Memoria JPA

ANIMALIA

Modelado del Software

Grupo E

05/10/2023

Integrantes:

Airam Martín Peraza

Álvaro Velasco García

Axel Kahou He Expósito

Carlos Mayorga Santiago

Carlo Dubini Marqués

Clara Sotillo Sotillo

Jorge Calvo Fernández

Mariana Cucicea

Sara Sacó Baños

Sergi Cristóbal Gambau

Sofía Postigo Ruiz

Steven Pérez Salinas

CONTENIDO

1. INTRODUCCIÓN

El objetivo de esta memoria es presentar la documentación de la segunda parte de nuestro zoológico Animalia y como se ha realizado. En esta segunda parte hemos programado y desarrollado una aplicación para gestionar la tienda de un zoológico a nivel administrativo como habíamos detallado previamente en la Especificación de Requisitos Software(SRS).

La elaboración de este proyecto ha transcurrido en la mitad del mes de Noviembre y hasta el 21 de Diciembre Este trabajo permite la venta de productos con sus marcas asociadas que se encuentran en stock en la tienda. También permite el seguimiento de los empleados que trabajan en ella y a qué departamento pertenecen. Las marcas que tenemos a la venta siempre son renovadas por nuestros proveedores contratados.

Este documento queda dividido las siguientes secciones:

- En el **Desarrollo,** podemos encontrar como se ha llevado a cabo el proceso del desarrollo del software y cómo se han realizado el reparto de tareas.
- En el apartado de **Producto** observamos el trabajo final con las características previamente establecidas en la SRS, que funcionalidades se han implementado y las interfaces utilizadas para el usuario.
- Hemos querido diseñar la **Arquitectura** de nuestra aplicación de la forma más precisa y apropiada para un correcto modelado del software, utilizando una serie de patrones de diseño.
- **La estructura de nuestro código** está organizada en carpetas para mantener las clases de Java ordenadas y otros archivos relativos a la base de datos, entre otros. Siguiendo la arquitectura multicapa.
- En nuestro proyecto hemos realizado un gran número de **Pruebas** para el perfecto funcionamiento de nuestro software, desde el nivel de requisitos a su propio funcionamiento.
- Para el desarrollo hemos utilizado una serie de **Recursos** empleados con el propósito de una funcionalidad más completa y segura.

Una vez definidas en qué consisten cada una de nuestras secciones para un mejor entendimiento de esta parte final de Animalia. A continuación nos centraremos en el desarrollo de la gestión de la tienda del zoológico.

2.DESARROLLO

2.1 RESUMEN

Una vez entregada la primera parte y la especificación de requisitos, planteamos todos los errores y cosas a mejorar tanto para terminar de perfeccionar estas apartes anteriores como para tenerlas en cuenta a la hora de desarrollar y planificar esta última parte del proyecto.

Después de seguir estas directrices, comenzamos enumerando todas las tareas que teníamos que realizar, la lista de diagramas que teníamos que presentar, el código y la elaboración de un documento que recogiera todos los datos relativos al proyecto. Primero dividimos las entidades por pareja para realizar los diagramas de clase. Cada pareja se encargaba de sus diagramas de clase y de la parte de código necesaria para esa entidad. También se le asignó a cada persona un diagrama de secuencia a realizar. Una vez cada uno iba acabando sus tareas ayudaban al resto del grupo que tuviese tareas más complejas o extensas.

Mientras se iban terminando funcionalidades, se iban pasando todas las pruebas necesarias para el perfecto funcionamiento de nuestro software. Cada paso que íbamos avanzando y terminando comprobábamos que cumpliese todos los requisitos especificados en la SRS.
Una vez comprobado que todo seguía el curso esperado, retocamos todo el trabajo realizado con el propósito de mejorar su legibilidad y optimizar todos los procesos.

Para una organización más óptima, realizamos reuniones continuas tanto físicas como virtuales para poner en contexto tanto las cosas que faltaban y requieren mayor urgencia, como procedimientos que podrían estar erróneos o que de cara al futuro podrían suponer un problema y poder cambiarlos a tiempo. Se ha promovido en todo momento que haya una comunicación activa y fluida tanto entre los miembros de cada grupo como con todos los integrantes del trabajo. Podemos asegurar que hemos tenido mucho trabajo en equipo constante. Todos éramos conscientes que si alguien tenía un conflicto o un problema, no era una disconformidad individual, si no era algo que teníamos que abordar todos juntos.

Junto con todo lo recopilado anteriormente podemos recalcar que hemos utilizado una metodología ágil, actualizando todo el material según iban avanzando los cambios. Hemos considerado que hacer ciertos cambios podrían ayudar mucho a las siguientes etapas del proyecto. Cada dato o procedimiento que se encuentra en el proyecto será sometido a una revisión por parte de nuestro profesor Antonio Navarro Martin para un mejor y correcto resultado.

2.2 PROCESO

Para una mejor lectura y comprensión de las tareas que se han realizado a lo largo de todo el desarrollo del proyecto, hemos recopilado todo lo necesario en tablas.

TAREA	AUTOR	FECHA
Diagramas de clase		
Marca Producto	Sara Sacó Carlo Dubini	19/11-30/11
Trabajador	Álvaro Velasco Jorge Calvo	19/11-30/11
Producto	Mariana Cucicea Clara Sotillo	19/11-30/11
Departamento	Axel Kahou Carlos Mayorga	19/11-30/11
Venta	Airam Martín Steven Perez	19/11-30/11
Proveedor	Sergi Cristóbal Sofía Postigo	19/11-30/11
Corrección y revisión	Todos	1/12
Memoria	Carlos Mayorga	18/12-20/12
FormularioCorrección	Carlos Mayorga	20/12

TAREA	AUTOR	FECHA
Diagramas de secuencia **(Negocio + Presentación + Command)**		
AltaDepartamento	Clara Sotillo	2/12-7/12
BajaDepartamento	Airam Martin	2/12-7/12
ListarDepartamento	Sara Sacó	2/12-7/12
Actualizar	Sergi Cristóbal	2/12-7/12
Leer	Mariana Cucicea	2/12-7/12
AltaTrabajador	Axel Kahou	2/12-7/12
VincularProveedor	Carlos Mayorga	2/12-7/12
DesvincularMarca	Sofía Postigo	2/12-7/12
DiagramaComponentes	Axel Kahou	2/12-7/12
CerrarVenta	Steven Perez	2/12-7/12
DevolverVenta	Steven Perez	2/12-7/12
DiagramaDespliegue	Sergi Cristóbal	2/12-7/12
CalcularNominaDepartamento	Sara Sacó	2/12-7/12
Revisión y corrección	Steven Perez Axel Kahou	8/12-10/12

TAREA	AUTOR	FECHA
Código		
Negocio Venta	Steven Perez Airam Martin	6/12-20/12
Presentación Venta	Steven Perez Airam Martin	6/12-20/12
Negocio Proveedor	Sofía Postigo	6/12-20/12
Presentación Proveedor	Sofía Postigo	6/12-20/12
Negocio MarcaProducto	Sara Sacó Carlo Dubini	6/12-20/12
Presentación MarcaProducto	Sara Sacó Carlo Dubini	6/12-20/12
Negocio Trabajador	Álvaro Velasco Jorge Calvo	6/12-20/12
Presentación Trabajador	Álvaro Velasco Jorge Calvo	6/12-20/12
Negocio Departamento	Axel Kahou Carlos Mayorga	6/12-20/12
Presentación Departamento	Axel Kahou Carlos Mayorga	6/12-20/12
Negocio Producto	Clara Sotillo Mariana Cucicea	6/12-20/12
Presentación Producto	Clara Sotillo Mariana Cucicea	6/12-20/12

vFailureDialog	Todos	6/12-20/12
MainView	Sofía Postigo	6/12-20/12
ComponentsBuilder	Sofía Postigo	6/12-20/12
Creación SQL de Base de datos	Axel Kahou	6/12-20/12
Corrección y revisión	Todos	20/12
Arreglos código y presentación	Todos	20/12

2.3 FACTORES DE CALIFICACIÓN

Hemos llegado a un consenso todo el equipo y hemos decidido que todo el mundo ha realizado su trabajo y ha puesto interés en cada tarea que se le ha encomendado. Por lo que hemos otorgado 1 punto de 12 a cada miembro del equipo.

Steven Perez	1
Sara Sacó	1
Sergi Cristóbal	1
Carlos Mayorga	1
Axel Kahou	1
Mariana Cucicea	1
Carlo Dubini	1
Airam Martin	1
Álvaro Velasco	1
Sofía Postigo	1
Clara Sotillo	1
Jorge Calvo	1

3. PRODUCTO

3.1. CASOS DE USO

- **Departamento**
  - Alta Departamento : registra un nuevo departamento.
  - Baja Departamento : desactiva un departamento.
  - Mostrar Departamento: Muestra todos los atributos de un departamento.
  - Modificar Departamento: Cambia los atributos de un departamento.
  - Listar Departamento: Muestra una lista de todos los departamentos de la tienda, junto con todos sus atributos.
  - Calcular Nomina Departamento: Muestra el sumatorio de todos los empleados activos que pertenecen al departamento.
- **Trabajador**
  - Alta Trabajador:: registra un nuevo trabajador.
  - Baja Trabajador : desactiva un empleado, indicando que no trabaja ya en el zoo.
  - Modificar Trabajador: Cambia los atributos del trabajador.
  - Mostrar Trabajador: Muestra todos los atributos de un trabajador.
  - Listar Trabajador: Muestra una lista de todos los trabajadores de la tienda.
  - Mostrar Trabajador por Departamento: Muestra una lista de todos los trabajadores de un departamento, junto a todos sus atributos.
  - Calcular Sueldo Trabajador: Calcula el sueldo del trabajador en base al tipo.
- **Venta**
  - Abrir Venta: Inicia una venta de un conjunto de productos.
  - Añadir producto a venta: Registra nuevos productos en la venta.
  - Quitar producto de venta: Descarta productos en la venta.
  - Cerrar venta: Finaliza el pedido
  - Mostrar venta: Muestra todos los atributos de la venta.
  - Listar ventas: Muestra una lista con todas las ventas realizadas y sus atributos.
  - Listar ventas por empleado: Muestra una lista con todas las ventas realizadas por un empleado.
  - Devolución (necesita cambio): Se encarga de realizar la devolución del dinero al cliente y de los productos correspondientes a la tienda de una venta.
  - Mostrar venta por producto: Muestra una lista con todas las ventas de un producto junto con sus atributos.

- **Producto**
    - Alta Producto : registra un nuevo producto.
    - Baja Producto : desactiva un producto indicando que ya no está en la tienda.
    - Modificar Producto: Cambia los atributos de un producto.
    - Mostrar Producto; Muestra todos los atributos de un producto.
    - Listar Productos: Muestra una lista de todos los productos de la tienda con sus atributos.
    - Listar Productos por Marcas de Producto: Muestra una lista con todos los productos de una marca que vende la tienda, junto con sus atributos.
- **Marca de Producto**
    - Alta Marca : registra una nueva marca.
    - Baja Marca: desactiva una marca indicando que esa marca ya no se vende en el zoo.
    - Modificar Marca: Cambia los atributos de una marca.
    - Mostrar Marca: Muestra los atributos de una marca.
    - Listar Marca: Muestra una lista con todas las marcas que vende la tienda y sus atributos.
    - Listar Marcas por Proveedor: Muestra una lista con todas las marcas que proporciona un proveedor a la tienda junto con sus atributos.
- **Proveedor**
    - Alta Proveedor: registra un nuevo proveedor.
    - Baja Proveedor: desactiva un proveedor indicando que ya no proporciona productos al zoo.
    - Modificar Proveedor: Cambia los atributos de un proveedor.
    - Mostrar Proveedor: Muestra todos los atributos de un proveedor.
    - Listar Proveedores: Muestra una lista con todos los proveedores de la tienda y sus atributos.
    - Listar proveedores por marca de producto: Muestra una lista con todos los proveedores que venden una marca a la tienda, junto con sus atributos.
    - Vincular proveedor con marca de producto: Relaciona a un proveedor con una marca que distribuye.

    - Desvincular proveedor con marca de producto: Elimina la relación entre un proveedor y una marca que ya no distribuye.

4. ARQUITECTURA

La arquitectura empleada en nuestro proyecto es multicapa. La arquitectura multicapa consiste en dividir el software en tres capas principales promoviendo el aislamiento entre capas. Si ocurre algún error o cambio, no afecta directamente en el resto de capas. Otras de las características de esta arquitectura son la encapsulación, mayor seguridad, mayor optimización y mejor empaquetado. Las capas son las siguientes:

1. Capa de presentación

   Es la encargada de dar servicio a todos los usuarios que acceden al sistema. Esta capa implementa la interfaz gráfica y los ActionListener de los componentes que crean los transfer y los envían al controlador. El controlador que también se encuentra en la capa de presentación sirve de intermediario con la capa de negocio. Por último y no menos importante se encuentran los comandos que invocan el servicio de aplicación, traduciendo el contexto que recibe del controller.

2. Capa de negocio

   En ella encuentras los servicios del sistema e implementa la lógica del negocio. Los transfers que se encuentran en esta capa representan las entidades. Los servicios de aplicación proporcionan la lógica y consolidan qué los datos de entrada cumplan las reglas establecidas y se guarden en el almacén persistente. Este almacén persistente es el enlace con la capa de integración.

3. Capa de integración

   Esta capa se encarga de la comunicación con los recursos y sistemas externos. Interactúa con la base de datos usando los DAOs que extraen y guardan los datos en el almacén persistente. Las transacciones, las queries y algunos patrones también son partes fundamentales de esta capa.

4.1 PATRONES

Numeración de todos los patrones utilizados en nuestro proyecto:

❖ **Patrón Transfer Object**

Es un conocido patrón j2ee que porta datos sobre alguna entidad del modelo entre capas, encapsulando sus atributos. Es utilizado (con operaciones mutadoras y accesoras) en todo nuestro proyecto, para guardar en objetos nuestras entidades del modelo.

❖ **Patrón Transfer Object Assembler (TOA)**

Se trata de un patrón que compone objetos Transfer a partir de otras fuentes de datos (comúnmente, otros transfers), así, simplifica algunas peticiones costosas haciendo menos llamadas. En nuestro proyecto, el módulo Venta hace uso de este patrón para evitar concatenaciones de Transfers, como proponía Fowler. Memoria – Gestión de Tienda 20

❖ **Patrón Application Service (AS)**

Se trata de un patrón que hemos aplicado en la capa de negocio, para centralizar la lógica de nuestro programa. Evalúa las condiciones estipuladas en nuestros casos de uso, a partir de un transfer generado en los ActionListener de cada panel. Los casos de uso tienen asociada una operación de negocio en cada Servicio de aplicación particular (particular a su módulo). Además, añade una capa más a negocio y permite un código reutilizable y no duplicado.

❖ **Patrón Service to Worker**

La finalidad de este patrón es llevar a cabo el manejo de peticiones y la invocación de lógica de negocio antes de pasar el control a la vista. Service to Worker articula diversos patrones de presentación (controlador frontal, controlador de aplicación, ayudante de vista). Sin embargo, en esta parte del proyecto no se ha implementado la versión original del patrón, sino una adaptada. Por ejemplo, no se utiliza el patrón ViewHelper ni el FrontController.

❖ **Patrón Context**

La finalidad de este patrón es evitar utilizar información del sistema específica del protocolo fuera de su contexto relevante, se encarga de encapsular el estado de una forma independiente del protocolo para ser compartida por toda la aplicación

❖ **Patrón Command**

La finalidad de este patrón es encapsular una petición en un objeto, permitiendo así parametrizar a los clientes con diferentes peticiones, hacer cola o llevar un registro de las peticiones, y poder deshacer las operaciones.

❖ **Patrón Singleton**

La finalidad de este patrón es garantizar que una clase solo tenga una instancia (o un número controlado de instancias) proporcionando un punto de acceso global a ella.

❖ **Patrón Abstract Factory**

La finalidad de este patrón es proporcionar una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

❖ **Patrón Business Object**

La finalidad de este patrón es la de separar los datos de negocio de la lógica de negocio. Para ello, se construye un modelo del dominio compuesto por varios objetos de negocio (Business Objects) que permiten reflejar el dominio de la aplicación.

❖ **Patrón Domain Store**

La finalidad de este patrón es la de separar la persistencia del modelo de objetos. Se emplea para persistir de manera transparente un modelo de objetos, y resuelve los problemas de Concurrencia, Carga dinámica, Transaccionalidad y Persistencia.

5. ESTRUCTURA DEL CÓDIGO

Todo nuestro código está originado una vez terminado los diagramas de clase. Hemos seguido estos modelos de referencia para una mejor organización y funcionamiento de nuestro software. En nuestro código mantenemos las clases e interfaces de la primera parte del proyecto, por eso hemos catalogado estas nuevas clases referidas a la tienda con una notación al final del nombre JPA. Algunas clases como departamento que no se encontraban en DAO no le hemos puesto la notación. En JPA no tenemos nada en la carpeta integración.

6. PRUEBAS

Para el perfecto funcionamiento de todo nuestro software lo hemos dividido en 2 categorías: pruebas unitarias SA, y comprobaciones manuales.

Las pruebas SA aseguran la implementación de las reglas lógicas y de la capa de negocio de nuestro proyecto.

Hemos realizado una serie de pruebas manuales para atestar ciertas funcionalidades como las interfaces. Con ellas hemos depurado cualquier error para cambiar u optimizar nuestro código.

A continuación listamos los test que hemos realizado y comprobado que funcionan a la perfección:

7. RECURSOS

-El principal recurso utilizado para la elaboración de nuestro software es el programa IBM Rational Software Architect Designer. Con ella hemos podido crear todos los diagramas, tanto los de clase,secuencia y despliegue como todo el código programado para el funcionamiento de la aplicación.

-Para la gestión de base de datos hemos utilizado MySQL, y como entorno de trabajo el MySQL Workbench. Nos ha permitido el modelado de datos y el desarrollo de SQL para probar nuestro software.

-El lenguaje de programación que hemos utilizado es Java17.

-Para la gestión de la información y la escritura de la memoria se han utilizado documentos de google en un drive compartido con el acceso a la edición de todos los miembros del equipo.

Repositorio código: https://github.com/anavarro-fdi-ucm-es/ms2324animaliacod
Repositorio modelo: https://github.com/anavarro-fdi-ucm-es/ms2324animaliamod