V 1.0

FRUTERÍA MR. PRESSMAN

Memoria

Diego Acuña Berguer
Daniel Calle Sánchez
Guillermo Cortina Fernández
Guillermo Delgado Yepes
Manuel Guerrero Moñús
Zihao Hong

FRUTERÍA MR. PRESSMAN

Tabla de 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, Marca, País, Producto, Venta

Negocio: Cliente, Marca, País, Producto, Venta, Factoria.

Integración: Cliente, Marca, País, Producto, Venta, Factoria, DAOConnection.

Diagramas de secuencia:

Presentación:

País: ActualizarPais, AltaPais, BajaPais, BuscarPais, ListarPaises, ActualizarGUIPais, ControladorAccion.

Venta: addLineaVenta, updateLineaVenta.

Otros: GUIMain, Controlador.

Negocio:

Marca: create, delete, update.

País: create, delete, generateSAPais, getInstance, read, readAll, update.

Venta: closeSale, devolution, openSale.

Integración:

País: create, delete, findByName, generateDAOPais, getInstance, read, readAll, update.

Venta: create, read, readAll, update.

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.

REPOSITORIOS:

Documentación:

https://versiones.fdi.ucm.es:10001/svn/is1617gisAfruteriadoc

Modelo:

https://versiones.fdi.ucm.es:10001/svn/is1617gisAfruteriamod

Código:

https://versiones.fdi.ucm.es:10001/svn/is1617gisAfruteriacod

Proyecto	Alumno	Usuario	Contraseña
Frutería Mr. Pressman	Daniel Calle dacalle@ucm.es	is1617gisAdcalle	Natalia1254
	Diego Acuña dacuna@ucm.es	is1617gisAdacuna	Natalia3210
	Guillermo Cortina Guillcor@ucm.es	is1617gisAgcortina	Natalia9751
	Guillermo Delgado gdelga02@ucm.es	is1617gisAgdelgado	Natalia4457
	Manuel Guerrero mangue01@ucm.es	is1617gisAmguerrero	Natalia3012
	Zihao Hong zhong@ucm.es	is1617gisAzhong	Natalia0743