V2.0

# FRUTERÍA MR. PRESSMAN

## Memoria

Diego Acuña Berger
Daniel Calle Sánchez
Guillermo Delgado Yepes
Carlos Gómez Cereceda
Manuel Guerrero Moñús
Zihao Hong
FRUTERÍA MR. PRESSMAN

# Contenido

# AQUITECTURA:

En nuestro proyecto hemos integrado la arquitectura MVC (Modelo Vista Controlador) la cual divide nuestra aplicación en tres componentes:

El **modelo** contiene la funcionalidad básica de nuestra aplicación, es decir, las funciones a realizar y los datos que usarán.
Las **vistas** muestran y recogen información del usuario, dicha información es enviada al controlador y cuando éste de una respuesta las vistas mostrarán los datos y se actualizarán conforme a dicha respuesta.
Los **controladores** mediante entre las vistas y el modelo, al cual le pasamos datos, tales como los eventos que le llegan de las vistas. Dependiendo del evento, el controlador seleccionará el componente del modelo correspondiente para tratar los datos y devolverá una respuesta a las vistas, para que éstas se actualicen correctamente.

Además, hemos aplicado una arquitectura multicapa, en la cual hemos integrado el MVC. Ésta considera tres capas, una de presentación, otra de negocio y finalmente una de integración:

**Presentación:** ésta encapsula toda la lógica de presentación necesaria para dar servicio a los clientes que acceden al sistema. Aquí implementamos nuestra interfaz gráfica, los correspondientes ActionListener de los diferentes componentes gráficos usados y además un Controlador que se ocupará de mediar entre Presentación y Negocio, aquí es donde aplicaremos la arquitectura MVC descrita anteriormente. Normalmente aquí crearemos los transfers de las entidades y se los entregaremos al Controlador para que pase al tratamiento de dichos datos.
**Negocio:** en esta capa es donde implementaremos los servicios de aplicación y los transfers de las entidades del sistema, los cuales, tras ser llamados desde el Controlador debido a que se ha interaccionado con la vista, se ocuparán de las reglas de negocio que hemos especificado, tales como comprobar que algunos datos

existan o que los mismos sean correctos sintácticamente. Posteriormente si todo ha sido correcto se comunicará con la capa de Integración para guardar los datos.
**Integración:** esta capa es la responsable de interactuar con nuestra base de datos, tras acceder al servicio de aplicación en negocio y comprobar que todos los datos del transfer son correctos, se accederá al DAO (Data Access Object) que implementa las mismas operaciones que el servicio de aplicación, pero con la diferencia de que no debe preocuparse de comprobar si los datos son correctos o no, esta capa simplemente se ocupa de extraer y guardar datos que se le pasan.

Las grandes ventajas de aplicar esta arquitectura son, por ejemplo, que podemos modificar cualquier capa sin afectar a las demás, es decir, los cambios que se realicen en presentación no afectarán para nada a como tratan los datos los servicios de aplicación. También favorece la encapsulación y la reusabilidad del código. Obtendremos un sistema más fiable y con un mejor rendimiento y muy empaquetado, además nos permite dividirnos y trabajar en paralelo por capas, mejorando así el trabajo del equipo.

# LISTADO DE DIAGRAMAS:

## Diagramas de clase:

**Presentación:** Cliente, Producto, Venta, Departamento, Trabajador, Curso.

**Negocio:** Cliente, Producto, Venta, Factoría, Departamento, Trabajador, Curso.

**Integración:** Cliente, Producto, Venta, Factoría, DAOConnection.

## Diagramas de secuencia:

### PRESENTACIÓN:

**Producto:** ActualizarProducto, AltaProducto, BajaProducto, BuscarProducto, ListarProductos, ActualizarGUIProducto.

**Venta:** addLineaVenta, updateLineaVenta.

**Dispatcher:** generateVista.

**Command:** generateCommand, CommandAbrirVenta, CommandBuscarVenta, CommandCerrarVenta, CommandListarVentas, CommandDevolucionVenta, CommandAltaProducto, CommandActualizarProducto, CommandBuscarProducto, CommandBajaProducto, CommandListarProductos.

**Otros:** GUIMain, Controlador, ControladorAccion,

### NEGOCIO:

**Cliente:** productoMasComprado, productosClienteVIP

**Producto:** create, delete, generateSAProducto, getInstance, read, readAll, update.

**Venta:** closeSale, devolution, openSale.

## INTEGRACIÓN:

**Producto:** create, delete, findByName,

generateDAOProducto, getInstance, read, readAll, update.

**Venta:** create, read, readAll, update.

**Departamento:** AltaDepartamento, BajaDepartamento, ActualizarDepartamento, ListarDepartamentos, BuscarDepartamento, CalculoDeUnaNominaDeUnDepartamento, CalculoNomina.

**Trabajador:** create, delete,update, CalcularNominaTiempoCompleto, CalcularNominaTiempoParcial

**Curso:** matricularTrabajador, desmatricularTrabajador.

# Diagramas de despliegue:

# PATRONES EMPLEADOS:

## Patrón Singleton:

**OBJETIVO:**

Garantizar que una clase solo posea una sola instancia.

**CARACTERÍSTICAS:**

La propia clase es responsable de su única instancia.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN EL PROYECTO?:**

Factorías.
Interfaces de usuario.
Controlador.
Ventajas:
Acceso controlado a la única instancia.
Espacio de nombres reducido.
Permite el refinamiento de operaciones y la representación.
Permite un número variable de instancias.
Más flexibles que las operaciones de clase estáticas.

## Patrón Transfer:

**OBJETIVO:**

Independizar el intercambio de datos entre las capas.

**CARACTERÍSTICAS:**

Evita que una capa tenga que conocer la representación interna de una entidad.
Son objetos serializables.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

Capas: Presentación, negocio, integración.

# Patrón DAO:

**OBJETIVO:**

Acceder a la capa de datos.

**CARACTERÍSTICAS:**

Permite cambiar la capa de datos sin afectar a la capa de negocio.
Favorece la independencia de la representación, el acceso y el procesamiento de los datos.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

Capas: Integración y se llama en los Servicios de Aplicación de Negocio.

# Patrón Servicio de Aplicación:

**OBJETIVO:**

Centralizar la lógica del negocio.

**CARACTERÍSTICAS:**

Desacopla la lógica del negocio de otros objetos de esta capa y hace que ésta quede unificada bajo el encapsulamiento que ofrece la orientación a objetos.
Permite agrupar funcionalidades relacionadas.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?**

Capas: Negocio y en Presentación el Controlador se ocupa de llamarlo.

# Patrón Controlador:

**OBJETIVO:**

Centraliza y modulariza la gestión de acciones y vistas.

**CARACTERÍSTICAS:**

Permite reutilizar el código de gestión de vistas y acciones.
Mejora la extensibilidad del manejo de peticiones.
Favorece la modularidad del código y su mantenibilidad.

Sirve para interactuar entre Presentación y Negocio.

**¿DÓNDE HEMOS UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

Capas: Presentación.

# Patrón Factoría Abstracta:

**OBJETIVO:**

Proporcionar una interfaz para crear familias de objetos relacionados.

**CARACTERÍSTICAS:**

Independiza al sistema de cómo se crean y representan sus productos.
Permite configurar familias de productos dentro del sistema software.
Proporciona una biblioteca de clases de productos, revelando únicamente sus interfaces y no sus implementaciones.

**¿DÓNDE HEMOS UTILIZADO ESTE PATRÓN DENTRO DE NUESTRO PROYECTO?:**

Capas: Negocio (Servicio de aplicación) e Integración.

# Patrón Command:

**OBJETIVO:**

Encapsular las peticiones de nuestra aplicación en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones.

**CARACTERÍSTICAS:**

Desacopla el objeto que invoca la operación de aquel que sabe cómo realizarla, permite añadir nuevas órdenes de manera más fácil.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

En la capa de presentación, para desacoplar el controlador

de los servicios de aplicación.

# Patrón Query:

**OBJETIVO:**

Desvincular las query complejas de los DAOS

**CARACTERÍSTICAS:**

Cuando hay que seleccionar un conjunto de elementos que se corresponde con una query muy compleja se pueden incluir como objeto query desvinculado de los DAOs.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

Se ubican en integración y se usan en negocio, únicamente para Cliente, para realizar las consultas: ProductoMasCompradosPorUnCliente y ProductosCompradosPorClienteVip.

# Patrón Almacén del dominio:

**OBJETIVO:**

Separar la persistencia del modelo de objetos.

**CARACTERÍSTICAS:**

Hace que nuestro sistema sea persistente.
Nos permite gestionar carga dinámica.
Nos resuelve los problemas de gestión de transacciones y concurrencia, gracias a un Transaction Manager.

**¿DÓNDE HEMOS UTILIZADO ESTE PATRÓN DENTRO DE NUESTRO PROYECTO?:**

Capas: Integración.

# Patrón Mapper:

**OBJETIVO:**

Selecciona un objeto a partir de un evento

**CARACTERÍSTICAS:**

Se realiza con carga dinámica a través de archivos de configuración xml.

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

En Dispatcher y en FactoriaCommand

# Patrón Contexto

**OBJETIVO**

Se desea evitar utilizar información del sistema específica del protocolo fuera de su contexto relevante.

**CARACTERÍSTICAS**

Mejora reusabilidad y mantenibilidad
Mejora las pruebas
Reduce las restricciones en la evolución de GUIs

**¿DÓNDE SE HA UTILIZADO ESTE PATRÓN EN NUESTRO PROYECTO?:**

En presentación para encapsular el par de evento-dato y se utilizar para la comunicación entre GUIs y controlador.

# JPA:

En la parte del proyecto correspondiente a JPA hemos realizado una gestión de la concurrencia optimista, que ya está implementado en JPA, aunque nosotros debemos ocuparnos de gestionar los casos en los que haga falta el uso del bloqueo optimista o el bloqueo optimista con incremento forzado. Las clases Curso, Departamento y Trabajador son entidades JPA. Tenemos una función que calcula la nómina de un departamento, que usa el polimorfismo de la clase Trabajador con Tiempo Parcial y Tiempo Completo para que cada especialización de la clase calcule la nómina de una forma distinta.

# REPOSITORIOS:

**DOCUMENTACIÓN:**

https://versiones.fdi.ucm.es/svn/MS/ms1718fruteriadoc

**MODELO:**

https://versiones.fdi.ucm.es/svn/MS/ms1718fruteriamod

**CÓDIGO:**

https://versiones.fdi.ucm.es/svn/MS/ms1718fruteriacod

| Proyecto | Alumno | Usuario | Contraseña |
|---|---|---|---|
| **Frutería Mr. Pressman** | Daniel Calle dacalle@ucm.es | ms1718dcalle | parker7851 |
| | Diego Acuña dacuna@ucm.es | ms1718dacuna | Parker4405 |
| | Carlos Gómez cargom11@ucm.es | ms1718cgomez | parker0489 |
| | Guillermo Delgado Gdelgado02@ucm.es | ms1718gdelgado | parker6584 |
| | Manuel Guerrero mangue01@ucm.es | ms1718mguerrero | parker4621 |
| | Zihao Hong zhong@ucm.es | Ms1718zhong | parker1873 |