MEMORIA BICITRON

Juan Alberto Camino Sáez
Samuel Eugercios Nevado
Víctor Fernández Duque
Daniel García Molero
Manuel Gómez Lagóstena
Arturo Marino Quintana

ÍNDICE

Introducción:

Nuestro proyecto software se basa en la gestión de una compañía comercial dedicada a la venta de bicicletas llamada Bicitron, por lo tanto nuestro software trabaja con datos relativos a las bicicletas, los clientes y las compras, así como con los datos de los trabajadores y departamentos que conforman la entidad, con el objetivo de gestionar la venta de bicicletas por parte del usuario, así como su devolución en caso de que lo solicite el usuario (para más detalles consultar la SRS).

Dado que el sistema requiere rapidez, nuestro software posee una respuesta en tiempo real para evitar problemas posteriores, así como una seguridad óptima para evitar errores críticos que pongan en riesgo la aplicación, y además es capaz de gestionar la concurrencia, pudiendo dar servicio a varios clientes al mismo tiempo sin problemas.

A continuación se hará una breve descripción de la arquitectura y los patrones presentes en nuestro proyecto software, así como las razones de su utilización.

El proyecto expuesto a continuación se divide en 2 partes:

-Modelos: modelo de requisitos, de diseño y de despliegue. Para representar los modelos se ha utilizado el lenguaje UML que está orientado a objetos. También se encuentran los diagramas de casos de uso y de actividades.
-Código: En lenguaje java.

2. URLS de los repositorios

-> Acceso al proyecto:

De cara al cliente a falta de crear un usuario de lectura, usted puede acceder a nuestro proyecto a través de un usuario y contraseña. Accediendo de esta manera a nuestros repositorios para que pueda observar todo el “esqueleto” del proyecto (código, diagramas, etc...) y poder añadir comentarios con el fin de establecer una óptima comunicación entre el cliente y nosotros los desarrolladores.
De esta manera se podrán ir generando versiones actualizadas del proyecto hasta llegar a un entendimiento entre las necesidades del cliente y el trabajo de los ingenieros.

URLS:

-Para acceder al código del proyecto:
https://versiones.fdi.ucm.es:10005/svn/ms1617biciscod

-Para acceder a los modelos y diagramas del proyecto:
https://versiones.fdi.ucm.es:10005/svn/ms1617bicismod

-Para acceder a la documentación del proyecto:
https://versiones.fdi.ucm.es:10005/svn/ms1617bicisdoc

USUARIO	CONTRASEÑA
ms1617jacamino	**Omega7100**
ms1617saeugercios	**Omega6542**
ms1617vfernandez	**Omega1328**
ms1617dgarcia	**Omega3258**
ms1617mgomez	**Omega5417**
ms1617amarino	**Omega9851**

3. Diagramas y modelos

A continuación presentamos todos los diagramas que contiene nuestro proyecto:

- Diagramas de casos de uso: un diagrama de casos de uso es un diagrama que muestra un conjunto de casos de uso, actores y sus relaciones. Nuestro proyecto posee un diagrama de casos de uso por cada módulo.

- Diagrama de actividades: Los diagramas de actividades presenta las funciones de nuestro proyecto e informa de los pasos que sigue nuestra aplicación en lenguaje natural. Nuestro proyecto consta de un diagrama de actividad por cada función que realiza nuestra aplicación.

- Diagrama de clases: los diagramas de clases muestran las clases y las interfaces que se usan en el proyecto y sus relaciones. Estas relaciones tienen varias componentes: Nuestro proyecto tiene diagramas de clase de todos los módulos.

- Diagrama de secuencia: nuestros diagramas de secuencia muestran las interacciones entre los objetos organizados en una secuencia temporal. En estos diagramas se detallan todos los pasos que sigue nuestro programa para realizar una función a nivel código. Nuestro proyecto posee un diagrama de las vistas(vista alta cliente, ya que el resto son prácticamente iguales), el controlador, el dispatcher, el command y las querys. En cuanto a negocio e integración, debido a la semejanza entre los módulos, sólo están hechos los diagramas de las operaciones del módulo cliente. Aparte el módulo compra tiene el diagrama de cerrar compra, añadir artículo, eliminar artículo, devolver articulo e iniciar compra, el módulo clientes el de número clientes, y el módulo bicicleta el de número bicicletas, debido a que son las funciones que difieren significativamente al resto en cuanto a funcionalidad. En la parte de la tienda de nuestra aplicación, también tenemos diagramas de secuencia del modulo departamento, ya que se implementa con JPA, y en el módulo tienda de asignar y desasignar departamento, que son las funciones que difieren significativamente del resto.

- Diagrama de despliegue: se utiliza para modelar el hardware empleado en las implementaciones y las relaciones entres sus

componentes. Esto es, muestras las relaciones físicas entre los componentes hardware y software en el sistema final. Nuestro proyecto posee un diagrama de despliegue.

Para la implementación del código se han seguido las indicaciones del modelo.

4. Patrones

Un patrón describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema de tal modo que se puede aplicar esta solución un millón de veces, sin hacer lo mismo dos veces.

En nuestro proyecto hemos aplicado el modelo-vista-controlador, que aplicando una arquitectura multicapa separa la vista por la cual el usuario se comunica con la aplicación del modelo, que se encarga de realizar todas las operaciones, usando un controlador que media entre ellos. De esta manera conseguimos que el software sea fácilmente mantenible ya que tenemos las operaciones claramente diferenciadas. En nuestra aplicación se usa además el modelo pasivo, el cual el modelo se comunica con la vista a través del controlador, delegando en el dispatcher.

También hemos usado los siguientes patrones:

- Factoría abstracta: proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas. Hemos usado este patrón para crear las clases que actúan de servicio de aplicación y de DAO en nuestro proyecto, y también en la factoría de transacciones y de querys.

- Singleton: este patrón garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella. Hemos usado este patrón para las vistas, para el controlador, y para el transaction manager, para poder llamarlos desde cualquier parte, además de que cualquier cambio en estas clases por un cambio en los requisitos sería fácilmente llevado a cabo ya que solo habría que cambiar el objeto al que instancia la clase que actúa de singleton.

- Servicio de Aplicación: este patrón encapsula todas las reglas de negocio en una clase. Hemos usado este patrón para llevar a cabo las operaciones de nuestra aplicación, ya que cualquier cambio en estas reglas serían fácilmente integradas en nuestra aplicación ya que sólo habría que modificar esta clase que actúa con este patrón.

- DAO (Data Access Object): usado para comunicarse con datos externos. Hemos usado este patrón para guardar y cargar datos de nuestra base de datos, separando esta función del servicio de aplicación.

- Transfer: usado para guardar todos los datos en un único objeto y usarlo para transmitir los datos. Usamos este patrón para pasar el objeto necesitado entre todos los patrones de la aplicación.

- Contexto: hemos usado este patrón para devolver datos del modelo y poder enviarlos a la vista, así como un evento para poder mostrar la información necesaria.

- Controlador de aplicación: un controlador que intermedia entre la vista y el modelo, usando como refuerzo el patrón command y el dispatcher.

- Command: usado para ampliar la funcionalidad y la mantenibilidad de la aplicación, usando un comando para cada operación de nuestra aplicación.

- Dispatcher: usamos este patrón para mostrar las distintas vistas disponibles según se requiera, con la información necesaria para seguir correctamente el funcionamiento de la aplicación.

- Transaction Manager: usado para permitir la concurrencia en nuestra aplicación, usando una transacción cada vez que se vaya a llevar a cabo una operación de negocio.

- Service to worker: El patrón controlador frontal, el command y el dispatcher se integran en este patrón, que usamos en nuestro proyecto, salvo por el Front Controller, ya que no es necesario debido a que el contexto de ejecución de esta aplicación es siempre el mismo.

- Objetos de negocio: Usado para mantener la relación entre los objetos de la base de datos y los del código java, usado en JPA

- Almacén del dominio: Usado para implementar la parte de integración, implementado con el framework JPA.