DP1 2020-2021

Documento de Diseño del Sistema

Proyecto Roto's Pizza

https://github.com/gii-is-DP1/dp1-2020-g2-09.git

Miembros:

- Figueroa Gómez Servando
- García Cáceres María
- Parrado Gordón Raúl
- Roldán Cadena Jesús
- Sánchez González Álvaro
- Torres Gómez Servando

Tutor: Irene Bedilia Estrada Torres

GRUPO G2-9

Versión 1.2

10/01/2021

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
06/01/2021	V1	• Creación del documento	3
08/01/2021	V1.1	• Añadido diagrama de dominio/diseño	3
10/01/2021	V1.2	• Añadido diagrama de capas	3

Contents

Introducción

El proyecto en el que vamos a trabajar se trata de un sistema de información web de una pizzería llamada Roto's. El **sistema** ofrecerá las siguientes funcionalidades: realización de pedidos y control de su estado, control de la ocupación del local mediante la reserva de mesas y por último control del stock de los productos e ingredientes disponibles.

La **realización de pedidos online** tiene como objetivo ofrecer al cliente una forma cómoda y sencilla de disfrutar de los productos sin necesidad de consumirlos en el local. Para ello, se llevarán a cabo los siguientes pasos:
1.- Elegir los productos deseados y añadirlos al carrito.
2.- Añadir las promociones posibles para el pedido. Las promociones se introducirían mediante un código que aplicaría un determinado descuento al precio total de los productos añadidos al carrito, siempre que se cumplan ciertas condiciones.
3.- Elegir entre servicio a domicilio y recogida en el restaurante. En el caso de servicio a domicilio, el cliente deberá especificar su dirección y el método de pago (efectivo/tarjeta).
4.- Resumen del pedido. Se revisan si los datos introducidos son correctos, pudiéndolos editar en caso necesario, y se confirma o se cancela el pedido.
5.- Una vez finalizado el pedido, el cliente dispondrá de un pizza tracker que le informará del estado del pedido: Aceptado - En preparación - En camino - Entregado.

El **sistema de reserva de mesas** consistirá en especificar el número de personas que van a asistir y la hora de reserva. De este modo, dada la situación actual provocada por la pandemia de la COVID-19, se podrá controlar fácilmente el aforo disponible. Además, la ocupación de mesas mediante reservas eliminará en el cliente la duda de saber si va a encontrar mesa o no a la hora de ir al restaurante.

En relación con el **control del stock** de los ingredientes y productos, se pretende alcanzar dos objetivos. Por un lado, facilitar el trabajo a los cocineros. Para ello, el sistema mostrará en todo momento qué ingredientes se encuentran disponibles para la elaboración de los productos. Por otro lado, ofrecer al cliente los productos de la carta disponibles cuando realice el pedido.

Por último, el sistema que vamos a desarrollar va dirigido a dos perfiles:

- **Empleados de la pizzería**: administrador, cocineros y repartidores.
- **Clientes**.

Al existir dos perfiles bastante diferenciados, el sistema ofrecerá a cada uno una interfaz de usuario distinta, ofreciendo diferentes funcionalidades en cada caso.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

Hemos omitido las generalizaciones hacia BaseEntity y NamedEntity para simplificar el diagrama. Además, todas las clases poseen un validador específico.

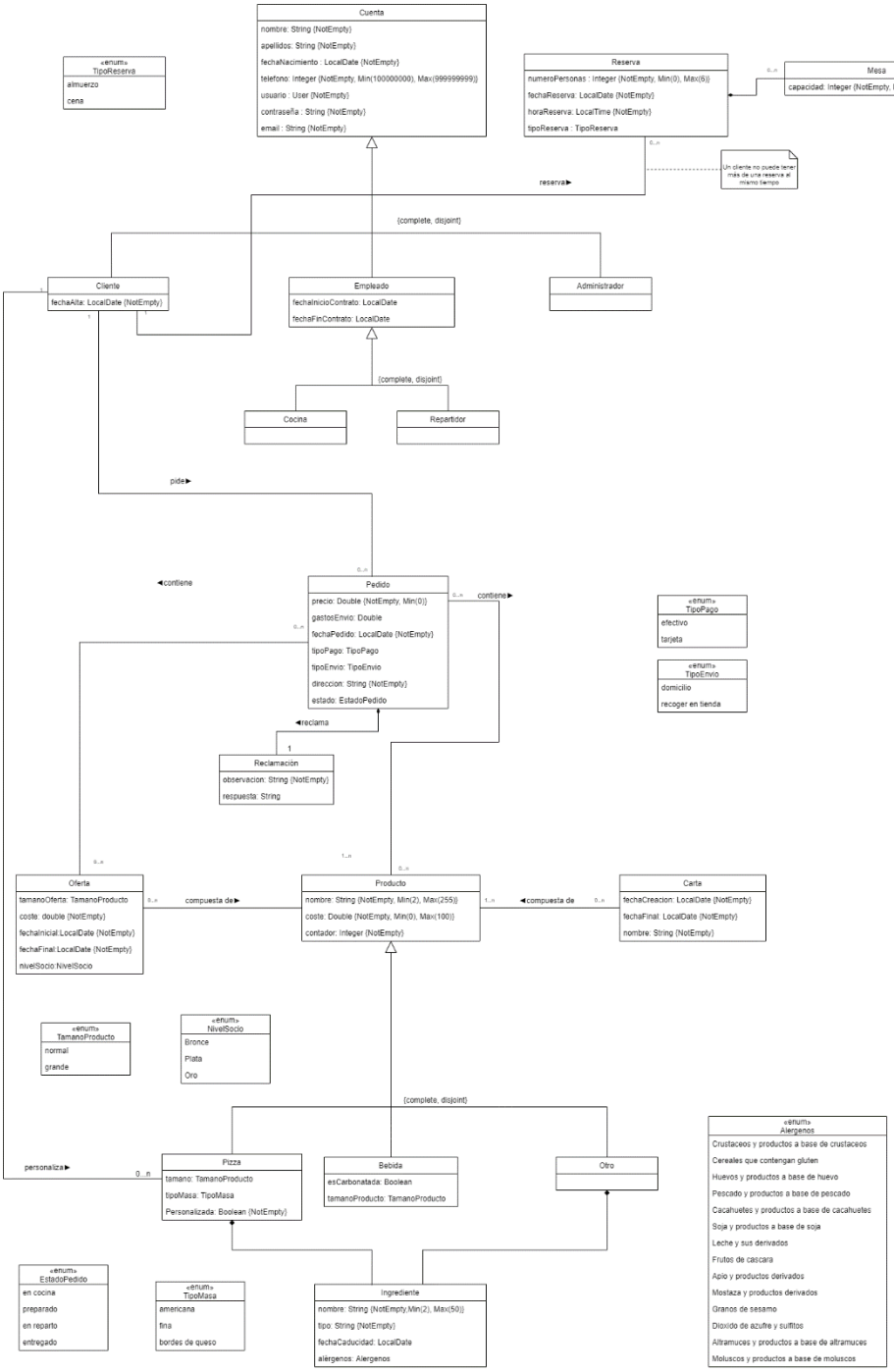
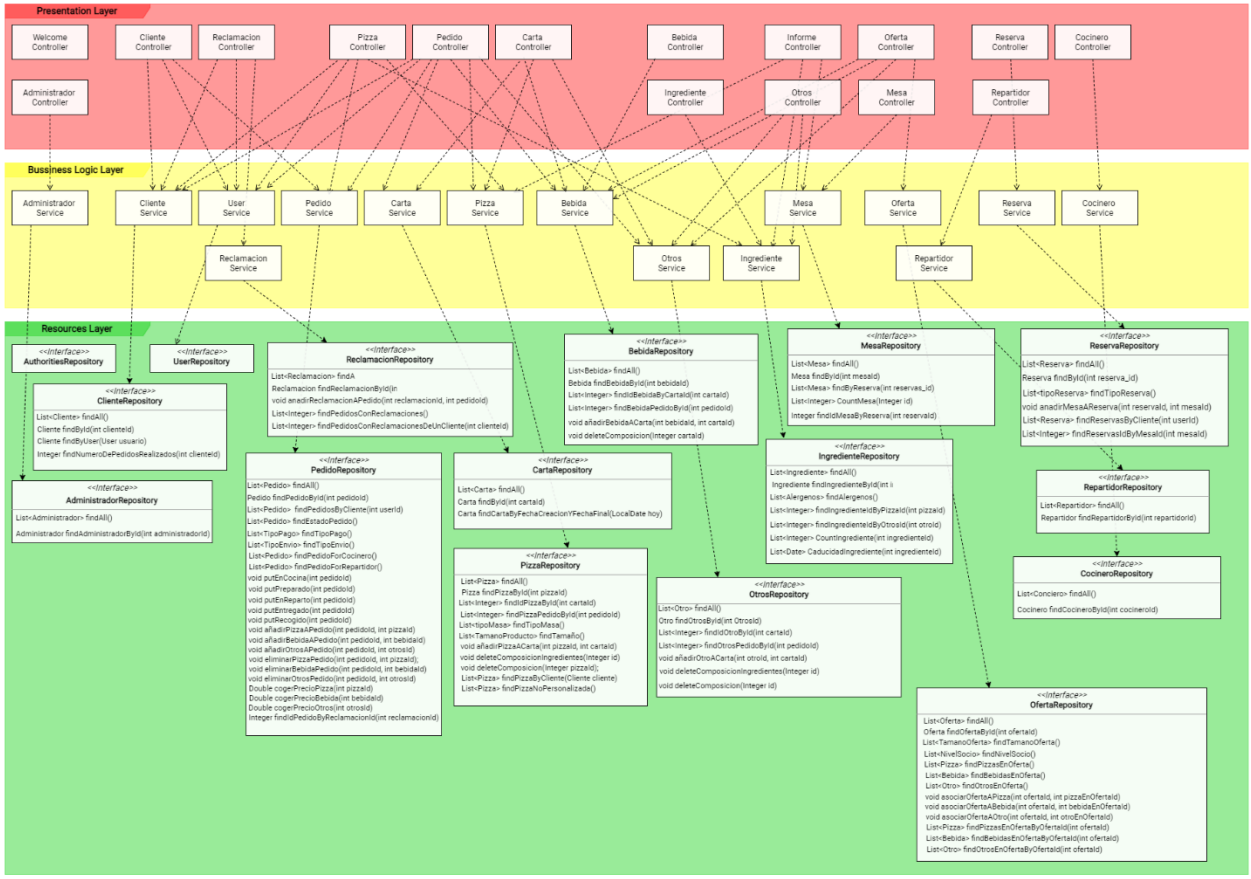


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)



Patrones de diseño y arquitectónicos aplicados

En esta sección de especificar el conjunto de patrones de diseño y arquitectónicos aplicados durante el proyecto. Para especificar la aplicación de cada patrón puede usar la siguiente plantilla:

Patrón: Modelo Vista Controlador

Tipo: Arquitectónico

Contexto de Aplicación

Al ser un patrón arquitectónico, se aplica a todo el proyecto.

Clases o paquetes creados

En el paquete org.springframework.samples.petclinic.model podemos encontrar todos los modelos que hemos utilizado, los controladores se encuentran en org.springframework.samples.petclinic.web y para acceder a las vistas es necesario acceder a src>main>webapp>WEB-INF> jsp.

Ventajas alcanzadas al aplicar el patrón

Al separar en modelo, vista y controlador es más fácil dividir responsabilidades, además aumenta la cohesión y disminuye el acoplamiento.

Decisiones de diseño

Decisión 1: Posibilidad de pedir de varias cartas

Descripción del problema:

Estuvimos planteando la posibilidad de que al cliente le mostráramos varias cartas (por ejemplo por si es celiaco y quisiera solamente visualizar los productos sin gluten). El problema era que nos resultaría muy complicado poder identificar de qué carta está pidiendo para mostrárselo al cocinero.

Alternativas de solución evaluadas:

Alternativa 1.a: Poder seleccionar entre varias cartas

Ventajas:

- Se pueden organizar mejor los productos.
- El cliente puede visualizar concretamente lo que desea.

Inconvenientes:

- El cocinero debe estar pendiente de la carta de la que viene el pedido.
- Para el cliente resulta incómodo pedir de diferentes cartas.

Alternativa 1.b: Al cliente solo se le muestra la última carta

Ventajas:

- Para el cliente es más fácil realizar un pedido.

- Nos resulta más cómodo de mostrar al cocinero.

Inconvenientes:

- Si el administrador desea añadir una nueva carta debe incluir muchos más productos que si estuviera dividida.

Justificación de la solución adoptada

Después de reunirnos tomamos la decisión unánime de elegir la alternativa 1.a ya que nos parecía más cómodo para el cliente, que es para quien principalmente está dirigido el proyecto.

Decisión 2: Responder a las reclamaciones

Descripción del problema:

Nos gustaría que el administrador pudiera contestar a las reclamaciones para que los clientes sientan que sus opiniones no están siendo ignoradas. El problema es que no sabíamos cuál era la manera más práctica de hacerlo.

Alternativas de solución evaluadas:

Alternativa 1.a: Añadir un atributo "respuesta" a la reclamación

Ventajas:

- No hay que añadir ninguna relación nueva.

Inconvenientes:

- Ralentiza todo el trabajo, por tener que añadir un atributo nuevo y porque nadie puede seguir trabajando con reclamación hasta que esté añadido.
- Hay que rehacer las pruebas.

Alternativa 1.b: Crear una clase "Respuesta" y relacionarla con la reclamación.

Ventajas:

- No afecta al trabajo que ya teníamos hecho (controller y pruebas) ya que es una clase independiente.

Inconvenientes:

- Es más complicado acceder a ella si estamos realizando una consulta.
- Debemos crearle un controller, un repository y un service además de sus pruebas.

Justificación de la solución adoptada

Como consideramos que lo más lógico que nos quitaba más carga de trabajo, decidimos optar por la alternativa de diseño 1.a.

Decisión 3: Nivel de cliente

Descripción del problema:

Para recompensar la fidelidad de nuestros clientes, decimos que al llegar a ciertas cantidades gastadas en total en sus pedidos (nivel de socio), pudieran tener acceso a mejores ofertas. El problema es cómo calcular el dinero que han gastado, si es mejor calcularlo muchas veces o dejarlo reflejado en algún sitio.

Alternativas de solución evaluadas:

Alternativa 1.a: Añadir un atributo "nivelDeSocio" al cliente y almacenar lo que llega gastado hoy en día.

Ventajas:

- Es más rápido de consultar.

Inconvenientes:

- Hay que añadir un nuevo atributo y crear una función para que vaya sumándole cada pedido que se realice.

Alternativa 1.b: Realizar consultas a cada pedido para obtener lo que ha gastado cada vez que inicie sesión para mostrarle las ofertas.

Ventajas:

-

Inconvenientes:

- El número de consultas aumenta considerablemente.

Justificación de la solución adoptada

Decidimos inclinarnos por la alternativa 1.a por ser más práctica y agilizar el sistema.

Decisión 4: Mostrar una carta sola al cliente

Descripción del problema:

El administrador puede crear varias cartas, pero a la hora de hacer un pedido al cliente solo le sale una carta en la que poder pedir. Esto es porque el administrador quiere que las cartas tengan fecha de caducidad por lo que habrá que hacer que no se creen dos cartas que se solapen para no provocar problemas.

Alternativas de solución evaluadas:

Alternativa 1.a: Crear una regla de negocio para que salte una excepción y no cree esa carta que solapa.

Ventajas:

- No se producirá el problema de que solo se muestra una carta y al haber más de una carta saltaría un error.

Inconvenientes:

- Debemos crear la regla de negocio y antes de crear la carta recorrer todas las cartas para ver que no solapa a ninguna ya creada.

Alternativa 1.b: Hacer que no se solapen creando la fecha de creación como el día siguiente o el mismo día de caducidad de la última carta creada.

Ventajas:

- No habría días que no se muestren cartas.

Inconvenientes:

- Ninguna.

Justificación de la solución adoptada

Hemos optado por a alternativa 1.b dado que parecía más sencilla que la 1.a.