Universidade Norte do Paraná

SISTEMA DE ENSINO PRESENCIAL CONECTADO
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

U-LIZ-SYS V.0.5:

Um ensaio de sistema de gerenciamento de produção e venda de sapatos

Caicó
2017

SAINT-CLAIR DA CUNHA LIMA

U-LIZ-SYS V.0.5:

Um ensaio de sistema de gerenciamento de produção e venda de sapatos

Trabalho em grupo apresentado à Universidade Norte do Paraná - UNOPAR, como requisito parcial para a obtenção de média semestral nas seguintes disciplinas: Banco de Dados II; Programação para Web I; Análise Orientada a Objetos II; Programação Orientada e Objetos; e Seminários IV.

Orientadores: Profº. Leonardo Ferrareto
Profº. Paulo Henrique Terra
Profª. Iolanda C.S. Catarino
Profº. Anderson E. M. Gonçalves
Profº. Roberto Yukio Nishimura

Caicó
2017

SUMÁRIO

ÍNDICE DE FIGURAS

1. INTRODUÇÃO

A fábrica de calçados Flor de Liz, especializada em produção, venda e distribuição de calçados é uma empresa em expansão. Sua produção aumentou em 10 vezes em relação à produção inicial do empreendimento, de maneira que está assumindo proporções que requerem maior organização e robustez no gerenciamento de seus processos internos e interações com clientes.

Com o objetivo de ter essa necessidade sanada, tenciona a aquisição de um sistema de software que dê suporte às suas necessidades. O foco principal do sistema é o gerenciamento do processo de fabricação dos calçados, bem como a dinâmica de escoamento da produção, incluindo a logística de entrega.

Assim, objetiva-se, nesse estudo, o planejamento, esboço/prototipação e desenvolvimento (1) das estruturas de armazenamento de dados, (2) processos de negócio envolvendo a produção, venda e entrega de produtos, (3) sistema informacional de gerenciamento dos dados modelados e implementados na camada de armazenamento de dados. De forma a prover escalabilidade e facilidade de acesso do sistema (já se antecipando à possibilidade de posterior ampliação da empresa em diversas filiais), optar-se-á pelo uso de uma aplicação web, a qual será hospedada em um servidor local ou *cloud based*. Destarte, poder-se-á acessar a aplicação independentemente de restrições espaciais.

Para satisfazer a exigência de levantamento e análise de requisitos, será utilizada a Unified Modeling Language – UML como ferramenta para melhor compreensão das atividades a serem realizadas pelo sistema, descrevendo e especificando os requisitos funcionais do sistema. O uso da UML proporcionará, também, refinamento gradual das especificações do sistema, caminhando na direção da implementação definitiva da solução. Para se elaborar os diagramas em notação UML utilizar-se-á a ferramenta Visual Paradigm Community Edition, que possui algumas limitações, por ser versão não paga, sem, não obstante, interferir nas metas a serem alcançadas por este trabalho.

Como base para a modelagem e gerenciamento de dados, utilizar-se-á a ferramenta BrModelo para fazer a modelagem da estrutura de dados, consistindo em um Diagrama Entidade Relacionamento – DER, tanto em uma visão conceitual quanto uma visão lógica, mais perto da implantação em um Sistema de Gerenciamento de Banco de Dados – SGBD, propriamente dito.

Optou-se por utilizar um banco de dados seguindo o modelo Relacional, utilizando a Structured Query Language – SQL como linguagem de gerenciamento e consulta de dados. O SGBD utilizado será o PostgreSQL, juntamente com o programa PgAdmin, para gerenciamento do banco em alto nível.

A aplicação será desenvolvida utilizando tecnologia Java, que se comunicará com o banco de dados subjacente por meio do Java Database Connectivity – JDBC, responsável por fazer a interface entre aplicação e dados. Isso facilitará o envio de comandos SQL para o SGBD. Essa combinação de SGBD e JDBC permite realizar a comunicação entre aplicação e banco de dados de maneira fluida, leve e robusta.

A interface entre as classes/objetos Java e a apresentação das informações ao usuário será feita por meio do uso de JavaServer Pages, encapsulando muitos dos procedimentos repetitivos e passíveis de erro que consiste da geração manual e declarativa de páginas por meio de outras classes Java. Objetiva-se seguir o Padrão Model-View-Controller – MVC no desenvolvimento. Com isso, poderá ser mantida a modularidade de partes do sistema, de maneira a facilitar a manutenção e a organização das classes/objetos e outros componentes da aplicação.

De forma a otimizar o processo de desenvolvimento, o Ambiente Integrado de Desenvolvimento (IDE) utilizado será o Netbeans, integrando-o com o WebContainer Apache Tomcat, em sua versão 7.

De maneira a garantir concisão na descrição desse trabalho, o código fonte dos principais arquivos descritos no texto pode ser encontrado nos apêndices. Foram omitidos os métodos setters e getters que não possuíam comportamento específico diferente do de um Plain Old Java Object, apresentando somente os métodos que possuem alguma lógica interna específica ou cuja apresentação seja de relevância para a compreensão da descrição.

Além disso, diante da impossibilidade de incluir todo o código fonte no corpo deste trabalho, os arquivos resultantes do processo de desenvolvimento realizado podem ser acessados na íntegra no repositório criado pelo autor, hospedado na plataforma GitHub. O acesso é público e pode ser realizado pelos hyperlinks: https://goo.gl/EF51gJ ou http://bit.ly/portfolio3_saint.

Além das classes Java e dos arquivos JavaServer Pages, pode-se encontrar o arquivo em formato .sql gerador do banco de dados. Além do trecho desse mencionado arquivo que incluído no Apêndice A, responsável pela definição da estrutura de dados do banco, pode-se encontrar um script que, ao ser executado,

ralizará a população do banco com dados de teste, os quais foram utilizados durante o processo dedesenvolvimento. Ambos os blocos de comandos SQL foram gerados automaticamente por meio de dois scripts escritos em Python, também presentes no repositório.

Pode ser encontrado, ainda, uma pasta compactada com todo o conteúdo do projeto para ser importado no Netbeans, bem como uma pasta compactada com os diagramas UML utilizados bem como as imagens deles exportados.

2. MODELANDO O NEGÓCIO

2.1 INFORMAÇÕES ADICIONAIS SOBRE A EMPRESA

A fábrica de sapatos Flor de Liz atua na produção, venda e distribuição de calçados femininos de alta qualidade, com foco em duas linhas distintas de produção: (01) calçados de luxo voltados para consumidores da classe média alta e (02) calçados esporte voltados ao público de classe média. Sua necessidade maior, com base nas informações obtidas em Análise de Negócio e Levantamento de Requisitos, é um sistema que dê conta de registrar os processos de produção de calçados, estoque de produtos e geração de pedidos e entregas por parte dos clientes.

Embora a empresa conte com vários funcionários e precise de uma solução para gerenciá-los, bem como gerenciamento de transações financeiras, de entrada e saída de materiais, de fornecedores e outros aspectos necessários a uma empresa de seu porte, optou por enfocar apenas em sua necessidade atual.

2.2 REQUISITOS

O processo de Levantamento de Requisitos foi feito e indicou a necessidade de um controle com funções de inserção, atualização, consulta e exclusão de entidades diversas ligadas ao processo abordado pelo sistema a ser produzido. Como resultado do processo de identificação de requisitos, verificou-se as necessidades listadas no Quadro 1: Lista de Requisitos Funcionais.

Além disso, foram identificadas as seguintes regras de negócio foram apontadas pelos usuários:

- Inicialmente é feito o cadastro do calçado que será produzido.
- Em seguida, é cadastrado as produções de calçados que ficarão em estoque para serem vendidas.
- Depois cadastram-se os clientes e os pedidos dos clientes.
- E por último são geradas as ordens de entrega de pedido.
- Um calçado poderá ter diversas produções de calçados e cada produção de calçado sempre estará atrelado a um calçado específico.
- Um cliente pode fazer diversos pedidos e um pedido obrigatoriamente

estará vinculado a um cliente.

- Um pedido pode conter diversos itens de calçados e cada item de calçado será sempre de um único pedido.
- Cada item de pedido será referente a um calçado específico, mas cada calçado poderá estar em vários itens de pedidos.
- Um pedido pode ter várias entregas de pedido e cada entrega de pedido é sempre de um pedido específico.
- Uma entrega de pedido contém diversos itens de entrega e cada item de entrega será sempre de uma única entrega de pedido.
- Cada item de entrega de pedido é referente a um calçado específico e cada calçado poderá estar em vários itens de entrega de pedido.

Título do Requisito	**Descrição do Requisito**
Requisito 1	Controle de Calçado: Neste controle que representa os calçados, é descrito o nome do calçado, a qual coleção ele pertence, qual o menor tamanho e o maior tamanho, qual a cor do calçado e o preço de custo.
Requisito 2	Controle de Produção: Neste controle é armazenado as informações referentes ao lote de produção de calçados como um código de produção, código do calçado a ser produzido, data de produção, tamanho do calçado produzido e a quantidade de calçados produzida.
Requisito 3	Controle de Cliente: Neste controle, temos as informações do nosso cliente como nome do cliente, nome da loja do cliente, endereço do cliente, CNPJ do cliente, CPF do dono da loja, telefone de contato.
Requisito 4	Controle de Pedido: Neste controle, registramos os dados do pedido do cliente como o nome do cliente, o produto, o tamanho e a quantidade. Lembrando que um pedido pode conter diversos produtos, aliás quanto mais, melhor. Ainda no pedido temos o valor de cada item, um valor total por item, um valor prévio da soma de todos os itens, um desconto e o valor final a ser pago pelo cliente.
Requisito 5	Controle de Entrega dos Pedidos: Neste controle, precisamos associar os pedidos à entrega dos mesmos até o nosso cliente.

Quadro 1: Lista de Requisitos Funcionais

Adicionalmente, identificou-se os seguintes requisitos derivados da descrição feita pelos usuários:

- A cada pedido realizado, a quantidade de produtos presentes no pedido deve ser deduzida do total mantido em estoque.
- A cada lote de calçados produzido, a quantidade de itens que deram entrada no estoque deve ser incrementada para cada item produzido pelo lote.

Com base nos requisitos apontados e derivados, bem como nas regras de negócio percebidas no levantamento de requisitos, realizou-se o processo de análise dos requisitos levantados, bem como a modelagem dos processos realizados no setor da fábrica, os quais serão abordados pelo sistema proposto.

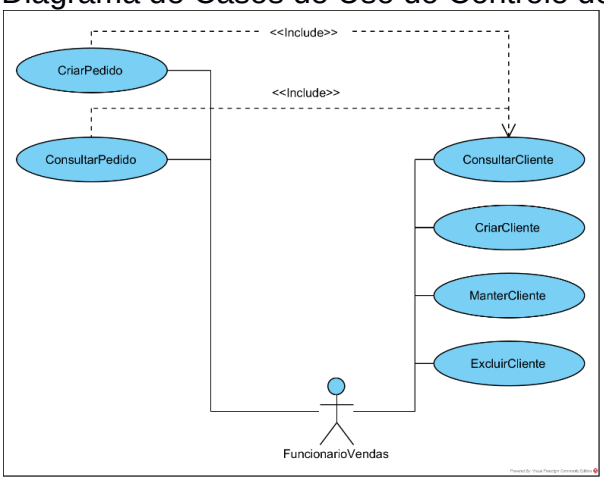
2.3 MAPEANDO OS REQUISITOS DO SISTEMA

De maneira a melhor compreender os detalhes de como o sistema deve funcionar, elaborou-se uma série de diagramas de Caso de Uso, seguindo o padrão UML, utilizando a ferramenta Visual Paradigm Community. O foco principal do sistema são as ações que atuam sobre as seguintes entidades (as quais foram também representadas em DER, mais adiante apresentado): (1) Modelos de Calçado, (2) Itens de Estoque, correspondendo a variações (em termos de tamanho e características individuais) de Modelos de Calçados, (3) Lotes de Produção, (4) Pedidos, (5) Entregas, as quais serão diretamente ligadas a pedidos específicos, e (6) Clientes aos quais os Pedidos e Entregas estarão vinculados.

Cada uma das entidades, bem como outras que se constituem em itens componentes do todo, devem ter operações a serem executadas pelo sistema que permitam a inclusão, alteração, consulta e exclusão (Create, Read, Update e Exclude – CRUD). Destarte, apresenta-se os Casos de Uso dos controles de cada uma das entidades apontadas.

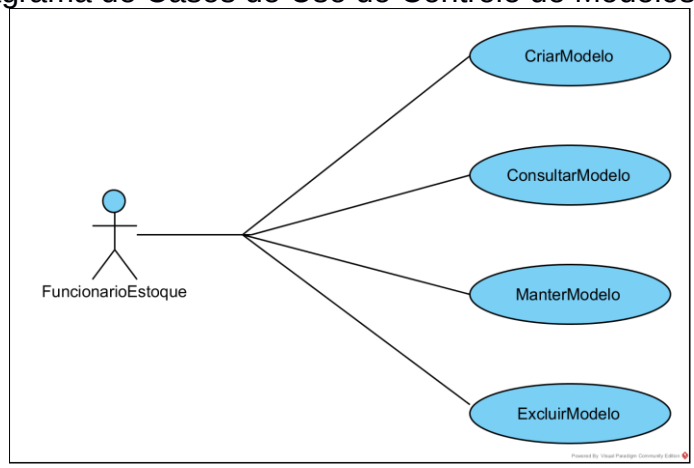
2.3.1 Diagramas de Casos de Uso

Figura 1: Diagrama de Casos de Uso do Controle de Clientes



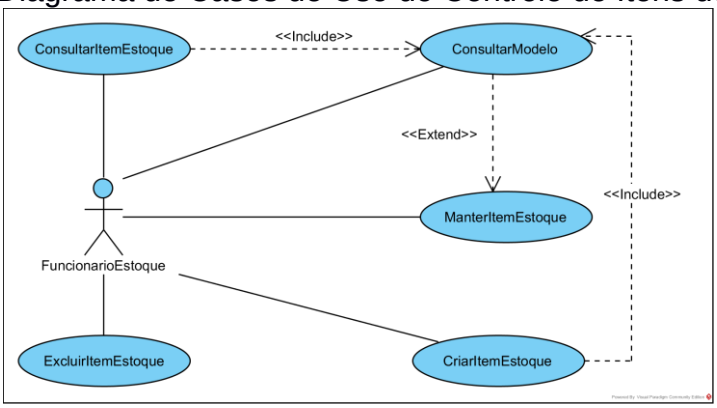
Fonte: Autoria Própria

Figura 2: Diagrama de Casos de Uso do Controle de Modelos de Calçados



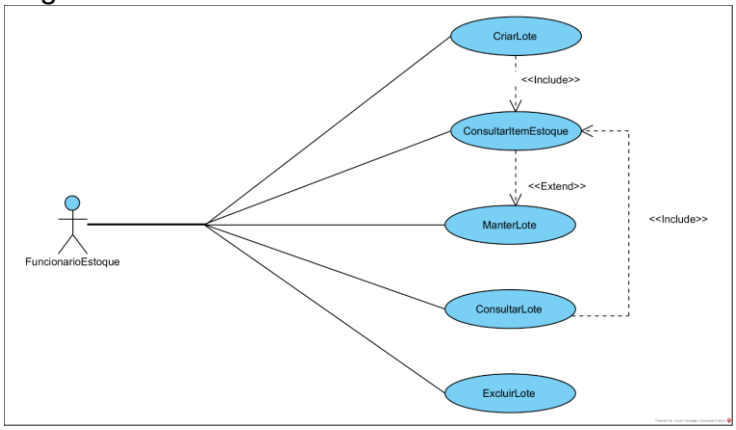
Fonte: Autoria Própria

Figura 3: Diagrama de Casos de Uso do Controle de Itens de Estoque



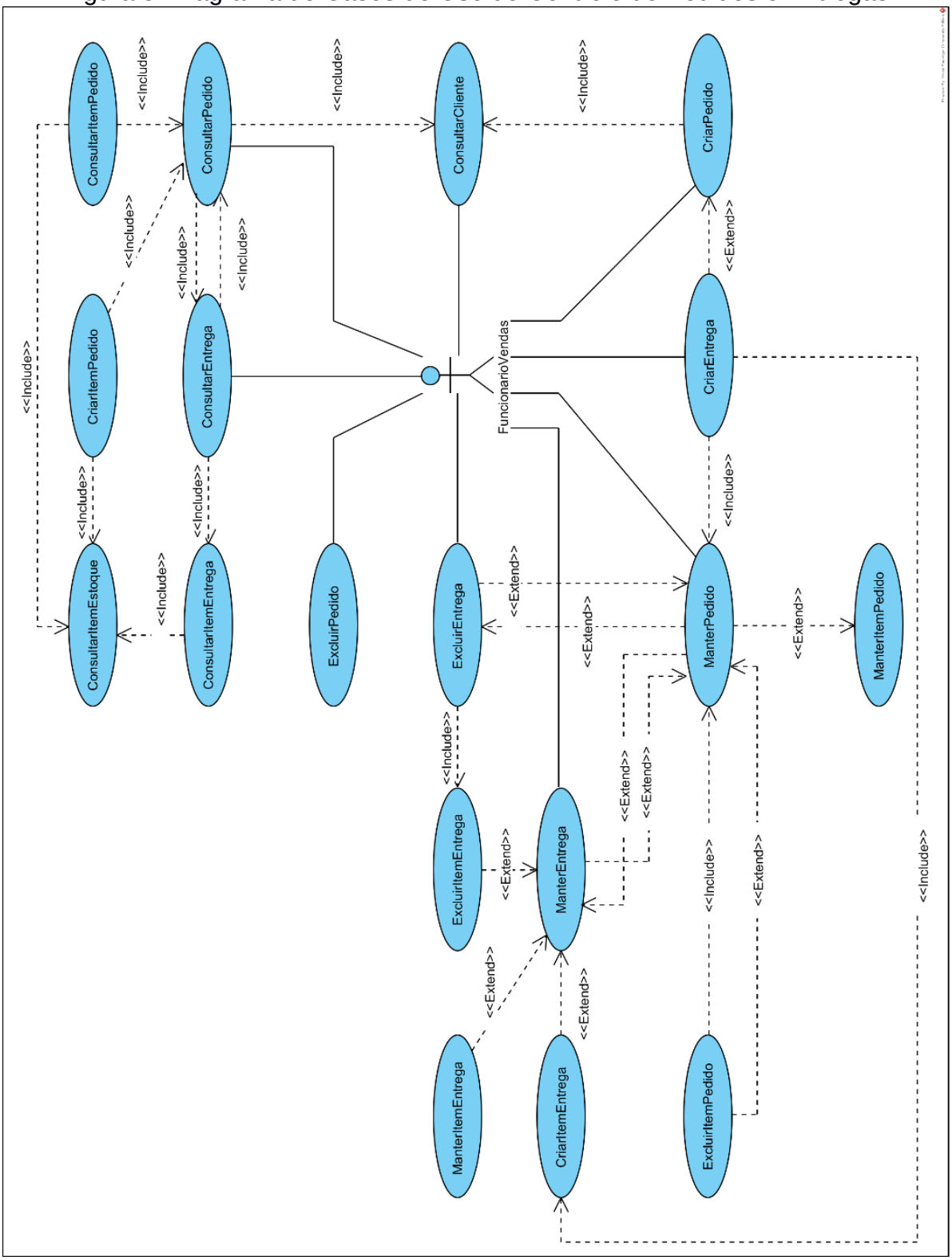
Fonte: Autoria Própria

Figura 4: Diagrama de Casos de Uso do Controle de Lotes de Produção



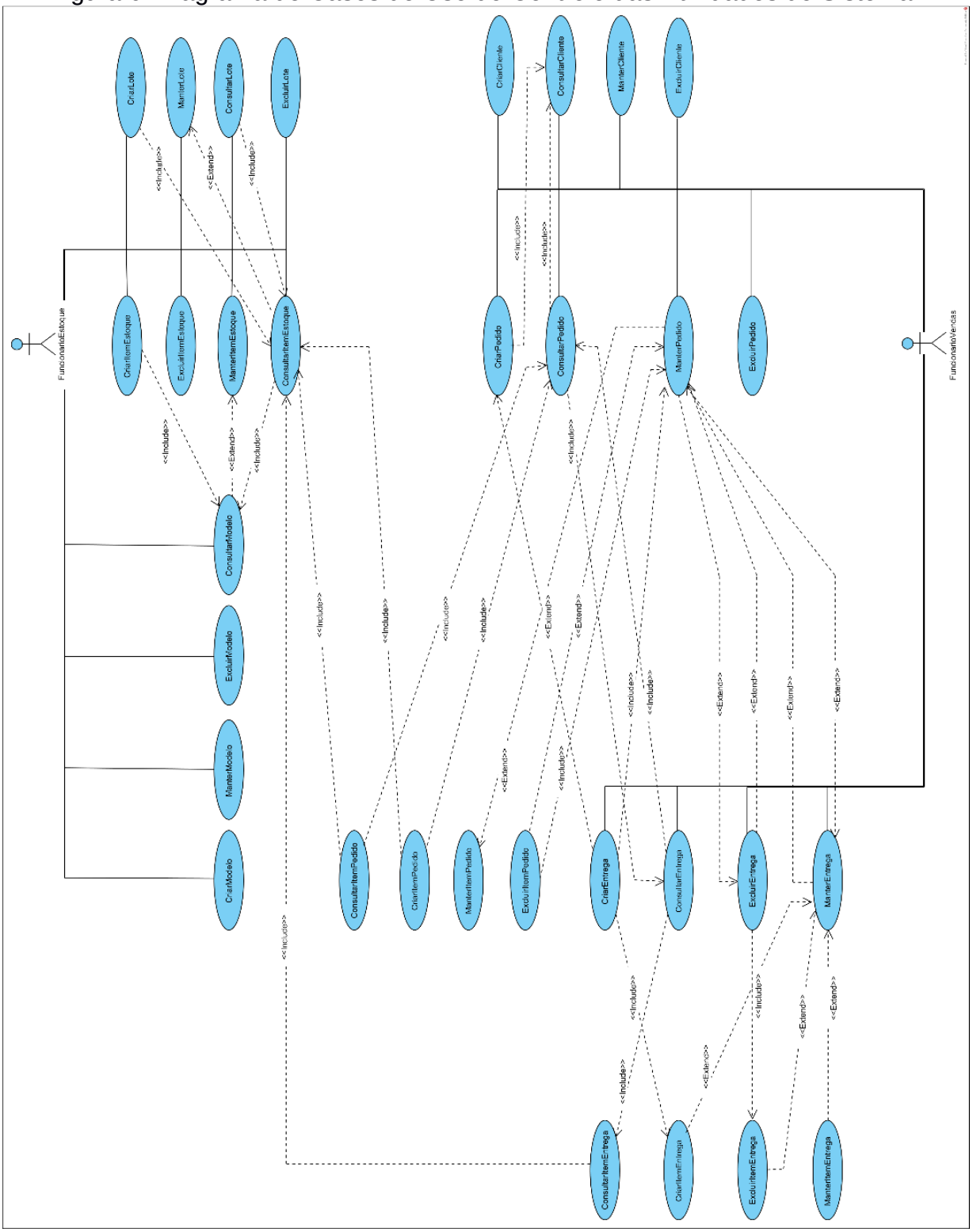
Fonte: Autoria Própria

Figura 5: Diagrama de Casos de Uso do Controle de Pedidos e Entregas



Fonte: Autoria Própria

Figura 6: Diagrama de Casos de Uso do Controle das Atividades do Sistema



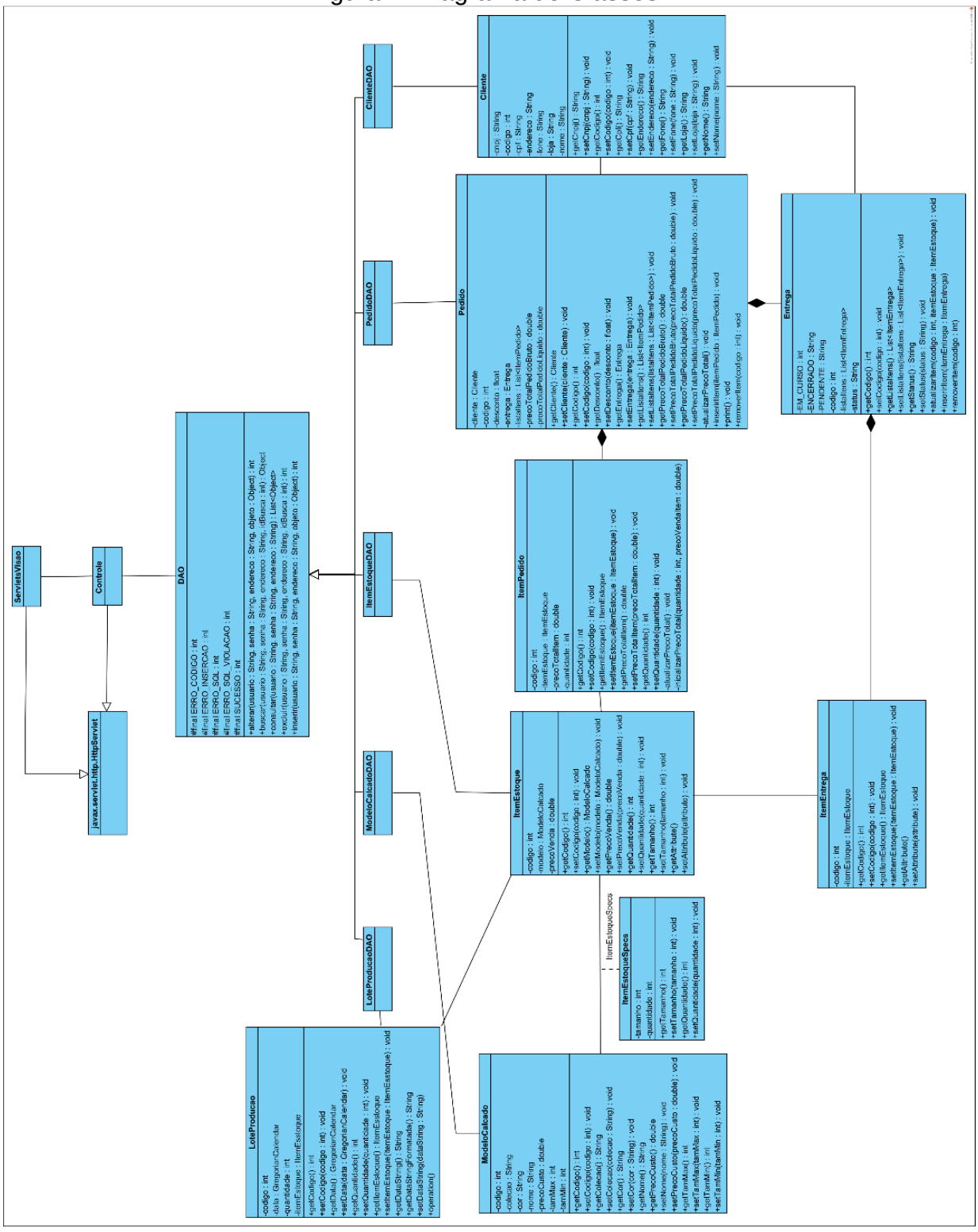
Fonte: Autoria Própria

2.3.2 Diagrama de Classes

Havendo sido definidos e especificados, por meio dos Diagramas de Casos de Uso, os processos a serem realizados pelo sistema prosseguiu-se com a elaboração do conceitual de como se daria a implementação *per se* do sistema, idealizando-se quais as classes que se poderia utilizar para modelar os objetos, mensagens e interações componentes do sistema. Com esse intuito, elaborou-se o Diagrama de Classes, tendo em mente as funcionalidades elencadas no Levantamento de Requisitos, bem como as especificações resultantes dos Diagramas de Caso de Uso anteriormente elaborados.

De igual importância para a definição de quais classes se utilizaria, bem como qual seriam as relações entre elas, foi a modelagem da estrutura de dados subjacente ao sistema (cujo DER será apresentado posteriormente).

Figura 7: Diagrama de Classes



Fonte: Autoria Própria

Note-se que as classes representadas no diagrama contemplam as classes necessárias à modelagem dos dados no sistema, bem como as classes que servirão de interface para gravação e recuperação de informações junto ao banco de dados,

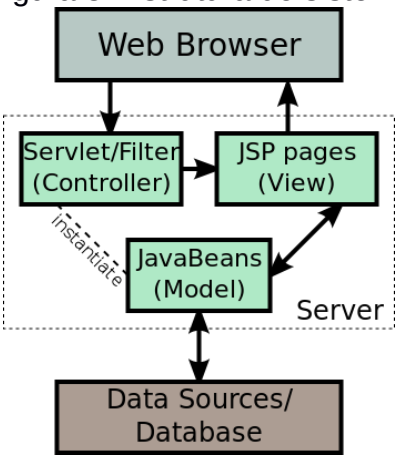
responsável fará a persistência dos dados. Além dessas classes, o sistema conta com dois conjuntos de Servlets Java.

O primeiro dos conjuntos contém Servlets que atuam como controladores, do padrão MVC, de forma a mediar o acesso ao banco de dados e a apresentação. Há um Servlet de Controle para cada uma das Classes de modelagem, o qual recebe os dados do usuário, faz o seu tratamento e, por meio da interface de acesso aos dados, os envia para a persistência.

Quanto ao segundo grupo de Servlets, trata-se, na verdade de um conjunto de páginas do tipo JavaServer Pages que, em tempo de execução, são convertidas em classes Servlets Java as quais são compiladas. Esses Servlets atuam como geradores dinâmicos de páginas HTML, encarregando-se de enviá-las ao Browser para interação com o usuário.

O fluxo de informação é orientado pelos Servlets de Controle, recebendo *inputs* do usuário, via requisições HTTP e gerando conteúdo por interação com o banco de dados subjacente, direcionando a resposta HTTP para as JSPs gerarem o conteúdo final a ser exibido ao usuário no Browser.

Figura 8: Estrutura do sistema

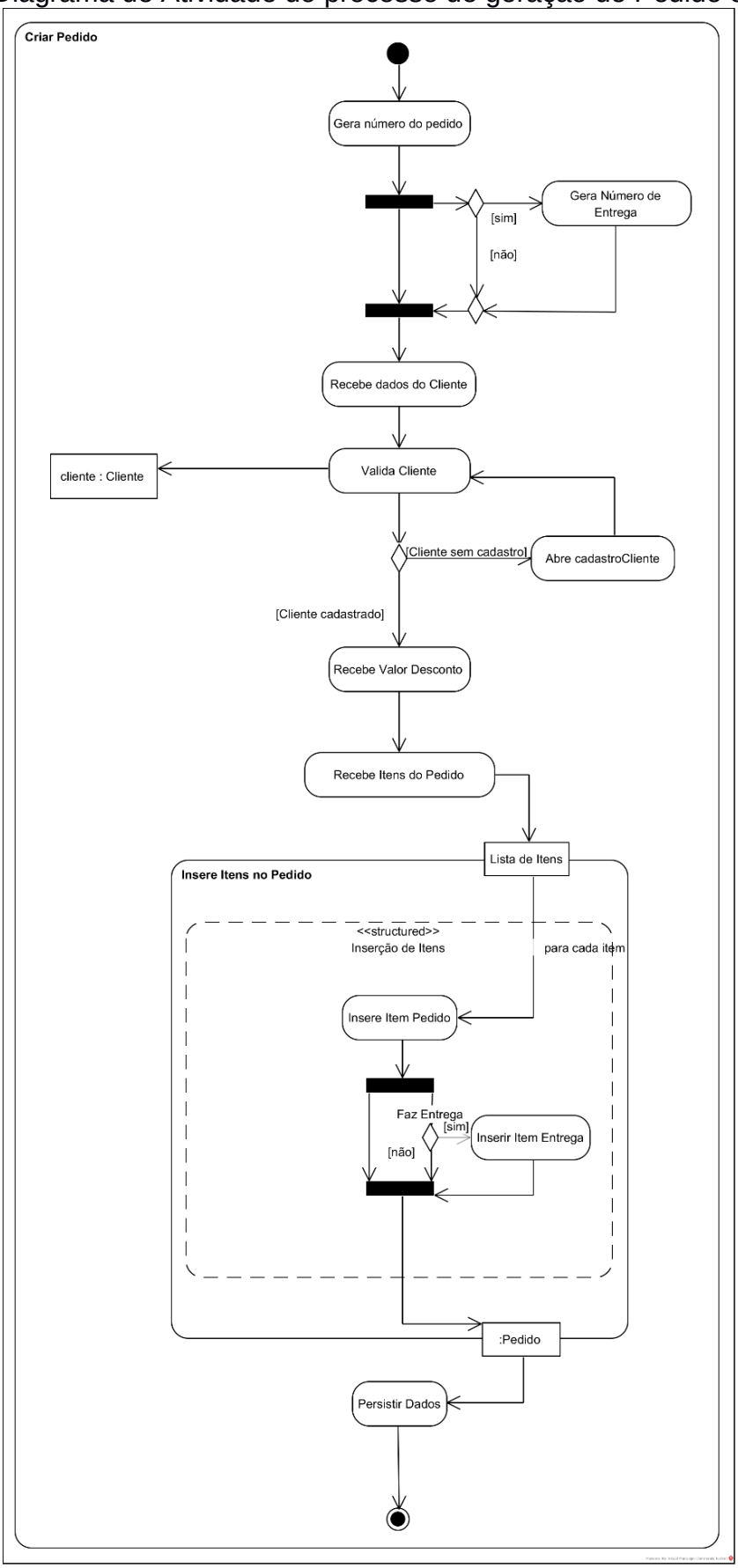


Fonte: https://en.wikipedia.org/wiki/JavaServer_Pages#/media/File:JSP_Model_2.svg

2.3.3 Realizar Pedido: Diagrama de Atividades

Considerando-se que a geração de Pedidos é a atividade mais dinâmica e expressiva da empresa, achou-se conveniente detalhar o processo. Para alcançar esse objetivo, utilizou-se um Diagrama de Atividades feito no Visual Paradigm Community.

Figura: 9 Diagrama de Atividade do processo de geração de Pedido e Entregas



Fonte: Autoria Própria

Para fins de simplicidade, optou-se por fazer o procedimento de inserção de Entrega juntamente com a geração do Pedido, visto que virtualmente todos os cenários em que serão geradas Entregas sempre haverá a geração de um Pedido de referência à própria Entrega. A inserção de um Pedido e sua Entrega são procedimentos relativamente simples, como se pode ver no Diagrama. Trata-se de um processo linear, basicamente, salvo pelas iterações necessárias à inclusão de cada item, individualmente, que farão a composição tanto da Entrega quanto do Pedido.

2.3.4 Cliente: Diagrama de Máquina de Estados

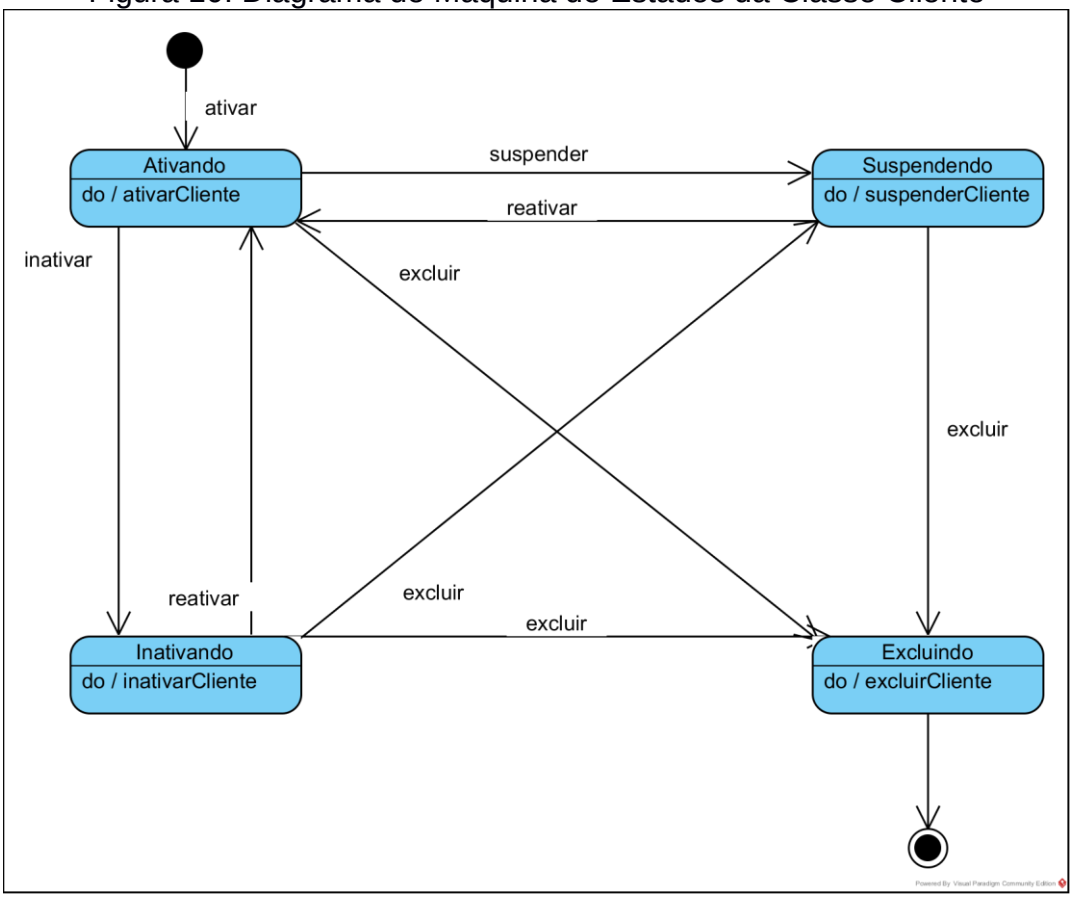
Ao se analisar os requisitos propostos pela empresa cliente, percebeu-se que não se deu uma devida atenção ao manejamento e manutenção de informações sobre os clientes da loja. Da forma como foi especificado, os Clientes são entidades passivas, estáticas: uma vez criados, não têm qualquer alteração de estado, permanecendo em mesmo estado quando da criação do objeto.

Todavia, de maneira a melhor representar os possíveis cenários de relacionamento entre a empresa e o cliente, poder-se-ia adicionar mais um campo para armazenar o estado do Cliente. Uma possibilidade de implementar isso seria estabelecendo uma lista de possíveis estados, dentre os quais um dos elementos faria a definição do estado do Cliente. Poderíamos ter, nesta lista, os seguintes estados:

- ativo – representaria a situação normal do cadastro do cliente, sendo o estado principal em que todo Cliente criado se encontraria;
- inativo – representaria um cliente que, por quaisquer razões (inadimplência, por exemplo) estaria impedido de fazer outros pedidos, podendo, contudo, realizar todas as outras ações normais que um cliente ativo realiza;
- suspenso – representaria um cliente que o sistema ignoraria, sem a possibilidade de realizar quaisquer atividades no sistema;
- excluído – representaria a situação em que o cadastro do cliente seria excluído do banco de dados.

Desta forma, com os estados do cliente listados acima, ficaria simples de se apresentar todo o ciclo de vida de um registro de cliente. Para isso, pode-se utilizar o Diagrama de Máquina de Estados, oferecido pela UML.

Figura 10: Diagrama de Máquina de Estados da Classe Cliente



Fonte: Autoria Própria

Embora tal implementação seja algo que agregaria mais possibilidades ao sistema, optou-se por seguir a orientação original definida pelos requisitos do cliente, sem incluir qualquer outro atributo não listado na descrição.

2.4 MODELAGEM DE DADOS

Outro passo importante para definição de todos os aspectos do sistema é a determinação de como os dados a serem utilizados pelo sistema serão estruturados. Isso impactará em todos os outros aspectos do desenvolvimento e implementação, visto que todo o desenvolvimento das outras partes do sistema precisa se conformar com o formato em que os dados, na camada mais baixa do sistema, estão organizados.

Tão importante é esse processo que, embora somente agora esteja sendo abordado diretamente, teve precedência sobre algumas das etapas já tratadas nesse

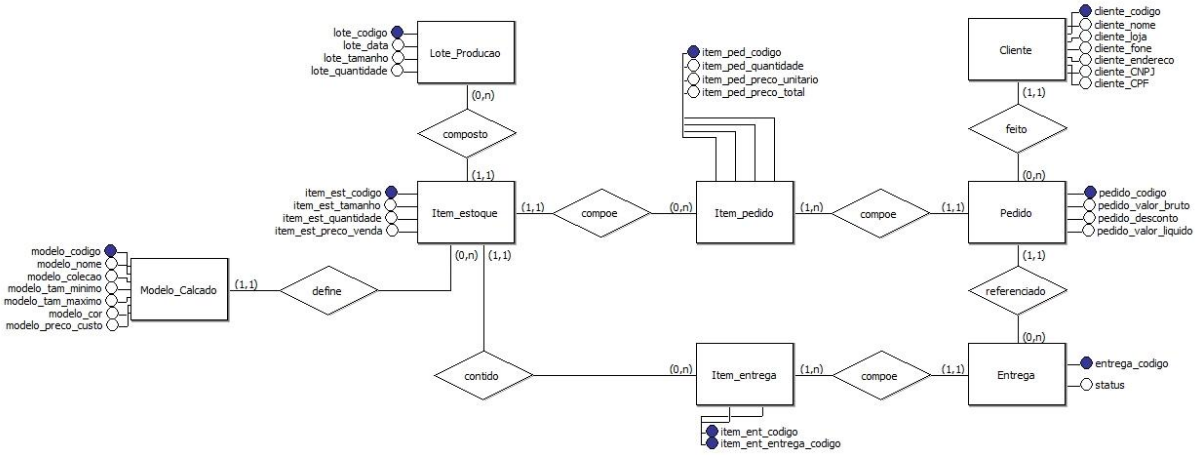
trabalho. Por exemplo, a estrutura das classes, representadas no Diagrama de Classes, foi determinada pela maneira que os dados foram modelados e estruturados.

Por meio da ferramenta BrModelo, foi feita a representação da estrutura que os dados do sistema devem assumir. Elaborou-se um Diagrama Entidade Relacionamento (DER) para se ter uma representação de alto nível da estrutura assumida pelos dados, sendo um modelo conceitual do banco de dados relacional a ser implementado. Por meio desse modelo criado, foi possível elaborar um modelo lógico, representando a forma com que os dados serão representados na camada mais baixa do sistema, mais especificamente no SGBD.

2.4.1 Modelo Conceitual

Por meio das especificações fornecidas pelos requisitos chegou-se a uma possível representação dos dados, elencando os atributos que permitiriam dar suporte a ações que garantam a implementação das regras de negócio da empresa. Definiu-se quais os dados adicionais que precisariam ser incluídos na representação, além dos listados nos requisitos funcionais, e que tipo de relacionamento cada entidade participante no sistema teria com as demais. Chegou-se ao seguinte Diagrama:

Figura 11: Modelo Entidade Relacionamento dos Dados do Sistema – Modelo Conceitual



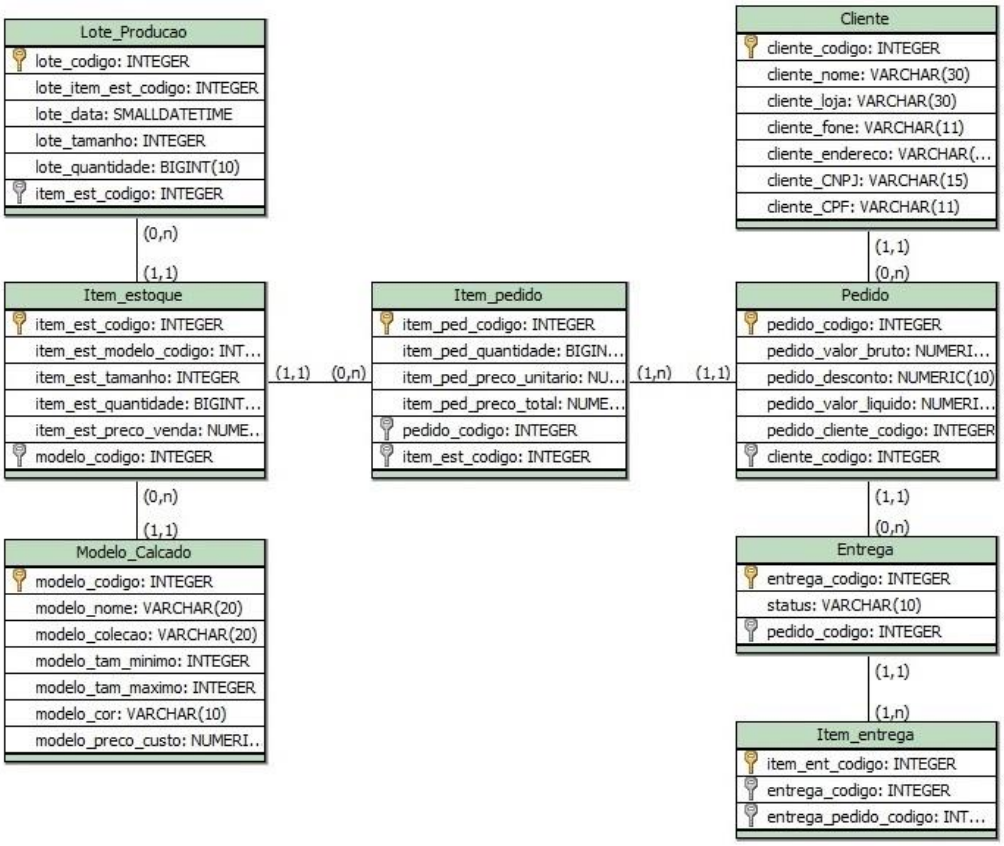
Fonte: Autoria Própria

2.4.2 Modelo Lógico

A partir do modelo conceitual elaborado, apresentado anteriormente, foi possível ter uma ideia de, em um banco de dados relacional, quais relações precisariam ser definidas para efetivar a implementação do que foi modelado. Além disso, com base nos atributos elencados no modelo conceitual, definiu-se os atributos de identificação de cada relação (Chaves Primárias) bem como os atributos identificadores de relacionamento entre as entidades (Chaves Estrangeiras.

Segue o modelo lógico elaborado seguindo as determinações do modelo conceitual:

Figura 12: Diagrama Entidade Relacionamento dos Dados do Sistema – Modelo Lógico



Fonte: Autoria Própria

2.4.3 Gerando o Banco de Dados

Havendo sido estabelecido o modelo lógico, prosseguiu-se para a criação efetiva

do banco de dados a ser utilizado pela aplicação web. Para tal, utilizou-se do PostgreSQL versão 9.6.1, sendo que, para facilitar a inspeção das estruturas criadas, utilizou-se também o PgAdmin III, o qual permite a visualização dos dados de forma tabular, bem como facilita algumas tarefas de ordem prática.

Foi criado um banco de dados e adicionadas as relações em conformidade com o que ficou definido nos modelos conceitual e lógico do DER. Com o intuito de facilitar a geração de dados e minimizar quaisquer erros de consistência, bem como violações de *constraints*, optou-se por utilizar sequências para geração de chaves primárias das tuplas em cada relação.

Adicionalmente, ainda nesse nível mais baixo, entendeu-se adequado implementar uma solução que desse conta da regra de negócio que trata da alteração das quantidades de itens em estoque em situações de (1) inserção de Lotes de produção de itens específicos, bem como (2) situações de saída de estoque, ocasionadas pelas gerações de Pedido e Entregas. Para isso, foram criados dois *procedures* e dois *triggers* a eles vinculados, os quais se encarregam de realizar a atualização do saldo de um produto específico no estoque. Dessa maneira, torna-se algo a menos para se preocupar em implementar em uma camada mais acima, reduzindo também o número de transações necessárias de serem invocadas pela aplicação.

Depois da definição das estruturas de dados, com o objetivo de providenciar dados que pudessem servir para testes no processo de desenvolvimento e implementação das camadas superiores da aplicação, foi gerado, por meio de *scripts* em Python dados para popular as tabelas. Tanto o código em Python gerador dos comandos SQL quanto os comandos em si serão omitidos aqui, para fins de brevidade.

Confira-se no Apêndice A o código com os comandos SQL utilizados para gerar o banco de dados com os detalhes descritos.

3. CRIANDO A APLICAÇÃO WEB

Havendo sido preparada toda a camada inferior da aplicação, prosseguiu-se com o desenvolvimento da aplicação em si. Foram escritas os JavaBeans para modelagem dos dados a serem recuperados do banco de dados, bem como as classes de Acesso Direto aos Dados (DAO Classes) para cada classe de modelagem. Algumas outras classes auxiliares foram construídas para simplificar determinadas tarefas e deixar cada classe responsável apenas pelas tarefas específicas de sua natureza.

Concluída a camada de Modelo, foram escritos os Servlets de Controle de cada conjunto de operações requerido pelas especificações e requisitos. Optou-se, nesse ponto do desenvolvimento, para se conformar apenas com as exigências expressas no documento que explica as tarefas motivadoras desse trabalho. Mais especificamente, incluiu-se nas classes de controle atributos constantes com nome de usuário, senha e endereço do banco de dados, de maneira que não se precisasse construir uma interface para execução de login, liberando tempo e recursos para enfocar nas tarefas especificadas.

Foram executados testes utilizando um simples programa de linha de comando, de forma a garantir o bom funcionamento das partes elaboradas até então. Diante dos resultados positivos dos testes, e havendo sido encerrado o desenvolvimento da camada de controle, deu-se início ao desenvolvimento da camada de visão.

Para a geração da interface gráfica de usuário (GUI), optou-se por utilizar JavaServer Pages – JSP – que, embora sejam mais trabalhosas que os frameworks oferecidos hodiernamente, permitiram uma maior compreensão dos detalhes nos procedimentos ocorrentes em segundo plano. Uma solução utilizando frameworks, pela própria transparência oferecida por estes, não permitiria a percepção exata do fluxo de interações entre os diversos elementos. Assim, o desenvolvimento certamente ficou mais custoso, porém o processo foi mais eficiente no sentido de permitir maior compreensão do processo de comunicação da GUI com as classes de controle.

Não se apresentará, de maneira a garantir a brevidade do trabalho, toda a coleção código nos arquivos JSP. Em vez disso, apresentar-se-á em detalhes apenas as interfaces implementadas como solução para dois Casos de Uso específicos, a saber: a inserção de Clientes e a inserção de Pedidos e Entregas.

3.1 PRIMEIRO PASSO: UMA ROTINA DE CADASTRO CRUD

Conforme solicitado, iniciou-se o processo de elaboração da camada de visão escolhendo uma das classes escritas até então e criando uma rotina de cadastro contendo as operações CRUD. Para garantir maior concisão e agilidade, escolheu-se a classe com menor número de dependência de outros componentes do sistema, a classe Cliente (ver Apêndice B). Trata-se de uma classe simples, com seus atributos e seus setters/getters, nada mais.

Em conjunto com ela, utilizamos uma classe para implementar o acesso ao banco de dados e dar conta das transações, chamando-a de `ClienteDAO` (ver Apêndice C) e uma classe `ControleCliente` (ver Apêndice D), que herda de `javax.servlet.http.HttpServlet`. Juntas, essas classes realizam o acesso ao banco de dados para recuperar informações e prepará-las para visualização na camada de visão, assim como recebem dados da GUI, tratam-nos e realizam a persistência no banco de dados.

Para as camadas de visão, utilizamos quatro JSPs principais (`inserirCliente.jsp`, `listarClientes.jsp`, `detalharCliente.jsp` e `alterarCliente.jsp`) e uma JSP alternativa chamada (`buscarCliente.jsp`). O código fonte de `inserirCliente.jsp` e `listarClientes.jsp` consta do Apêndice E.

Por meio de `inserirCliente.jsp`, o usuário pode informar e pré-validar dados a serem usados para geração de um novo cliente a ser inserido no banco de dados. Enquanto que `listarClientes.jsp` mostra uma lista dos clientes cadastrados com as opções de excluir (diretamente implementada na lista, sem qualquer outra interface intermediária), alterar (direcionando para `alterarCliente.jsp`) e detalhar, que abre uma tela contendo os detalhes do cliente (`detalharCliente.jsp`). A página `buscarCliente.jsp` pede um código de cliente e, encontrado o cliente no banco de dados, redireciona para a página `detalharCliente.jsp` com os dados do cliente buscado, tendo esta, também, as opções de alterar e excluir.

Completadas as etapas desse problema mais simples, segue-se para outro mais intrincado, o qual envolve não apenas uma, mas múltiplas classes.

3.2 AVANÇANDO: UMA ROTINA DE MOVIMENTO DE ESTOQUE

De maneira a implementar uma rotina que permita a movimentação de estoques, torna-se necessário abordar dois vetores opostos de fluxo de produtos: (1) o fluxo de entrada de produtos no estoque, por meio do cadastro de novos Lotes de Produção; e (2) o fluxo de saída de produtos, que se dá pela inclusão de Pedidos/Entregas no sistema. A estrutura base das operações é a mesma da utilizada no cadastro de Clientes, porém torna-se mais complexa por (1) essencialmente conter duas frentes de gerenciamento de dados (Pedidos e Lotes) e (2) por girarem em torno de classes formadas por composição e/ou agregação. Perceba-se que `Pedido`, além de referenciar `Cliente`, é composto por Itens de `Pedido` e `Entrega`, os quais também são formados por composição/agregação, visto que `ItemPedido` referencia `ItemEstoque`, e `Entrega` é formada por `ItemEntrega`. `ItemEntrega`, adicionalmente, referencia `ItemEstoque`, que, por sua vez, referencia `ModeloCalcado`. Não obstante, em grande parte, utilizou-se, para a resolução do problema, a mesma perspectiva utilizada no cadastro de Clientes.

Assim, das duas frentes de atuação na rotina de movimento de estoque a que tem menos diferenças em relação à rotina de cadastro de Clientes é a de gerenciamento de Lotes de Produção, tendo a mesma estrutura lógica do processo de inserção de Clientes. Tal semelhança se reflete na forma como as Classes de representação, controle de fluxo de dados e apresentação foram desenvolvidas.

Destarte, utilizou-se, para a camada de modelo, uma classe `LoteProducao` (ver Apêndice F), a qual é faz referência à classe `ItemEstoque`, a qual, por sua vez, aponta para a classe `ModeloCalcado`. Já de maneira a fazer a interface com o banco de dados, lançou-se mão da classse `LoteProducaoDAO` (ver Apêndice G), e de uma classe `ControleLote` (ver Apêndice H), o qual é uma subclasse de `javax.servlet.http.HttpServlet`, a qual servirá como elemento controlador no modelo MVC. Adicionalmente, para dar conta da interface com o usuário, utilizou-se um conjunto de JavaServer Pages (`inserirLote.jsp`, `listarLotes.jsp`, `detalharLote.jsp` e `buscarLote.jsp`, listadas no Apêndice I) com o fim de prover uma GUI de interação.

Como antes exposto, a rotina de realização de pedidos envolve todas as outras classes utilizadas no sistema, exceto `LoteProducao`, bem como todas as suas

classes de acesso ao banco de dados (classes DAO), o que a torna mais detalhada que as rotinas de gerenciamento de Clientes e Lotes de Produção. Adicionalmente, incluir pedidos implica tem uma classe Pedido, a qual é formada por um número variável de ItemPedido, tornando o tráfgo de dados entre a interface e o Servlet de Controle um pouco mais trabalhoso.

Nesses termos, a abordagem para seu desenvolvimento foi a mesma utilizada para a inserção de Clientes: utilizou-se os JavaBeans para modelar os dados recuperados por meio das classes DAO, as quais concentraram todo o trabalho de construção das classes compostas a partir dos dados obtidos, repassando-os para os Servlets de Controle. Toda a coordenação de recebimento de dados da GUI, processamento e persistência, bem como elaboração de respostas a serem devolvidas à interface de usuário ficou centralizada nessa camada de controle providenciada por meio dos Servlets.

O mesmo conjunto de páginas para inclusão, alteração, detalhamento, listagem e busca (as quais realizam também a operação de exclusão) foi utilizado, tanto na rotina de controle de pedidos e entregas, quanto na de controle de lotes de produção.

Para fins de brevidade, não se listará o código fonte das classes e JSPs utilizadas na rotina de gerenciamento de Pedidos e Entregas. Todavia, conforme mencionado na introdução deste trabalho, todos os arquivos de código fonte podem ser vistos em sua completude na página do repositório em que foi salvo o projeto, a qual pode ser acessada por meio dos seguintes hyperlinks: https://goo.gl/EF51gJ ou http://bit.ly/portfolio3_saint.

.CONCLUSÃO

Após a realização do trabalho de planejamento e desenvolvimento do que foi solicitado, foi possível ganhar uma compreensão maior de todo o processo de desenvolvimento de software, em especial no que se refere a um ambiente Web, utilizando tecnologia Java. A exposição aos padrões de modelagem da UML, bem como aos conceitos da arquitetura MVC, garantiu uma visão bottom up holística do sistema, ao mesmo tempo em que tornou explícitos os detalhes de cada um dos módulos (modelo, visão e controle), suas características, seus papéis a serem desempenhados no sistema, assim como o escopo e extensão de sua atuação. Além disso, ganhou-se uma maior compreensão das interações entre as camadas da dita arquitetura, agindo em colaboração para fornecer uma solução para o problema proposto.

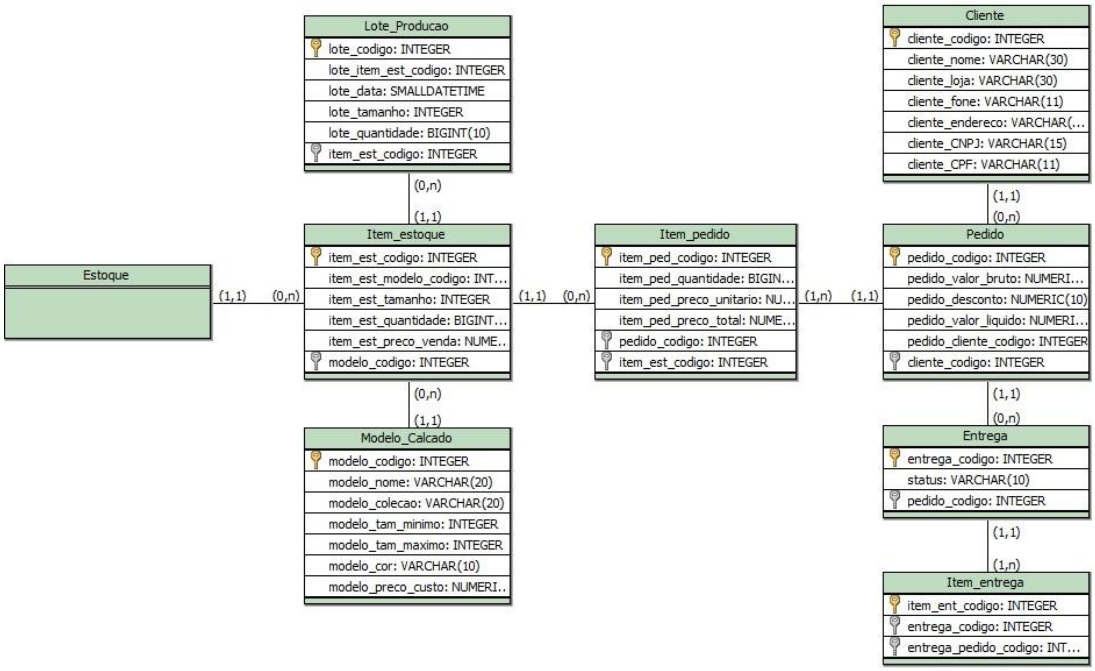
Após ser elaborada essa proposta de solução para o problema, foi possível ganhar uma visão mais abrangente do sistema de produção da fábrica. Com isso, algumas possibilidades de melhorias poderiam ser feitas, de maneira a otimizar a forma como a empresa trabalha, ou, pelo menos, aprimorar a forma como seus dados e informações deveriam ser representados – além de expandir o conjunto de dados a serem armazenados.

A primeira coisa a ser sugerida seria a inclusão de um marcador em cada uma das entidades informadas para elaboração do sistema. Em vez de ter a opção de excluir um determinado Cliente ou Item de Estoque do sistema, seria interessante incluir um atributo que registrasse o estado atual daquele registro específico (ativo ou inativo), bem como um campo para anotações sobre quaisquer mudanças de estado, de maneira a dar maiores detalhes do estado atual do item registrado (digamos, explicando que o item de estoque foi descontinuado da produção, ou que foi criado um registro alternativo por uma ou outra razão). Seria, enfim, algo semelhante ao que foi sugerido anteriormente para a classe Cliente, quando da criação do Diagrama de Máquina de Estados.

Em se tratando de escalabilidade, percebe-se que a fábrica está em expansão e que o sistema deverá dar suporte a esse crescimento. Levando em conta que cada uma das fábricas filiais deverá ter seu próprio estoque, seria de especial interesse que se mantivesse um controle de diferentes estoques existentes, bem como dos itens lá disponíveis. Para isso, iniciando-se do nível mais baixo, poder-se-ia incluir a entidade

Estoque no Diagrama Entidade Relacionamento, vinculando-o à entidade ItemEstoque. Igualmente, dever-se-ia refletir essa alteração no diagrama de classes, incluindo o desenvolvimento do código fonte da classe, bem como sua classe auxiliar DAO de acesso aos dados e sua interface de gerenciamento.

Figura 13: Diagrama Entidade Relacionamento – Modelo Lógico Incluindo Estoque



Fonte: Autoria Própria

Outra possibilidade de expansão, no lado do frontend, seria, em vez de realizar as operações navegando entre diversas páginas dinamicamente geradas, utilizar-se de alguma tecnologia de alteração de página de forma assíncrona (como AJAX, por exemplo) ou algum framework que trate dessa questão de maneira transparente ao usuário. Isso reduziria o tráfego de dados na rede e garantiria uma experiência de usabilidade mais fluida, sem quebras por envio de informações e carregamento de novas páginas.

Em suma, a aplicação web até o momento desenvolvida, como todo e qualquer sistema computacional em sua primeira versão, funciona adequadamente, segundo as especificações e requisitos fornecidos, porém tem muito espaço para refinamento, ampliação, crescimento e aperfeiçoamento, os quais serão desenvolvidos, implementados e ajustados à medida que o feedback do usuário final, durante a fase de testes, chega à equipe de desenvolvedores. Não obstante, o processo de desenvolvimento foi de grande valia e satisfação, alcançando o propósito de aplicação

dos conceitos aprendidos, bem como obtenção de experiência que apenas uma abordagem hands on poderia garantir.

REFERÊNCIAS

CARDOSO, Virgínia Mara. **Ferramentas para Sistemas WEB**. Londrina: Editora e Distribuidora Educacional S. A., 2017.

CEZAR, Douglas Fujita de Oliveira. **Banco de Dados II**. Londrina: Editora e Distribuidora Educacional S. A., 2017.

FABRIS, Polyanna Pacheco Gomes; CATARINO, Iolanda Cláudia Sanches. **Análise Orientada a Objetos II**. Londrina: Editora e Distribuidora Educacional S. A., 2017.

FABRIS, Polyanna Pacheco Gomes; PERINI, Luis Cláudio. **Processos de Software**. Londrina: Editora e Distribuidora Educacional S. A., 2014.

MATOS, Maria Clotilde Pires, **Metodologia Científica**. Londrina: Editora e Distribuidora Educacional S. A., 2014.

MOZER, Merris. **Sistemas WEB**. Londrina: Editora e Distribuidora Educacional S. A., 2014.

NISHIMURA, Roberto Yukio. **Banco de Dados I**. São Paulo, Pearson Prentice Hall, 2009.

TANAKA, Simone Sawasaki. **Análise de sistemas I**. São Paulo, Pearson Prentice Hall, 2009.

UML DIAGRAMS. **UML 2.5 Diagrams Overview**. Disponível em: . Acessado em 17 de setembro de 2017, às 08:43.

VENTURA, Plínio. **Caso de Uso – Include, Extend e Generalização.** Disponível em: . Acessado em 16 de setembro de 2017, às 16:01.

APÊNDICES

APÊNDICE A

Código Fonte para Geração do Banco de Dados

```
drop database liz;
create database liz;

create table modelo_calcado (modelo_codigo int not null constraint modelo_pk primary key,
              modelo_nome varchar(25) not null,
              modelo_colecao varchar(25) not null,
              modelo_tam_min int not null,
              modelo_tam_max int not null,
              modelo_cor varchar(15) not null,
              modelo_preco_custo numeric not null);

create table item_estoque(item_estoque_codigo int not null constraint item_estoque_pk primary
key,
             item_estoque_tamanho int not null,
             item_estoque_quantidade int not null,
             item_estoque_preco_venda numeric not null,
             item_estoque_modelo_codigo int references modelo_calcado(modelo_codigo) on
update cascade on delete cascade,
             constraint item_estoque_unq unique(item_estoque_tamanho,
item_estoque_modelo_codigo));

create table lote_producao(lote_codigo int not null constraint lote_producao_pk primary key,
              lote_data date not null,
              lote_tamanho int not null,
              lote_quantidade int not null,
              lote_item_est_codigo int not null references item_estoque(item_estoque_codigo)
on update cascade);

create table cliente(cliente_codigo int not null constraint cliente_pk primary key,
           cliente_nome varchar(35) not null,
           cliente_loja varchar(35),
           cliente_fone varchar(11),
           cliente_endereco varchar(100) not null,
           cliente_cpf varchar(11) not null unique,
           cliente_cnpj varchar(15));

create table pedido (pedido_codigo int not null constraint pedido_pk primary key,
           pedido_desconto numeric default 0,
           pedido_cliente_codigo int not null references cliente(cliente_codigo) on update
cascade);

create table item_pedido(item_ped_codigo int not null,
            item_ped_quant int not null,
            item_ped_pedido_codigo int not null references pedido(pedido_codigo) on update
cascade on delete cascade,
            item_ped_it_est_codigo int not null references item_estoque(item_estoque_codigo)
on update cascade,
            constraint item_pedido_pk primary key (item_ped_pedido_codigo,
item_ped_it_est_codigo));

create table entrega (entrega_codigo int not null constraint entrega_pk primary key,
           entrega_status varchar(10) not null,
```

```
            entrega_pedido_codigo int not null references pedido(pedido_codigo) on update
cascade on delete cascade);

create table item_entrega(item_ent_codigo int not null,
              item_ent_entrega_codigo int not null references entrega(entrega_codigo) on
update cascade on delete cascade,
              item_ent_it_est_codigo int not null references item_estoque(item_estoque_codigo)
on update cascade,
              constraint item_entrada_pk primary key(item_ent_entrega_codigo,
item_ent_it_est_codigo));

create or replace function fn_registrar_entrada_estoque()
  returns trigger as'
  begin
  update item_estoque set item_estoque_quantidade = item_estoque.item_estoque_quantidade +
new.lote_quantidade
  where item_estoque.item_estoque_codigo = new.lote_item_est_codigo;
  return new;
  end;
  '
  language plpgsql;

  create trigger tg_registrar_entrada_estoque
  after insert on lote_producao
  for each row execute procedure fn_registrar_entrada_estoque();

create or replace function fn_registrar_saida_estoque()
  returns trigger as'
  begin
  update item_estoque set item_estoque_quantidade = item_estoque.item_estoque_quantidade -
new.item_ped_quant
  where item_estoque.item_estoque_codigo = new.item_ped_it_est_codigo;
  return new;
  end;
  '
  language plpgsql;

  create trigger tg_registrar_saida_estoque
  after insert on item_pedido
  for each row execute procedure fn_registrar_saida_estoque();

create sequence seq_modelo_calcado_codigo increment 1 minvalue 1 maxvalue 9223372036854775807
start 21 cache 1;
create sequence seq_lote_producao_codigo increment 1 minvalue 1 maxvalue 9223372036854775807
start 442 cache 1;
create sequence seq_item_estoque_codigo increment 1 minvalue 1 maxvalue 9223372036854775807
start 123 cache 1;
create sequence seq_cliente_codigo increment 1 minvalue 1 maxvalue 9223372036854775807 start
251 cache 1;
create sequence seq_pedido_codigo increment 1 minvalue 1 maxvalue 9223372036854775807 start
101 cache 1;
create sequence seq_entrega_codigo increment 1 minvalue 1 maxvalue 9223372036854775807 start
101 cache 1;
```

APÊNDICE B

Código fonte da Classe Cliente

```
package br.com.flordeliz.modelo;
import br.com.flordeliz.utils.CNP;
import br.com.flordeliz.utils.Utils;

public class Cliente {
  private int codigo;
  private String nome;
  private String loja;
  private String fone;
  private String endereco;
  private String cpf;
  private String cnpj;

  public Cliente(int codigo, String nome, String loja, String fone, String endereco, String
cpf, String cnpj) throws InsercaoException{
    this.setCodigo(codigo);
    this.setNome(nome);
    this.setLoja(loja);
    this.setFone(fone);
    this.setEndereco(endereco);
    this.setCpf(cpf);
    this.setCnpj(cnpj);
  }

  public void setCodigo(int codigo) throws InsercaoException {
    if (codigo < 0){
      throw new InsercaoException("Codigo invalido: valor menor que zero");
    } else {
      this.codigo = codigo;
    }
  }

  public void setNome(String nome) throws InsercaoException {
    if (nome.length() > 35){
      throw new InsercaoException("Nome inválido: número de caracteres maior que 35");
    } else {
      this.nome = nome;
    }
  }

  public void setLoja(String loja) throws InsercaoException {
    loja = Utils.checaNull(loja);
    if (loja.length() > 35){
      throw new InsercaoException("Nome da loja inválido: número de caracteres maior que 35");
    } else {
      this.loja = loja;
    }
  }
  public void setFone(String fone) throws InsercaoException {
    fone = Utils.checaNull(fone);

    if (fone.length() > 11){
```

```
      throw new InsercaoException("Numero de telefone inválido: mais que 15 caracteres");
    } else {
      this.fone = fone;
    }
  }

  public void setEndereco(String endereco) throws InsercaoException {
    if (endereco.length() > 100){
      throw new InsercaoException("Endereco inválido: mais que 100 caracteres");
    } else {
      this.endereco = endereco;
    }
  }

  public void setCpf(String cpf)  throws InsercaoException {
    if (! CNP.isValidCPF(cpf)){
      throw new InsercaoException("CPF invalido");
    } else {
      this.cpf = cpf;
    }
  }

  public void setCnpj(String cnpj)  throws InsercaoException {
    cnpj = Utils.checaNull(cnpj);

    if (cnpj.equals("")){
      this.cnpj = cnpj;
    }else if (! CNP.isValidCNPJ(cnpj)){
      throw new InsercaoException("CNPJ invalido");
    } else {
      this.cnpj = cnpj;
    }
  }
}
```

APÊNDICE C

Código fonte da Classe ClienteDAO

```
package br.com.flordeliz.dao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import br.com.flordeliz.modelo.Cliente;
import br.com.flordeliz.modelo.EntidadeNulaException;
import br.com.flordeliz.modelo.InsercaoException;
import br.com.flordeliz.utils.Utils;

public class ClienteDAO extends DAO{

  public int inserir(String usuario, String senha, String endereco, Cliente cliente) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "INSERT INTO cliente (cliente_codigo, cliente_nome, cliente_loja,
cliente_fone, cliente_endereco, cliente_cpf, cliente_cnpj)
VALUES(NEXTVAL('seq_cliente_codigo'),?,?,?,?,?,?)";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      Utils.checaNull(ps,"PreparedStatement",
Thread.currentThread().getStackTrace()[2].getLineNumber());

      ps.setString(1, cliente.getNome());
      ps.setString(2, cliente.getLoja());
      ps.setString(3, cliente.getFone());
      ps.setString(4, cliente.getEndereco());
      ps.setString(5, cliente.getCpf());
      ps.setString(6, cliente.getCnpj());
      ps.execute();
      conexao.close();
      return this.SUCESSO;
    } catch (SQLException e) {
      System.err.println("ERRO: falha ao inserir cliente no banco" + "\r\n " +
e.getMessage());
      return this.ERRO_SQL;
    }
  }

  public int alterar(String usuario, String senha, String endereco, Cliente cliente) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "UPDATE cliente SET cliente_nome = ?, cliente_loja = ?, cliente_fone
= ?, cliente_endereco = ?, cliente_cpf = ?, cliente_cnpj = ? WHERE cliente_codigo =?";
```

```
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setString(1, cliente.getNome());
      ps.setString(2, cliente.getLoja());
      ps.setString(3, cliente.getFone());
      ps.setString(4, cliente.getEndereco());
      ps.setString(5, cliente.getCpf());
      ps.setString(6, cliente.getCnpj());
      ps.setInt(7, cliente.getCodigo());
      ps.executeUpdate();
      conexao.close();

      return this.SUCESSO;
    } catch (org.postgresql.util.PSQLException e){
      System.err.println("ERRO: Falha ao alterar cliente de codigo " + cliente.getCodigo() +
". Violação de valor único\r\n " + e.getMessage());
      return this.ERRO_SQL;
    } catch (SQLException e) {
      System.err.println("ERRO: Falha ao alterar cliente de codigo " + cliente.getCodigo() +
"\r\n " + e.getMessage());
      return this.ERRO_SQL;
    }
  }

  public int excluir(String usuario, String senha, String endereco, int idExclusao) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());
    int resultadoOperacao = 0;
    try {
      String comandoSQL = "SELECT cliente_codigo FROM cliente WHERE cliente_codigo = ?";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setInt(1, idExclusao);
      ResultSet rs = ps.executeQuery();

      if (rs.next()){
        comandoSQL = "DELETE FROM cliente WHERE cliente_codigo =?";
        /*PreparedStatement*/ ps = conexao.prepareStatement(comandoSQL);
        ps.setInt(1, idExclusao);
        ps.executeUpdate();
      } else {
        System.err.println("ALERTA: Cliente com código " + idExclusao + " não
existente.\r\nExclusão não realizada");
        resultadoOperacao = ClienteDAO.ERRO_CODIGO;
      }

      conexao.close();
      return resultadoOperacao = ClienteDAO.SUCESSO;
    } catch (SQLException e) {
      System.err.println("Não foi possível excluir o Cliente" + "\nErro : " + e.getMessage());
      resultadoOperacao = ClienteDAO.ERRO_SQL;
      return resultadoOperacao;
    }
  }

  public List<Cliente> consultar(String usuario, String senha, String endereco) {
    List<Cliente> listaCliente = new ArrayList<Cliente>();
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
```

```
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "SELECT cliente_codigo, cliente_nome, cliente_loja, cliente_fone,
cliente_endereco, cliente_cpf, cliente_cnpj FROM cliente ORDER BY cliente_codigo";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ResultSet rs = ps.executeQuery();
      while (rs.next()) {
        Cliente cliente = null;
        try{
          cliente = new Cliente(rs.getInt("cliente_codigo"),
                        rs.getString("cliente_nome"),
                        rs.getString("cliente_loja"),
                        rs.getString("cliente_fone"),
                        rs.getString("cliente_endereco"),
                        rs.getString("cliente_cpf"),
                        rs.getString("cliente_cnpj"));
        }catch(InsercaoException e){
          System.err.println("ERRO: Falha na consulta por clientes:" + "\r\n" +
e.getMessage());
        }
        listaCliente.add(cliente);
        Utils.checaNull(cliente, "Cliente",
Thread.currentThread().getStackTrace()[2].getLineNumber());
      }

      conexao.close();
    } catch (SQLException e) {
       System.err.println("Consulta inválida!" + "\nErro : " + e.getMessage());
    }
    return listaCliente;
  }

  public Cliente buscar(String usuario, String senha, String endereco, int idBusca) throws
EntidadeNulaException {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());
    Cliente cliente = null;

    try {
      String comandoSQL = "SELECT cliente_codigo, cliente_nome, cliente_loja, cliente_fone,
cliente_endereco, cliente_cpf, cliente_cnpj FROM cliente WHERE cliente_codigo = ?";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setInt(1, idBusca);
      ResultSet rs = ps.executeQuery();
      while (rs.next()) {
        try{
          cliente = new Cliente(rs.getInt("cliente_codigo"),
                        rs.getString("cliente_nome"),
                        rs.getString("cliente_loja"),
                        rs.getString("cliente_fone"),
                        rs.getString("cliente_endereco"),
                        rs.getString("cliente_cpf"),
                        rs.getString("cliente_cnpj"));
        }catch(InsercaoException e){
```

```
          System.err.println("ERRO: Falha na busca por cliente:" + "\r\n" + e.getMessage());
        }
        Utils.checaNull(cliente, "Cliente",
Thread.currentThread().getStackTrace()[2].getLineNumber());
      }

      conexao.close();
    } catch (SQLException e) {
       System.err.println("Consulta inválida!" + "\nErro : " + e.getMessage());
    }
    Utils.checaNull(cliente, "Cliente",
Thread.currentThread().getStackTrace()[2].getLineNumber());
    if (cliente==null){
      throw new EntidadeNulaException("Cliente não encontrado no banco de dados");
    }
    return cliente;
  }
}
```

APÊNDICE D

Código Fonte da Classe ControleCliente

```
package br.com.flordeliz.controlador;

import br.com.flordeliz.dao.ClienteDAO;
import br.com.flordeliz.modelo.Cliente;
import br.com.flordeliz.modelo.EntidadeNulaException;
import br.com.flordeliz.modelo.InsercaoException;
import java.io.IOException;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ControleCliente extends HttpServlet implements InterfaceControle{
  public static final int LISTAR_CLIENTES = 4;
  public static final int DETALHAR_CLIENTE = 5;
  public static final int ATUALIZAR_CLIENTE = 6;
  public static final int INSERIR_CLIENTE = 7;
  public static final int BUSCAR_CLIENTE = 8;

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    String acao = request.getParameter("acao");
    int resultadoOperacao = 20;
    if (acao==null) acao = "";
    if (acao.equals("incluir")){
      resultadoOperacao = this.INSERIR_CLIENTE;

    } else if (acao.equals("inserir")){
      resultadoOperacao = inserirCliente(request, response);

    } else if (acao.equals("alterar")){
      resultadoOperacao = alterarCliente(request, response);

    }else if (acao.equals("atualizar")){
      try {
        Cliente cliente = buscarCliente(request);
        request.setAttribute("cliente", cliente);
        resultadoOperacao = this.ATUALIZAR_CLIENTE;
      } catch (EntidadeNulaException ex) {
        resultadoOperacao = ClienteDAO.ERRO_CODIGO;
      }

    } else if (acao.equals("excluir")){
      resultadoOperacao = excluirCliente(request, response);

    } else if (acao.equals("consultar")){
      List<Cliente> lista = consultarClientes();
      request.setAttribute("listaClientes", lista);
      resultadoOperacao = this.LISTAR_CLIENTES;
```

```
    } else if (acao.equals("buscar")){
      resultadoOperacao = this.BUSCAR_CLIENTE;

    } else if (acao.equals("detalhar")){
      try {
        Cliente cliente = buscarCliente(request);
        request.setAttribute("cliente", cliente);
        resultadoOperacao = this.DETALHAR_CLIENTE;
      } catch (EntidadeNulaException ex) {
        resultadoOperacao = ClienteDAO.ERRO_CODIGO;
      }
    }

    RequestDispatcher rd;
    request.setAttribute("title","Cliente - " + acao);
    switch (resultadoOperacao){
      case INSERIR_CLIENTE:
        rd = request.getRequestDispatcher("inserirCliente.jsp");
        break;
      case ClienteDAO.SUCESSO:
        String processadas = " processadas ";
        if (acao.equals("inserir")){
          processadas = " inseridas ";
        } else if (acao.equals("excluir")){
          processadas = " excluídas ";
        } else {
          processadas = " alteradas ";
        }
        request.setAttribute("conteudo","<h1>Informações sobre o cliente" + processadas + "com
sucesso</h1>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case ClienteDAO.ERRO_SQL:
        request.setAttribute("conteudo","<h1>Falha ao Acessar o Banco</h1>"
                  + "<p>Erro durante o acesso (leitura ou escrita) ao banco de dados. Comando
SQL malformado. Procure o Webmaster</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case ClienteDAO.ERRO_INSERCAO:
        request.setAttribute("conteudo","<h1>Falha na Operação</h1>"
                        + "<p>Os dados inseridos eram inválidos. Por favor verifcar e tentar
outra vez</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case ClienteDAO.ERRO_CODIGO:
        request.setAttribute("conteudo","<h1>Falha na Operação</h1>"
                        + "<p>O Código informado nao corresponde a um código de cliente
presente no banco de dados</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case LISTAR_CLIENTES:
        rd = request.getRequestDispatcher("listarClientes.jsp");
        break;
      case BUSCAR_CLIENTE:
        rd = request.getRequestDispatcher("buscarCliente.jsp");
        break;
```

```
        case DETALHAR_CLIENTE:
          rd = request.getRequestDispatcher("detalharCliente.jsp");
          break;
        case ATUALIZAR_CLIENTE:
          rd = request.getRequestDispatcher("atualizarCliente.jsp");
          break;
        default:
          rd = request.getRequestDispatcher("buscarCliente.jsp");
      }

      rd.forward(request, response);
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
      processRequest(request, response);
    }

    @Override
    public String getServletInfo() {
      return "Short description";
    }// </editor-fold>

    protected int inserirCliente(HttpServletRequest request, HttpServletResponse response){
      int resultadoOperacao = 0;
      try {
        ClienteDAO clienteDAO = new ClienteDAO();
        Cliente cliente = new Cliente(0,
                        request.getParameter("cliente_nome"),
                        request.getParameter("cliente_loja"),
                        request.getParameter("cliente_fone"),
                        request.getParameter("cliente_endereco"),
                        request.getParameter("cliente_cpf"),
                        request.getParameter("cliente_cnpj"));
        resultadoOperacao = clienteDAO.inserir(this.usuario, this.senha, this.endereco,
cliente);
        return resultadoOperacao;
      } catch (InsercaoException ex) {
        //tratamento de falha na insercao
        return ClienteDAO.ERRO_INSERCAO;
      }
    }

    protected int alterarCliente(HttpServletRequest request, HttpServletResponse response){
      try {
        ClienteDAO clienteDAO = new ClienteDAO();
        Cliente cliente = new Cliente(Integer.parseInt(request.getParameter("cliente_codigo")),
                        request.getParameter("cliente_nome"),
                        request.getParameter("cliente_loja"),
                        request.getParameter("cliente_fone"),
                        request.getParameter("cliente_endereco"),
```

```
                    request.getParameter("cliente_cpf"),
                    request.getParameter("cliente_cnpj"));
      int resultadoOperacao = clienteDAO.alterar(this.usuario, this.senha, this.endereco,
cliente);
      return resultadoOperacao;
    } catch (InsercaoException ex) {
      //tratamento de falha na insercao
      return ClienteDAO.ERRO_INSERCAO;
    }
  }

  protected int excluirCliente(HttpServletRequest request, HttpServletResponse response){
    ClienteDAO clienteDAO = new ClienteDAO();
    int clienteCodigo = Integer.parseInt(request.getParameter("cliente_codigo"));
    int resultadoOperacao = clienteDAO.excluir(this.usuario, this.senha, this.endereco,
clienteCodigo);
    return resultadoOperacao;
  }

  protected List<Cliente> consultarClientes(){
    ClienteDAO clienteDAO = new ClienteDAO();
    return clienteDAO.consultar(this.usuario, this.senha, this.endereco);
  }

  protected Cliente buscarCliente(HttpServletRequest request) throws EntidadeNulaException{
    int clienteCodigo = Integer.parseInt(request.getParameter("cliente_codigo"));
    ClienteDAO clienteDAO = new ClienteDAO();
    return clienteDAO.buscar(this.usuario, this.senha, this.endereco, clienteCodigo);
  }
}
```

APÊNDICE E

Código Fonte das JavaServer Pages de Manipulação de Clientes

A. inserirCliente.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Cliente - Inserir</title>

    <jsp:include page="/cabecalho.jsp"/>

    <h1>Novo Cliente</h1>
    <form action="ControleCliente" method="post">
      <input type="hidden" name="acao" value="inserir"/>
      <label for="cliente_nome">Nome</label>
      <input type="text" name="cliente_nome" maxlength="35"/><br/>
      <label for="cliente loja">Nome da Loja</label>
      <input type="text" name="cliente loja" maxlength="35"/><br/>
      <label for="cliente_fone">Telefone</label>
      <input type="text" name="cliente_fone" maxlength="11"/><br/>
      <label for="cliente_endereco">Endereço</label>
      <input type="text" name="cliente endereco" maxlength="100"/><br/>
      <label for="cliente cpf">CPF</label>
      <input type="text" name="cliente cpf" maxlength="11"/><br/>
      <label for="cliente_cnpj">CNPJ</label>
      <input type="text" name="cliente_cnpj" maxlength="15"/><br/>
      <input type="submit" value="Incluir Cliente"/>
      <input type = "button" value = "Cancelar" onClick = "window.location =
'<%=request.getContextPath() %> /home.jsp';"/><br/>
    </form>

    <a href="ControleCliente?acao=consultar">Listar Clientes</a><br/>
    <a href="../index.xhtml">Voltar</a>
  </body>
```

B. listarClientes.jsp

```
<%@page import="br.com.flordeliz.modelo.Cliente"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%@ page import="java.util.List" %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Cliente - Listar</title>

    <jsp:include page="/cabecalho.jsp"/>

    <h1>Lista de Clientes Registrados</h1>
    <a href="ControleCliente?acao=incluir">Inserir Cliente</a><br/>
    <a href="ControleCliente?acao=buscar">Buscar Cliente por Código</a>

    <table border="1">
      <tr>
        <th>Código</th>
        <th>Nome</th>
        <th>Nome da Loja</th>
        <th>Número de Telefone</th>
        <th>Endereco</th>
        <th>CPF</th>
        <th>CNPJ</th>
        <th colspan="2">Ações</th>
      </tr>
      <%
```

```
        List<Cliente> lista = (List<Cliente>) request.getAttribute("listaClientes");
        for (Cliente cliente : lista) {%>
          <tr>
            <td><%= cliente.getCodigo() %></td>
            <td>
              <a href="ControleCliente?acao=detalhar&cliente_codigo=<%= cliente.getCodigo()
%>">
                <%= cliente.getNome() %>
              </a>
            </td>
            <td><%= cliente.getLoja() %></td>
            <td><%= cliente.getFone() %></td>
            <td><%= cliente.getEndereco() %></td>
            <td><%= cliente.getCpf() %></td>
            <td><%= cliente.getCnpj() %></td>
            <td>
              <a href="ControleCliente?acao=atualizar&cliente_codigo=<%= cliente.getCodigo()
%>">
                Atualizar
              </a>
            </td>
            <td>
              <a href="ControleCliente?acao=excluir&cliente_codigo=<%= cliente.getCodigo()
%>">
                Excluir
              </a>
            </td>
          </tr>
        <%}
      %>
      </table>

  </body>
</html>
```

APÊNDICE F

Código Fonte da Classe LoteProducao

```
package br.com.flordeliz.modelo;

import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.GregorianCalendar;

public class LoteProducao {
  private int codigo;
  private GregorianCalendar data;
  private int quantidade;
  private ItemEstoque itemEstoque;

  public LoteProducao(int codigo, String data, int quantidade, ItemEstoque itemEstoque) throws
InsercaoException, ParseException{
    this.setCodigo(codigo);
    this.setDataString(data);
    this.setQuantidade(quantidade);
    this.setItemEstoque(itemEstoque);
  }

  public LoteProducao(int codigo, GregorianCalendar data, int quantidade, ItemEstoque
itemEstoque) throws InsercaoException, ParseException{
    this.setCodigo(codigo);
    this.setData(data);
    this.setQuantidade(quantidade);
    this.setItemEstoque(itemEstoque);
  }

  public void setCodigo(int codigo) throws InsercaoException {
    if(codigo < 0){
      throw new InsercaoException("Codigo invalido: valor negativo");
    } else {
      this.codigo = codigo;
    }
  }

  public String getDataString() {
    Date dataString = (Date) this.data.getTime();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    return sdf.format(dataString);
  }

  public Date getDataAsDate(){
    return new Date(this.getData().getTimeInMillis());
  }

  public java.sql.Date getDataAsSQLDate(){
    return new java.sql.Date(this.getData().getTimeInMillis());
  }
```

```
  public String getDataStringFormatada() {
    Date dataString = (Date) this.data.getTime();
    SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
    return sdf.format(dataString);
  }

  public void setDataString(String dataString) throws ParseException {
    DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
    Date date = (Date) df.parse(dataString);
    this.data = new GregorianCalendar();
    this.data.setTime(date);
  }

  public void setQuantidade(int quantidade) throws InsercaoException {
    if(quantidade < 0){
      throw new InsercaoException("Quantidade invalida: valor negativo");
    } else {
      this.quantidade = quantidade;
    }
  }

  public void setItemEstoque(ItemEstoque itemEstoque) {
    this.itemEstoque = itemEstoque;
  }
}
```

APÊNDICE G

Código Fonte da Classe LoteProducaoDAO

```
package br.com.flordeliz.dao;
import br.com.flordeliz.modelo.EntidadeNulaException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import br.com.flordeliz.modelo.LoteProducao;
import br.com.flordeliz.modelo.InsercaoException;
import br.com.flordeliz.modelo.ItemEstoque;
import br.com.flordeliz.modelo.ModeloCalcado;
import br.com.flordeliz.utils.Utils;
import java.text.ParseException;

public class LoteProducaoDAO extends DAO{
  public int inserir(String usuario, String senha, String endereco, LoteProducao lote) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "INSERT INTO lote_producao(lote_codigo, lote_data, lote_tamanho,
lote_quantidade, lote_item_est_codigo) VALUES(NEXTVAL('seq_lote_producao_codigo'),?,?,?,?)";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      Utils.checaNull(ps,"PreparedStatement",
Thread.currentThread().getStackTrace()[2].getLineNumber());

      ps.setDate(1, lote.getDataAsSQLDate());
      ps.setInt(2, lote.getItemEstoque().getTamanho());
      ps.setInt(3, lote.getQuantidade());
      ps.setInt(4, lote.getItemEstoque().getCodigo());
      ps.execute();
      conexao.close();
      return this.SUCESSO;
    } catch (SQLException e) {
      System.err.println("ERRO: falha ao inserir lote no banco" + "\r\n " + e.getMessage());
      return this.ERRO_SQL;
    }
  }

  public void alterar(String usuario, String senha, String endereco, LoteProducao lote) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "UPDATE lote_producao SET lote_data = ?, lote_tamanho = ?,
lote_quantidade = ?, lote_item_est_codigo = ? WHERE lote_codigo =?";

      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setDate(1, lote.getDataAsSQLDate());
```

```
      ps.setInt(2, lote.getItemEstoque().getTamanho());
      ps.setInt(3, lote.getQuantidade());
      ps.setInt(4, lote.getItemEstoque().getCodigo());
      ps.setInt(5, lote.getCodigo());
      ps.executeUpdate();
      conexao.close();
    } catch (SQLException e) {
      System.err.println("ERRO: Falha ao alterar lote de codigo " + lote.getCodigo() + "\r\n "
+ e.getMessage());
    }
  }

  public int excluir(String usuario, String senha, String endereco, int idExclusao) {
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());
    int resultadoOperacao = 0;
    try {
      String comandoSQL = "SELECT lote_codigo FROM lote_producao WHERE lote_codigo = ?";
      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setInt(1, idExclusao);
      ResultSet rs = ps.executeQuery();

      if (rs.next()){
        comandoSQL = "DELETE FROM lote_producao WHERE lote_codigo =?";
        /*PreparedStatement*/ ps = conexao.prepareStatement(comandoSQL);
        ps.setInt(1, idExclusao);
        ps.executeUpdate();
      } else {
        System.err.println("ALERTA: Lote com código " + idExclusao + " não
existente.\r\nExclusão não realizada");
        resultadoOperacao = LoteProducaoDAO.ERRO_CODIGO;
      }

      conexao.close();

      return resultadoOperacao = LoteProducaoDAO.SUCESSO;
    } catch (SQLException e) {
      System.err.println("Não foi possível excluir o lote" + "\nErro : " + e.getMessage());
      resultadoOperacao = LoteProducaoDAO.ERRO_SQL;
      return resultadoOperacao;
    }

  }

  public List<LoteProducao> consultar (String usuario, String senha, String endereco) {
    List<LoteProducao> listaLote = new ArrayList<LoteProducao>();
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());

    try {
      String comandoSQL = "SELECT lote_codigo,"
                + "lote_data, lote_tamanho, lote_quantidade, lote_item_est_codigo,"
                + "item_estoque_codigo, item_estoque_tamanho, item_estoque_quantidade,
item_estoque_preco_venda, item_estoque_modelo_codigo,"
```

```
                + "modelo_codigo, modelo_nome, modelo_colecao, modelo_tam_min, modelo_tam_max,
modelo_cor, modelo_preco_custo "
                + "FROM lote_producao, item_estoque, modelo_calcado "
                + "WHERE lote_item_est_codigo = item_estoque_codigo and
item_estoque_modelo_codigo = modelo_codigo "
                + "ORDER BY lote_codigo";

      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ResultSet rs = ps.executeQuery();
      while (rs.next()) {
        LoteProducao lote = null;
        try{
          ModeloCalcado modelo = new ModeloCalcado(rs.getInt("modelo_codigo"),
                                rs.getString("modelo_nome"),
                                rs.getString("modelo_colecao"),
                                rs.getInt("modelo_tam_min"),
                                rs.getInt("modelo_tam_max"),
                                rs.getString("modelo_cor"),
                                rs.getDouble("modelo_preco_custo"));

          ItemEstoque itemEstoque = new ItemEstoque(rs.getInt("item_estoque_codigo"),
                                 rs.getInt("item_estoque_tamanho"),
                                 rs.getInt("item_estoque_quantidade"),
                                 rs.getDouble("item_estoque_preco_venda"),
                                 modelo);

          lote = new LoteProducao(rs.getInt("lote_codigo"),
                      rs.getString("lote_data"),
                      //rs.getInt("lote_tamanho"),
                      rs.getInt("lote_quantidade"),
                      itemEstoque);
        }catch(SQLException e){
          System.err.println("ERRO: Falha na consulta por lotes:" + "\r\n" + e.getMessage());
        }catch(InsercaoException e){
          System.err.println("ERRO: Falha incluir informacao de item de estoque no lote:" +
"\r\n" + e.getMessage());
        }catch (ParseException e) {
          System.err.println("ERRO: data com formato incorreto registrada no banco:" + "\r\n"
+ e.getMessage());
        }
        listaLote.add(lote);
        Utils.checaNull(lote, "LoteProducao",
Thread.currentThread().getStackTrace()[2].getLineNumber());
      }
      conexao.close();
    } catch (SQLException e) {
       System.err.println("Consulta inválida!" + "\nErro : " + e.getMessage());
    }
    return listaLote;
  }

  public LoteProducao buscar (String usuario, String senha, String endereco, int idBusca)
throws EntidadeNulaException {
    LoteProducao lote = null;
    Connection conexao = ConectaBD.conectarBanco(usuario, senha, endereco);
    Utils.checaNull(conexao, "Connection",
Thread.currentThread().getStackTrace()[2].getLineNumber());
```

```
    try {
      String comandoSQL = "SELECT lote_codigo,"
          + "lote_data, lote_tamanho, lote_quantidade, lote_item_est_codigo,"
          + "item_estoque_codigo, item_estoque_tamanho, item_estoque_quantidade,
item_estoque_preco_venda, item_estoque_modelo_codigo,"
          + "modelo_codigo, modelo_nome, modelo_colecao, modelo_tam_min, modelo_tam_max,
modelo_cor, modelo_preco_custo "
          + "FROM lote_producao, item_estoque, modelo_calcado "
          + "WHERE lote_item_est_codigo = item_estoque_codigo and item_estoque_modelo_codigo =
modelo_codigo "
          + "AND lote_codigo = ?;";

      PreparedStatement ps = conexao.prepareStatement(comandoSQL);
      ps.setInt(1, idBusca);
      ResultSet rs = ps.executeQuery();
      while (rs.next()) {
        try{
          ModeloCalcado modelo = new ModeloCalcado(rs.getInt("modelo_codigo"),
                                rs.getString("modelo_nome"),
                                rs.getString("modelo_colecao"),
                                rs.getInt("modelo_tam_min"),
                                rs.getInt("modelo_tam_max"),
                                rs.getString("modelo_cor"),
                                rs.getDouble("modelo_preco_custo"));

          ItemEstoque itemEstoque = new ItemEstoque(rs.getInt("item_estoque_codigo"),
                                 rs.getInt("item_estoque_tamanho"),
                                 rs.getInt("item_estoque_quantidade"),
                                 rs.getDouble("item_estoque_preco_venda"),
                                 modelo);

          lote = new LoteProducao(rs.getInt("lote_codigo"),
                      rs.getString("lote_data"),
                      //rs.getInt("lote_tamanho"),
                      rs.getInt("lote_quantidade"),
                      itemEstoque);
        }catch(SQLException e){
          System.err.println("ERRO: Falha na consulta por lotes:" + "\r\n" + e.getMessage());
        }catch(InsercaoException e){
          System.err.println("ERRO: Falha incluir informacao de item de estoque no lote:" +
"\r\n" + e.getMessage());
        }catch (ParseException e) {
          System.err.println("ERRO: data com formato incorreto registrada no banco:" + "\r\n"
+ e.getMessage());
        }

        Utils.checaNull(lote, "LoteProducao",
Thread.currentThread().getStackTrace()[2].getLineNumber());
      }
      conexao.close();
    } catch (SQLException e) {
       System.err.println("Consulta inválida!" + "\nErro : " + e.getMessage());
    }
    Utils.checaNull(lote, "Lote", Thread.currentThread().getStackTrace()[2].getLineNumber());
    if (lote==null){
      throw new EntidadeNulaException("Lote não encontrado no banco de dados");
```

```
    }
    return lote;

  }
}
```

APÊNDICE H

Código Fonte da Classe ControleLote

```
package br.com.flordeliz.controlador;

import br.com.flordeliz.dao.ItemEstoqueDAO;
import br.com.flordeliz.dao.LoteProducaoDAO;
import br.com.flordeliz.modelo.LoteProducao;
import br.com.flordeliz.modelo.EntidadeNulaException;
import br.com.flordeliz.modelo.InsercaoException;
import br.com.flordeliz.modelo.ItemEstoque;
import java.io.IOException;
import java.text.ParseException;
import java.util.GregorianCalendar;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ControleLote extends HttpServlet implements InterfaceControle{
  public static final int LISTAR_LOTES = 4;
  public static final int DETALHAR_LOTE = 5;
  public static final int ATUALIZAR_LOTE = 6;
  public static final int INSERIR_LOTE = 7;
  public static final int BUSCAR_LOTE = 8;

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    String acao = request.getParameter("acao");
    if (acao==null) acao = "";
    int resultadoOperacao = 20;
    if (acao.equals("consultar")){
      List<LoteProducao> lista = consultarLotes();
      request.setAttribute("listaLotes", lista);
      resultadoOperacao = this.LISTAR_LOTES;


    } else if (acao.equals("inserir")){
      List<ItemEstoque> listaItensEstoque = this.consultarItensEstoque();
      request.setAttribute("listaItensEstoque", listaItensEstoque);
      resultadoOperacao = this.INSERIR_LOTE;


    } else if (acao.equals("incluir")){
      resultadoOperacao = this.inserirLote(request, response);


    } else if (acao.equals("buscar")){
      resultadoOperacao = this.BUSCAR_LOTE;
```

```
    } else if (acao.equals("detalhar")){
      try {
        LoteProducao lote = buscarLote(request);
        request.setAttribute("lote", lote);
        resultadoOperacao = this.DETALHAR_LOTE;
      } catch (EntidadeNulaException ex) {
        resultadoOperacao = LoteProducaoDAO.ERRO_CODIGO;
      }
    } else if (acao.equals("excluir")){
      resultadoOperacao = excluirLote(request, response);

    }

    RequestDispatcher rd;
    request.setAttribute("title","Lote - " + acao);
    switch (resultadoOperacao){

      case LoteProducaoDAO.SUCESSO:
        String processadas = " processadas ";
        if (acao.equals("incluir")){
          processadas = " inseridas ";
        } else if (acao.equals("excluir")){
          processadas = " excluídas ";
        } else {
          processadas = " alteradas ";
        }
        request.setAttribute("conteudo","<h1>Informações sobre o lote"+ processadas + "com
sucesso</h1>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case LoteProducaoDAO.ERRO_SQL:
        request.setAttribute("conteudo","<h1>Falha ao Acessar o Banco</h1>"
                  + "<p>Erro durante o acesso (leitura ou escrita) ao banco de dados. Comando
SQL malformado. Procure o Webmaster</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case LoteProducaoDAO.ERRO_INSERCAO:
        request.setAttribute("conteudo","<h1>Falha na Operação</h1>"
                        + "<p>Os dados inseridos eram inválidos. Por favor verifcar e tentar
outra vez</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case LoteProducaoDAO.ERRO_CODIGO:
        request.setAttribute("conteudo","<h1>Falha na Operação</h1>"
                        + "<p>O Código informado nao corresponde a um código de lote presente
no banco de dados</p>");
        rd = request.getRequestDispatcher("/resultadoOperacao.jsp");
        break;
      case LISTAR_LOTES:
        rd = request.getRequestDispatcher("listarLotes.jsp");
        break;
      case INSERIR_LOTE:
        rd = request.getRequestDispatcher("inserirLote.jsp");
        break;
      case BUSCAR_LOTE:
        rd = request.getRequestDispatcher("buscarLote.jsp");
```

```
            break;
          case DETALHAR_LOTE:
            rd = request.getRequestDispatcher("detalharLote.jsp");
            break;
          default:
            rd = request.getRequestDispatcher("buscarLote.jsp");
            break;
        }
        rd.forward(request, response);
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        processRequest(request, response);
    }

    protected int inserirLote(HttpServletRequest request, HttpServletResponse response){
        int resultadoOperacao = 0;
        try {
            LoteProducaoDAO loteProducaoDAO = new LoteProducaoDAO();
            ItemEstoqueDAO itemEstoqueDAO = new ItemEstoqueDAO();

            ItemEstoque item = itemEstoqueDAO.buscar(usuario, senha, endereco,
    Integer.parseInt(request.getParameter("itemEstoque")));
            LoteProducao lote = new LoteProducao(0,
                                new GregorianCalendar(),
                                Integer.parseInt(request.getParameter("quantidade")),
                                item);

            resultadoOperacao = loteProducaoDAO.inserir(usuario, senha, endereco, lote);
            System.out.print("Deu Certo");
            return resultadoOperacao;

        } catch (InsercaoException ex) {
            System.out.print("Insercao Exception");
            return LoteProducaoDAO.ERRO_INSERCAO;
        } catch (ParseException ex) {
            System.out.print("Parse Exception");
            return LoteProducaoDAO.ERRO_INSERCAO;
        }
    }

    protected List<LoteProducao> consultarLotes(){
        LoteProducaoDAO loteDAO = new LoteProducaoDAO();
        return loteDAO.consultar(this.usuario, this.senha, this.endereco);
    }

     protected LoteProducao buscarLote(HttpServletRequest request) throws EntidadeNulaException{
        int loteCodigo = Integer.parseInt(request.getParameter("lote_codigo"));
        LoteProducaoDAO loteDAO = new LoteProducaoDAO();
```

```
    return loteDAO.buscar(this.usuario, this.senha, this.endereco, loteCodigo);
  }

   protected List<ItemEstoque> consultarItensEstoque(){
     ItemEstoqueDAO itemEstoqueDAO = new ItemEstoqueDAO();
     return itemEstoqueDAO.consultar(this.usuario, this.senha, this.endereco);
   }

  private int excluirLote(HttpServletRequest request, HttpServletResponse response) {
    LoteProducaoDAO loteDAO = new LoteProducaoDAO();
    int loteCodigo = Integer.parseInt(request.getParameter("lote_codigo"));
    int resultadoOperacao = loteDAO.excluir(this.usuario, this.senha, this.endereco,
loteCodigo);
    return resultadoOperacao;
  }
}
```

APÊNDICE I

Código Fonte das JavaServer Pages de Manipulação de Lotes

```
<%@page import="br.com.flordeliz.modelo.ItemEstoque"%>
<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Lote - Inserir</title>

    <jsp:include page="/cabecalho.jsp"/>

    <script type="text/javascript">
      // Evita que seja feito pedido com quantidade maior que a disponível em estoque
      function validarQuantidade() {
        var e = document.getElementById("itemEstoque")
        var quantidade = parseInt(e.options[e.selectedIndex].getAttribute("data-quantidade"));
        console.log(quantidade);
        document.getElementById("quantidade").max=quantidade;
      }
    </script>

    <h1>Inserir Lote no Cadastro</h1>
    <form action="ControleLote" method="post">
      <input type="hidden" name="acao" value="incluir"/>
      <select name="itemEstoque" id="itemEstoque">
        <%
          List<ItemEstoque> listaItens = (List<ItemEstoque>)
request.getAttribute("listaItensEstoque");
          for (ItemEstoque item:listaItens){
            if (item.getQuantidade() > 0){
        %>
            <option value ="<%=item.getCodigo()%>" data-
quantidade="<%=item.getQuantidade()%>">
              <%=item.getModelo().getNome() + " | Tamanho " + item.getTamanho()+ " | " +
item.getModelo().getCor() + " (" + item.getQuantidade() + ")"%>
            </option>
        <%
            }
          }
        %>
      </select><br/>

      <label for="quantidade"><b>Quantidade: </b></label>
      <input type="number" min="0" step="1" name="quantidade" id="quantidade"/><br/>
      <input type="submit" value="Incluir Lote"/>
      <input type="button" value ="Cancelar"
onClick="window.location='<%=request.getContextPath() %>/home.jsp';"/><br/>
    </form>

  </body>
</html>



<%@page import="br.com.flordeliz.modelo.LoteProducao"%>
<%@page import="java.util.List"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Lote - Listar</title>

    <jsp:include page="/cabecalho.jsp"/>

    <h1>Lista de Lotes Registrados</h1>
    <a href="ControleLote?acao=incluir">Inserir Lote</a><br/>
    <a href="ControleLote?acao=buscar">Buscar Lote por Código</a>

    <table border="1">
```

```
        <tr>
          <th>Código</th>
          <th>Data de Produção</th>
          <th>Modelo</th>
          <th>Coleção</th>
          <th>Tamanho</th>
          <th>Quantidade</th>
          <th>Cor</th>
          <th colspan="3">Ações</th>
        </tr>

        <%
          List<LoteProducao> lista = (List<LoteProducao>) request.getAttribute("listaLotes");
          for (LoteProducao lote : lista) {%>
          <tr>
            <td><%=lote.getCodigo()%></td>
            <td><%=lote.getDataStringFormatada()%></td>
            <td><%=lote.getItemEstoque().getModelo().getNome()%></td>
            <td><%=lote.getItemEstoque().getModelo().getColecao()%></td>
            <td><%=lote.getItemEstoque().getTamanho()%></td>
            <td><%=lote.getQuantidade()%></td>
            <td><%=lote.getItemEstoque().getModelo().getCor()%></td>
            <td>
              <a href="ControleLote?acao=detalhar&lote_codigo=<%= lote.getCodigo() %>">
                Detalhar
              </a>
            </td>
            <td>Atualizar</td>
            <td>
              <a href="ControleLote?acao=excluir&lote_codigo=<%= lote.getCodigo() %>">
                Excluir
              </a>
            </td>
          </tr>
          <%}
        %>
      </table>
    </body>
  </html>
```

APÊNDICE J

Código Fonte da Classe Pedido

```
package br.com.flordeliz.modelo;

import br.com.flordeliz.utils.Utils;
import java.util.ArrayList;
import java.util.List;

public class Pedido {
  private int codigo;
  private float desconto;
  private Cliente cliente;
  private List <ItemPedido> listaItens = new ArrayList<ItemPedido>();
  private Entrega entrega;
  private double precoTotalPedidoBruto;
  private double precoTotalPedidoLiquido;

  public Pedido (int codigo, float desconto, Cliente cliente){
    this.setCodigo(codigo);
    this.setDesconto(desconto);
    this.setCliente(cliente);
    this.atualizarPrecoTotal();
  }

  public Pedido (int codigo, float desconto, Cliente cliente, List <ItemPedido> listaItens){
    this.setCodigo(codigo);
    this.setDesconto(desconto);
    this.setCliente(cliente);
    this.setListaItens(listaItens);
    this.atualizarPrecoTotal();
  }

  public Pedido (int codigo, float desconto, Cliente cliente, List <ItemPedido> listaItens,
Entrega entrega){
    this.setCodigo(codigo);
    this.setDesconto(desconto);
    this.setCliente(cliente);
    this.setListaItens(listaItens);
    this.setEntrega(entrega);
    this.atualizarPrecoTotal();
  }

  private void atualizarPrecoTotal(){
    double precoTotal = 0;
    for (ItemPedido item : this.listaItens){
      precoTotal += item.getPrecoTotalItem();
    }
    this.precoTotalPedidoBruto = precoTotal;
    this.setPrecoTotalPedidoLiquido(this.getPrecoTotalPedidoBruto() * this.desconto);
  }

  public void inserirItem(ItemPedido itemPedido) throws InsercaoException {
    boolean jaInserido = false;
    for (ItemPedido item : this.listaItens){
      if(item.getItemEstoque().getCodigo() == itemPedido.getItemEstoque().getCodigo()){
        jaInserido = true;
        break;
      }
    }
    if (!jaInserido){
      this.listaItens.add(itemPedido);
    }else{
      throw new InsercaoException("ERRO: Item já presente no pedido");
    }

    this.atualizarPrecoTotal();
  }

  public void removerItem(int codigo){
    for (ItemPedido item : this.listaItens){
      if (item.getCodigo() == codigo){
        this.listaItens.remove(item);
```

```
        break;
      }
    }

    this.atualizarPrecoTotal();
  }

  public ItemPedido getItem(int codigo){
    ItemPedido itemRetorno = null;
    for (ItemPedido item : this.listaItens){
      if (item.getCodigo() == codigo){
        itemRetorno = item;
        break;
      }
    }
    return itemRetorno;
  }

  public void atualizarItem(int codigo, int quantidade, ItemEstoque itemEstoque){
    for (ItemPedido item : this.listaItens){
      if (item.getCodigo() == itemEstoque.getCodigo()){
        item.setQuantidade(quantidade);
        item.setItemEstoque(itemEstoque);
        break;
      }
    }

    this.atualizarPrecoTotal();
  }

  public void setDesconto(float desconto) {
    this.desconto = desconto;

    this.atualizarPrecoTotal();
  }

  public void setListaItens(List <ItemPedido> listaItens) {
    this.listaItens = listaItens;

    this.atualizarPrecoTotal();
  }

  public void setCliente(Cliente cliente) {
    this.cliente = (Cliente) Utils.checaNull(cliente);
  }
}
```