Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Sistema de controle de estoque doméstico

Autor:

_____

Alexandre Figueiredo Nazareth

Orientador:

_____

Prof. Antônio Cláudio Gómez de Souza, M.Eng.

Examinador:

_____

Prof. Flávio Luis de Mello, D.Sc.

Examinador:

_____

Prof. Sergio Barbosa Villas-Boas, D.Sc.

DEL

Novembro de 2010

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica – Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro – RJ CEP 21949-900

Este exemplar é de propriedade da Universidade Federal do Rio de Janeiro, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde3 que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

AGRADECIMENTO

Dedico este trabalho a todos que ao longo dos anos de curso me apoiaram, não deixando que as dificuldades me tirassem do caminho rumo à minha graduação e àqueles que não deixaram de dar seu apoio mesmo quando distantes. Este projeto é a prova que todo o apoio não foi em vão e ele não seria possível sem essa ajuda.

RESUMO

Esse trabalho se propõe a apresentar uma solução para o desenvolvimento de uma ferramenta de software capaz de permitir um controle de pequenos estoques, mais especificamente os estoques domésticos.

Utilizando tecnologias de ponta para o desenvolvimento, como a linguagem Java e o gerenciador de bancos de dados PostgreSQL, através de uma arquitetura orientada a objetos, será apresentada uma solução simples e eficaz para o gerenciamento cotidiano do estoque de diversos itens de consumo diário que são estocados em nossas casas.

Palavras-Chave: estoque, doméstico, Java, PostgreSQL.

ABSTRACT

This project intends to present a solution for the development of a software application capable of allowing the control of small stocks, specifically domestic stocks.

Using state-of-the-art technology for the development, like the Java programming language and the PostgreSQL database manager, through a object oriented architecture, a simple and effective solution will be presented for the day-to-day management of the inventory of several daily consumer items which we need to stock in our households.

Key-words: stock, domestic, Java, PostgreSQL.

SIGLAS

IDE – *Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)

SGBD – Sistema Gerenciador de Banco de Dados

MVC – *Model-View-Controller* (Modelo-Visão-Controlador)

UML – *Unified Modeling Language* (Linguagem de Modelagem Unificada)

JPA – *Java Persistence API* (API de Persistência Java)

API – *Application Programming Interface* (Interface de Programação de Aplicações)

Java EE – *Java Enterprise Edition* (Java Edição Empresarial)

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 – Tema

Esse trabalho tem como tema o desenvolvimento de uma ferramenta de software capaz de auxiliar no gerenciamento de estoques de pequeno porte. Com ênfase nos estoques domésticos, este trabalho tratará da apresentação e implementação de uma solução para agilizar o registro e acompanhamento das entradas e saídas desses estoques, com o objetivo de facilitar o controle de itens de consumo utilizados na residência do usuário.

1.2 – Delimitação

O objeto do projeto são os estoques de pequeno porte, mais especificamente os estoques domésticos. Sejam estoques de gêneros, matérias de limpeza ou suprimentos médicos, todos os tipos de estoque de itens de consumo imediatos do cotidiano fazem parte do escopo da aplicação a ser desenvolvida por esse projeto.

No entanto, ainda que o foco específico seja nos estoques domésticos, com pequenos ajustes, a solução apresentada poderá também ser aplicada a pequenos estoques comerciais, como armarinhos, pequenas farmácias ou mesmo lojas de ferragens.

1.3 – Justificativa

Atualmente, cada vez mais o tempo se torna um recurso bastante escasso para cada um de nós. Dessa forma, as pessoas são levadas a procurar otimizar o tempo gasto em cada uma das atividades, cotidianas ou não, para que esse recurso escasso seja aproveitado ao máximo sem ser desperdiçado.

Temos então que as atividades domésticas, especificamente as relacionadas com a administração de um lar, precisam de ferramentas modernas para que a otimização do tempo gasto seja possível. Assim nos voltamos para os estoques que mantemos em nossos lares. Desde os gêneros em despensa aos produtos de limpeza ou mesmo os remédios que mantemos para uso eventual, temos a necessidade de controlarmos o que possuímos disponível para executar as tarefas de manutenção do lar e aquilo que necessitamos e precisamos repor.

É justamente nesse controle que se torna necessária uma ferramenta que permita de forma ágil registrar todas as entradas e saídas, assim como acompanhar todo o material disponível em casa, verificando a evolução desse pequeno estoque e avaliando os custos previsíveis para o futuro próximo.

Com isso em mente, o presente projeto é então voltado para desenvolver uma ferramenta simples para atender essa necessidade, contribuindo com a agilidade e confiabilidade desse controle, procurando a manutenção da disponibilidade das informações para o devido acompanhamento a todo momento.

1.4 – Objetivos

O objetivo do projeto é então desenvolver uma ferramenta de software capaz de agilizar o controle de entradas e saídas em estoques domésticos, permitindo o acompanhamento da evolução do mesmo e possibilitando a previsão de custos de reposição em estoque dos itens considerados necessários em futuro próximo.

1.5 – Metodologia

Procurando simplificar a abordagem tradicional sobre gerenciamento de estoques de forma a obter uma solução simples a ser utilizada em pequenos estoques com facilidade, esse projeto irá focar em três macro-operações, cadastro, entrada/saída e acompanhamento.

O cadastro irá então ser constituído dos registros relacionados às entidades utilizadas nas operações de entrada e saída (itens, estoques, etc.), evitando que os dados das mesmas sejam necessários no momento da operação principal do sistema (entrada e

saída). A entrada/saída, sendo a macro-operação principal do sistema irá efetivamente registrar os itens de consumo incluídos ou excluídos no estoque, individualmente ou em conjunto. O acompanhamento será então constituído de relatórios que auxiliaram na visualização da situação do estoque em determinado momento.

Esse projeto irá utilizar técnicas de orientação a objetos já consagradas no mercado para a modelagem e o desenvolvimento da ferramenta objeto do mesmo, utilizando um banco de dados relacional para o armazenamento das informações do estoque.

O êxito do projeto será determinado pelo desenvolvimento completo da referida ferramenta, de forma que a mesma possa ser instalada e utilizada em qualquer estoque de pequeno porte, especificamente nos estoques domésticos.

1.6 – Descrição

No capítulo 2, será apresentada uma descrição mais detalhada sobre o problema a ser solucionado com o projeto, a solução proposta que será adotada no desenvolvimento da ferramenta, o método a ser utilizado para tal e o ciclo de vida do projeto.

O capítulo 3 mostrará a análise realizada para se chegar à solução proposta, sendo apresentados cada um dos produtos da citada análise.

A partir do modelo resultante da análise apresentada no capítulo 3, será descrito no capítulo 4 o desenho da implementação da ferramenta, apresentando também os produtos resultantes desse processo.

No capítulo 5 serão apresentadas algumas breves observações relacionadas à codificação a ao processo de teste do sistema. Nesse mesmo capítulo serão apresentados os resultados obtidos.

Ao fim, serão apresentadas as conclusões tiradas do projeto no capítulo 6, avaliando o método de implementação, as ferramentas utilizadas no desenvolvimento, incluindo IDE, SGBD e a própria linguagem de programação. Nesse mesmo capítulo também serão sugeridas melhorias a serem feitas futuramente, assim como possíveis expansões para a idéia original do sistema. Como último item do capítulo 6, serão citadas as lições aprendidas com a execução do projeto.

Capítulo 2

O problema, a solução, o método e o ciclo de vida

2.1 – O problema

O tempo nos dias atuais é cada vez mais um recurso escasso em nossas vidas. No crescente dinamismo da vida moderna, entre o trabalho e diversas formas de entretenimento, certas tarefas antes vistas como naturais e parte do cotidiano vem sendo relegadas a um segundo plano, onde se busca formas mais eficientes para as mesmas consumirem o mínimo de tempo possível das pessoas.

Dentro desse universo de tarefas que não mais são vistas como dignas da dedicação de muito tempo, a administração das nossas casas é uma que não podemos ignorar. E tratando-se de administração doméstica, o controle dos recursos materiais necessários para a execução das diversas tarefas de manutenção do lar se torna extremamente importante, sendo no entanto necessária a otimização do tempo gasto no mesmo.

O conhecimento de quais itens temos em casa para executarmos as diversas tarefas, como produtos de limpeza para uma eventual faxina, ou gêneros para a preparação de uma refeição, produtos de higiene ou mesmo medicamentos de uso comum é essencial para que em um momento de necessidade tais itens não faltem.

Ao mesmo tempo, termos como prever os custos que a reposição de itens que venham a sobrar em quantidade inferior àquela que achamos necessária é importantíssimo em um planejamento de gastos.

Para que esse controle então seja possível, se torna necessária uma ferramenta que permita o devido registro de compras e de consumo dos itens que mantemos em casa. Ferramenta essa que permita evidenciar aquilo que falta em nosso lar e quanto devemos gastar para que possamos repor os itens necessários.

2.1.1 – Requisitos de negócio

Tendo em vista que o sistema tem por função principal o controle de entrada e saída de itens em estoque, temos as seguintes funcionalidades principais do sistema:

- ✓ Cadastro de itens.
- ✓ Entrada de estoque.
- ✓ Saída de estoque.
- ✓ Geração de relatórios de acompanhamento e previsão de gastos.

2.1.2 – Diagrama de casos de uso de negócio

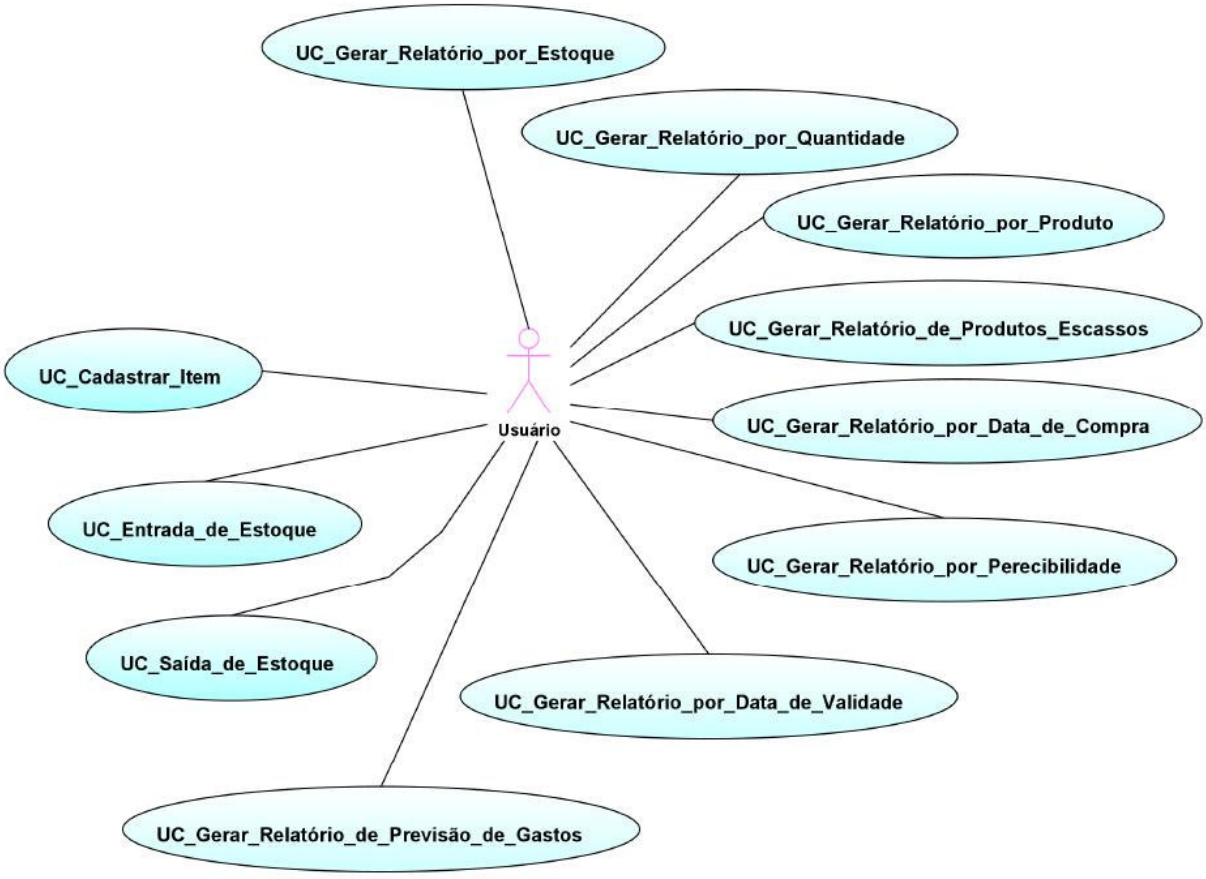

Figura 1 - Diagrama de casos de uso de negócio

2.1.3 – Funcionalidades extras

Após uma primeira análise dos requisitos básicos de negócio do sistema, foram observadas algumas funcionalidades que não haviam sido consideradas na definição dos requisitos de negócio.

- ✓ Registro de usuários, para possibilitar a validação dos usuários utilizando o sistema.
- ✓ Cadastro de novas unidades de medida.
- ✓ Cadastro de novos estoques, eventualmente não previstos ou definidos como padrão.
- ✓ Cadastro de receitas, para permitir a saída combinada de itens de forma pré-programada.
- ✓ Execução de receitas, para realizar as saídas programadas segundo o cadastro de receitas.

2.1.4 – Requisitos de sistema

Temos então, com as funcionalidades básicas de negócio e as funcionalidades extras, a lista dos casos de uso do sistema:

- ✓ Registrar Usuário
- ✓ Cadastrar Unidade de Medida
- ✓ Cadastrar Item
- ✓ Cadastrar Receita
- ✓ Cadastrar Estoque
- ✓ Entrada de Estoque
- ✓ Saída de Estoque
- ✓ Executar Receita
- ✓ Gerar Relatório por Estoque
- ✓ Gerar Relatório por Quantidade
- ✓ Gerar Relatório por Produto
- ✓ Gerar Relatório por Data de Compra
- ✓ Gerar Relatório por Perecibilidade
- ✓ Gerar Relatório por Data de Validade
- ✓ Gerar Relatório de Produtos Escassos
- ✓ Gerar Relatório de Previsão de Gastos

2.1.5 – Diagrama de casos de uso de sistema

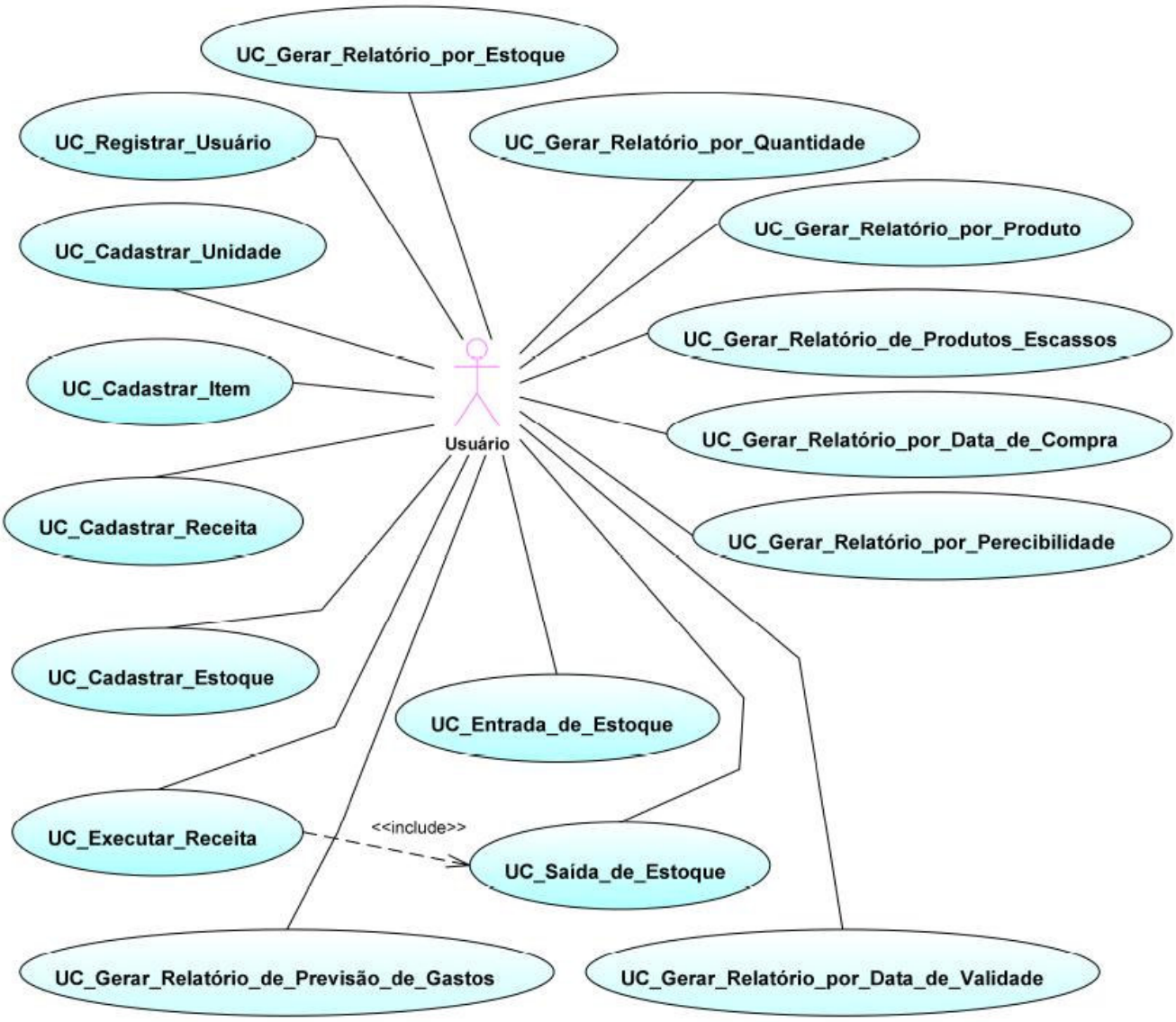

Figura 2 – Diagrama de casos de uso de sistema

2.1.6 – Descrição dos casos de uso

2.1.6.1 – Registrar Usuário

Descrição: O acesso ao sistema é feito somente por usuários registrados, com seus respectivos logins e senhas às funcionalidades cadastradas em seus respectivos perfis.

Ator: Usuário com acesso à funcionalidade *"Registrar Usuário"*.

Curso Típico

- [P1] O sistema lista os usuários já cadastrados.
- O ator seleciona a opção para registrar novo usuário [A1] [A2].
- O sistema solicita o preenchimento dos dados do novo usuário (Nome, login, senha e confirmação de senha).
- O ator informa os dados solicitados.
- [P2] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema exibe a tela para o cadastro do perfil do novo usuário [A3].
- O ator informa as funcionalidades às quais o usuário irá ter acesso.
- O sistema insere os dados do novo usuário na base de dados.
- O sistema exibe mensagem informando do registro bem-sucedido do novo usuário. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona um usuário para alteração

- O ator seleciona um usuário da lista para alteração.
- O sistema apresenta as opções de alteração (dados ou perfil de acesso).
- O ator seleciona a opção de alteração de dados [A4].
- O sistema solicita o preenchimento dos dados a serem alterados.
- O ator informa os dados solicitados.
- [P3] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema solicita confirmação da alteração dos dados [A5].
- O ator confirma a alteração dos dados informados.
- O sistema altera os dados do usuário na base de dados.
- O sistema exibe mensagem informando da alteração bem-sucedida dos dados do usuário. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona um usuário para exclusão

- O ator seleciona um usuário da lista para exclusão.
- O sistema solicita confirmação da exclusão do usuário.

- O ator confirma a exclusão do usuário.
- O sistema exclui o usuário da base de dados.
- O sistema exibe mensagem informando da exclusão bem-sucedida do usuário. Volta ao passo [P1].

Curso Alternativo [A3] – O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P2].

Curso Alternativo [A4] – O ator seleciona a opção de alteração de perfil

- O ator seleciona a opção de alteração de perfil
- O sistema solicita as funcionalidades às quais o usuário irá ter acesso.
- O sistema solicita confirmação da alteração do perfil.
- O ator confirma a alteração do perfil do usuário.
- O sistema altera o perfil do usuário na base de dados.
- O sistema exibe mensagem informando da alteração bem-sucedida do perfil do usuário. Volta ao passo [P1].

Curso Alternativo [A5] - O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P3].

2.1.6.2 – Cadastrar Unidade de Medida

Descrição: Unidades de medida devem ser cadastradas para serem usadas como referência de quantidades para os itens de estoque.

Ator: Usuário com acesso ao menu "*Cadastro*" e à funcionalidade *"Cadastrar Unidade"*.

Curso Típico

- [P1] O sistema apresenta as unidades de medida cadastradas.
- O ator seleciona a opção para cadastrar nova unidade de medida [A1] [A2].
- O sistema solicita o preenchimento dos dados da nova unidade de medida (Descrição e sigla).
- O ator informa os dados solicitados.
- [P2] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema insere a nova unidade de medida na base de dados [A3].
- O sistema exibe mensagem informando do registro bem-sucedido da nova unidade de medida. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona unidade de medida para alteração

- O ator seleciona unidade de medida para ser alterada.
- O sistema solicita os dados da unidade de medida a serem alterados.
- O ator informa os dados solicitados.
- [P3] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema solicita confirmação da alteração dos dados [A4].
- O ator confirma a alteração.
- O sistema altera os dados da unidade de medida na base de dados.
- O sistema exibe mensagem informando da alteração bem-sucedida dos dados da unidade de medida. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona unidade de medida para exclusão

- O ator seleciona unidade de medida para exclusão.
- O sistema solicita confirmação da exclusão da unidade de medida.

- O ator confirma a exclusão da unidade de medida.
- O sistema exclui a unidade de medida da base de dados.
- O sistema exibe mensagem informando da exclusão bem-sucedida da unidade de medida. Volta ao passo [P1].

Curso Alternativo [A3] - O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P2].

Curso Alternativo [A4] - O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P3].

2.1.6.3 – Cadastrar Item

Descrição: Itens de estoque devem ser cadastrados para permitir a entrada dos mesmos no estoque e dar subsídio para os relatórios de acompanhamento e evolução do estoque.

Ator: Usuário com acesso ao menu "*Cadastro*" e à funcionalidade *"Cadastrar Item"*.

Curso Típico

- [P1] O sistema apresenta os itens cadastrados.
- O ator seleciona a opção para cadastrar novo item [A1] [A2].
- O sistema solicita o preenchimento dos dados do novo item (Descrição, quantidade mínima necessária em estoque).
- O ator informa os dados solicitados.

- O sistema lista as unidades de medida padrão e solicita ao ator qual será utilizada para o novo item.
- O ator informa a unidade de medida padrão.
- [P2] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema insere o novo item na base de dados [A3].
- O sistema exibe mensagem informando do registro bem-sucedido do novo item. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona item para alteração

- O ator seleciona item para alteração.
- O sistema solicita os dados do item a serem alterados.
- O ator informa os dados solicitados.
- O sistema lista as unidades de medida padrão e solicita ao ator qual será utilizada para o item.
- [P3] O sistema verifica a validade dos dados informados.
- Com os dados válidos, o sistema solicita confirmação da alteração dos dados [A4].
- O ator confirma a alteração.
- O sistema altera os dados do item na base de dados.
- O sistema exibe mensagem informando da alteração bem-sucedida dos dados do item. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona item para exclusão

- O ator seleciona item para exclusão.
- O sistema solicita confirmação da exclusão do item.
- O ator confirma a exclusão do item.
- O sistema exclui o item da base de dados.
- O sistema exibe mensagem informando da exclusão bem-sucedida do item. Volta ao passo [P1].

Curso Alternativo [A3] - O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P2].

Curso Alternativo [A4] - O sistema verifica que os dados informados são inválidos

- O sistema verifica que os dados informados são inválidos e solicita a correção dos mesmos.
- O ator corrige os dados inválidos. Volta ao passo [P3].

2.1.6.4 – Cadastrar Receita

Descrição: Receitas podem ser cadastradas a partir dos itens existentes para otimizar a saída de estoque.

Ator: Usuário com acesso ao menu "*Cadastro*" e à funcionalidade *"Cadastrar Receita"*.

Curso Típico

- [P1] O sistema apresenta as receitas cadastradas.
- O ator seleciona a opção para cadastrar nova receita [A1] [A2] [A3] [A4] [A5].
- O sistema solicita a descrição da receita.
- O ator informa a descrição da receita.
- [P2] O sistema apresenta os itens cadastrados para serem incluídos da receita.
- O ator seleciona a opção de finalizar a receita [A6].
- O sistema solicita a confirmação da inclusão da nova receita.

- O ator confirma a inclusão da nova receita.
- O sistema insere a nova receita na base de dados.
- O sistema inclui os itens da receita na base de dados.
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona uma receita para alteração da descrição

- O ator seleciona uma receita para alteração da descrição.
- O sistema solicita a nova descrição da receita.
- O ator informa a nova descrição da receita.
- O sistema altera a descrição da receita na base de dados. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona uma receita para alteração da quantidade de item da receita

- O ator seleciona uma receita para alteração da quantidade de item da receita.
- O sistema apresenta a lista dos itens da receita.
- O ator seleciona o item da receita a ter sua quantidade alterada.
- O sistema solicita a nova quantidade do item na unidade padrão do mesmo.
- O ator informa a nova quantidade do item.
- O sistema altera a quantidade do item da receita na base de dados. Volta ao passo [P1].

Curso Alternativo [A3] – O ator seleciona uma receita para exclusão de item da receita

- O ator seleciona uma receita para exclusão de item da receita.
- O sistema apresenta a lista dos itens da receita.
- O ator seleciona o item da receita a ser excluído.
- O sistema solicita a confirmação da exclusão.
- O ator confirma a exclusão do item da receita.
- O sistema exclui o item da receita na base de dados. Volta ao passo [P1].

Curso Alternativo [A4] – O ator seleciona uma receita para inclusão de item na receita

- O ator seleciona uma receita para inclusão de item na receita.
- [P3] O sistema apresenta os itens cadastrados para serem incluídos da receita.
- O ator seleciona o item a ser incluído na receita.
- O sistema solicita a quantidade do novo item da receita na unidade padrão do mesmo.
- O ator informa a quantidade do novo item.
- O sistema solicita a confirmação da inclusão do novo item na receita.
- O ator confirma a inclusão do novo item.
- O sistema inclui o novo item na receita.
- O sistema pergunta se o ator deseja incluir novo item ou finalizar a inclusão de novos itens.
- O ator seleciona a opção de finalizar a inclusão de novos itens na receita [A7].
- O sistema inclui os novos itens da receita na base de dados. Volta ao passo [P1].

Curso Alternativo [A5] – O ator seleciona uma receita para exclusão

- O ator seleciona uma receita para exclusão.
- O sistema solicita a confirmação da exclusão.
- O ator confirma a exclusão da receita.
- O sistema exclui a receita da base de dados. Volta ao passo [P1].

Curso Alternativo [A6] – O ator seleciona um item para inclusão na receita

- O ator seleciona um item para inclusão na receita.
- O sistema solicita a quantidade do item a ser incluído na receita na unidade padrão do mesmo.
- O ator informa a quantidade do item.
- O sistema solicita a confirmação da inclusão do item na receita.
- O ator confirma a inclusão do item.

- O sistema inclui o item na receita.
- Volta ao passo [P2].

Curso Alternativo [A7] – O ator seleciona a opção para incluir novo item na receita

- O ator seleciona a opção para incluir novo item na receita.
- O sistema apresenta os itens cadastrados para serem incluídos da receita.
- O ator seleciona o item a ser incluído na receita.
- O sistema solicita a quantidade do novo item da receita na unidade padrão do mesmo.
- O ator informa a quantidade do novo item.
- O sistema solicita a confirmação da inclusão do novo item na receita.
- O ator confirma a inclusão do novo item.
- O sistema inclui o novo item na receita.
- Volta ao passo [P3].

2.1.6.5 – Cadastrar Estoque

Descrição: Estoques devem ser cadastrados para ser possível a execução da entrada de itens.

Ator: Usuário com acesso ao menu "*Cadastro*" e à funcionalidade *"Cadastrar Estoque"*.

Curso Típico

- [P1] O sistema apresenta os estoques cadastrados.
- O ator seleciona a opção para cadastrar novo estoque [A1] [A2].
- O sistema solicita a descrição do novo estoque.
- O ator informa a descrição do estoque.
- [P2] O sistema verifica a validade da descrição informada.
- Com a descrição válida, o sistema insere o novo estoque na base de dados [A3].

- O sistema exibe mensagem informando do registro bem-sucedido do novo estoque. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona um estoque para alteração da descrição

- O ator seleciona um estoque para alteração da descrição.
- O sistema solicita a nova descrição do estoque.
- O ator informa a nova descrição do estoque.
- [P3] O sistema verifica a validade da descrição informada.
- Com a descrição válida, o sistema altera a descrição do estoque na base de dados [A4]. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona um estoque para exclusão

- O ator seleciona um estoque para exclusão.
- O sistema verifica se o estoque está vazio e assim válido para exclusão.
- Com a exclusão válida, o sistema exclui o estoque da base de dados [A5]. Volta ao passo [P1].

Curso Alternativo [A3] – O sistema verifica que a descrição informada para o novo estoque é inválida

- O sistema verifica que a descrição informada para o novo estoque é inválida.
- O ator corrige a descrição inválida. Volta ao passo [P2].

Curso Alternativo [A4] – O sistema verifica que a nova descrição informada para o estoque é inválida

- O sistema verifica que a nova descrição informada para o estoque é inválida.
- O ator corrige a descrição inválida. Volta ao passo [P3].

Curso Alternativo [A5] - O sistema verifica que a exclusão do estoque é inválida

- O sistema verifica que a exclusão do estoque é inválida.
- O sistema exibe mensagem informando o ator que a exclusão do estoque não é válida, pois o estoque não está vazio. Volta ao passo [P1].

2.1.6.6 – Entrada de Estoque

Descrição: A entrada de estoque é a funcionalidade que permite adicionar itens a um estoque cadastrado.

Ator: Usuário com acesso ao menu "*Estoque*" e à funcionalidade *"Entrada de Estoque"*.

Curso Típico

- [P1] O sistema apresenta os itens cadastrados.
- O ator seleciona o item a ser inserido em um dos estoques.
- O sistema solicita os dados de entrada para o item selecionado (Estoque no qual o item será inserido, quantidade na unidade padrão do item, data de validade, preço de compra e data de compra).
- O ator informa os dados de entrada.
- [P2] O sistema valida as datas informadas.
- Com as datas válidas, o sistema solicita a confirmação da entrada do item selecionado no estoque escolhido [A1].
- O ator confirma a entrada.
- O sistema inclui o registro da entrada de estoque na base de dados. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O sistema verifica que uma das datas informadas é inválida

- O sistema verifica que uma das datas informadas é inválida.
- O ator corrige a data inválida. Volta ao passo [P2].

2.1.6.7 – Saída de Estoque

Descrição: A saída de estoque é a funcionalidade mais simples referente à retirada individual dos itens em estoque para consumo.

Ator: Usuário com acesso ao menu "*Estoque*" e à funcionalidade *"Saída de Estoque"*.

Curso Típico

- [P1] O sistema apresenta os itens cadastrados.
- O ator seleciona o item a ser retirado de um dos estoques.
- O sistema solicita os dados de saída para o item selecionado (Estoque do qual o item será retirado e quantidade na unidade padrão do item).
- O ator informa os dados de entrada.
- [P2] O sistema verifica se a quantidade informada existe em estoque.
- Existindo a quantidade no estoque, o sistema solicita a confirmação da retirada do item selecionado do estoque escolhido [A1].
- O ator confirma a retirada.
- O sistema subtrai na base de dados a quantidade retirada das entradas na validade (Com perecibilidade "*Perecível"* dentro da validade ou "*Não perecível*") mais antigas. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] - O sistema verifica que a quantidade em estoque é insuficiente para a retirada informada

- O sistema verifica que a quantidade em estoque é insuficiente para a retirada informada.
- O ator corrige a quantidade. Volta ao passo [P2].

2.1.6.8 – Executar Receita

Descrição: A execução de receitas é a funcionalidade que permite a retirada única de diversos itens de estoque de acordo com uma receita pré-cadastrada, de forma a aperfeiçoar a saída de estoque de itens consumidos freqüentemente em conjunto.

Ator: Usuário com acesso ao menu "*Estoque*" e à funcionalidade *"Executar Receita"*.

Curso Típico

- [P1] O sistema apresenta as receitas cadastradas.
- O ator seleciona a receita a ser executada.
- O sistema verifica se as quantidades dos itens da receita selecionada existem em estoque.
- Existindo as quantidades no estoque, o sistema solicita a confirmação da execução da receita [A1].
- O ator confirma a execução.
- O sistema subtrai na base de dados as quantidades dos itens da receita das entradas na validade (Com perecibilidade "*Perecível"* dentro da validade ou "*Não perecível*") mais antigas. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] - O sistema verifica que as quantidades dos itens em estoque são insuficientes para a execução da receita selecionada

- O sistema verifica que as quantidades dos itens em estoque são insuficientes para a execução da receita selecionada.
- O sistema informa que não há itens em estoque suficientes para a execução da receita selecionada. Volta ao passo [P1].

2.1.6.9 – Gerar Relatório por Estoque

Descrição: Gera o relatório, agrupado por item, com todos os itens constantes no estoque selecionado, informando as datas de compra e validade, preço de compra, quantidade e perecibilidade de cada entrada do item no estoque, assim como a descrição e a quantidade mínima dos itens em suas unidades padrão.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema apresenta os estoques cadastrados.
- O ator seleciona o estoque desejado.
- O sistema consulta na base de dados todos os itens constantes no estoque selecionado.
- O sistema exibe para o ator o relatório dos itens constantes no estoque selecionado. Volta ao passo [P1].
- ** Fim do caso de uso **

2.1.6.10 – Gerar Relatório por Quantidade

Descrição: Gera o relatório, agrupado por item, dos itens constantes em estoque com a quantidade solicitada, informando as datas de validade e compra, preço de compra, perecibilidade e estoque de cada entrada do item, assim como a descrição e quantidade mínima dos itens em suas unidades padrão.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema solicita a quantidade e unidade padrão pelas quais buscar os itens no estoque.
- O ator informa a quantidade e a unidade padrão para a busca dos itens.
- O sistema busca todos os itens em estoque com quantidade solicitada.
- O sistema exibe o relatório com os itens com a quantidade solicitada. Volta ao passo [P1].

- ** Fim do caso de uso **

2.1.6.11 – Gerar Relatório por Produto

Descrição: Gera o relatório, agrupado por estoque, das entradas em estoque do item selecionado, informando a descrição e quantidade mínima do item na unidade padrão, assim como todas as entradas em estoque do item, com datas de validade e compra, preço de compra, quantidade e perecibilidade.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema apresenta os itens existentes em estoque.
- O ator seleciona o item para gerar o relatório.
- O sistema consulta todas as entradas de estoque do item selecionado.
- O sistema exibe o relatório com as entradas de estoque do item selecionado. Volta ao passo [P1].
- ** Fim do caso de uso **

2.1.6.12 – Gerar Relatório por Data de Compra

Descrição: Gera o relatório, agrupado por estoque, das entradas em estoque com a data de compra selecionada, informando o item, quantidade na unidade padrão do mesmo, data de validade, perecibilidade e estoque de cada entrada.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema solicita a data de compra pela qual gerar o relatório.
- O ator informa a data de compra.

- O sistema consulta as entradas em estoque com a data de compra informada.
- O sistema exibe o relatório com as estradas em estoque com a data de compra informada. Volta ao passo [P1].
- ** Fim do caso de uso **

2.1.6.13 – Gerar Relatório por Perecibilidade

Descrição: Gera o relatório, agrupado por itens, das entradas em estoque com a perecibilidade selecionada, informando datas de validade e compra, preço de compra e estoque das entradas, assim como descrição e quantidade mínima dos itens em suas unidades padrão.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema solicita a perecibilidade pela qual gerar o relatório.
- O ator seleciona a perecibilidade.
- O sistema consulta as entradas em estoque com a perecibilidade selecionada.
- O sistema exibe o relatório com as entradas em estoque com a perecibilidade selecionada. Volta ao passo [P1].
- ** Fim do caso de uso **

2.1.6.14 – Gerar Relatório por Data de Validade

Descrição: Gera o relatório, agrupado por estoque, das entradas em estoque com a data de validade selecionada, informando o item, quantidade na unidade padrão do mesmo, data de compra, perecibilidade e estoque de cada entrada.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- [P1] O sistema solicita a data de validade pela qual gerar o relatório.
- O ator informa a data de validade.
- O sistema consulta as entradas em estoque com a data de validade informada.
- O sistema exibe o relatório com as estradas em estoque com a data de validade informada. Volta ao passo [P1].
- ** Fim do caso de uso **

2.1.6.15 – Gerar Relatório de Produtos Escassos

Descrição: Gera o relatório, agrupado por itens, dos itens com quantidade em estoque inferior à quantidade mínima cadastrada para o tem, informando o estoque, quantidade, datas de compra e validade, preço e perecibilidade de cada entrada do item em estoque, assim como descrição, quantidade mínima na unidade padrão e quantidade total em estoque do item.

Ator: Usuário com acesso ao menu "*Relatórios*".

Curso Típico

- O sistema consulta os itens que possuem quantidade em estoque inferior à quantidade mínima cadastrada.
- O sistema exibe o relatório com os itens com quantidade em estoque inferior à quantidade mínima cadastrada.
- ** Fim do caso de uso **

2.1.6.16 – Gerar Relatório de Previsão de Gastos

Descrição: Gera o relatório de previsão de gastos a partir de uma lista de itens selecionados pelo ator, informando descrição, data de última compra e previsão

de preço, assim como quantidade, preço de compra e data de compra das três últimas entradas de estoque de cada item.

Ator: Usuário com acesso ao menu "*Relatórios*.

Curso Típico

- [P1] O sistema pergunta ao ator se ele deseja selecionar os itens para compor uma lista de compras ou gerar o relatório a partir da lista de produtos escassos.
- O ator seleciona gerar o relatório a partir da lista de produtos escassos [A1].
- O sistema consulta os itens escassos na base de dados.
- O sistema calcula as estimativas de preço, assim como o total da compra.
- O sistema exibe o relatório com a lista dos itens e suas previsões de preço. Volta ao passo [P1].
- ** Fim do caso de uso **

Curso Alternativo [A1] – O ator seleciona gerar uma lista de compras

- O ator seleciona gerar uma lista de compras.
- [P2] O sistema apresenta os itens cadastrados.
- O ator seleciona um item para incluir na lista.
- O sistema solicita a quantidade do item na unidade padrão do mesmo.
- O ator informa a quantidade a ser incluída na lista.
- O sistema inclui o item com a quantidade informada na lista de compras.
- O sistema pergunta se o ator que incluir mais itens ou fechar a lista.
- O ator seleciona a opção de fechar a lista [A2].
- O sistema consulta os itens selecionados na base de dados.
- O sistema calcula as estimativas de preço, assim como o total da compra.
- O sistema exibe o relatório com a lista dos itens e suas previsões de preço. Volta ao passo [P1].

Curso Alternativo [A2] – O ator seleciona a opção de incluir novo item na lista

- O ator seleciona a opção de incluir novo item na lista. Volta ao passo [P2].

2.2 – A solução

A solução proposta por esse projeto é uma ferramenta desenvolvida com tecnologia Java EE 6 através da IDE NetBeans 6.5, funcionando instalada em máquina local, utilizando um banco de dados PostgreSQL 8.3, sendo acessado pelo sistema através da API de persistência (JPA) nativa da plataforma. O sistema irá funcionar preferencialmente na mesma máquina local com sistema operacional Windows XP Professional.

Essa solução visa implementar a totalidade das funcionalidades definidas durante a fase de levantamento dos requisitos do sistema. Assim sendo, serão contempladas as funcionalidades de cadastro, de entrada e saída dos itens de estoque e os relatórios de acompanhamento.

Dessa forma, o sistema irá então permitir um controle de entrada e saída dos itens domésticos de forma ágil para permitir o acompanhamento do estoque e eventual previsão dos gastos de reposição necessários.

2.3 – O método

A aplicação foi desenvolvida com arquitetura em camadas, com paradigmas de orientação a objetos, seguindo fundamentos da arquitetura MVC, onde foram implementadas classes de formulário que fazem o papel de visão e de controlador, classes do modelo e classes de persistência, criadas para o acesso ao banco de dados.

Dessa forma então, a classe de formulário, que define a interface com o usuário, utiliza instâncias das classes do modelo para executar as funcionalidades de cada entidade relacionada com o formulário. As classes do modelo por sua vez, utilizam uma classe de gerência de entidades do banco de dados (*Entity Manager*), sendo usado nesse caso o padrão de projeto *Factory Method* para instanciá-la, padrão esse utilizado pela JPA do NetBeans.

Através dessa classe de gerência de entidades, as classes do modelo utilizam então as classes de persistência para realizarem acessos ao banco de dados, seja para consultas através de *queries* pré-definidas ou personalizadas, seja para realizar a persistência dos dados no banco.

Os formulários desenvolvidos para as funcionalidades de cadastro seguem um padrão em abas, de forma a destacar as operações de inclusão, alteração ou exclusão e facilitar a atuação usuário na execução dessas funcionalidades.

2.4 – Ciclo de vida

Para refinar os requisitos do sistema, o projeto se iniciará com uma etapa de levantamento, de forma a reavaliar os requisitos funcionais e complementares podendo então serem realizados ajustes nas funcionalidades a serem implementadas ou melhor especificação dos requisitos definidos.

Após essa primeira fase, o projeto passará à fase de modelagem que, baseada nos requisitos levantados, definirá os modelos atinentes ao projeto conceitual do sistema. Na terceira fase será então desenvolvido o modelo de dados a ser utilizado para a implementação do sistema.

A fase seguinte será a de modelagem para a implementação, expandindo o modelo conceitual e funcional do sistema para contemplar todos os componentes auxiliares da arquitetura de implementação do sistema.

A quinta fase consistirá na implementação de fato do sistema, com a criação das interfaces e codificação das funcionalidades definidas na primeira fase e modeladas nas fase seguintes. Finalmente, a última fase consistirá na documentação final do projeto, sendo realizadas as considerações finais pertinentes.

2.4.1 - Cronograma

Fase 1: Levantamento dos requisitos do sistema.
Fase 2: Modelagem do sistema a partir dos requisitos definidos na fase 1.
Fase 3: Modelagem do banco de dados do sistema.
Fase 4: Criação do modelo de implementação.
Fase 5: Implementação do modelo definido na fase 2.
Fase 6: Documentação geral do projeto.

Tabela 1 – Cronograma de Atividades

Semana:	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Mês:	maio				junho				julho				agosto				setembro				outubro			
Fase 1	>	>																						
Fase 2			>	>	>	>																		
Fase 3							>	>																
Fase 4									>	>														
Fase 5											>	>	>	>	>	>	>	>	>	>	>	>		
Fase 6																							>	>

Capítulo 3

Análise

3.1 – Modelo Conceitual

Considerando-se os requisitos funcionais do sistema, onde temos o cadastro de itens, entrada e saída de estoque, temos então a definição das duas classes principais do sistema, *Item* e *Estoque*. A funcionalidade primordial do sistema se baseia na interação entre essas duas entidades e, por esse motivo, temos então o principal relacionamento também definido como uma classe, *ItemEstoque*.

Para definirmos então um item, são necessárias algumas informações como parâmetros. Dessa forma, temos que a entidade Item deverá possuir como atributos um identificador, que será gerado automaticamente pelo banco de dados de forma incremental; uma descrição para fácil identificação pela interface; a data em que o item teve a última entrada em estoque; e a quantidade mínima do item que deverá estar em estoque.

Ao falarmos da quantidade mínima do item, ou da quantidade do item em estoque, surge a necessidade de definirmos em qual unidade de medida tais quantidades serão referenciadas. Assim, define-se uma outra classe, diretamente relacionada ao item, a *Unidade*. Essa nova entidade possuirá como atributos um identificador controlado pelo banco de dados (como no caso do item), uma descrição que define a unidade e uma sigla para facilitar a visualização na interface com o usuário.

O estoque então terá como atributos apenas um identificador gerado pelo banco de dados e uma descrição para facilitar a identificação pelo usuário. Quando associamos um item a um estoque temos um item de estoque (classe *ItemEstoque* citada acima), que terá como atributos, além do item e do estoque propriamente ditos, um identificador próprio, cuja função principal é identificar diferentes partidas de um mesmo item em um mesmo estoque (por exemplo, duas entradas no estoque “geladeira” do item “carne” que foram compradas em datas distintas e possuem datas de validade também distintas); a quantidade do item em sua unidade padrão no estoque (quantidade especificada para cada partida entrada no estoque); o preço de compra por unidade padrão do item; a data

da compra do item; e a data de validade da partida do item, no caso de partidas que sejam perecíveis e de fato possuam validade.

Como tratamos da validade dos itens de estoque, considerando que apenas alguns itens de fato possuirão essa data definida, definimos então uma classe relacionada com a classe ItemEstoque para facilitar a classificação das partidas quanto a sua validade, a classe *Perecibilidade*, por padrão definida simplesmente como "perecível" ou "não perecível".

Para a implementação do cadastro de receitas, visando a funcionalidade para realizar saídas pré-programadas de estoque de vários itens, definimos então uma nova classe, *Receita*, que possuirá como atributos apenas um identificador gerado pelo banco de dados, um título para facilitar a identificação da receita pelo usuário e uma descrição explicativa da execução da receita.

Assim como no relacionamento entre Item e Estoque criamos a classe para implementá-lo, no caso das receitas, criamos uma classe para relacionar itens com a receita, implementando assim a lista de itens que fazem parte da receita. A classe *ItemReceita* então criada terá como atributos, além do identificador da receita e do item relacionado, um identificador próprio gerado pelo banco de dados e a quantidade do item em sua unidade padrão utilizado na receita.

Finalmente, temos a funcionalidade de cadastro de usuários. A classe *Usuário* então é criada para o devido registro dos usuários autorizados a utilizar o sistema. Essa classe terá como atributos um identificador gerado pelo banco de dados, o nome do usuário, o login a ser utilizado pelo mesmo e sua senha para autenticação.

De forma a flexibilizar a definição de quais funcionalidades cada usuário registrado poderá acessar, definimos a classe *Perfil*, que definirá quais os menus do sistema estarão disponíveis para o usuário que possuir dado perfil. A classe perfil terá então como atributos um identificador gerado pelo banco de dados, e uma série de atributos booleanos que definem quais menus o perfil disponibilizará ao usuário. Com a definição do perfil, voltando à classe *Usuário*, acrescentamos o atributo perfil ao conjunto de atributos já definidos para o usuário.

3.2 – Diagrama de Classes

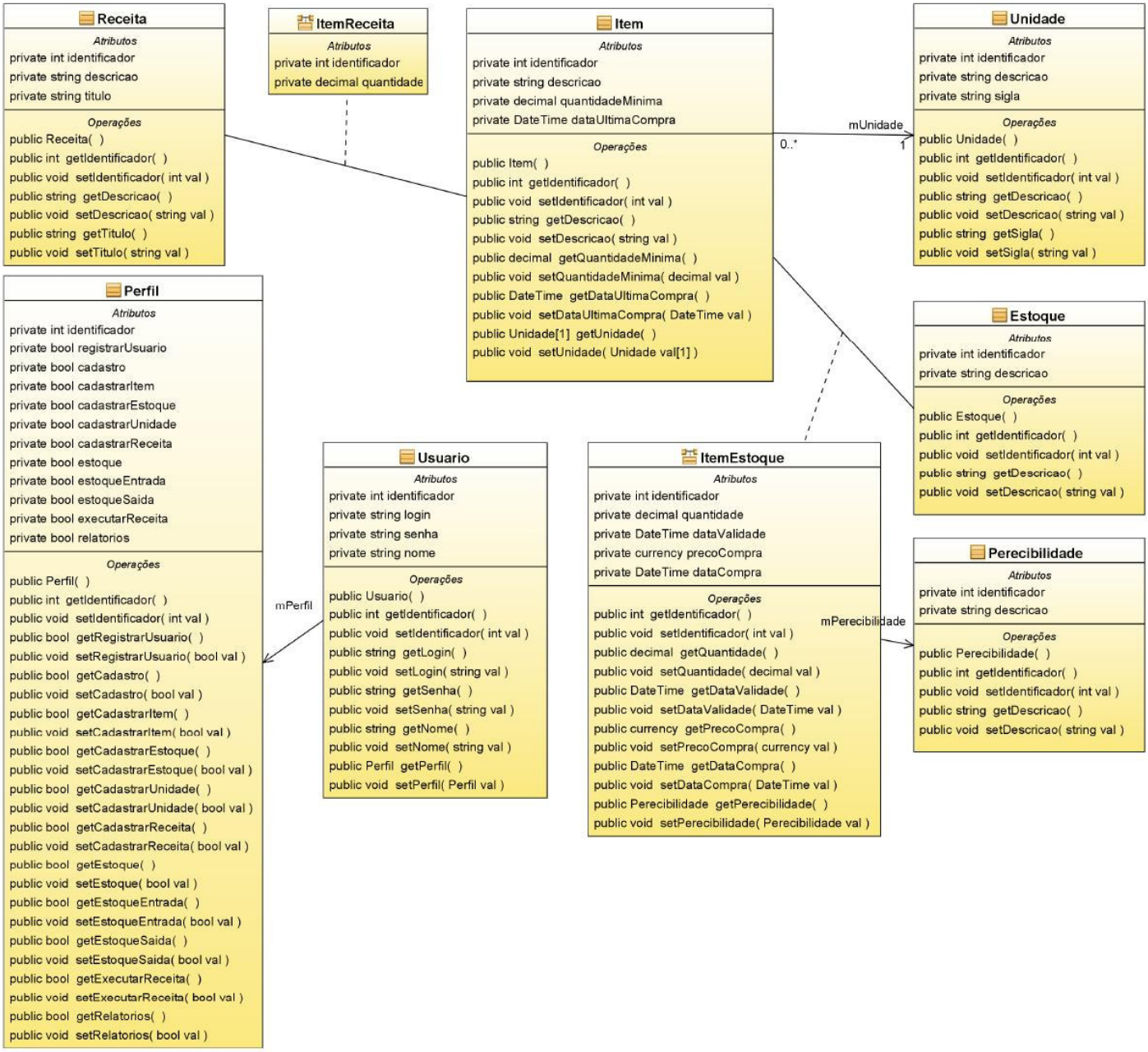

Figura 3 – Diagrama de classes

Capítulo 4

Desenho

4.1 – Modelo de implementação

A implementação do projeto em uma arquitetura de camadas exige a definição de outro conjunto de classes além daquele definido no modelo conceitual do sistema. Esse conjunto de classes implementará a camada *Menu*, que será responsável pela interface com o usuário, efetivamente representando as camadas *Visão* e *Controle* do modelo MVC.

As classes da camada *Menu* representam cada uma um caso de uso específico, definindo o desenho da interface com o usuário e os controles e validações de cada funcionalidade do sistema.

Temos assim a equivalência direta entre os casos de uso com as classes como a seguir:

Caso de uso	**Classe da camada *Menu***
Registrar Usuário	RegistrarUsuario
Cadastrar Unidade de Medida	CadastrarUnidade
Cadastrar Item	CadastrarItem
Cadastrar Receita	CadastrarReceita
Cadastrar Estoque	CadastrarEstoque
Entrada de Estoque	EntradaEstoque
Saída de Estoque	SaidaEstoque
Executar Receita	ExecutarReceita
Gerar Relatório por Estoque	RelatorioEstoque
Gerar Relatório por Quantidade	RelatorioQuantidade
Gerar Relatório por Produto	RelatorioProduto
Gerar Relatório por Data de Compra	RelatorioDataCompra
Gerar Relatório por Perecibilidade	RelatorioPerecibilidade
Gerar Relatório por Data de Validade	RelatorioDataValidade
Gerar Relatório de Produtos Escassos	RelatorioProdutosEscassos
Gerar Relatório de Previsão de Gastos	RelatorioPrevisaoGastos

Temos ainda a definição de um terceiro grupo de classes, além do Modelo e do Menu, que são as classes de persistência criadas pela JPA para acesso ao banco de dados. Essa camada de *Dados* é diretamente equivalente à camada *Modelo* (ver item 4.5, Modelo de dados), possuindo uma classe de persistência para cada classe do modelo conceitual.

4.2 – Diagrama de componentes

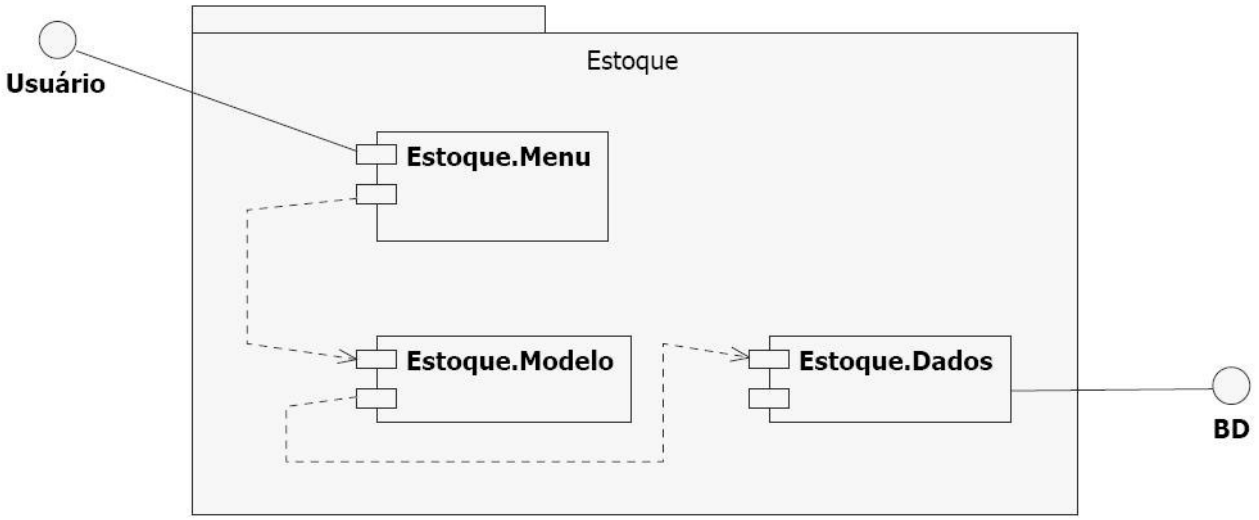

Figura 4 – Diagrama de componentes

4.3 – Diagrama de classes de implementação

Podemos então expandir o diagrama de classes do modelo conceitual, mostrando assim a interação da camada *Menu* com a camada *Modelo*.

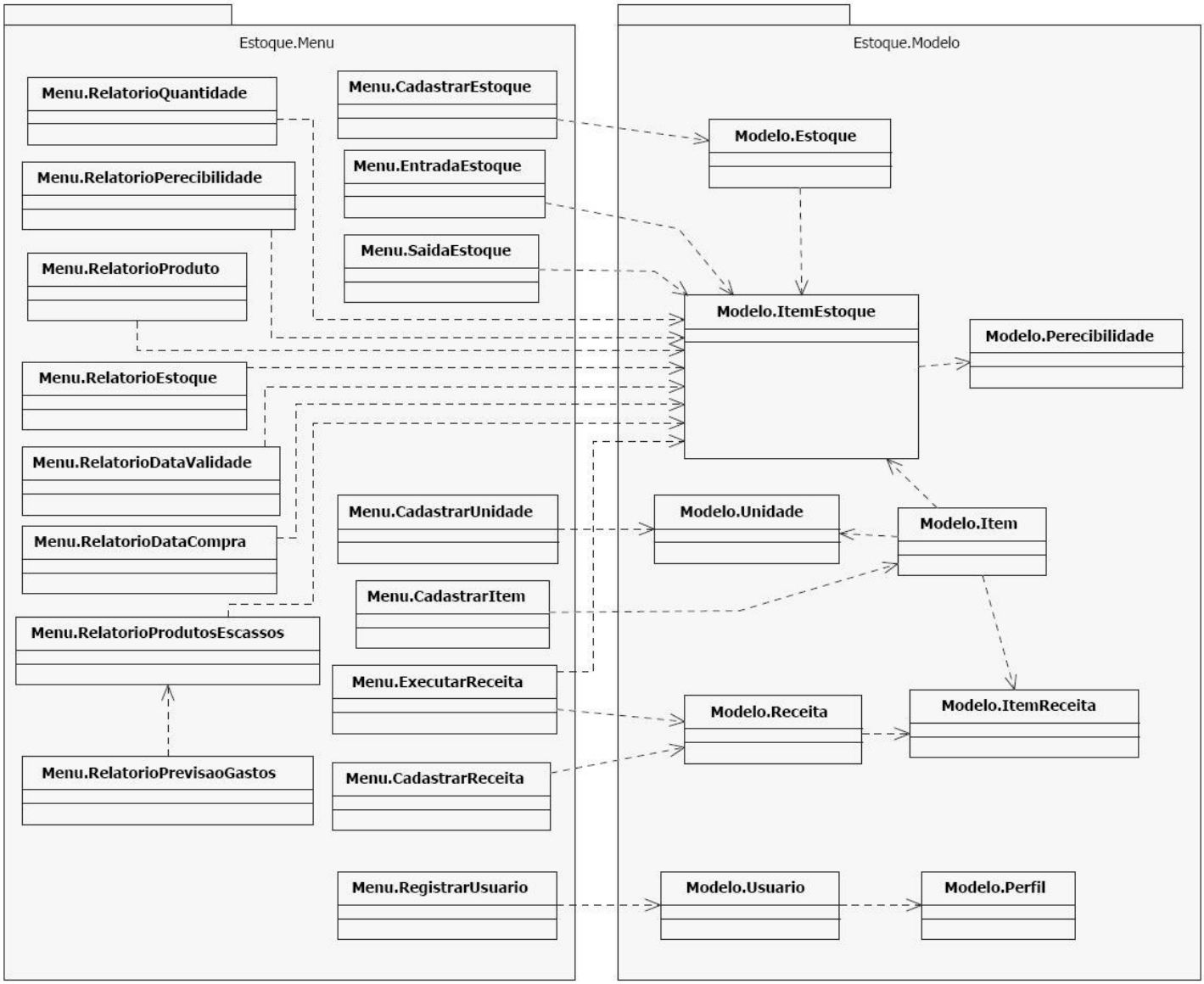

Figura 5 – Diagrama de classes de implementação

4.4 – Diagramas de sequência

Considerando a definição inicial do projeto das funcionalidades principais baseadas nos requisitos de negócio do sistema, podemos destacar os seguintes diagramas de sequência que descrevem tais funcionalidades.

São então apresentados nessa seção os diagramas que descrevem o funcionamento dos casos de uso Cadastrar Item (cursos típico, alternativo A1 e alternativo A2), Entrada de Estoque, Saída de Estoque, Gerar Relatório de Produtos Escassos e Gerar Relatório de Previsão de Gastos (os dois últimos sendo considerados os casos de uso relacionados com os relatórios mais importantes para o atendimento dos requisitos funcionais do sistema).

4.4.1 – Cadastrar Item (curso típico – incluir novo item)

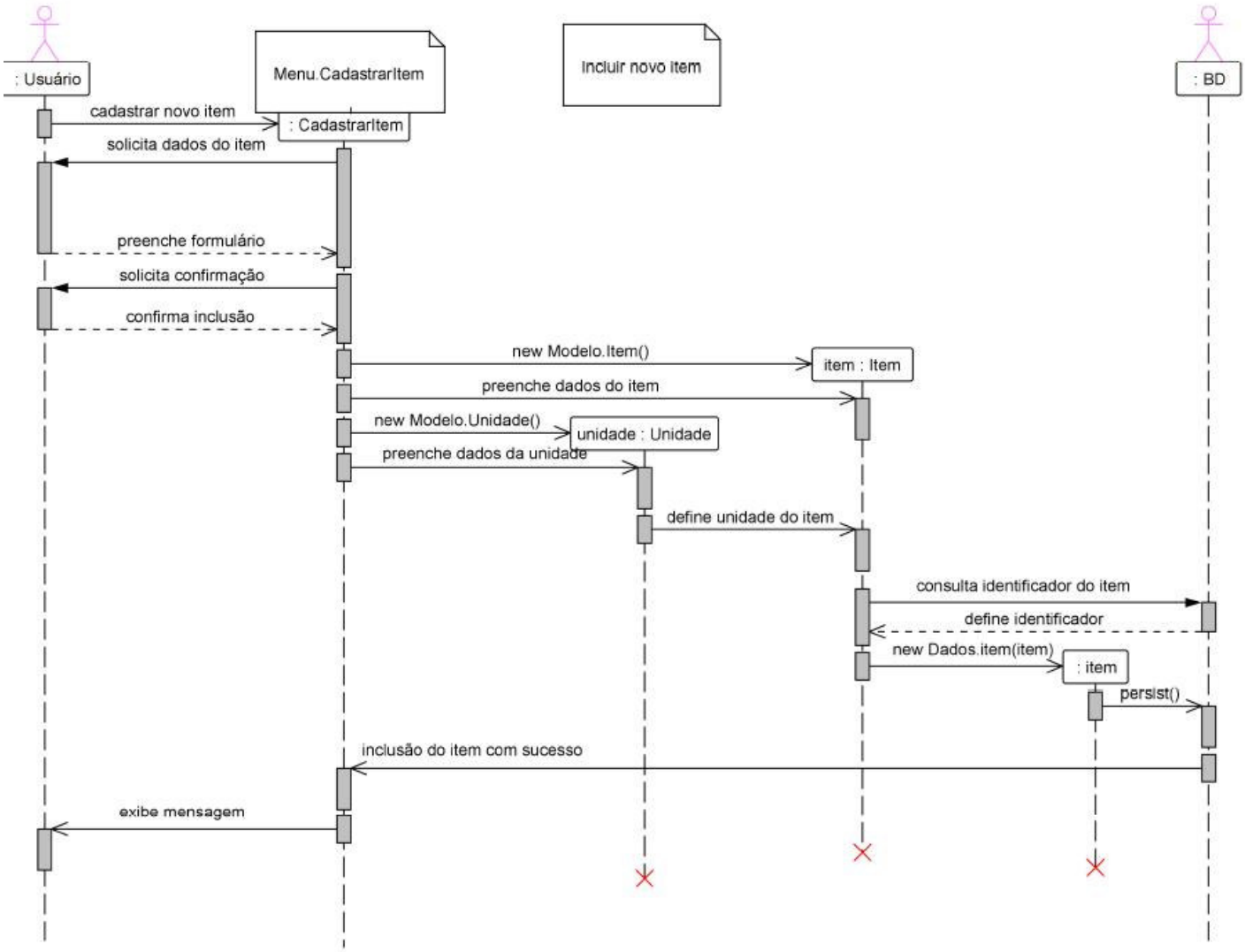

Figura 6 – Diagrama de sequência – Incluir novo item

4.4.2 – Cadastrar Item (curso alternativo A1 – alterar item)

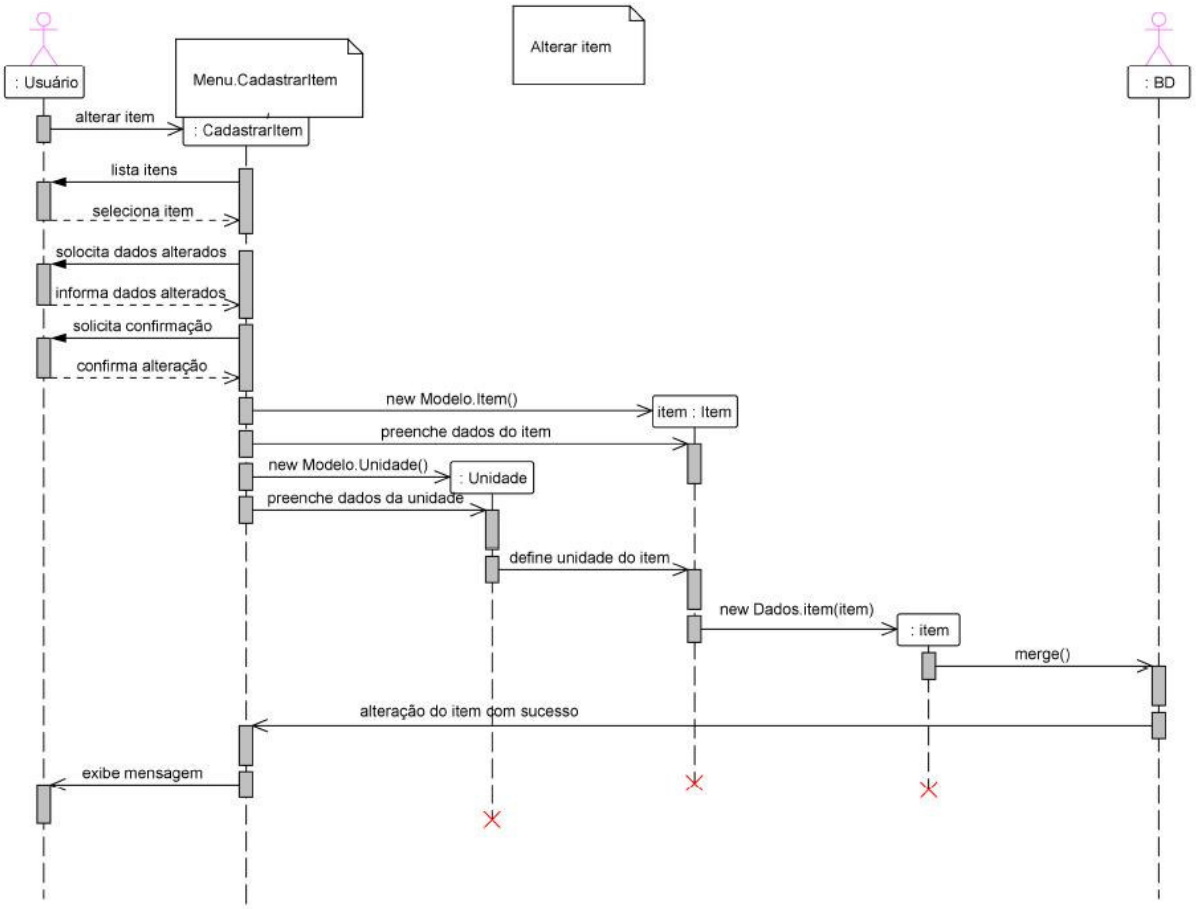

Figura 7 – Diagrama de sequência – Alterar item

4.4.3 – Cadastrar Item (curso alternativo A2 – excluir item)

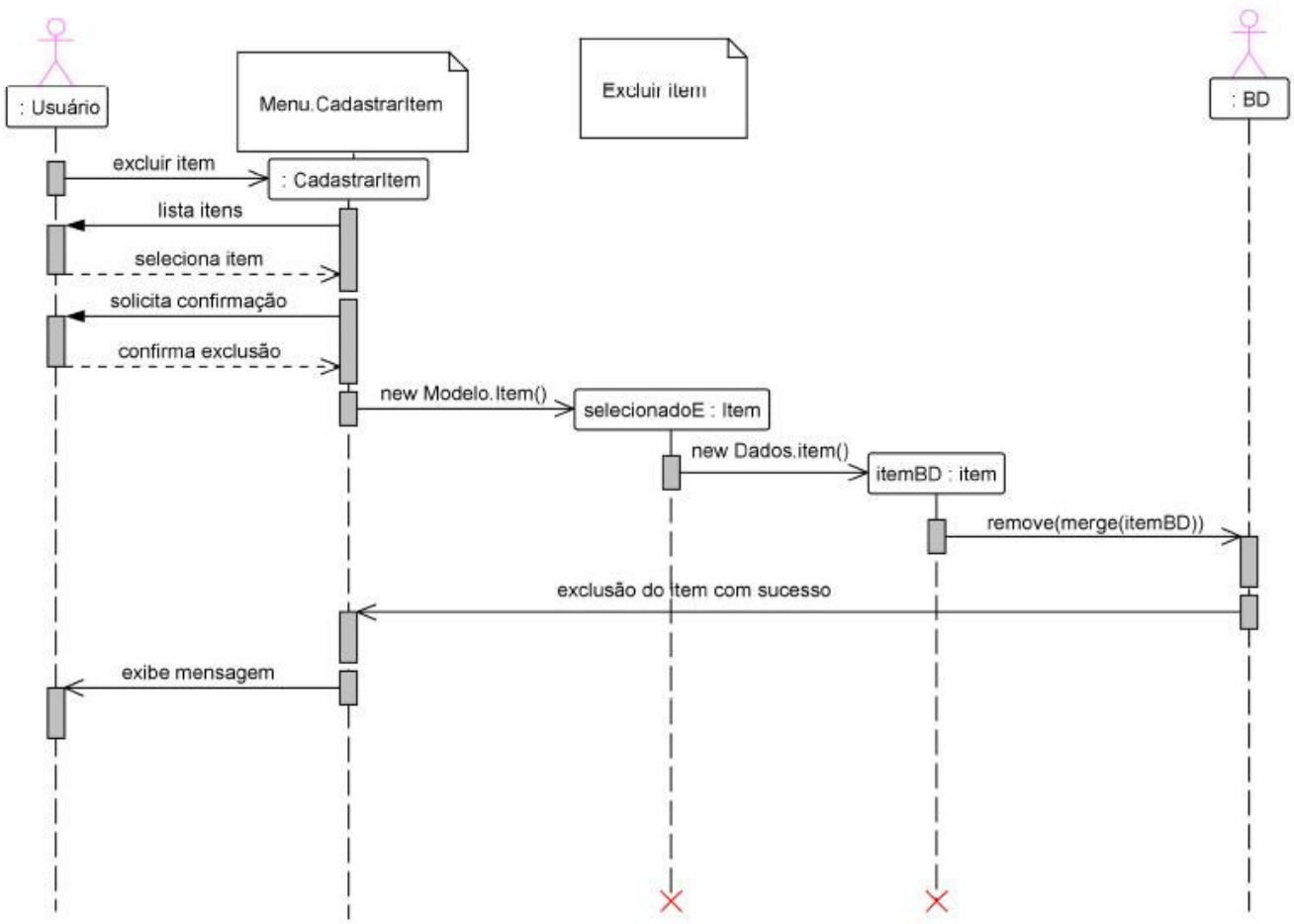

Figura 8 – Diagrama de sequência – Excluir item

4.4.4 – Entrada de Estoque (curso típico)

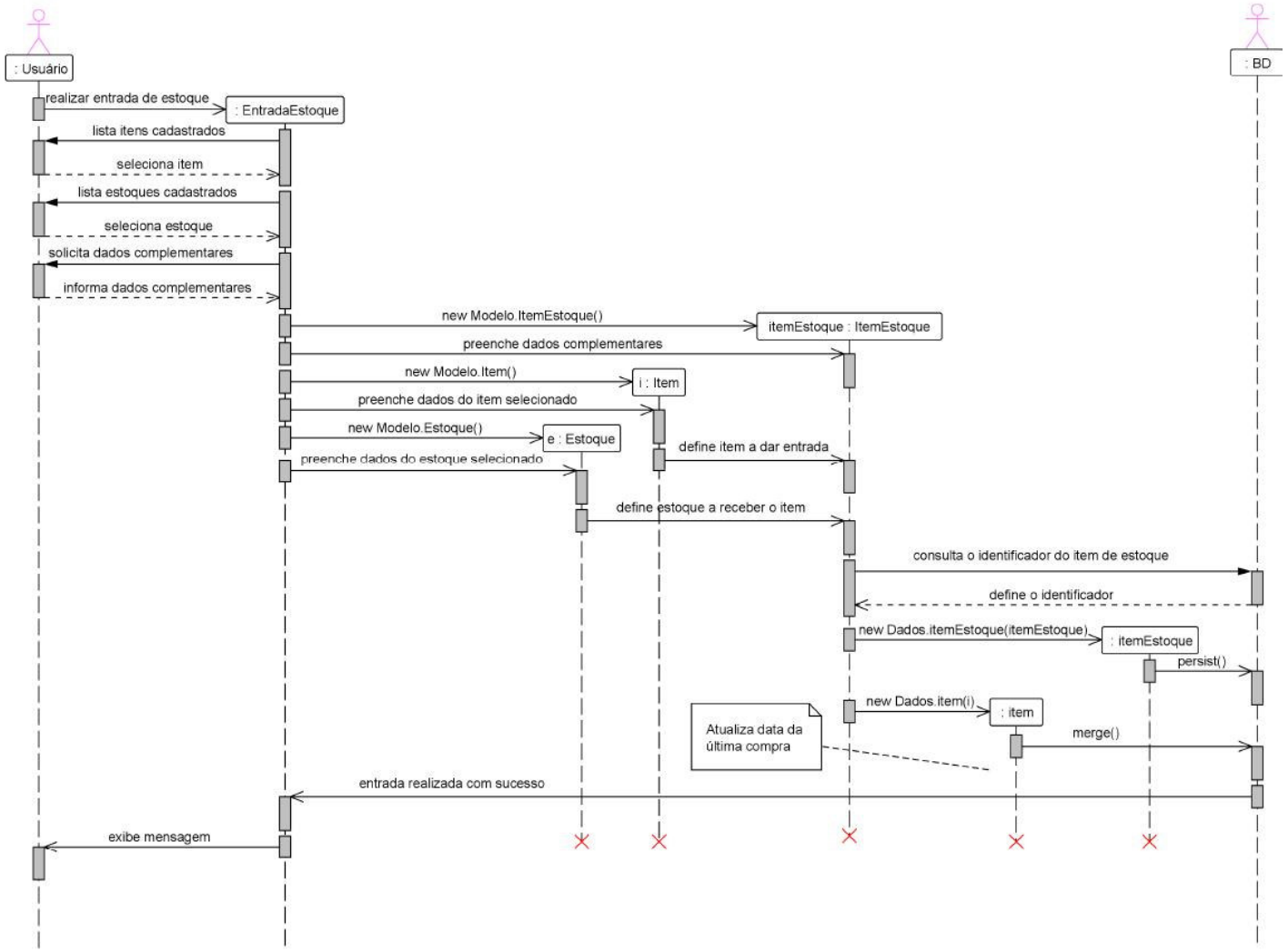

Figura 9 – Diagrama de sequência – Entrada de estoque

4.4.5 – Saída de Estoque (curso típico)

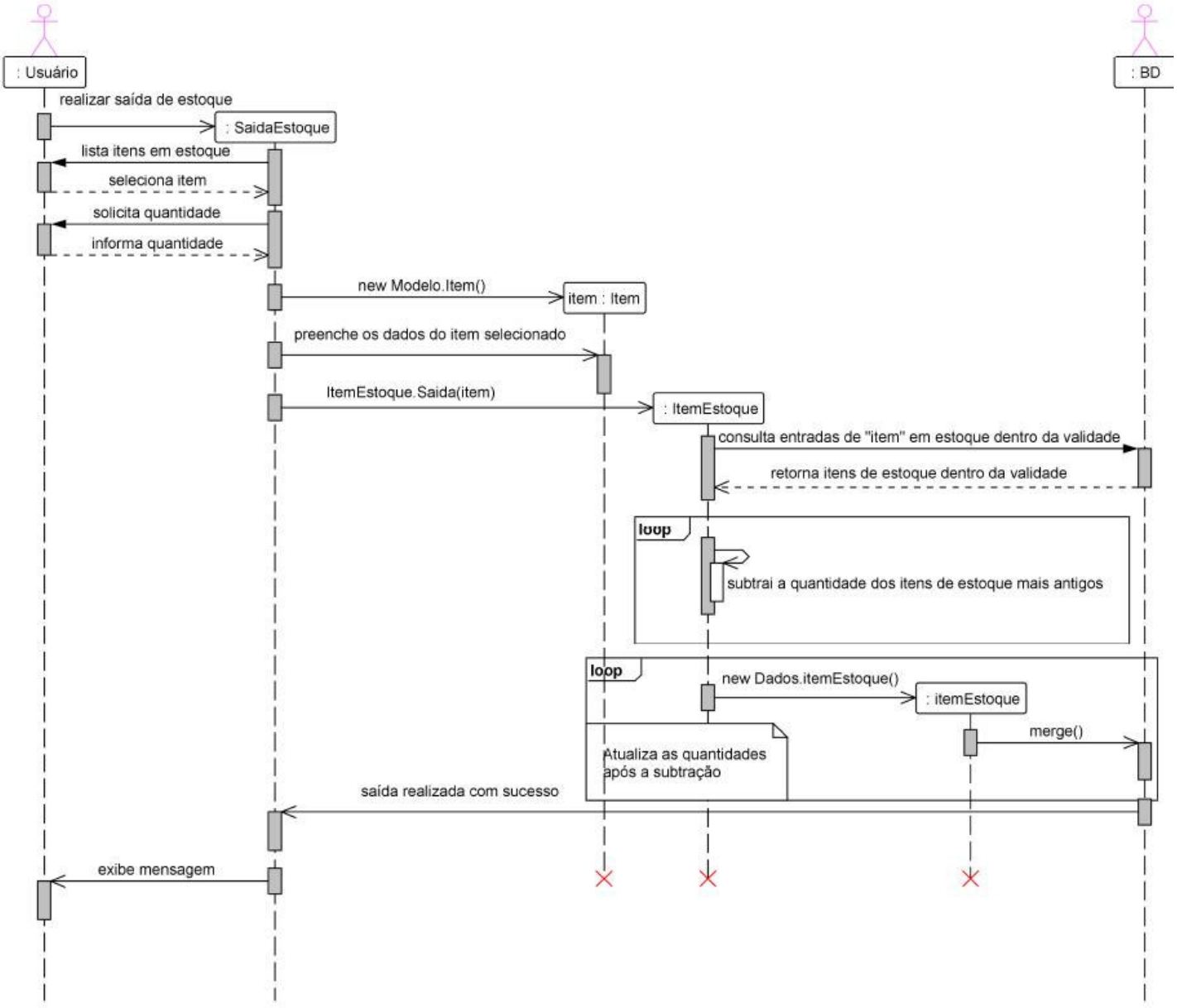

Figura 10 – Diagrama de sequência – Saída de estoque

4.4.6 – Gerar Relatório de Produtos Escassos (curso típico)

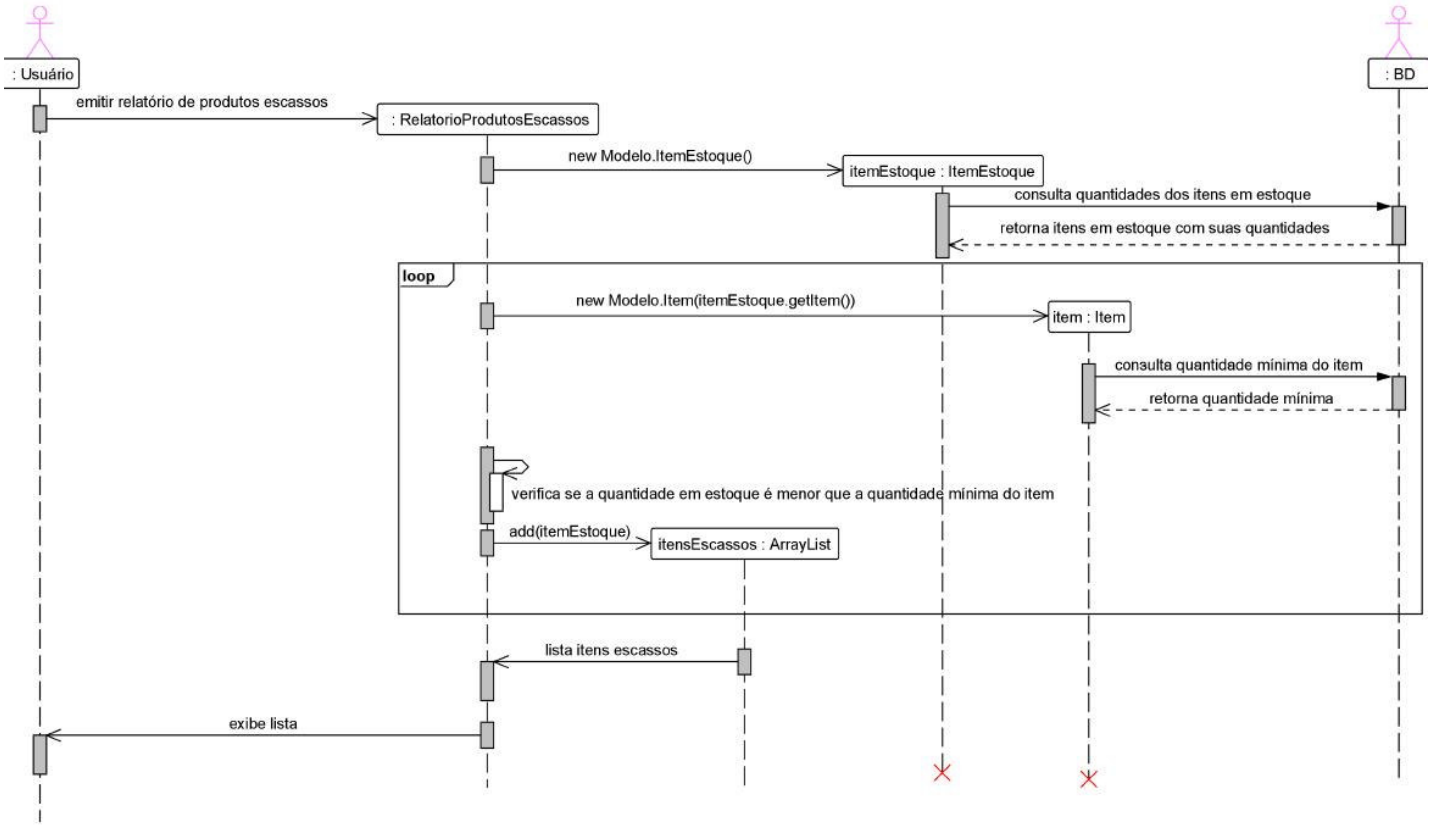

Figura 11 – Diagrama de sequência – Gerar relatório de produtos escassos

4.4.7 – Gerar Relatório de Previsão de Gastos (curso típico – produtos escassos)

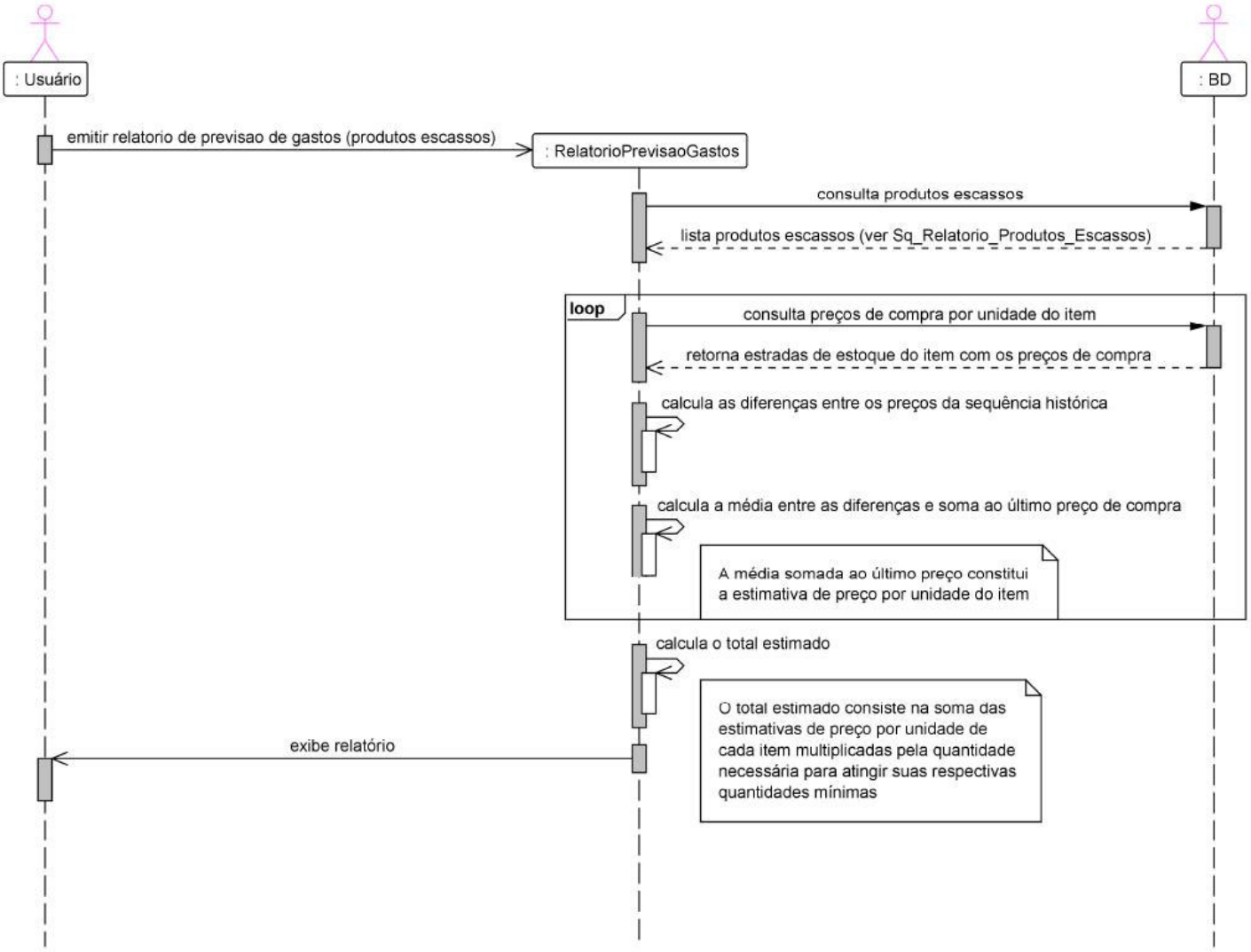

Figura 12 – Diagrama de sequência – Gerar relatório de previsão de gastos

4.5 – Modelo de dados

Analisando novamente o modelo de classes definido no capítulo 3, podemos perceber que as entidades de dados são diretamente representadas pelas classes do modelo. Dessa forma, o modelo de dados a ser implementado no banco utilizado pelo sistema será equivalente ao modelo de classes.

Temos assim as principais classes relacionadas ao controle de estoque, *Item*, *Estoque* e *Receita* que são representadas por tabelas de mesmo nome, item, estoque e receita. As classes secundárias do controle de estoque (*Unidade* e *Perecibilidade*) são também representadas por tabelas de mesmo nome (unidade e perecibilidade), cujos identificadores serão chaves estrangeiras das tabelas relacionadas (item e item_estoque, respectivamente). As classes de relacionamento (*ItemEstoque* e *ItemReceita*) são representadas por tabelas com nome análogo (item_estoque e item_receita), as quais possuem como chaves estrangeiras as chaves das tabelas às quais se relacionam, conforme apresentado no diagrama abaixo.

Finalmente, temos as classes do controle de usuários, *Usuário* e *Perfil*, que da mesma forma que as classes principais do controle de estoque são representadas por tabelas de mesmo nome, usuário e perfil. Entretanto, a tabela perfil, representando conceitualmente um atributo do usuário, tem seu identificador definido então como uma chave estrangeira da tabela usuário.

Considerando então o modelo de classes como o modelo lógico a ser seguido na implementação do banco de dados, temos modelo físico apresentado na figura 13 abaixo.

No apêndice 1 está apresentado o dicionário de dados que descreve cada um dos atributos de cada classe do modelo. Como o modelo de classes é diretamente equivalente ao modelo de dados, cada atributo descrito no dicionário apresentado é correspondente então a um campo nas tabelas equivalentes às classes.

O diagrama do modelo físico do banco de dados, no entanto, apresenta explicitamente as chaves estrangeiras de cada tabela, deixando mais claro o relacionamento entre as entidades.

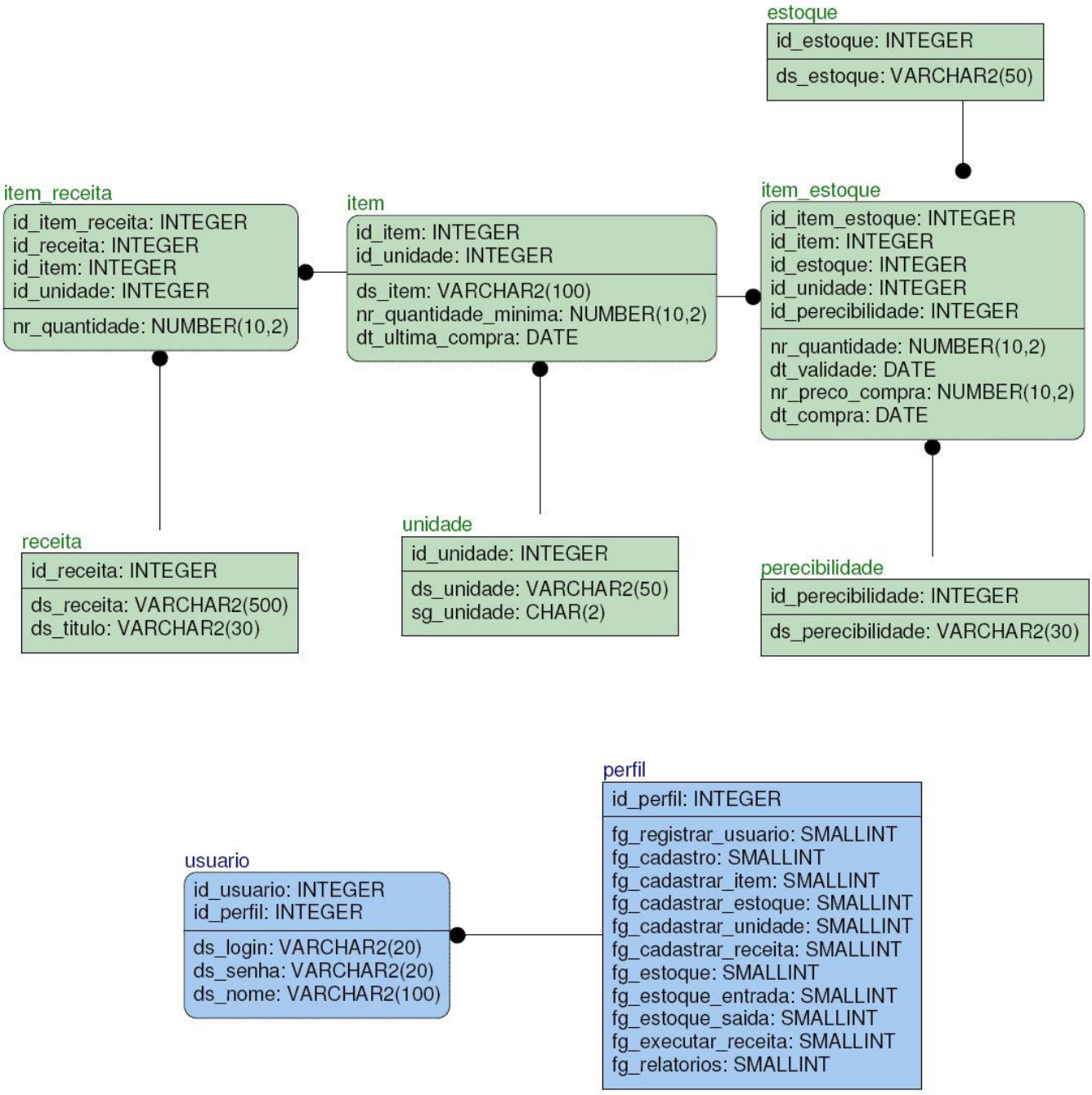

Figura 13 – Modelo físico de dados

Capítulo 5

Codificação, testes e resultados

5.1 – Codificação

Visto que o sistema foi modelado segundo uma arquitetura em camadas, podemos então explorar algumas das definições que foram utilizadas no desenvolvimento de cada uma dessas camadas, de forma a padronizar o código resultante do projeto.

A camada *Menu*, que implementa a visão e o controle do modelo MVC, é constituída dos formulários que implementam os casos de uso definidos para o sistema. Podemos então dividir as classes da camada *Menu* em três grupos, Cadastro, Entrada/Saída e Relatórios, de acordo com as macro-operações anteriormente definidas.

A visão implementada para os formulários de cadastro segue um modelo em abas, separando as funções de inclusão, alteração ou exclusão de registros. A aba de inclusão (a Figura 14 mostra um exemplo para o cadastro de itens) traz então todos os campos pertinentes de cada registro vazios para o devido preenchimento.

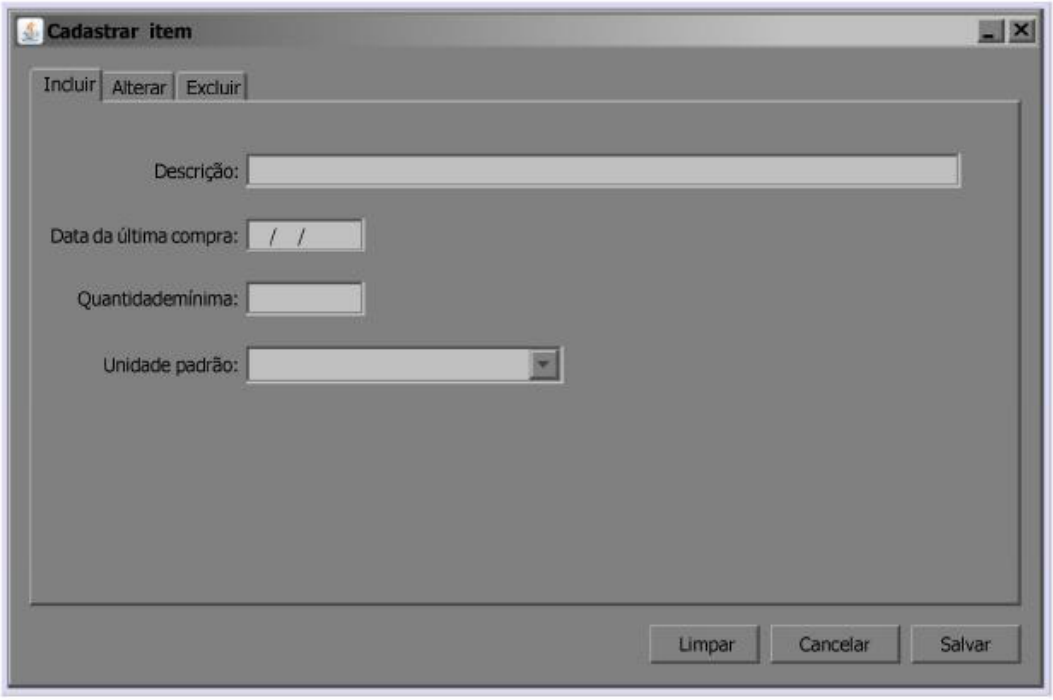

Figura 14 – Formulário de cadastro de itens, aba de inclusão

A aba de alteração, também dividida em duas abas (selecionar e alterar) traz primeiramente uma lista com os registros cadastrados para que seja selecionado o registro a ser alterado (aba selecionar, a Figura 15 mostra um exemplo para o cadastro de itens), sendo então exibido o formulário com os campos pertinentes preenchidos com os dados do registro selecionado para as devidas alterações.

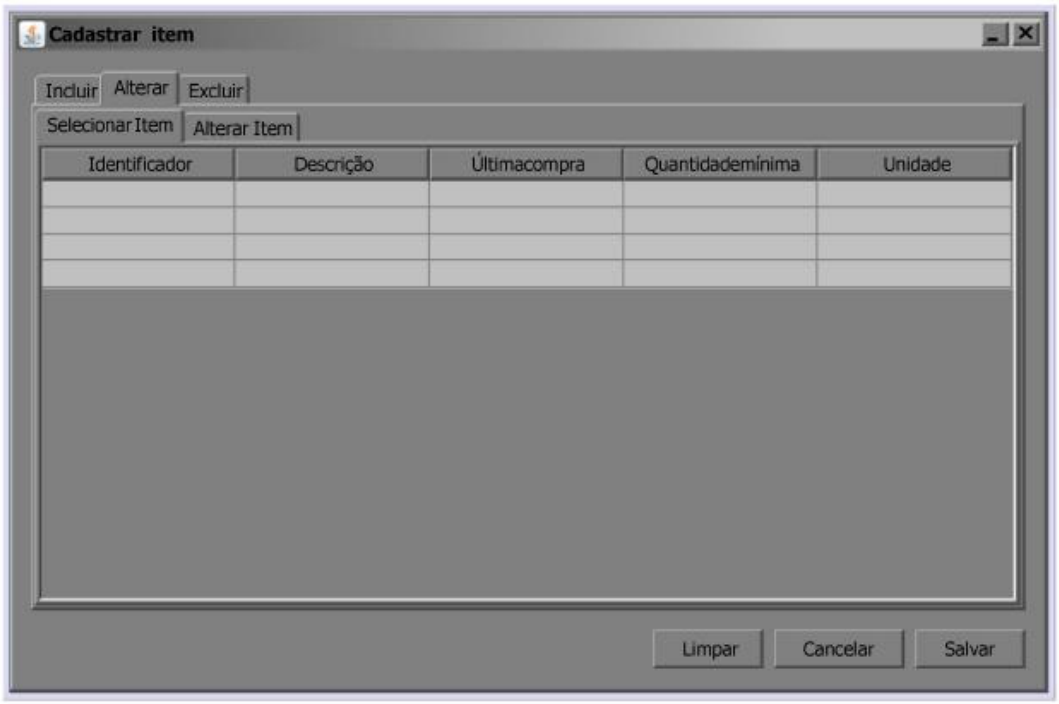

Figura 15 – Formulário de cadastro de itens, aba de alteração (selecionar)

A última aba (exclusão) apresenta simplesmente a lista dos registros cadastrados (como na aba selecionar da alteração, ver Figura 15) para que seja selecionado o registro que será excluído.

Finalmente, além dos formulários, temos também na visão do cadastro três botões, Limpar, Cancelar e Salvar. O botão limpar simplesmente apaga os dados preenchidos do formulário, permitindo ao usuário recomeçar o preenchimento dos dados. O botão cancelar apaga os dados preenchidos (como o botão limpar) e fecha a janela do formulário.

```
@Action
public void doLimpar() {
    ddlUnidade.setSelectedIndex(0);
    ddlUnidadeA.setSelectedIndex(0);
    tblItensA.clearSelection();
    tblItensE.clearSelection();
    txtDataCompra.setText("");
    txtDataCompraA.setText("");
    txtDescricao.setText("");
    txtDescricaoA.setText("");
    txtQuantidade.setText("");
    txtQuantidadeA.setText("");
    selecionadoA = null;
    selecionadoE = null;
}
```

Figura 16 – Código-fonte da ação executada pelo botão limpar (cadastro de itens)

```
@Action
public void doCancelar() {
    doLimpar();
    this.dispose();
}
```

Figura 17 – Código-fonte da ação executada pelo botão cancelar (cadastro de itens)

O botão salvar executa a inclusão ou alteração dos dados, dependendo exclusivamente de qual aba esteja sendo exibida. No caso da exclusão, a operação é executada através da simples seleção do registro, não utilizando o botão salvar para isso.

```
public void doSalvar() {
    //Salvar
    if(pnlIncluir.isShowing())
    {
        if(isValido())
        {
            Item item = new Item();
            item.setDescricao(txtDescricao.getText());
            if(txtQuantidade.getText().isEmpty()
                || Double.parseDouble(txtQuantidade.getText().replace(',', '.')) == 0)
            {
                item.setQuantidadeMinima(BigDecimal.ZERO);
            }
            else
                item.setQuantidadeMinima(new BigDecimal(txtQuantidade.getText().replace(",", ".")));

            int ano = Integer.parseInt(txtDataCompra.getText().substring(6,10)) - 1900;
            int mes = Integer.parseInt(txtDataCompra.getText().substring(3,5)) - 1;
            int dia = Integer.parseInt(txtDataCompra.getText().substring(0,2));
            item.setDataUltimaCompra(new Date(ano,mes,dia));

            confirma = javax.swing.JOptionPane.showConfirmDialog(null, "Tem certeza que deseja incluir o item " + txtDescricao.getText() + "?",
            if(confirma == 0)
            {
                try {
                    Unidade unidade = new Unidade();
                    String descricao = ddlUnidade.getSelectedItem().toString();
                    item.setUnidade(unidade.ConsultarDescricao(descricao));

                    item.Inserir();
                }catch (Exception e) {
                    JOptionPane.showMessageDialog(null, e.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
                    e.printStackTrace();
                }
                finally {
                    JOptionPane.showMessageDialog(null, "Item registrado com sucesso!", this.getTitle(), JOptionPane.INFORMATION_MESSAGE);
                    this.dispose();
                }
            }
        }
        else
            JOptionPane.showMessageDialog(null, erro, this.getTitle(), JOptionPane.ERROR_MESSAGE);
    }
```

Figura 18 – Código-fonte da ação executada pelo botão salvar, quando a aba de inclusão está visível (cadastro de itens)

```
if(pnlAlterar.isShowing())
{
    if(isValido())
    {
        Item item = new Item();
        item.setIdentificador(selecionadoA.getIdentificador());
        item.setDescricao(txtDescricaoA.getText());
        if(txtQuantidadeA.getText().isEmpty()
            || Double.parseDouble(txtQuantidadeA.getText().replace(',', '.')) == 0)
        {
            item.setQuantidadeMinima(BigDecimal.ZERO);
        }
        else
            item.setQuantidadeMinima(new BigDecimal(Double.parseDouble(txtQuantidadeA.getText().replace(',', '.'))));

        int ano = Integer.parseInt(txtDataCompraA.getText().substring(6));
        int mes = Integer.parseInt(txtDataCompraA.getText().substring(3,5));
        int dia = Integer.parseInt(txtDataCompraA.getText().substring(0,2));
        item.setDataUltimaCompra(new Date(ano - 1900,mes - 1,dia));

        confirma = JOptionPane.showConfirmDialog(null, "Tem certeza que deseja alterar o item " + txtDescricaoA.getText() + "?", this
        if(confirma == 0)
        {
            //SALVAR
            try
            {
                Unidade unidade = new Unidade();
                item.setUnidade(unidade.ConsultarDescricao(ddlUnidadeA.getSelectedItem().toString()));

                item.Alterar();

                JOptionPane.showMessageDialog(null, "Item alterado com sucesso!", this.getTitle(), JOptionPane.INFORMATION_MESSAGE);
                this.dispose();
            }catch (Exception e) {
                JOptionPane.showMessageDialog(null, e.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
                e.printStackTrace();
            }

        }
    }
    else
        JOptionPane.showMessageDialog(null, erro, this.getTitle(), JOptionPane.ERROR_MESSAGE);
}
```

Figura 19 – Código-fonte da ação executada pelo botão salvar, quando a aba de alteração está visível (cadastro de itens)

```
private void tblItensEMouseClicked(java.awt.event.MouseEvent evt) {
    int linha = tblItensE.getSelectedRow();
    int identificador = Integer.parseInt((String)tblItensE.getModel().getValueAt(linha, 0));
    selecionadoE = new Item();
    selecionadoE.setIdentificador(identificador);
    try
    {
        selecionadoE = selecionadoE.ConsultarIdentificador();
    } catch (Exception e) {
        e.printStackTrace();
        if(e instanceof javax.persistence.NoResultException)
            JOptionPane.showMessageDialog(null, "Item inexistente!", this.getTitle(), JOptionPane.ERROR_MESSAGE);
    }
    //CONFIRMAÇÃO DA EXCLUSÃO SEGUIDA DA EXCLUSÃO DO BD
    confirma = JOptionPane.showConfirmDialog(null, "Tem certeza que deseja excluir o item " + selecionadoE.getDescricao() + "?", this.getTitle
    if(confirma == 0)
    {
        //SALVAR
        try
        {
            selecionadoE.Excluir();
            JOptionPane.showMessageDialog(null, "Item excluído com sucesso!", this.getTitle(), JOptionPane.INFORMATION_MESSAGE);
        }catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
            e.printStackTrace();
        }
        finally {
            this.dispose();
        }
    }
}
```

Figura 20 – Código-fonte executado ao ser selecionado um registro na aba de exclusão

Ainda na camada *Menu*, são feitas validações do preenchimento dos campos dos formulários, principalmente relacionadas à obrigatoriedade de preenchimento e à duplicidade de campos identificadores.

```
private boolean isValido(){
    //VALIDAR DADOS A SEREM SALVOS
    try {
        ArrayList i = new Item().ConsultarDescricao(txtDescricao.getText());
        if(i.isEmpty())
            return true;
    } catch (Exception ex) {
        if(pnlIncluir.isShowing() && txtDescricao.getText().isEmpty())
        {
            erro = "A descrição do item deve ser informada.";
            return false;
        }
        else if(pnlAlterar.isShowing() && txtDescricaoA.getText().isEmpty())
        {
            erro = "A descrição do item deve ser informada.";
            return false;
        }
        else if(pnlAlterar.isShowing() && (txtQuantidadeA.getText().isEmpty()
                || Double.parseDouble(txtQuantidadeA.getText().replace(',', '.')) == 0))
        {
            erro = "A quantidade mínima do item deve ser informada.";
            return false;
        }
        else if(pnlIncluir.isShowing() && txtDataCompra.getText().replace("/", "").trim().isEmpty())
        {
            erro = "A data de última compra do item deve ser informada.";
            return false;
        }
        else if(pnlAlterar.isShowing() && txtDataCompraA.getText().replace("/", "").trim().isEmpty())
        {
            erro = "A data de última compra do item deve ser informada.";
            return false;
        }
        else return true;
    }
    erro = "Já existe um item com a descrição informada. Não podem haver dois itens com a mesma descrição.";
    return false;
}
```

Figura 21 – Código-fonte da validação (cadastro de itens) de preenchimento e duplicidade (descrição)

A visão implementada para os formulários de entrada e saída seguem um padrão mais simples, apresentando apenas os campos a serem preenchidos necessários para a execução das funcionalidades. Ao contrário das funcionalidades de cadastro, os formulários de entrada e saída exercem função única (sem a separação entre inserção, alteração ou exclusão), não existindo assim a necessidade para o modelo em abas.

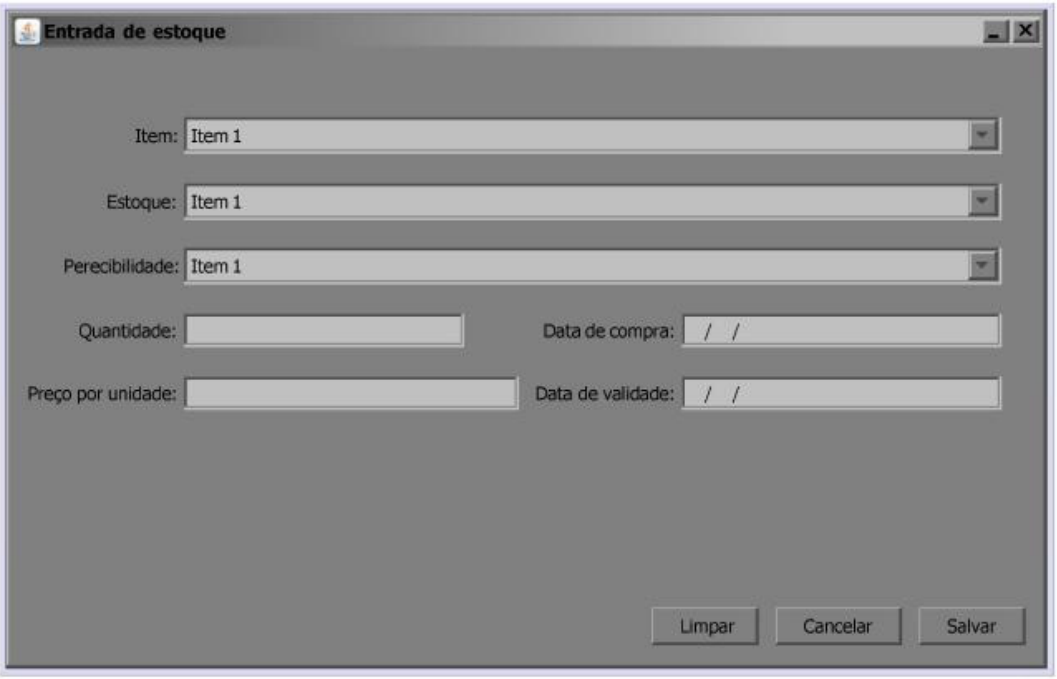

Figura 22 – Formulário de entrada de itens em estoque

No entanto, os formulários de entrada e saída possuem também os botões padrão existentes nos formulários de cadastro, com funções análogas às já explicadas, botão Limpar para apagar os dados preenchidos, botão Cancelar para apagar os dados e fechar a janela e botão Salvar para gravar a operação no banco de dados.

```
@Action
public void doLimpar() {
    ddlEstoque.setSelectedIndex(0);
    ddlItem.setSelectedIndex(0);
    ddlPerecibilidade.setSelectedIndex(0);
    txtDataCompra.setText("");
    txtDataValidade.setText("");
    txtPreco.setText("");
    txtQuantidade.setText("");
}
```

Figura 23 – Código-fonte da ação executada pelo botão limpar (entrada de estoque)

Como nos formulários de entrada e saída não existem as abas constantes nos formulários de cadastro, a ação executada pelo botão salvar é única, consistindo na validação de preenchimento e na persistência da operação no banco de dados.

```
@Action
public void doSalvar() {
    //SALVAR ENTRADA
    ItemEstoque itemEstoque = new ItemEstoque();
    Item i = new Item();
    Estoque e = new Estoque();
    Perecibilidade p = new Perecibilidade();
    try {
        if(isValido())
        {
            String dit = ddlItem.getSelectedItem().toString();
            i = (Item)i.ConsultarDescricao(dit).get(0);
            itemEstoque.setItem(new item(i));
            String des = ddlEstoque.getSelectedItem().toString();
            itemEstoque.setEstoque(new estoque((Estoque)e.ConsultarDescricao(des).get(0)));
            String dpe = ddlPerecibilidade.getSelectedItem().toString();
            itemEstoque.setPerecibilidade(new perecibilidade((Perecibilidade)p.ConsultarDescricao(dpe).get(0)));
            itemEstoque.setQuantidade(new BigDecimal(txtQuantidade.getText().replace(",", ".")));
            itemEstoque.setPreco(new BigDecimal(txtPreco.getText().replace(",", ".")));

            int anoC = Integer.parseInt(txtDataCompra.getText().substring(6,10)) - 1900;
            int mesC = Integer.parseInt(txtDataCompra.getText().substring(3,5)) - 1;
            int diaC = Integer.parseInt(txtDataCompra.getText().substring(0,2));
            itemEstoque.setDataCompra(new Date(anoC,mesC,diaC));
            i.setDataUltimaCompra(new Date(anoC,mesC,diaC));

            if(!txtDataValidade.getText().replace("/", "").trim().isEmpty())
            {
                int anoV = Integer.parseInt(txtDataValidade.getText().substring(6,10)) - 1900;
                int mesV = Integer.parseInt(txtDataValidade.getText().substring(3,5)) - 1;
                int diaV = Integer.parseInt(txtDataValidade.getText().substring(0,2));
                itemEstoque.setValidade(new Date(anoV,mesV,diaV));
            }
            //VALIDA ITEM-ESTOQUE (POR DATA DE COMPRA)\\
            if(itemEstoque.ConsultarDuplicados(itemEstoque.getItem().getIdItem(), itemEstoque.getEstoque().getIdEstoque(), itemEstoque.getDataComp
            {
                //INCLUIR
                confirma = javax.swing.JOptionPane.showConfirmDialog(null, "Tem certeza que deseja dar entrada do item " + dit + " no estoque " +
                if(confirma == 0)
                {
                    itemEstoque.Inserir();
                    i.Alterar();
                    JOptionPane.showMessageDialog(null, "Entrada realizada com sucesso!", this.getTitle(), JOptionPane.INFORMATION_MESSAGE);
                    doLimpar();
                }
            }
            else
                JOptionPane.showMessageDialog(null, "Já existe uma entrada do item " + itemEstoque.getItem().getDsItem() + " no estoque " + itemE
        }
        else
            JOptionPane.showMessageDialog(null, erro, this.getTitle(), JOptionPane.ERROR_MESSAGE);
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
        ex.printStackTrace();
    }
}
```

Figura 24 - Código-fonte da ação executada pelo botão salvar (entrada de estoque)

```
private boolean isValido() {
    if(ddlEstoque.getSelectedIndex() == 0)
    {
        erro = "O estoque de entrada deve ser informado.";
        return false;
    }
    else if(ddlItem.getSelectedIndex() == 0)
    {
        erro = "O item de entrada deve ser informado.";
        return false;
    }
    else if(ddlPerecibilidade.getSelectedIndex() == 0)
    {
        erro = "A perecibilidade do item de entrada deve ser informada.";
        return false;
    }
    else if(txtDataCompra.getText().replace("/", "").trim().isEmpty())
    {
        erro = "A data de compra deve ser informada.";
        return false;
    }
    else if(txtPreco.getText().isEmpty()
                || Double.parseDouble(txtPreco.getText().replace(',', '.')) == 0)
    {
        erro = "O preço de compra deve ser informado.";
        return false;
    }
    else if(txtQuantidade.getText().isEmpty()
                || Double.parseDouble(txtQuantidade.getText().replace(',', '.')) == 0)
    {
        erro = "A quantidade de entrada deve ser informada.";
        return false;
    }
    else if(ddlPerecibilidade.getSelectedItem() != null
                && ddlPerecibilidade.getSelectedItem().toString().equals("Perecível")
                && txtDataValidade.getText().replace("/", "").trim().isEmpty())
    {
        erro = "A data de validade de itens perecíveis deve ser informada.";
        return false;
    }
    else return true;
}
```

Figura 25 - Código-fonte da validação (entrada de estoque) de preenchimento

Finalizando a camada *Menu*, temos os formulários implementados para os relatórios. Esses consistem basicamente em uma lista com os registros enquadrados nos critérios específicos de cada relatório.

Além da referida lista de registros, cada formulários apresenta também o campo para o preenchimento do critério do relatório (quantidade, produto, estoque, perecibilidade, data de validade ou data de compra), exceto o relatório de produtos escassos, cujo critério parte da definição do mesmo, e o relatório de previsão de gastos, que pode ser criado a partir de uma lista de compras ou a partir da lista de produtos escassos.

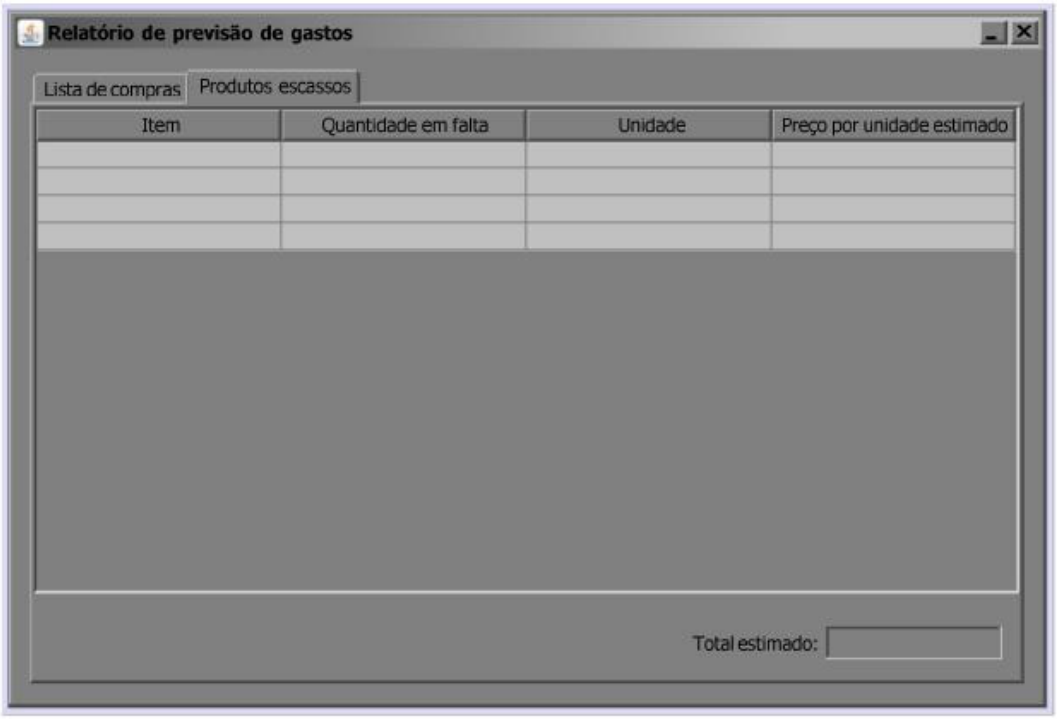

Figura 26 – Tela do relatório de previsão de gastos no caso da utilização da lista de produtos escassos. Esse relatório conta com um campo totalizador logo abaixo da lista de registros.

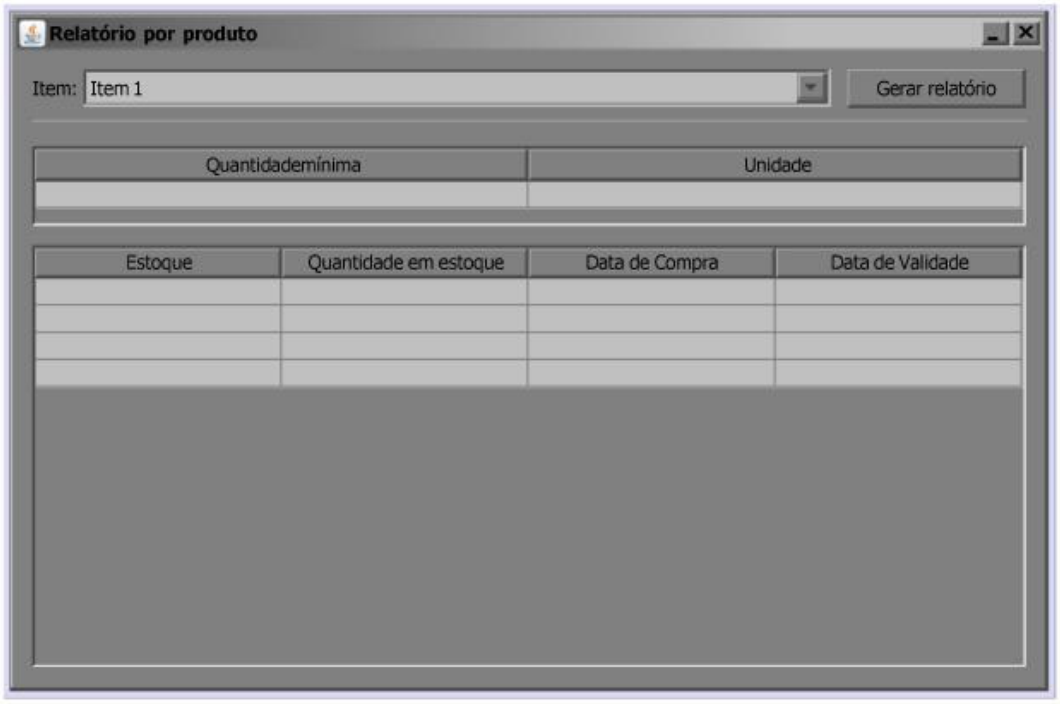

Figura 27 – Tela do relatório por produto, exibindo os campos relacionados ao item selecionado e a lista de entradas do mesmo em estoque.

O relatório de produtos escassos e o relatório de previsão de gastos a partir da lista de produtos escassos são relatório automáticos que não possuem critérios de busca definidos pelo usuário. Dessa forma, ambos são gerados no momento em que a tela do relatório é exibida.

```
private void pnlProdutosEscassosComponentShown(java.awt.event.ComponentEvent evt) {
    Item item = new Item();
    ItemEstoque itemEstoque = new ItemEstoque();
    ArrayList<ItemEstoque> itensEstoque = new ArrayList();
    ArrayList<ItemEstoque> itensEscassos = new ArrayList();
    BigDecimal totalEstimado = BigDecimal.ZERO;
    BigDecimal precoEstimado = BigDecimal.ZERO;
    try
    {
        itensEstoque = itemEstoque.ConsultarQuantidadeTotal();
        for(int i = 0; i < itensEstoque.size(); i++)
        {
            itemEstoque = itensEstoque.get(i);
            item = new Item(itemEstoque.getItem());
            item = item.ConsultarIdentificador();
            if(itemEstoque.getQuantidade().compareTo(item.getQuantidadeMinima()) < 0)
            {
                itemEstoque.setItem(new item(item));
                itensEscassos.add(itemEstoque);
            }
        }

        if(!itensEscassos.isEmpty())
        {
            ((DefaultTableModel)tblEscassosRelatorio.getModel()).setRowCount(itensEscassos.size());
            for (int linha=0; linha<itensEscassos.size(); linha++) {
                tblEscassosRelatorio.getModel().setValueAt(itensEscassos.get(linha).getItem().getDsItem(), linha, 0);

                BigDecimal quantidadeMinima = itensEscassos.get(linha).getItem().getNrQuantidadeMinima();
                BigDecimal quantidadeEstoque = itensEscassos.get(linha).getQuantidade();
                tblEscassosRelatorio.getModel().setValueAt(String.format("%1$1.2f", quantidadeMinima.subtract(quantidadeEstoque)), linha, 1);

                tblEscassosRelatorio.getModel().setValueAt(itensEscassos.get(linha).getItem().getIdUnidade().getDsUnidade(), linha, 2);
                precoEstimado = CalcularPrecoEstimado(new Item(itensEscassos.get(linha).getItem()));
                tblEscassosRelatorio.getModel().setValueAt(String.format("%1$1.2f", precoEstimado), linha, 3);
                totalEstimado = totalEstimado.add(precoEstimado.multiply(quantidadeMinima.subtract(quantidadeEstoque)));
            }
            txtTotalEscassos.setText(String.format("%1$1.2f", totalEstimado));

        }
        else
        {
            while (((DefaultTableModel)tblEscassosRelatorio.getModel()).getRowCount() > 0)
            {
                ((DefaultTableModel)tblEscassosRelatorio.getModel()).removeRow(0);
            }
            txtTotalEscassos.setText("");
        }
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null, e.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}
```

Figura 28 – Código-fonte executado ao abrir a aba *Produtos Escassos* do relatório de previsão de gastos

Já os relatórios que utilizam critérios definidos pelo usuário possuem então um botão para acionar a geração do relatório (como visto na figura 27). Esse botão executará então a ação do sistema que irá buscar no banco de dados os registros pertinentes ao relatório de acordo com o critério definido.

```
@Action
public void doReport() {
    ArrayList<ItemEstoque> itensEstoque = new ArrayList();
    String dit = ddlItem.getSelectedItem().toString();
    Item item = new Item();
    try {
        if(ddlItem.getSelectedIndex() != 0)
        {
            item = (Item)item.ConsultarDescricao(dit).get(0);
            itensEstoque = new ItemEstoque().ConsultarItem(item.getIdentificador());
            ((DefaultTableModel)tblRelatorioI.getModel()).setRowCount(1);
            tblRelatorioI.getModel().setValueAt(String.format("%1$1.2f", item.getQuantidadeMinima()), 0, 0);
            tblRelatorioI.getModel().setValueAt(item.getUnidade().getDescricao(), 0, 1);

            if(!itensEstoque.isEmpty())
            {
                ((DefaultTableModel)tblRelatorioIE.getModel()).setRowCount(itensEstoque.size());
                for (int linha=0; linha<itensEstoque.size(); linha++) {
                    tblRelatorioIE.getModel().setValueAt(itensEstoque.get(linha).getEstoque().getDsEstoque(), linha, 0);
                    tblRelatorioIE.getModel().setValueAt(String.format("%1$1.2f", itensEstoque.get(linha).getQuantidade()), linha, 1);

                    String dataCompra = "";
                    if(itensEstoque.get(linha).getValidade() != null)
                    {
                        dataCompra = String.format("%1$02d", itensEstoque.get(linha).getDataCompra().getDate());
                        dataCompra += "/";
                        dataCompra += String.format("%1$02d", itensEstoque.get(linha).getDataCompra().getMonth() + 1);
                        dataCompra += "/";
                        dataCompra += String.format("%1$04d", (itensEstoque.get(linha).getDataCompra().getYear() + 1900));
                    }
                    else
                        dataCompra = "N/A";
                    tblRelatorioIE.getModel().setValueAt(dataCompra, linha, 2);

                    String dataValidade = "";
                    if(itensEstoque.get(linha).getValidade() != null)
                    {
                        dataValidade = String.format("%1$02d", itensEstoque.get(linha).getValidade().getDate());
                        dataValidade += "/";
                        dataValidade += String.format("%1$02d", itensEstoque.get(linha).getValidade().getMonth() + 1);
                        dataValidade += "/";
                        dataValidade += String.format("%1$04d", (itensEstoque.get(linha).getValidade().getYear() + 1900));
                    }
                    else
                        dataValidade = "N/A";
                    tblRelatorioIE.getModel().setValueAt(dataValidade, linha, 3);
                }
            }
            else
            {
                while (((DefaultTableModel)tblRelatorioIE.getModel()).getRowCount() > 0)
                {
                    ((DefaultTableModel)tblRelatorioIE.getModel()).removeRow(0);
                }
            }
        }
        else
            JOptionPane.showMessageDialog(null, "Um estoque deve ser selecionado!", this.getTitle(), JOptionPane.ERROR_MESSAGE);
    } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, ex.getMessage(), this.getTitle(), JOptionPane.ERROR_MESSAGE);
            ex.printStackTrace();
    }
}
```

Figura 29 – Código-fonte da ação executada pelo botão Gerar Relatório do relatório por produto.

A camada *Modelo* é composta das classes que implementam as entidades do modelo de negócio do sistema. Cada uma dessas classes é composta de seus atributos, métodos de acesso aos atributos e métodos de comunicação com o banco de dados (realizada através das consultas definidas na camada de *Dados*).

```
public class Item {
    private String descricao;
    private BigDecimal quantidadeMinima;
    private Date dataUltimaCompra;
    private Integer identificador;
    private Unidade unidade;
    private Collection<ItemEstoque> itensEstoque;
    private Collection<ItemReceita> itensReceita;

    public Item() {...}

    public Item(item item) {...}

    /**...*/
    public String getDescricao() {
        return descricao;
    }

    /**...*/
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    /**...*/
    public BigDecimal getQuantidadeMinima() {
        return quantidadeMinima;
    }

    /**...*/
    public void setQuantidadeMinima(BigDecimal quantidadeMinima) {
        this.quantidadeMinima = quantidadeMinima;
    }

    /**...*/
    public Date getDataUltimaCompra() {
        return dataUltimaCompra;
    }

    /**...*/
    public void setDataUltimaCompra(Date dataUltimaCompra) {
        this.dataUltimaCompra = dataUltimaCompra;
    }
```

Figura 30 – Atributos da classe Item e alguns dos seus métodos de acesso.

```
//CONSULTAS
public Item ConsultarIdentificador () throws Exception
{
    Item item = new Item();
    ArrayList<item> listaItens = new ArrayList();

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("EstoquePU");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    try
    {
        listaItens.add((item) em.createNamedQuery("item.findByIdItem").setParameter("idItem", this.identificador).getSingleResult());
        item = new Item(listaItens.get(0));
    } catch (Exception e) {
        em.getTransaction().rollback();
        throw e;
    } finally {
        em.close();
    }
    return item;
}

public ArrayList<Item> ConsultarTodos () throws Exception
{
    ArrayList<Item> itens = new ArrayList();
    ArrayList<item> listaItens = new ArrayList();

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("EstoquePU");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    try
    {
        listaItens.addAll(em.createNamedQuery("item.findAll").getResultList());
        if(!listaItens.isEmpty())
        {
            for(int i = 0; i < listaItens.size(); i++)
            {
                itens.add(new Item((item) listaItens.get(i)));
            }
        }
    } catch (Exception e) {
        em.getTransaction().rollback();
        throw e;
    } finally {
        em.close();
    }
    return itens;
}
```

Figura 31 – Código-fonte de dois métodos de consulta da classe Item (consulta por identificador e consulta por todos os itens cadastrados).

```
public void Inserir () throws Exception
{
    //int registros = 0;
    //int idUnidade = 0;
    ArrayList<Integer> lista = new ArrayList();

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("EstoquePU");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    try
    {
        lista.add(new Integer(em.createNativeQuery("SELECT nextval('\"ITEM_id_item_seq\"'::regclass)").getSingleResult().toString().replace("[",
        this.setIdentificador(lista.get(0));

        em.persist(new item(this));
        em.getTransaction().commit();
    } catch (Exception e) {
        em.getTransaction().rollback();
        throw e;
    } finally {
        em.close();
    }
}

public void Alterar () throws Exception
{
 //UPDATE
    //int idUnidade = 0;

    EntityManagerFactory emf = Persistence.createEntityManagerFactory("EstoquePU");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    try
    {
        item i = new item(this);
        em.merge(i);
        em.getTransaction().commit();
    } catch (Exception e) {
        em.getTransaction().rollback();
        throw e;
    } finally {
        em.close();
    }
}

public void Excluir () throws Exception
{
 //DELETE
    item itemBD = new item(this);
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("EstoquePU");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    try
    {
        em.remove(em.merge(itemBD));
        em.getTransaction().commit();
    } catch (Exception e) {
        em.getTransaction().rollback();
        throw e;
    } finally {
        em.close();
    }
}
```

Figura 32 – Código-fonte dos métodos de inserção, alteração e exclusão de itens (classe Item).

A camada *Dados* é composta pelas classes criadas pela JPA do NetBeans para realizar a pertinência dos dados de acordo com o modelo criado para o banco de dados no PostgreSQL. Essas classes possuem as consultas ao banco de dados pré-definidas relacionadas com a entidade que é implementada por cada classe, além dos campos definidos por cada tabela.

```
@Entity
@Table(name = "item")
@NamedQueries({ @NamedQuery(name = "item.findAll", query = "SELECT i FROM item i"),
                @NamedQuery(name = "item.findByDsItem", query = "SELECT i FROM item i WHERE i.dsItem = :dsItem"),
                @NamedQuery(name = "item.findByNrQuantidadeMinima", query = "SELECT i FROM item i WHERE i.nrQuantidadeMinima = :nrQuantidadeMinima"
                @NamedQuery(name = "item.findByDtUltimaCompra", query = "SELECT i FROM item i WHERE i.dtUltimaCompra = :dtUltimaCompra"),
                @NamedQuery(name = "item.findByIdItem", query = "SELECT i FROM item i WHERE i.idItem = :idItem"),
                @NamedQuery(name = "item.findByIdUnidade", query = "SELECT i FROM item i WHERE i.idUnidade = :idUnidade")})
public class item implements Serializable {
    private static final long serialVersionUID = 1L;
    @Basic(optional = false)
    @Column(name = "ds_item")
    private String dsItem;
    @Column(name = "nr_quantidade_minima")
    private BigDecimal nrQuantidadeMinima;
    @Basic(optional = false)
    @Column(name = "dt_ultima_compra")
    @Temporal(TemporalType.DATE)
    private Date dtUltimaCompra;
    @Id
    @Basic(optional = false)
    @Column(name = "id_item")
    private Integer idItem;
    @JoinColumn(name = "id_unidade", referencedColumnName = "id_unidade")
    @ManyToOne(optional = false)
    private unidade idUnidade;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idItem")
    private Collection<itemEstoque> itemEstoqueCollection;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "idItem")
    private Collection<itemReceita> itemReceitaCollection;
```

Figura 33 – Classe *item* (camada de *Dados*), suas consultas pré-definidas e os campos da tabela item.

5.2 – Testes

O ambiente de testes usado para esse sistema, considerando que o mesmo foi projetado para funcionar em estação local, utilizando apenas um usuário com perfil de acesso a todas as funcionalidades do mesmo, foi um computador AMD Athlon64 X2 2.6GHz, com 3GB de memória RAM, rodando o sistema operacional Microsoft Windows XP Professional SP3.

Para a realização dos testes de sistema, objetivando a simulação da operação real do sistema, foi realizada uma linha cronológica de operações que compreende a quase totalidade das funcionalidades do sistema, iniciando pelo **registro de um novo usuário**, o **cadastro de novos itens**, **cadastro de um novo estoque**, **entrada em estoque** dos itens cadastrados, **saída de estoque** dos itens cadastrados, **cadastro de um nova receita** com os itens cadastrados, **execução da receita** cadastrada e geração de **relatórios relacionados (por produto, por estoque, de produtos escassos e previsão de gastos)**.

Visto que os testes de unidade foram realizados concomitantemente com a fase de desenvolvimento e os testes de aceitação foram realizados junto com os testes de sistema, não houve necessidade de ser feito um plano de testes nesses casos, sendo aproveitadas as situações de teste já sendo utilizadas.

Tabela 2 – Casos de teste

Caso de Uso	Descrição do teste
Registrar Usuário	Registro de um novo usuário
Cadastrar Item	Cadastro de novos itens
Cadastrar Estoque	Cadastro de um novo estoque
Entrada de Estoque	Entrada em estoque dos itens cadastrados
Saída de Estoque	Saída de estoque de alguns itens
Cadastrar Receita	Cadastro de uma nova receita
Executar Receita	Execução da receita cadastrada
Gerar Relatório por Produto	Geração do relatório por produtos cadastrados
Gerar Relatório por Estoque	Geração do relatório pelo estoque cadastrado
Gerar Relatório de Produtos Escassos	Geração do relatório de produtos escassos
Gerar Relatório de Previsão de Gastos	Geração do relatório de previsão de gastos

5.3 – Resultados

Os resultados encontrados na fase de testes (teste de sistema/aceitação) foram todos dentro do esperado, sendo observado que o sistema executava as operações em tempo bastante satisfatório sem a ocorrência de erros inesperados e cumprindo os requisitos funcionais definidos na fase inicial do projeto.

A interface amigável e de resposta rápida se demonstrou bastante intuitiva na operação das funcionalidades do sistema, sem que fosse apresentada nenhuma dificuldade por parte do usuário de explorar o mesmo sem muita necessidade de suporte.

Os testes de unidade, realizados no decorrer do desenvolvimento, trouxeram alguns problemas na execução de algumas funções internas do sistema, mas ao serem identificados, esses problemas foram prontamente solucionados e não mais ocorreram nas fases subseqüentes do projeto.

Capítulo 6

Conclusões

A utilização da modelagem orientada a objetos permitiu a fácil transposição dos conceitos de negócio relacionados com a gestão de um estoque doméstico para um padrão de componentes de software utilizados no desenvolvimento do sistema. Ainda que para um software de arquitetura local, o método facilita a criação do aplicativo em blocos funcionais isolados e reutilizáveis, como por exemplo as classes definidas na camada *Modelo*.

Aliado ao método de modelagem orientada a objetos, a arquitetura utilizada baseada no modelo MVC também contribuiu para essa segmentação em blocos funcionais, isolando componentes de modelagem conceitual (*Modelo*) e de modelagem funcional (*Menu*). Ainda que o padrão MVC não tenha sido utilizado de forma plena, visto que o sistema é local e não se fez necessário o isolamento das camadas de visão e controle, o isolamento em camadas contribui essencialmente no desenvolvimento e na manutenção do sistema, permitindo melhorias mais ágeis e facilitando o processo de adaptação a eventuais novos requisitos.

O ambiente de desenvolvimento do NetBeans, bastante amigável e fácil de utilizar, propicia por padrão uma série de componentes para a criação de interfaces que torna bastante intuitiva a construção das telas do sistema. Isso gera uma grande economia no tempo despendido na criação das interfaces, permitindo o maior gasto do tempo programado no desenvolvimento relacionado com as definições funcionais do sistema.

O NetBeans também possui uma ferramenta de modelagem UML que inicialmente é bastante interessante, visto que proporciona uma melhor integração da fase de projeto com a fase de desenvolvimento. No entanto, os recursos dessa ferramenta não são muito extensos e de forma complementar foi utilizado o StarUML, que é um aplicativo específico para modelagem UML e possui uma gama de recursos bastante extensa para a construção dos modelos de projeto nas diversas fases do mesmo.

A utilização da linguagem Java tornou o uso da arquitetura orientada a objetos trivial, já que a mesma tem dentre os seus princípios de criação a implementação dessa arquitetura. Por sua simplicidade e similaridade com as linguagens C e C++, utilizadas

durante o decorrer do curso de graduação, a linguagem Java permitiu um desenvolvimento rápido e eficiente de cada uma das funcionalidades do sistema, sem ocorrerem muitos problemas nessa fase do projeto. As poucas dificuldades ocorridas foram facilmente sanadas, visto que através de breves pesquisas na *web* é possível obter extenso material sobre o uso da linguagem.

O PostgreSQL, sendo um banco de dados objeto-relacional, permite com maior naturalidade a implementação do modelo conceitual do sistema no modelo de dados utilizado. Entretanto, como o acesso ao banco de dados no sistema foi inteiramente realizado através da JPA do NetBeans, os recursos da linguagem de consultas do PostgreSQL não foram utilizados.

Futuramente, no entanto, existem algumas melhorias que poderão ser feitas no sistema para que o uso do sistema seja mais abrangente, não só melhorando a utilização do mesmo pelo usuário doméstico, como adaptando o projeto para o melhor uso do sistema em estoques comerciais.

A primeira melhoria nesse sentido seria modificar a arquitetura do sistema para permitir o acesso do sistema através de vários terminais de forma simultânea, criando um ambiente multi-usuário que a arquitetura local não permite. Para isso, o banco de dados deverá ser centralizado, criando uma arquitetura cliente-servidor.

Com uma arquitetura cliente-servidor, o isolamento de camadas do sistema deve ser otimizado, efetivamente separando as camadas de visão e controle para permitir que apenas a visão seja necessária ao cliente e as outras camadas fiquem a cargo do servidor. Com esse isolamento efetivo, será possível ainda modificar o sistema para que o mesmo passe a ser acessado através de uma arquitetura web, em uma intranet ou mesmo através da internet.

Outra melhoria possível a ser realizada é a criação de uma funcionalidade que permita a listagem de itens a serem comprados, de forma que o usuário possa planejar a próxima compra e, após a mesma efetuada, realizar a entrada conjunta em estoque dos itens da lista de compras.

Ainda, o sistema, através dessa funcionalidade de lista de compras, poderia ser integrado com um sistema de compras, a fim de permitir que a gerência de um estoque comercial possa ser mais eficiente na reposição de itens.

Esse projeto de graduação, como forma de complementar a formação acadêmica, permitiu a aplicação na prática de conceitos, métodos e tecnologias estudados durante o decorrer do curso, contribuindo assim para um ganho de experiência no desenvolvimento de projetos de software.

Através desse projeto, pude utilizar os conceitos da UML para uma aplicação real, analisando as ferramentas que o método fornece e observando os resultados decorrentes do uso do mesmo.

Tendo sido estabelecido um cronograma bem definido nas fases iniciais do projeto, o mesmo me ajudou a realizar um planejamento eficiente de todas as tarefas necessárias à análise e ao desenvolvimento do sistema final. Esse planejamento contribuiu para o desenvolvimento de um controle do tempo despendido em cada uma das tarefas a serem executadas, evitando desperdício e trazendo maior eficiência ao processo de desenvolvimento.

Também me permitiu a utilização prática de ferramentas como o NetBeans e o PostgreSQL, me garantindo experiência com um ferramental mais amplo, além daquele utilizado didaticamente no decorrer do curso de graduação. Ferramentas essas que além de possuírem recursos bastante vastos, são amplamente utilizadas no mercado de trabalho, me abrindo então um maior leque de possibilidades de trabalho após a conclusão da faculdade.

Além disso, decorrente do negócio atendido pelo sistema, o presente projeto me propiciou uma visão diferenciada com relação à gestão operacional de estoques, área da gestão administrativa que hoje é muito importante para o êxito de diversas empresas no mercado.

Bibliografia

[1] Documentação online do NetBeans < http://netbeans.org/>.

[2] Documentação online do PostgreSQL < http://www.postgresql.org/>.

[3] Documentação online sobre UML < http://www.uml.org/>.

[4] GARCIA, Eduardo S.; REIS, Leticia M. T. V.; MACHADO, Leonardo R.; FERREIRA, Virgílio J. M. "*Gestão de estoques: otimizando a logística e a cadeia de suprimentos"*. Rio de Janeiro, E-papers Servicos Editoriais Ltda., 2006. Disponível em: < http://books.google.com/books?id=AvfRM51NLcQC&dq=pt-PT>.

[5] TEMPELMEIER, Horst. "*Inventory Management in Supply Networks"*. Norderstedt (Books on Demand) 2006.

[6] LARMAN, Craig. "*Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado*". Bookman, 2004.

Apêndice A - Dicionário de dados

Receita	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Descrição	Descrição da receita.
Título	Título da receita.

ItemReceita	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Quantidade	Quantidade do item utilizado na receita.
Item (FK)	Identificador do item.
Receita (FK)	Identificador da receita.

Item	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Descrição	Descrição do item.
Quantidade mínima	Quantidade mínima necessária do item em estoque.
Data de última compra	Data em que o item deu entrada no estoque pela última vez.
Unidade	Identificador da unidade padrão do item.

Unidade	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Descrição	Descrição da unidade.
Sigla	Sigla utilizada para a unidade.

Estoque	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Descrição	Descrição do estoque.

ItemEstoque	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Quantidade	Quantidade do item no estoque.
Data de validade	Data de validade da partida, caso seja perecível.
Preço de compra	Preço por unidade pago pelo item.
Data de compra	Data em que foi realizada a compra do item.
Item (FK)	Identificador do item.
Estoque (FK)	Identificador do estoque.
Perecibilidade (FK)	Identificador da perecibilidade (perecível ou não perecível).

Perecibilidade	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Descrição	Descrição da perecibilidade (perecível ou não perecível).

Usuário	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Login	Login do usuário para acesso ao sistema.
Senha	Senha do usuário para aceso ao sistema.
Nome	Nome do usuário.
Perfil	Identificador do perfil do usuário.

Perfil	
Identificador	Número identificador seqüencial gerado pelo banco de dados.
Registrar Usuário	Define se o usuário tem permissão para registrar novos usuários, alterar ou excluir usuários já registrados.
Cadastro	Define se o usuário tem permissão para acessar o menu 'Cadastro' do sistema.
Cadastrar Item	Define se o usuário tem permissão para realizar o cadastro de novos itens, alterar ou excluir itens já cadastrados. (É necessário que o usuário tenha permissão para acessar o menu 'Cadastro')
Cadastrar Estoque	Define se o usuário tem permissão para realizar o cadastro de novos estoques, alterar ou excluir estoques já cadastrados. (É necessário que o usuário tenha permissão para acessar o menu 'Cadastro')
Cadastrar Unidade	Define se o usuário tem permissão para realizar o cadastro de novas unidades, alterar ou excluir unidades já cadastradas. (É necessário que o usuário tenha permissão para acessar o menu 'Cadastro')
Cadastrar Receita	Define se o usuário tem permissão para realizar o cadastro de novas receitas, alterar ou excluir receitas já cadastradas. (É necessário que o usuário tenha permissão para acessar o menu 'Cadastro')
Estoque	Define se o usuário tem permissão para acessar o menu 'Estoque' do sistema.
Entrada de estoque	Define se o usuário tem permissão para realizar entrada de itens em estoque. (É necessário que o usuário tenha permissão para acessar o menu 'Estoque')
Saída de estoque	Define se o usuário tem permissão para realizar a retirada de itens do estoque. (É necessário que o usuário tenha permissão para acessar o menu 'Estoque')
Executar receita	Define se o usuário tem permissão para realizar a retirada de itens do estoque através da execução de uma receita. (É necessário que o usuário tenha permissão para acessar o menu 'Estoque')
Relatórios	Define se o usuário tem permissão para acessar o menu 'Relatórios' do sistema, de forma a gerar qualquer dos relatórios disponíveis.