ROTISOFT

Sistema de administración de rotiserías

Utilización de Patrones

Comida Árabe

Utilización de Patrones

Versión 1.1

Historia de Revisión

Fecha	Versión	Descripción	Autor
09/10/2013	1.0	Creación del documento	Matías
11/11/2013	1.1	Se agregan detalles de patrones	Matías

Tabla de Contenidos

INTRODUCCIÓN 5

PROPÓSITO 5
AUDIENCIA 5

PATRONES 6

N-TIER CLIENT – SERVER: 6
PUBLISHER – SUBSCRIBER: 7
PATRÓN – MVP 8
PATRÓN – SINGLETON 9
ENTITY FRAMEWORK 10

Introducción

Propósito

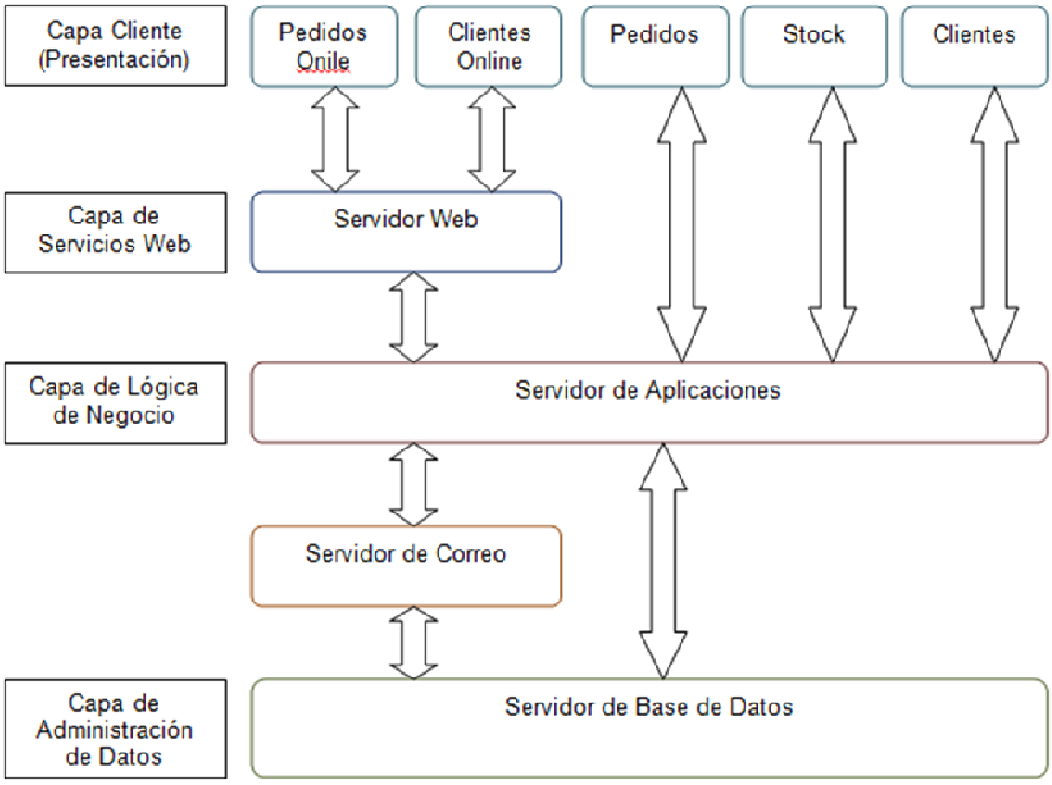
El presente documento permite conocer los patrones utilizados en el diseño y desarrollo del sistema informático Rotisoft.

Audiencia

El documento se encuentra dirigido al equipo de desarrollo del software Rotisoft y al departamento de sistemas del cliente si lo hubiese.

Patrones

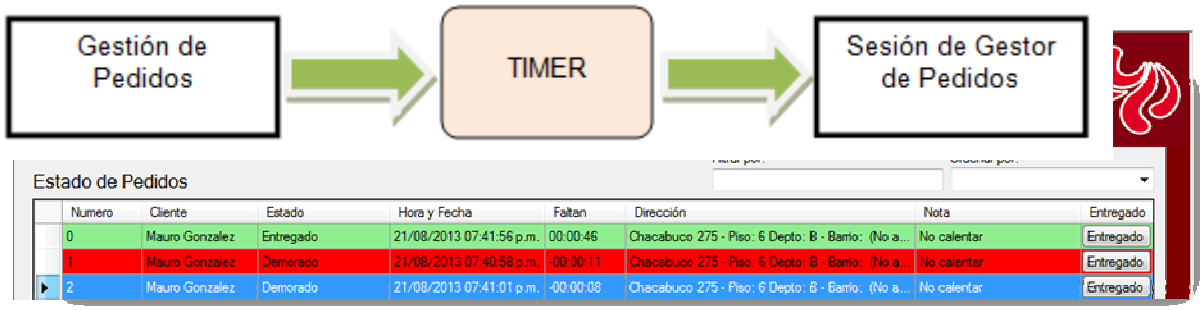
N-Tier Client – Server:



Explicación:

Este patrón fue utilizado para armar la arquitectura del sistema Rotisoft, la cual al contar con una interfaz web y otra de escritorio requiere de una división en capas para simplificar su desarrollo y su posterior mantenimiento.
Esta división en capas se ve mejor explicada en el diagrama de despliegue, el cual muestra cada uno de los componentes del sistema Rotisoft y cómo interactúan entre sí.
Además al utilizar un framework de persistencia como el Entity Framework, pudimos separar la lógica de negocio y modelo de objetos de la implementación relacional de entidades en la base de datos.

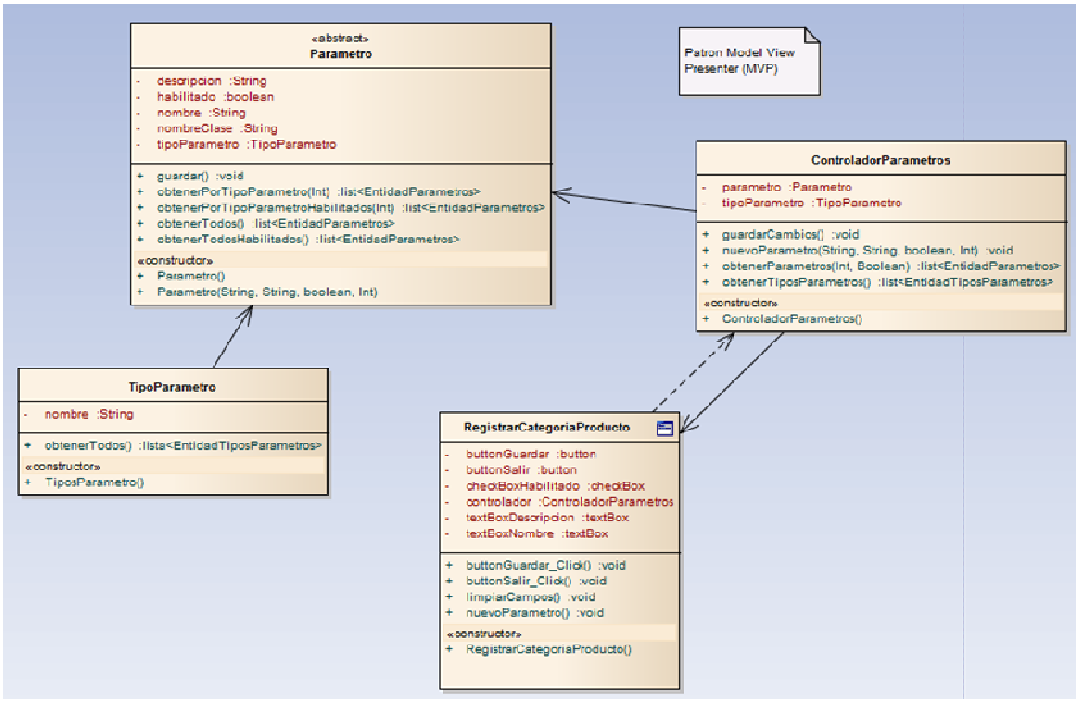
Publisher – Subscriber:



Explicación:

En la pantalla de Estado de Pedidos se implementó el patrón "**Publisher - Subscriber**" que funciona de la siguiente forma:
Existe un **Timer** que a intervalos regulares de tiempo revisa si existen nuevos Pedidos Web o si el estado de algún pedido cambió a "Demorado", dependiendo de su fecha y hora de entrega.
Esto lo hace en base a la información que la **Gestión de Pedidos** le entrega, ya que esta es quien conoce los pedidos y el estado de ellos.
Una vez se dispara algún evento, ya sea porque hay un pedido web nuevo o porque cambió el estado a Demorado de algún pedido, el "**Timer**" le notifica a la **Sesión de Gestor de Pedidos** para que la misma se actualice mostrando la información en pantalla, cambiando los colores del pedido o agregando el nuevo pedido web a la lista.

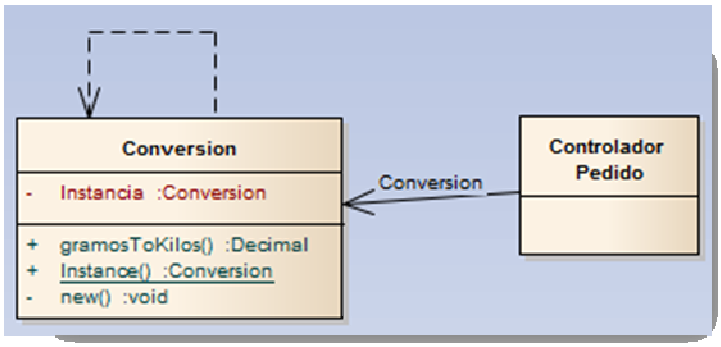
Patrón – MVP



Explicación:

El patrón MVP (Modelo – Vista – Presentador) fue utilizado a lo largo de todas las pantallas de escritorio para separar la lógica de negocio de los formularios de usuario.
Para esto por cada paquete de negocio se creó un Controlador que conoce las clases de negocio y sus métodos y es quien trabaja de nexo entre los disparadores de la pantalla y las acciones que lleva a cabo las clases de negocio.
De esta forma las pantallas no se comunican directamente con las clases de negocio y se hace más fácil su mantenimiento o su posterior reemplazo.

Patrón – Singleton

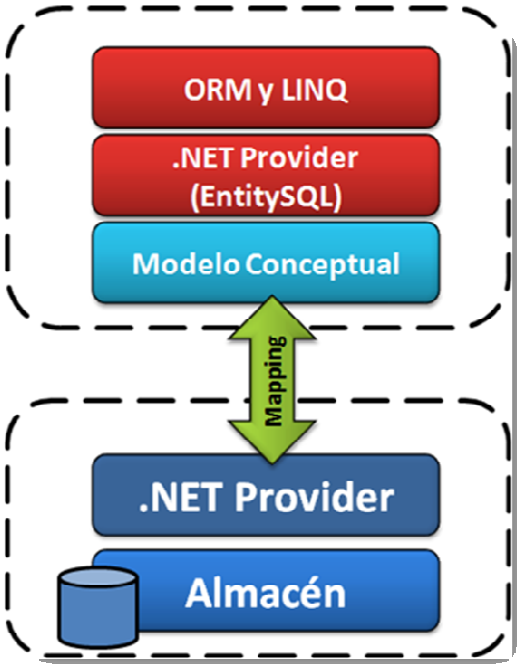



```
//Metodo de Acceso a la clase
Conversion clase = Conversion.Instance;
clase.gramosToKilos(1000);
```

Explicación:

El patrón Singleton fue utilizado en aquellas clases de las cuales se necesitaba que en todo momento haya una sola instancia.
La pantalla Principal es un Singleton ya que en todo momento se debe tener solo una y además esto facilitó la obtención de variables gobales utilizadas en el sistema.
Otro ejemplo de la utilización de este patrón fue la clase Conversión, la cual es utilizada para convertir unidades de una medida a otra.
Por último el Controlador de Sesión también fue necesario que sea único ya que contiene la información del usuario de aplicación actual, así como sus roles y a donde puede acceder el mismo.

Entity Framework



Explicación:

La persistencia en la base de datos fue implementada a través del Entity Framework, el cual facilita la traducción del modelo orientado a objetos a las entidades relacionales de la base.
Este Framework del cual Microsoft es propietario, está diseñado para tener una fácil integración con la herramienta de desarrollo Visual Studio y provee de los métodos necesarios para hacer persistente una clase de C#.
En el diseño de Rotisoft, cada clase de negocio contiene una Entidad la cual está asociada con una tabla relacional, esto permite que la clase implemente los métodos necesarios para transformarse en Entidad y a su vez dicha Entidad conoce que comandos ejecutar en la base para finalmente hacerse persistente, proveyendo de mucha flexibilidad y sencillez al desarrollo ya que el desarrollador no necesita conocer los comandos para insertar un registro en la base de datos y haciendo el código independiente de la tecnología de la base de datos.