Relatório do Projeto SrEstoque

Alunos Wellington Lins Claudino Duarte

1 - Introdução

Neste documento será abordada uma descrição simples de um sistema de controle de estoque. Para construir esse pequeno sistema foi feito um breve estudo sobre o minimundo que envolve o tema do projeto . Ao longo de pequeno trajeto pude constatar que sua criação inicia –se simples mas à medida que o projeto vai tomando corpo, percebe—se que o controle de estoque é uma aplicação que não é tão trivial assim. Logo para que a sua concepção seja feita de forma satisfatória é necessário mais informações para lapidar o projeto, seria necessário uma entrevistas com pessoas que trabalham com armazenagem de produtos, pois conhecem de perto quais o processos necessários para um controle de estoque sem falhas grosseiras.

Aqui está uma ideia inicial do que seria um sistema de controle de estoque , que atende requisitos mínimos.

Temos uma breve descrição do que foi usado em termos de design de projeto no sistema que chamamos de “SrEstoque” e uma visão geral de como trabalhamos com o banco de dados.

2- Objetivos do sistema

Oferecer um sistema para o gerenciamento de estoque de um centro de distribuição de mercadorias de uma rede de lojas. As mercadorias compradas pela rede serão armazenadas no estoque central. A partir disso, à medida que as lojas necessitem das mercadorias, o pedido será feito para o estoque central, as mercadorias serão entregues as lojas por transportadoras e serão armazenadas as datas de entrega e saída de tais mercadorias.

O “SrEstoque” é um sistema que poderá ser aplicado para controlar o estoque de uma rede de lojas em qualquer ramo.

3- Descrição do sistema

· O sistema deve ter a capacidade de armazenar os produtos contidos no estoque, para que esses possam ser controlados individualmente. Outro detalhe importante no cadastro do produto é armazenar a quantidade mínima que deverá ter desse produto no estoque.

· Cada produto terá um fornecedor relacionado a ele, sendo possível controlar os produtos divididos por fornecedores.

· Os produtos devem ser divididos por categoria, ou seja, cada produto terá uma categoria.

· As entradas e saídas dos produtos deverão ser registradas no programa, para futuramente obtermos um histórico completo de todo o trajeto do produto dentro do centro de distribuição.

· Na entrada do produto será necessário armazenar a data do pedido e a data de entrega da mercadoria, para depois podermos analisar quanto tempo o pedido demora a chegar ao estoque.

· Na saída, obrigatoriamente será informada a loja para a qual a mercadoria foi enviada, pois ao final do mês devemos fazer o fechamento do faturamento para saber qual é a loja que mais obteve vendas.

· Outra capacidade que o sistema deverá ter é calcular o peso total de uma entrada ou de uma saída.

· No programa, devem-se apresentar os produtos nos quais a sua quantidade total em estoque é menor ou igual à quantidade mínima requerida em estoque definida previamente.

· A transportadora será outro item importante, pois devemos saber qual transportadora é mais utilizada para fazer a entrega dos produtos e qual é a mais utilizada para fazer a saída.

· Uma questão que o cliente deseja observar é qual categoria possui mais item no local do estoque.

4- O que fizemos:

Sobre o banco:

Como SGBD para o gerenciamento do banco de dados, será utilizado o PostgreSQL, que foi escolhido por tratar-se de um banco com licença gratuita, amplamente usado por desenvolvedores e empresas e que possui competitividade com ferramentas pagas e já consolidadas no mercado.

Nessa primeira versão do sistema foi criado apenas CRUDS para as seguintes entidades:

Categoria, Cidade , Entrada , Fornecedor, ItemEntrada, ItemSaida, Loja, Produto,

Saída, Transportadora e Usuário.

Assim é possível para o usuário poder criar , remover , editar ou visualizar qualquer uma da entidades.

1. Categoria -> entidade responsável por separar as mercadorias;
2. Cidade -> entidade responsável pela cidade onde existe uma ou mais lojas da rede;
3. Entrada -> entidade responsável pela entrada de mercadorias
4. Fornecedor -> entidade responsável por fornecer mercadoria para as
5. ItemEntrada -> entidade responsável por manter os itens de entrada de mercadorias
6. ItemSaida -> entidade responsável por manter os itens de saída de mercadorias
7. Loja -> entidade responsável manter as lojas cadastradas no sistema
8. Produto -> entidade responsável manter os produtos do estoque
9. Saída -> entidade responsável pela saída de mercadorias
10. Transportadora --> entidade responsável pelo transporte das mercadorias
11. Usuário -> entidade responsável por armazenar os usuários do sistema

Sobre a implementação em java:

MVC

Usamos o design pattern MVC (Model View Controll ), em português modelo-visão-controlador ,que separa a apresentação da informação da interação do usuário com modelo (classes pojo ou beans ,regras de negócios, lógica e funções.) que fazem parte do domínio de negócio, ou seja, o mundo da aplicação e não são de interesse do usuário essa parte para ele é transparente.

Para a parte de visão que pode ser qualquer saída de representação dos dados, usamos o JSF

Toda a manipulação dos dados pelo cliente é feita por intermédio do browser onde as páginas web apresentam as interfaces gráficas que são muito simples e intuitivas de serem usadas.

Usamos ainda um componente bootstrap para disponibilizar uma interface amigável e moderna para o usuário poder usar o sistema de forma mais agradável.

O controlador faz uma ponte dos dados inseridos pelos clientes através da interface gráfica de entrada, convertendo-a em comandos para o modelo ou visão.

Usamos o MVC para facilitar a compreensão de design do projeto e para termos a reusabilidade de código e separação de conceitos em camadas.

DAO

Usamos o padrão DAO (acrônimo de Data Access Object), objeto de acesso a dados que é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados, assim não sofremos pelo árduo caminho do acoplamento. Todas as funcionalidades de bancos de dados, tais como obter as conexões, mapear objetos Java para tipos de dados SQL ou executar comandos SQL, devem ser feitas por classes DAO.

Usando objetos de acesso a dados temos a vantagem da separação simples e exata entre duas partes importantes de uma aplicação que não podem conhecer quase que nada uma da outra, e que podem evoluir de forma independente quantas vezes forem necessárias. No mundo atual com a dinâmica constate a mudança de código de uma aplicação é algo inevitável e alterar a lógica de negócio pode-se esperar apenas a implementação de uma interface, ao passo que modificações na lógica de persistência não alteram a lógica de negócio, desde que, não alteramos a interface entre elas.

Esse padrão esconde todos os detalhes relativos a armazenamento de dados do resto da aplicação, o que é muito bom para todo o sistema, pois cada parte trabalha em conjunto com o todo, como se fossem componentes lego que se encaixam sem quebrar uma as outras para montar uma estrutura final.

Nas classes de acesso ao banco de dados usamos o JPA para o mapeamento das entidades, e deixamos o gerenciamento da aplicação ser feito pelo Container web Glassfish , garantido para a nossa aplicação segurança , escalabilidade e transacionabilidade.

Abstract Factory é um padrão de projeto que permite a criação de famílias de objetos relacionados ou dependentes por meio de uma única interface e sem que a classe concreta seja especificada.

Sentimos a necessidade de usar esse padrão porque nossa aplicação foi concebida exatamente com essa concepção de métodos assinados por interfaces.

Esse padrão ajuda a monitorar as classes de objetos criadas por uma aplicação. Como a fábrica encapsula a responsabilidade e o processo de criar objetos, isso isola os clientes das classes de implementação. Os clientes manipulam as instâncias através das suas interfaces abstratas

Factory Method

Factory Method possui a seguinte intenção:

“Definir uma interface para criar um objeto, mas deixar as subclasses decidirem que classe instanciar. O Factory Method permite adiar a instanciação para subclasses.”

A vantagem em utilizar o padrão Factory Method é a grande facilidade para incluir novos objetos e não é necessário alterar código algum, apenas criamos o objeto e a sua fábrica. Todo o código já escrito não será alterado de modo algum.

FACADE

Usamos o padrão Facade para distanciar ainda mais o usuário das regras de negócio, juntamente com o MVC . Assim a view (interface gráfica) comunica-se com a fachada que por sua vez vai chamar os controladores dessa forma criamos mais uma “camada” que vai deixar mais densa a distância entre as regras de negócio e seus consumidores.

A intenção do padrão se encaixa perfeitamente em um requisito conceitual por parte de uma necessidade especifica do projeto, pois sua definição justifica claramente o seu uso :

“Fornecer uma interface unificada para um conjunto de interfaces em um subsistema. Facade define uma interface de nível mais alto que torna o subsistema mais fácil de ser usado.”

O Facade vai definir operações a serem realizadas com estes subsistemas, sua utilização foi bem simples, apenas foram criadas as classes fachadas, foi criada uma fachada para cada entidade, que iram se comunicar com os subsistemas no lugar no cliente.

