DP1 2020-2021

Documento de Diseño del Sistema

Proyecto Talleres La Plata

< https://github.com/gii-is-DP1/dp1-2020-g2-11.git >

Miembros:

- Álvaro Alcón Granado
- Francisco José Anillo Carrasco
- Daniel Dorante Sánchez
- Alejandro Sevillano Barea
- Eugenio Benito Vicente Vázquez
- Jesús Viera Vázquez

Tutor: **CARLOS GUILLERMO MULLER CEJAS**

GRUPO G2-11

Versión 2.0

9-2-2020

Historial de versiones

Estos son ejemplo del contenido que debería tener el historial de cambios del documento a entregar a lo largo de los sprints del proyecto

Fecha	Versión	Descripción de los cambios	Sprint
28/12/2020	V1	Creación del documento	3
09/2/2021	V2	Finalización y revisión del documento	4

Contents

Introducción

Este proyecto está basado en un taller, el objetivo requerido es conocer cuáles son las necesidades a satisfacer, que permita aportar más ventajas a la hora de realizar la gestión de su negocio. De esta manera, se centrará principalmente en concertar citas a través de Internet, cuestión que a día de hoy no se realiza, y facilitaría la tarea tanto para el propietario, como para el cliente del taller.

Además, guardar información acerca del problema de su vehículo, así como de la gestión administrativa con proveedores y clientes ofreciendo nuestra aplicación la posibilidad de organizar los pedidos que hace el taller a los diferentes proveedores.

La finalidad sería facilitar la gestión y el uso de los diferentes servicios que ofrece el taller para todos los usuarios de manera que se realice de forma óptima.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

Diagrama de Cliente

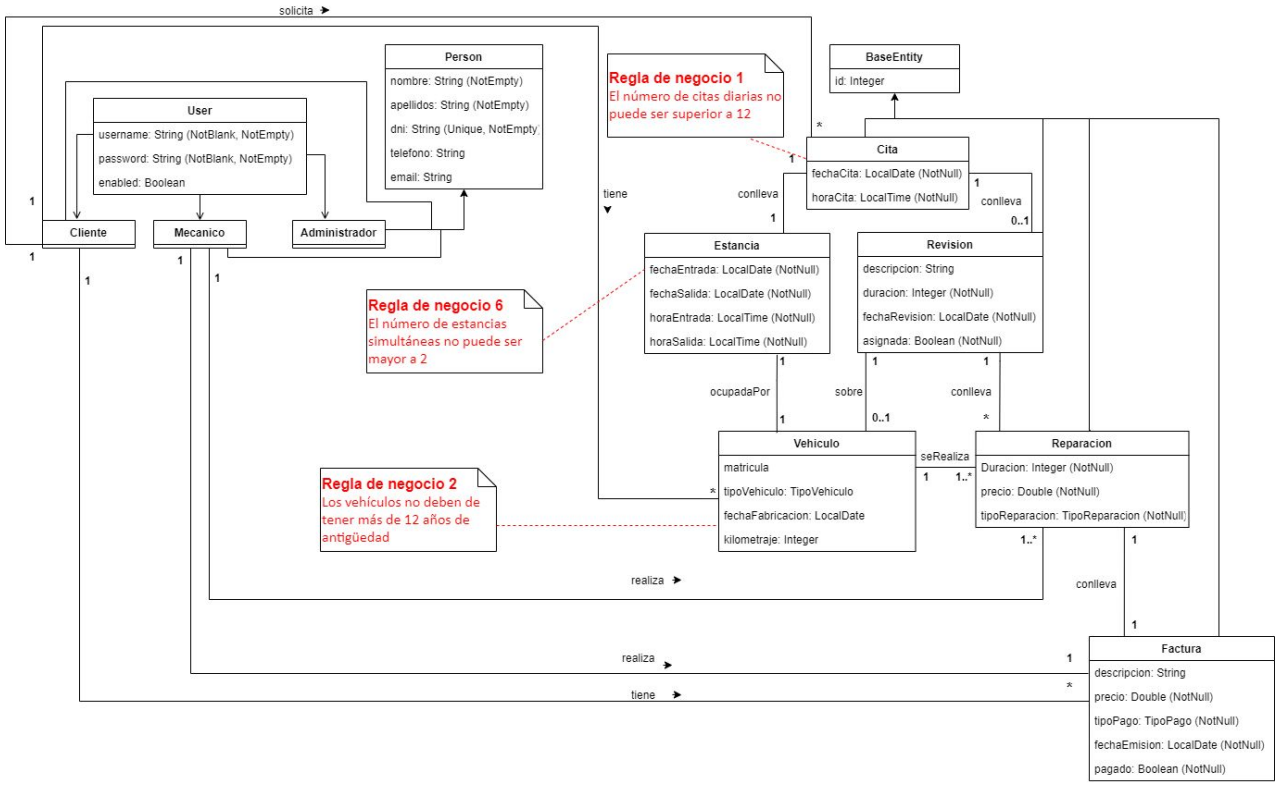


Diagrama de Pedido

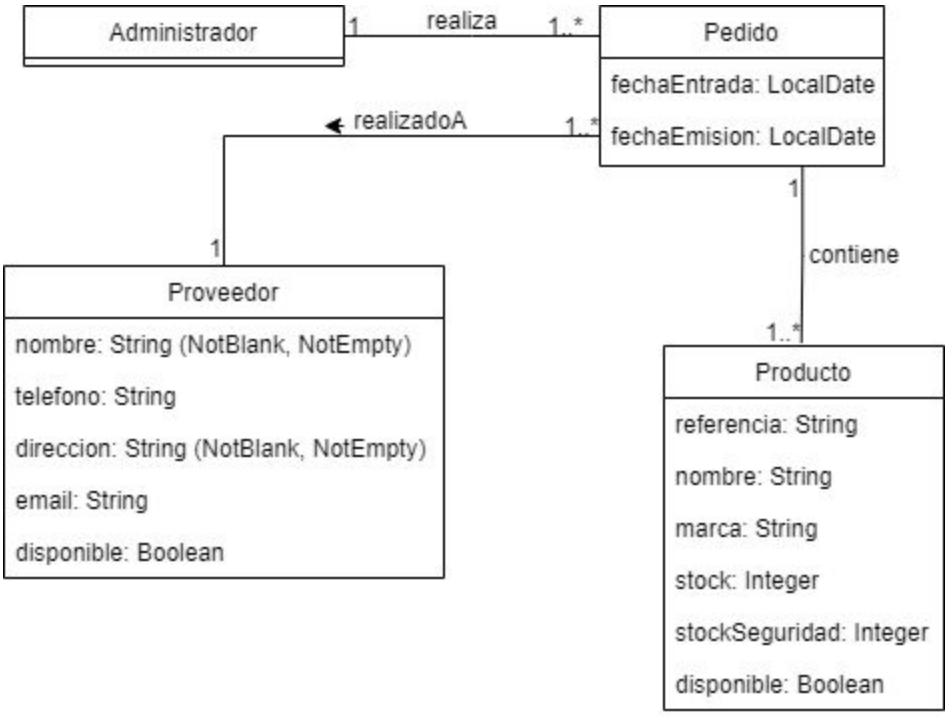
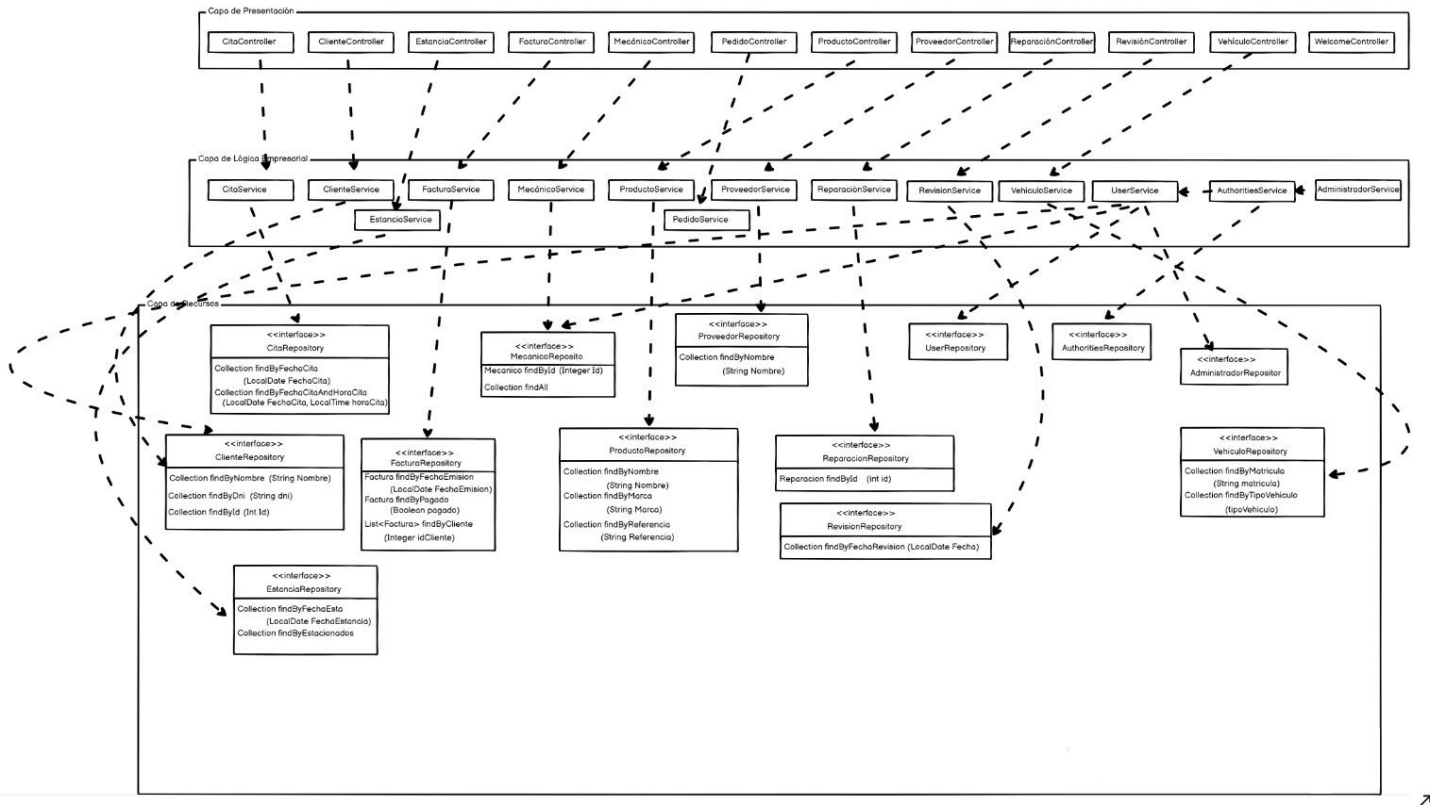


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)



Patrones de diseño y arquitectónicos aplicados

Patrón: Modelo-Vista-Controlador

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

Nuestra aplicación tiene tres capas, la primera es la capa de recursos, los modelos, donde creamos las clases con los distintos atributos y métodos, necesarios para cambiar la información de la base de datos, estos son los paquetes: service, model y repository.

Por otro lado, nosotros tenemos la capa de vista, que son los archivos jsp, encargados de generar la interfaz donde los usuarios de la página podrán interactuar con esta. Se encuentra en la carpeta “WEB-INF”.

Por último están los controladores, que son los que realizan los métodos implementados en la capa de recursos. Estos se encuentran en el paquete web.

Clases o paquetes creados

Hay clases de modelos, en el paquete model, también hay clases service y repository y controladores, en el paquete web, tenemos paquetes para las pruebas de funcionamiento y un paquete para la base de datos y por último tenemos las vistas en la carpeta “WEB-INF”.

Ventajas alcanzadas al aplicar el patrón

Las ventajas de este patrón son: que al ser de capas, facilita el manejo de los errores, además tiene una estructura más fácil de controlar y tiene separados los datos de la representación visual.

Patrón: Capa supertipo (Layer Supertype)

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

Tenemos varias clases que usan objetos semejantes, por ejemplo un id, que lo usan todas las clases del sistema, o los atributos nombre, apellidos, teléfono... en las clases de los usuarios. Para ello usamos el patrón capa supertipo.

Este patrón consiste en crear una clase abstracta para todas las entidades que contengan dichos atributos en ella. De manera que en cada clase podemos crear atributos diferentes y a parte los de la clase supertipo.

Clases o paquetes creados

Este patrón lo hemos implementado en dos casos.

El primer caso es en el campo del identificador de cada objeto o actor. Todas las entidades llevan un id, para ello se creó la clase BaseEntity, que genera un id automático.

Por otro lado, también creamos la clase Person, la cual contiene los atributos de una persona, nombre, apellidos, teléfono... Esta clase es extendida por los actores de nuestro proyecto, ya que todos tienen atributos semejantes.

Ventajas alcanzadas al aplicar el patrón

Las ventajas de este patrón son: la facilidad que genera al no tener que crear los mismos atributos varias veces, si no que solo se crean una vez. Es mucho más rápido y sencillo.

Patrón: De Repositorio

Tipo: Arquitectónico | de Diseño

Contexto de Aplicación

En nuestra aplicación usamos este patrón para dar la impresión de que estás accediendo a una colección de objetos en vez de a la base de datos.

Los repositorios son clases que encapsulan la lógica requerida para acceder a las fuentes de datos.

Clases o paquetes creados

Nosotros creamos un repositorio por cada entidad. Por lo tanto tenemos creados, un total de 14. Estos son los siguientes: AdministradorRepository, AuthoritiesRepository, CitaRepository, ClienteRepository, EstanciaRepository, FacturaRepository, MecanicoRepository, PedidoRepository, ProductoRepository, PorveedorRepository, ReparacionRepository, RevisionRepository, UserRepository y VehiculoRepository.

Ventajas alcanzadas al aplicar el patrón

Las ventajas de este patrón son: que crean la conexión entre nuestro sistema y la base de datos.

Decisiones de diseño

Decisión 1: Ocultar Productos agotados.

Descripción del problema:

A la hora de borrar un producto porque no disponemos stock de éste, nos dimos cuenta de que ese producto no iba a estar disponible durante un tiempo, pero que después de un tiempo podría volver a estar disponible.

Alternativas de solución evaluadas:

Alternativa 1.a: No eliminar el producto, sino ocultarlo durante el tiempo que no esté disponible.

Ventajas:

- Se evita tener que estar añadiendo y eliminando los mismos productos.

Inconvenientes:

- Estos productos consumen espacio en la base de datos y recursos de la aplicación, aunque visiblemente no están disponibles en la web.

Alternativa 1.b: Eliminar el producto de la base de datos.

Ventajas:

- El producto no consume espacio en la base de datos, ni recursos en la aplicación.

Inconvenientes:

- Hay que estar añadiendo y eliminando productos que ya han estado en la web.

Justificación de la solución adoptada

Como hemos evitado el tener que estar añadiendo y eliminando los productos que dejan de estar disponibles durante un tiempo, hemos decidido tomar la alternativa 1a.

Decisión 2: Ocultar Proveedores con los que no se trabaja durante un tiempo.

Descripción del problema:

A la hora de eliminar un proveedor porque éste deja de suministrar productos al taller durante un tiempo, observamos que ese proveedor podría volver a suministrar productos en un futuro.

Alternativas de solución evaluadas:

Alternativa 1.a: No eliminar el proveedor, sino ocultarlo durante el tiempo que no suministre productos al taller..

Ventajas:

- Se evita tener que estar añadiendo y eliminando los mismos proveedores.

Inconvenientes:

- Estos proveedores consumen espacio en la base de datos y recursos de la aplicación, aunque visiblemente no están disponibles en la web.

Alternativa 1.b: Eliminar el proveedor de la base de datos.

Ventajas:

- El proveedor no consume espacio en la base de datos, ni recursos en la aplicación.

Inconvenientes:

- Hay que estar añadiendo y eliminando proveedores que ya han estado en la web.

Justificación de la solución adoptada

Como hemos evitado el tener que estar añadiendo y eliminando los proveedores que dejan de suministrar productos durante un tiempo, hemos decidido tomar la alternativa 1a.

Decisión 3: El administrador registra a los mecánicos.

Descripción del problema:

A la hora de eliminar un proveedor porque éste deja de suministrar productos al taller durante un tiempo, observamos que ese proveedor podría volver a suministrar productos en un futuro.

Alternativas de solución evaluadas:

Alternativa 1.a: Poder elegir dentro del formulario de usuario el tipo de usuario que era.

Ventajas:

- Los mecánicos se añadían ellos mismos.

Inconvenientes:

- El problema era que un cliente cualquiera se podría registrar como mecánico y tener acceso a cosas que no debería.

Alternativa 1.b: El administrador es el encargado de registrar a los mecánicos.

Ventajas:

- Solo se crean los mecánicos que el administrador desee.

Inconvenientes:

- Es un poco tedioso para el administrador, ya que tiene que encargarse él de crearlos.

Justificación de la solución adoptada
Por temas de seguridad en el uso de la página web y para evitar que un usuario cualquiera pueda ver datos que no debe ver, hemos elegido la alternativa 1b.

Decisión 4: Establecer la factura como pagada.

Descripción del problema:
A la hora de saber si una factura estaba pagada o no.

Alternativas de solución evaluadas:
Alternativa 1.a: Editar las facturas para cambiar el boolean.

Ventajas:

- Puedes editar otros datos.

Inconvenientes:

- Es mucho más trabajoso para el administrador

Alternativa 1.b: Poner un botón que transforme el boolean de no pagado a pagado.

Ventajas:

- Es más rápido que la opción a, ya que solo se le tiene que dar a un botón.

Inconvenientes:

- Solo se puede cambiar ese valor.

Justificación de la solución adoptada
Ya que las facturas no tienen mucho que editar, nosotros hemos optado por acoger la opción b, ya que es más rápida y fácil de implementar.

Decisión 5: Validación de formulario producto

Descripción del problema:
A la hora de validar los campos del formulario producto, para comprobar que no era nulo.

Alternativas de solución evaluadas:
Alternativa 1.a: Usar el método *"isBlank()"*, que comprueba si una cadena está vacía.

Ventajas:

- Nos ahorraba trabajo y esfuerzo.

Inconvenientes:

- Se implementa a partir de Java 14 y no sabíamos si podríamos meterlo

Alternativa 1.b: Comprobar si la cadena empieza por un espacio, lo que supondría que la cadena está vacía o que se ha equivocado.

Ventajas:

- Versión de Java menor a la indicada, por lo tanto, está disponible para todos.

Inconvenientes:

- Más largo de implementar.

Justificación de la solución adoptada

Usamos la opción b, ya que no queríamos correr el riesgo de que al corrector no le funcionase el método al corrector.