DP1 2020-2021

Documento de Diseño del Sistema

Proyecto Pizzería y Pollería Curro Gas

https://github.com/gii-is-DP1/dp1-2020-g3-07

Miembros:

- Francisco José Brenes Lozano
- Ismael Luna Atienza
- Roberto Paz Rivera
- Daniel Rico Ostos
- Gonzalo Rodríguez Terrón
- Rodrigo Sánchez González

Tutor: Bedilia Estrada Torres

GRUPO G3-07

Versión 1

10/01/2021

Historial de versiones

Fecha	Versión	Descripción de los cambios	Sprint
10/01/2021	V1	• Creación del documento • Añadido diagrama de dominio/diseño • Añadido patrones • Añadido decisiones	3

Contents

Introducción

Este proyecto va a consistir en un sistema de gestión de pedidos online, donde el cliente va a poder pedir distintos productos ofrecidos por el establecimiento, realizando pago online y optando al reparto a domicilio o recogida en el propio local.

Dependiendo del usuario el sistema tendrá diferentes funcionalidades, tales como añadir distintos productos a la carta (en el caso del usuario "Administrador"), realizar un pedido (en el caso del usuario "Cliente") o ver la lista de pedidos a repartir y realizar una asignación de estos a sí mismo (en el caso del usuario "Repartidor"), además de varias funcionalidades más.

El objetivo principal de este sistema es abrir las puertas de este negocio al mundo de pedidos online lo que le supondrá mayores beneficios al dueño y mayor organización para sus empleados.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

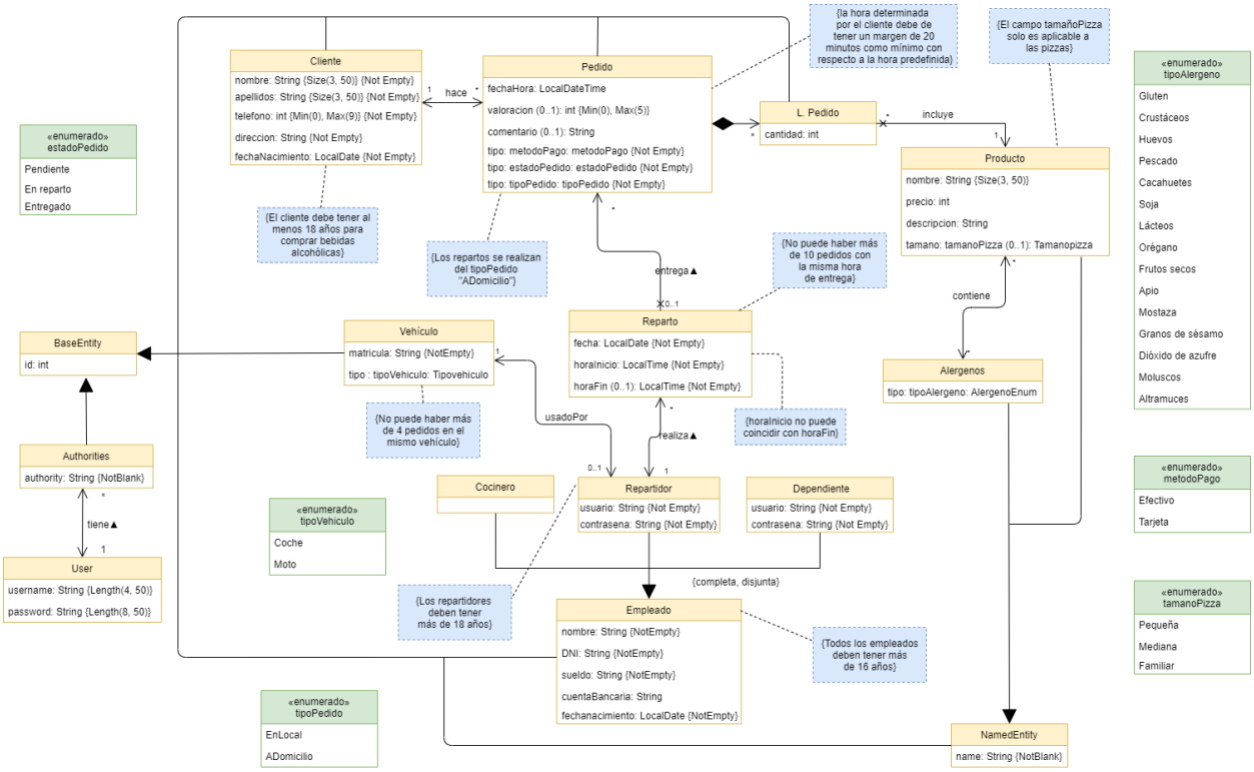
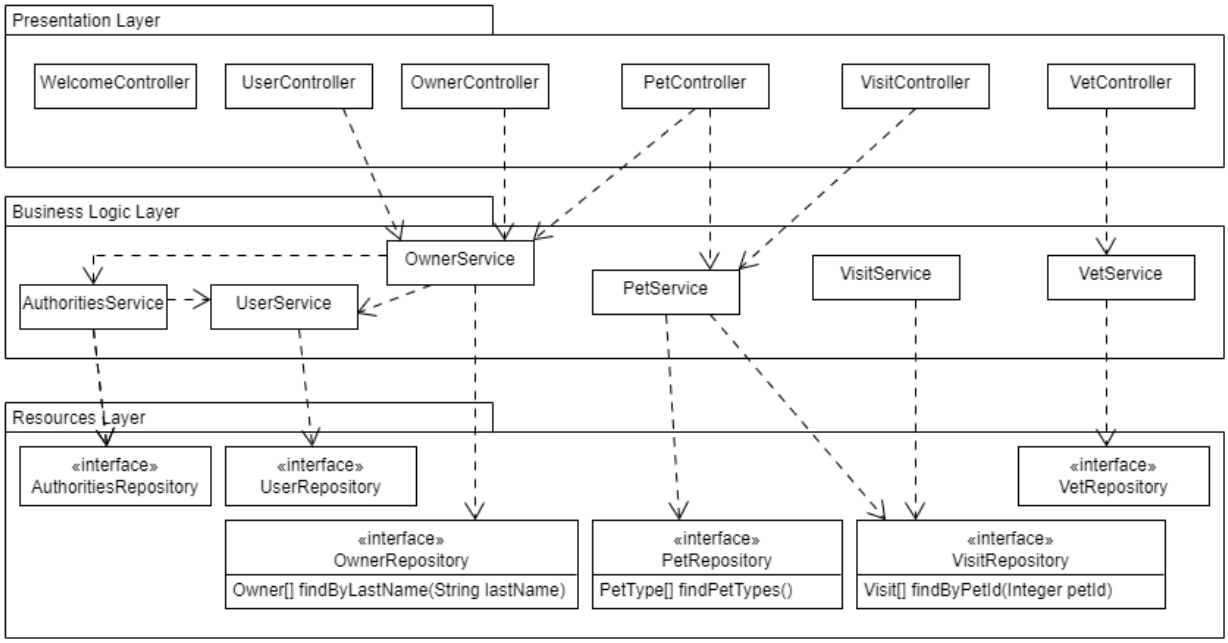


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

En esta sección debe proporcionar un diagrama UML de clases que describa el conjunto de controladores, servicios, y repositorios implementados, incluya la división en capas del sistema como paquetes horizontales tal y como se muestra en el siguiente ejemplo:



El diagrama debe especificar además las relaciones de uso entre controladores y servicios, entre servicios y servicios, y entre servicios y repositorios.

Tal y como se muestra en el diagrama de ejemplo, para el caso de los repositorios se deben especificar las consultas personalizadas creadas (usando la signatura de su método asociado).

Patrones de diseño y arquitectónicos aplicados

Patrón: Por capas

Tipo: Arquitectónico

Contexto de Aplicación

Nuestro proyecto lo hemos separado en distintas capas, estas son:

- Capa de recursos: formada por los modelos, repositorios y servicios.
- Capa de lógica de negocio: formada por los controladores.
- Capa de presentación: formada por las vistas (.jsp)

Clases o paquetes creados

- Capa recursos: org.springframework.samples.petclinic.model, org.springframework.samples.petclinic.repository, org.springframework.samples.petclinic.service
- Capa lógica de negocio: org.springframework.samples.petclinic.web
- Capa presentación: src/main/webapp/WEB-INF/jsp, src/main/webapp/WEB-INF/tags

Ventajas alcanzadas al aplicar el patrón

Hemos aplicado este patrón debido a que nos permite conseguir una alta cohesión y un bajo acoplamiento, separación de responsabilidades, las capas son independientes y fácil de sustituir y finalmente permite el desarrollo simultáneo de sus diferentes capas.

Patrón: Front Controller

Tipo: Diseño

Contexto de Aplicación

Hemos usado el framework Spring Boot para desarrollar nuestro proyecto el cual incluye un front controller llamado Dispatcher Servlet que recoge todas las solicitudes y desvía al controlador apropiado dicha solicitud.

Clases o paquetes creados

Incluido en el framework de Spring.

Ventajas alcanzadas al aplicar el patrón

Al usar este patrón hacemos uso del front controller que lleva un seguimiento de los usuarios. Se encarga de que todas las solicitudes sean seguras, y además añade funcionalidad a través de procesamiento de peticiones para la validación de parámetros y su transformación.

Patrón: Model View Controller (MVC)

Tipo: Arquitectónico

Contexto de Aplicación

Se puede observar claramente como nuestro proyecto está estructurado por Model, View, y Controller diferenciándose por paquete como los que mencionamos en el siguiente punto.

Clases o paquetes creados

- Model: org.springframework.samples.petclinic.model
- View: src/main/webapp/WEB-INF/jsp, src/main/webapp/WEB-INF/tags
- Controller: org.springframework.samples.petclinic.web

Ventajas alcanzadas al aplicar el patrón

Al usar este patrón observamos como nos ayuda dando suporte a varias vistas y favoreciendo a la alta cohesión, bajo acoplamiento y separación de responsabilidades.

Patrón: Inyección de dependencia

Tipo: Diseño

Contexto de Aplicación

Spring se encarga de crear instancias de determinados objetos, en concreto de aquellos que tengan una anotación @Autowired. Lo hemos usado en los modelos con la anotación @Entity, en los servicios con la anotación @Service y en los controladores con la anotación @Controller.

Clases o paquetes creados

En las clases de los paquetes org.springframework.samples.petclinic.model, org.springframework.samples.petclinic.service y org.springframework.samples.petclinic.web

Ventajas alcanzadas al aplicar el patrón

Obtenemos beneficio usando este patrón de diseño, por ejemplo, asegurándonos que se crea una sola instancia de cada clase y no varias a la vez.

Patrón: Template View

Tipo: Diseño

Contexto de Aplicación

En concreto hemos usado JSP/JSTL que componen todas las vistas de nuestra página web.

Clases o paquetes creados

Paquetes creados como: src/main/webapp/WEB-INF/jsp, src/main/webapp/WEB-INF/tags

Ventajas alcanzadas al aplicar el patrón

Podemos representar atributos del model con gran facilidad en nuestro código en HTML.

Decisiones de diseño

Decisión 1: Asignación de pedidos

Descripción del problema:

Necesitábamos la manera de asignar una serie de pedidos a un mismo reparto.

Alternativas de solución evaluadas:

Alternativa 1.a: Utilizar form:checkbox importado de Spring

Ventajas:

- Seguimos trabajando con el framework con el que manejamos todo el proyecto que es la gran novedad de este curso.
- Es una forma práctica de seleccionar datos

Inconvenientes:

- Puede que haya menos documentación comparándolo con lo que puede haber de HTML5.

Alternativa 1.b: Utilizar <select> con HTML5 base.

Ventajas:

- Hubiésemos acabado antes la historia de usuario que tiene que ver con este problema al encontrar información de manera más rápida

Inconvenientes:

- Dejamos de estar usando las librerías que nos presta Spring

Justificación de la solución adoptada

Ya que queríamos seguir usando todo lo proporcionado por Spring decidimos optar por la Alternativa 1.a.

Decisión 2: Forma de hacer enumerados

Descripción del problema:

Teníamos que decidir de la forma en la que íbamos a implementar los enumeraos para nuestras clases del paquete model (lo que serían algunos de los atributos de las entidades del Modelo Conceptual).

Alternativas de solución evaluadas:

Alternativa 2.a: Hacer los enumerados como clases Java

Ventajas:

- Se asemeja a la manera con la que siempre hemos trabajado con los enumerados
- La aplicación PetClinic hace los enumerados como clases e insertan los distintos valores a través del archivo data.sql

Inconvenientes:

- Se tarda más tiempo en insertar los valores de ese enumerado a través de la base de datos

Alternativa 2.b: Hacer los enumerados como enum

Ventajas:

- Más sencillo que insertar los datos a través de data.sql

Inconvenientes:

- PetClinic no nos servía de ejemplo en este caso ya que no implementa los enumerados de esta forma.

Justificación de la solución adoptada

Ya que pensamos que iba a ser más sencillo e intuitivo de trabajar con ellos posteriormente, decidimos implementar los enumerados como en la Alternativa 2.b.

Decisión 3: Inicio de sesión

Descripción del problema:

Teníamos que decidir cómo íbamos a realizar el inicio de sesión de los distintos usuarios que iba a tener la aplicación

Alternativas de solución evaluadas:

Alternativa 3.a: Clases Cliente, Dependiente y Repartidor (todas estas clases son usuarios de la aplicación) con atributos usuario, contraseña.

Ventajas:

- Sencillo de trabajar con estos atributos
- No necesita de entender como funciona la entidad User de PetClinic

Inconvenientes:

- No proporciona ninguna seguridad

Alternativa 3.b: Reutilizar la clase User de Petclinic junto con Authorities

Ventajas:

- Gracias a esto, se proporciona mayor seguridad al inicio de sesión
- Podemos usar de ejemplo la aplicación PetClinic si nos hiciera falta.

Inconvenientes:

- En un principio no sabíamos como trabajar con el Authentication pero conseguimos entenderlo.

Justificación de la solución adoptada

Ya que consideramos que es mucho más completo y real a lo que sería una página web profesional de este estilo, el utilizar la Alternativa 3.b es para nosotros la mejor opción

Decisión 4: Forma de hacer enumerados

Descripción del problema:

Necesitábamos decidir de qué manera íbamos a implementar Alérgenos ya que desde el principio barajamos distintas posibilidades

Alternativas de solución evaluadas:

Alternativa 4.a: Hacer Alérgenos como enumerado

Ventajas:

- Sencillo de implementar
- Fácil de trabajar con él

Inconvenientes:

- No se asemeja a lo que PetClinic ya tenía

Alternativa 4.b: Crear una clase llamada AlergenoType

Ventajas:

- Podríamos tomar Pet y PetType como ejemplo perfecto

Inconvenientes:

- En un principio no veíamos claro como íbamos a poder trabajar con esto a lo largo del desarrollo del proyecto.

Justificación de la solución adoptada

Vimos más asequible y sencillo el implementar Alérgenos como en la Alternativa 4a ya que cómo esta comentado anteriormente también íbamos a tener que trabajar con enumerados en otras clases.