";

}

}

@SuppressWarnings("unchecked")

public int getCantidadConNombre(String nombre, long promocionId) throws Exception {

String sql = "select count(0) as total from Categoria where estado = 1 and nombre = :nombre and promocion\_id = :promocionId";

P p = P.getP();

p.put("nombre", nombre);

p.put("promocionId", promocionId);

List<BigDecimal> l = masterDao.findAllNativeQuery(sql, p);

return l != null && l.size() > 0 ? l.get(0).intValue() : 0;

}

public void save(Object entidad) throws Exception {

masterDao.save(entidad);

}

public void remove(Object entidad) throws Exception {

masterDao.remove(entidad);

}

public void update(Object entidad) throws Exception {

masterDao.update(entidad);

}

@SuppressWarnings("rawtypes")

public Object find(Object entityKey, Class clase) throws Exception {

return masterDao.find(entityKey, clase);

}

@SuppressWarnings({ "rawtypes" })

public List findAll() throws Exception {

String sql = "select l from Categoria l";

return masterDao.findAllQuery(Categoria.class, sql, null);

}

}



### Implementación de dato

package heladeria.bambi.dao;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import javax.annotation.Resource;

import javax.inject.Named;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import javax.persistence.Query;

import javax.transaction.UserTransaction;

@Named

public class MasterDao implements Serializable {

private static final long serialVersionUID = 1L;

@PersistenceContext

private transient EntityManager entityManager;

@Resource

private transient UserTransaction transaction;

public void save(Object entidad) throws Exception {

transaction.begin();

entityManager.joinTransaction();

try {

entityManager.persist(entidad);

transaction.commit();

} catch (Exception e) {

transaction.rollback();

throw e;

}

}

public void remove(Object entidad) throws Exception {

transaction.begin();

entityManager.joinTransaction();

try {

entityManager.remove(entityManager.contains(entidad) ? entidad : entityManager.merge(entidad));

transaction.commit();

} catch (Exception e) {

transaction.rollback();

throw e;

}

}

public void update(Object entidad) throws Exception {

transaction.begin();

entityManager.joinTransaction();

try {

entityManager.merge(entidad);

transaction.commit();

} catch (Exception e) {

transaction.rollback();

throw e;

}

}

@SuppressWarnings({ "unchecked", "rawtypes" })

public Object find(Object entityKey, Class clase) throws Exception {

try {

return entityManager.find(clase, entityKey);

} catch (Exception e) {

throw e;

}

}

@SuppressWarnings({ "rawtypes", "unchecked" })

public List findAllQuery(Class clase, String sql, P p) throws Exception {

Map<String, Object> parametros = p != null ? p.getParametros() : null;

try {

Query query = entityManager.createQuery(sql, clase);

if (parametros != null) {

List<String> keys = new ArrayList<String>(parametros.keySet());

for (String key : keys) {

query.setParameter(key, parametros.get(key));

}

}

return query.getResultList();

} catch (Exception e) {

throw e;

}

}

@SuppressWarnings({ "rawtypes" })

public List findAllNativeQuery(String sql, P p) throws Exception {

Map<String, Object> parametros = p != null ? p.getParametros() : null;

try {

Query query = entityManager.createNativeQuery(sql);

if (parametros != null) {

List<String> keys = new ArrayList<String>(parametros.keySet());

for (String key : keys) {

query.setParameter(key, parametros.get(key));

}

}

return query.getResultList();

} catch (Exception e) {

throw e;

}

}

@SuppressWarnings({ "rawtypes" })

public List findAllNativeQuery(Class clase, String sql, P p) throws Exception {

Map<String, Object> parametros = p != null ? p.getParametros() : null;

try {

Query query = entityManager.createNativeQuery(sql, clase);

if (parametros != null) {

List<String> keys = new ArrayList<String>(parametros.keySet());

for (String key : keys) {

query.setParameter(key, parametros.get(key));

}

}

return query.getResultList();

} catch (Exception e) {

throw e;

}

}

public void executeUpdateNativeQuery(String sql, P p) throws Exception {

Map<String, Object> parametros = p != null ? p.getParametros() : null;

transaction.begin();

entityManager.joinTransaction();

try {

Query query = entityManager.createNativeQuery(sql);

if (parametros != null) {

List<String> keys = new ArrayList<String>(parametros.keySet());

for (String key : keys) {

query.setParameter(key, parametros.get(key));

}

}

query.executeUpdate();

transaction.commit();

} catch (Exception e) {

transaction.rollback();

throw e;

}

}

}

### 5.1.5. Modelo de despliegue para la solución

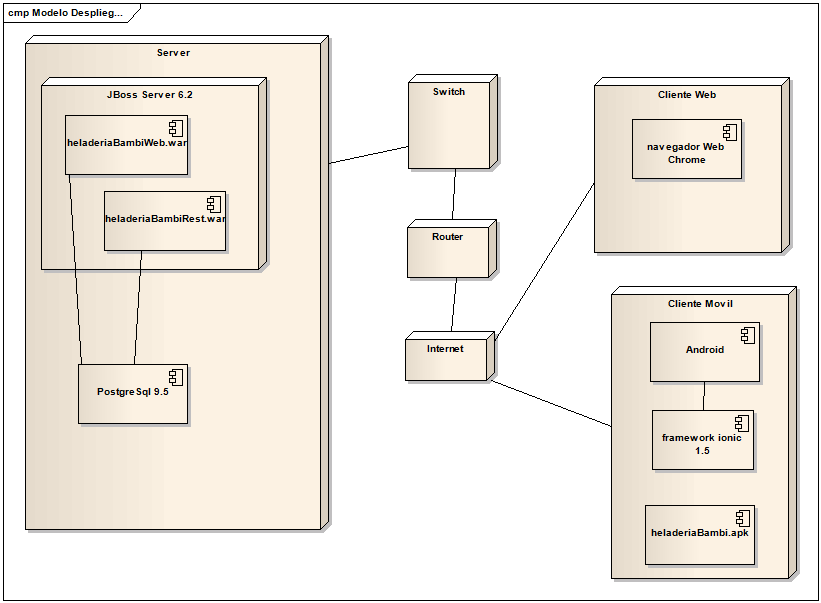


Ilustración 9 Modelo de Despliegue



# CAPÍTULO VI

# PRUEBA

# 6.1. PRUEBAS DE SOFTWARE

|  |
| --- |
| TEST CASE |
| Identificador: 002  Título: Probar que guarde una categoría correctamente.  Descripción: Introducir los datos de una categoria. |
| Pasos |
| 1. Ejecutar la aplicación  2. Ingresar a Categoría  3. Seleccionar registrar nueva categoría  4. Ingresar los datos de la categoría (Nombre , Descripcion) El código se genera automáticamente  Si deja nombre en blanco no le permitirá registrar la categoría  5. Si no indica que el nombre está en blanco. No pasa la prueba  6. Seleccionar Guardar  7. Mostrará un mensaje del nuevo registro. “Se ha registrado de forma exitosa”  8. Mostrará la lista de las categorías |
| Resultado Esperado |
| * Deberá indicar complete este campo. |
|  |
|  |

|  |
| --- |
| TEST CASE |
| Identificador: 003  Título: Probar que guarde los datos requeridos en el registrar producto  Descripción: Introducir los datos del nuevo producto que se registrara. |
| Pasos |
| 1. Ejecutar la aplicación  2. Abrir la opción productos  3. Presionar registrar producto(mostrara un formulario para rellenar datos de un nuevo producto)  4. Rellenar los campos (seleccionar). Todos los campos deberán estar rellenados(el código se generara automáticamente)  Si hay campos vacíos mostrara un mensaje de “completa este campo” en el campo vacío  5. Seleccionar guardar  7. validar los datos  Si hay campos que no corresponden a su tipo de dato o validación correspondiente mostrara mensajes de lo que no se pudieron validar, por ejemplo   * El campo stock debe ser un número. (solo admite valor numérico) * El campo precio debe ser un número. (solo admite valor numérico)   Seleccionar Guardar  8. Mostrará un mensaje del nuevo registro. “Se ha registrado de forma exitosa”  9. Mostrará la lista de productos con su correspondiente imagen |
| Resultado Esperado |
|  |
| ¿Automatizado?  no  Autor Alejandra  Cuando se ejecutan |
|  |

|  |
| --- |
| TEST CASE |
| Identificador: 005  Título: Probar que no se vendan productos con cantidad inferior a la solicitada en el pedido.  Descripción: Seleccionar un detalle, con una cantidad de productos con una cantidad de alimentos disponibles en la heladeria. |
| Pasos |
| 1. Ejecutar la aplicación  2. Buscar en el listado de pedido uno en particular, seleccionarlo e ingresar la cantidad de productos inferior a la cantidad en stock del producto.  3. El sistema debe de decirle inmediatamente de que el producto ya se terminó, o no cumple con la cantidad específica de lo solicitado.  4. Al especificar un detalle, debe de disminuir la cantidad disponible de productos.  5. Seleccionar el mismo producto y agregar al pedido. |
| Resultado Esperado |
| * Deberá indicar aumentar la cantidad del producto en el detalle del carrito y actualizar los demás datos del mismo. |
| ¿Automatizado?  Sí  Cuando se ejecutan: Al momento de hacer el pedido  Resultado Actual: Satisfactorio |
|  |

|  |
| --- |
| TEST CASE |
| Identificador: 004  Título: Gestionar Mesa  Descripción: Permite registrar, editar y dar de baja el registro de una mesa. |
| Pasos |
| REGISTRAR UN NUEVO ELEMENTO  1. Abrir la aplicación  2. Ejecutar la opción Mesas  3. El sistema muestra una lista de todas las mesas que están disponible.  4. Marcar el botón Registrar mesas nuevas y llenar lo campos de:  Numero : nro 1  Descripción: Mesa Nro. 1  5. Presionar registrar  EDITAR UN ELEMENTO REGISTRADO  1. Abrir la aplicación  2. Ejecutar la opción Mesas  3. El sistema muestra una lista de todas las mesas que están disponible.  4. Seleccionar la fila que necesita modificar y marcar el icono con imagen de lápiz  5. El sistema muestra los campos de NUMERO y DESCRIPCION para poder modificar. |
|  |
|  |

# 7. REFERENCIAS

## 7.1. BIBLIOGRAFÍA

***- Schwaber, Ken;***[***Sutherland, Jeff***](http://en.wikipedia.org/wiki/Jeff_Sutherland)*2012 - Software in 30 Days: How Agile Managers Beat the Odds, Delight Their Customers, And Leave Competitors In the Dust*

**- Takeuchi, H y Nonaka, I** *1986 - The New Product Development Game.*

**- Kent Beck***1999 - Extreme Programming explained: Embrace Change.*

***- Fernández Escribano, Gerardo*** *2002- Ingeniería de Software II. Introducción a Extreme Programming.*

***- Palacio, Juan*** *2007 - Flexibilidad con Scrum, principios de diseño e implantación en campos Scrum.*

**- Kniberg Pr, H., & SUTHERLAND-M.** *(s.f.). SCRUM y XP desde las trincheras.*

**- Schwaber, k., & Sutherland**, **j**. (2012*). SCRUM\_GUIDE. UUEE: SCRUM Aliance.*

## 7.2. ENLACES

* + *Metodología Scrum. Recuperado en:*[*https://www.softeng.es/es-es/empresa/metodologias-de-*](https://www.softeng.es/es-es/empresa/metodologias-de-)*trabajo/metodologia-scrum.html*
  + *Mountain Goat Software. Scrum. Recuperado en https://www.mountaingoatsoftware.com/agile/scrum*

1. El Número de Identificación Tributaria (**NIT**) es un código privado, secreto e intransferible que solamente debe conocer el contribuyente.  [↑](#footnote-ref-1)