UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CÂMPUS PATO BRANCO
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

EMERSON LUIS KOSLOVSKI

SISTEMA PARA GERENCIAMENTO DE LANCHONETES

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2014

EMERSON LUIS KOSLOVSKI

SISTEMA PARA GERENCIAMENTO DE LANCHONETES

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: profa. Beatriz Terezinha Borsoi

PATO BRANCO
2014

ATA Nº: **253**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **EMERSON LUIS KOSLOVSKI**.

Às 13:30 hrs do dia 18 de dezembro de 2014, Bloco V da UTFPR, Câmpus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Edilson Pontarolo (Orientador), Robison Cris Brito (Convidado) e Kathya Silvia Collazos Linares (Convidada), para avaliar o Trabalho de Diplomação do aluno Emerson Luis Koslovski, matrícula 01066846, sob o título **Sistema para Gerenciamento de Lanchonetes**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, COADS. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 14:05 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientador

Prof. Robison Cris Brito, M.Sc.
Convidado

Profa. Kathya Silvia Collazos Linares, Dr.
Convidada

Profa. Eliane Maria de Bortoli Fávero, M.Sc
Coordenadora do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

KOSLOVSKI, Emerson Luis. Sistema para gerenciamento de lanchonetes. 2014. 92 f. Relatório de estágio (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Pato Branco, 2014.

A facilidade de acesso aos recursos computacionais caracterizada como um computador com ou sem acesso à Internet e a quantidade de dados que as empresas, mesmo de pequenos negócios, manipulam, tem justificado o uso desses recursos para gerenciamento das atividades dessas instituições. O segmento de lanchonetes - incluindo nessa denominação, bares, panificadoras e restaurantes que atendem pedidos de clientes sob demanda e que oportunizam o consumo no local ou entrega do pedido – pode ser beneficiado com o uso de um sistema para gerenciamento dos pedidos dos clientes e o controle das contas e de estoque. Assim, propõe-se neste trabalho a implementação de um sistema para gerenciamento de lanchonetes visando facilitar o trabalho dos garçons pelo controle dos pedidos e das entregas, agilizar os serviços de cozinha pelo gerenciamento da fila de pedidos e facilitar o controle de estoques e das contas a pagar e a receber. Para a implementação foram utilizadas tecnologias para desenvolvimento web e caracterizada como *Rich Internet Application*, enfatizando a linguagem Java, o *framework* JavaServer Faces e a biblioteca PrimeFacces. A escolha de um sistema para *web* tem como justificativa a de facilitar e a não necessidade de implantar uma rede corporativa, visto que o sistema será utilizado por vários usuários e com acessos distintos como gerente, garçons e cozinheiros.

Palavras-chave: Gerenciamento de lanchonetes. Rich Internet Application. JavaServer Faces. PrimeFaces.

LISTA DE FIGURAS

LISTAGENS DE CÓDIGO

LISTAGENS DE QUADROS

LISTA DE ABREVITURAS E SIGLAS

API	*Application Programing Interfaces*
CSS	*Cascading Style Sheet*
HTML	*HiperText Markup Language*
ORM	*Object Relational Mapping*
RF	Requisito funcional
RIA	*Rich Internet Applications*
RNF	Requisito não funcional
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais, os objetivos do sistema e a justificativa para a realização deste trabalho.

1.1 CONSIDERAÇÕES INICIAIS

As atividades realizadas em uma lanchonete, como fazer pedidos, controlar a etapa do preparo e a entrega aos clientes, efetuar o controle do fluxo de caixa e financeiro, tem sido, em muitos casos, feitos por meio de anotações em papel e/ou por planilhas de dados. Os atores envolvidos no processo de negócio de uma lanchonete são garçons, cozinheiros, atendente de caixa e gerente. Cada um desses atores realiza tarefas distintas, mas todas relacionadas ao atendimento dos pedidos dos clientes.

Sem o auxílio de sistemas computacionais, o processo desses pedidos é feito pelo garçom, que anota os pedidos manualmente em comandas e as encaminha para a cozinha para o preparo. Após esse procedimento o pedido é entregue pelo garçom ao cliente. O processo de entrega externa à lanchonete é feito por entregadores que tem como base as mesmas comandas anotadas em papel. O processo de caixa é feito pelo atendente que tem acesso as planilhas nas quais informa o pedido, o que foi consumido e os valores gastos. Já o processo de controle interno é realizado pela gerência do estabelecimento que analisa as planilhas de dados nas quais estão anotados os dados dos pedidos, entregas, entradas e saída do caixa e dados não vinculados ao processo de negócio como cartão ponto e folha de pagamento. Mesmo com o auxílio de planilhas eletrônicas de dados, o gerenciamento e controle das atividades realizadas, sejam as relacionadas ao negócio da empresa como as de manutenção como pagamentos e folha ponto, fica bastante difícil e pode ser parcial e por vezes inconsistente.

Diante deste contexto, percebeu-se a importância de um sistema computacional que permita a todos os envolvidos nos procedimentos de uma lanchonete, lançar pedidos, efetuar o preparo e realizar o controle das entregas. O controle financeiro, de fluxo de caixa e dos funcionários contribui, assim para a realização mais eficaz e segura das atividades envolvidas nos processos de negócio.

Um sistema *web* facilita o acesso de todos os envolvidos, desde o garçom até o gerente, a todas as etapas do processo, sem a necessidade de uma plataforma definida, podendo ser acessado de qualquer navegador até mesmo por plataformas móveis.

Como forma de atender essa necessidade, neste trabalho de conclusão de curso é realizada a implementação de um sistema para gerenciamento de lanchonetes. A modelagem do sistema foi realizada como relatório de estágio do autor deste trabalho. O resultado dessa modelagem está presente neste texto para facilitar o entendimento das funcionalidades implementadas.

1.2 OBJETIVOS

O objetivo geral está relacionado com o resultado esperado na realização deste trabalho, já os objetivos específicos complementam os gerais em termos de funcionalidade do sistema.

1.2.1 Objetivo geral

Implementar um sistema que permita aos funcionários de uma lanchonete realizar suas tarefas, visando melhor desempenho em termos de processamento e fornecer serviços melhores aos seus clientes.

1.2.2 Objetivos específicos

- Facilitar a emissão dos pedidos junto aos clientes com o envio direto do pedido para o setor da cozinha.
- Agilizar o processo de preparo dos pedidos e a entrega dos mesmos.
- Facilitar o controle financeiro e de fluxo de caixa.
- Facilitar o fechamento do pedido e a confirmação de pagamento pelo cliente junto ao caixa.
- Facilitar o processo de cotação junto a fornecedores.

1.3 JUSTIFICATIVA

A falta de praticidade no uso de comandas em papel e de eficácia nesse modo de anotar pedidos e de um controle mais eficiente dos processos compõem a principal justificativa para o desenvolvimento deste trabalho. Para os garçons que fazem os pedidos, o uso de comandas escritas à mão nem sempre é eficiente. Para os funcionários do setor da cozinha, o uso de tais comandas pode implicar erros na realização dos pedidos pela interpretação incorreta das muitas formas de escritas a mão. Para o funcionário do caixa a identificação dos itens do pedido na hora do pagamento torna-se complicado. Para os gerentes o controle das atividades é quase nulo, pois não há um sistema que possa mostrar em que cada funcionário trabalhou.

Para os integrantes do processo, um sistema *web* torna mais fácil o processo de registro e de acompanhamento do trabalho na lanchonete. Para os garçons, que realizam o pedido, além da não necessidade de anotar pedidos a mão, é muito prático realizar a emissão dos pedidos com apenas alguns toques em um computador. Uma vez feito, o pedido estará disponível para o setor da cozinha efetuar o preparo. Para o setor da cozinha, o preparo se torna mais eficiente, pois pode contar com controles de saldo de itens e o preparo sempre será na ordem em que foi feito, sem possibilidade de avançar pedidos e deixar outros em tempo demasiado em espera.

A justificativa de aplicabilidade do resultado deste trabalho se fundamenta na necessidade percebida de facilitar o processo de trabalho de uma lanchonete como um todo. Em termos de tecnologias, a escolha de um sistema *web* decorre pela facilidade de acesso que se tem por meio de servidores de aplicações *web*, sem vínculo a uma plataforma de sistema operacional específica.

1.4 ORGANIZAÇÃO DO TEXTO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia do sistema, incluindo os objetivos e a justificativa.

No Capítulo 2 está o referencial teórico sobre desenvolvimento para ambiente *web* de aplicações caracterizadas como de interface rica. No Capítulo 3 está o método, que é a sequência geral de passos definidos para a realização do trabalho e os materiais utilizados. Os

materiais se referem ao que foi utilizado para modelar e exemplificar a implementação do sistema. O Capítulo 4 contém a modelagem realizada do sistema e códigos que apresentam o uso das tecnologias adotadas na implementação de operações básicas. No Capítulo 5 está a conclusão com as considerações finais. Por fim estão as referências bibliográficas utilizadas para compor o texto dos Capítulos 2 e 3.

2 DESENVOLVIMENTO DE APLICAÇÕES WEB

Este capítulo apresenta o referencial que fundamenta o sistema proposto, sendo este um sistema *web* utilizando recursos que a caracterizam como de interface rica.

2.1 APLICAÇÕES WEB COM INTERFACE RICA

As aplicações *web* e *desktop* estão convergindo rapidamente (STEARN, 2007): os desenvolvedores *web* estão adicionando características de aplicações em seus sites e as funcionalidades, definidas como recursos de interação, das aplicações *desktop* estão sendo portadas para aplicações *web* visando incrementar as funcionalidades dessas aplicações.

A convergência entre *desktop* e *web* conduz ao desenvolvimento de aplicações Internet com tecnologias relacionadas à *HiperText Markup Language* (HTML). Contudo, HTML foi originalmente criada como uma linguagem de marcação de hipertexto que um navegador *web* pode utilizar para interpretar e compor páginas *web* estáticas (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). HTML não permite atualizações parciais da página, ou seja, apenas em determinadas partes da página sendo exibida. Isso é possível somente com a inclusão de programação em outras linguagens que processam no cliente, como JavaScript que podem ser utilizadas para manipular elementos específicos de uma página *web.*

Linguagens de programação como, por exemplo, PHP, ASP e Java podem ser usadas para gerar páginas HTML dinamicamente. HTML por si só não permite o desenvolvimento de aplicações de negócio com recursos de interação que a caracterizam como rica. Esses recursos são os semelhantes às aplicações *desktop*. Isso é possível pelo uso de tecnologias adicionais tais como Ajax (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012). Esses autores ressaltam que a necessidade de usar várias tecnologias torna o desenvolvimento mais dispendioso em termos de tempo e custo, se comparado ao desenvolvimento das aplicações *desktop* tradicionais. E destacam ainda que o principal problema está na necessidade de criar o lado servidor (utilizando linguagens de programação que geram HTML) separadamente do lado cliente (usando tecnologias Ajax para dinamicamente alterar partes da tela e *Cascading Style Sheet* (CSS) para definir a interface gráfica com o usuário.

Uma forma de resolver esse problema e eliminar a necessidade de usar tecnologias

distintas para implementar cliente e servidor em uma aplicação *web* é por meio do desenvolvimento de um servidor que permite a execução de aplicações na Internet. O cliente é um programa que a partir da descrição da interface gráfica e janela de eventos do lado servidor renderiza a interface no cliente. Essas aplicações são comumente conhecidas como *Rich Internet Applications* (RIA) denominadas de aplicações Internet ricas em decorrência dos recursos de interface que elas possuem.

Iniciando em 2005, as aplicações *web* evoluíram para as RIAs (TRAMONTANA; AMALFITANO; FASOLINO, 2013). As RIAs são aplicações *web* que contêm características e funcionalidades de uma aplicação *desktop* tradicional. Por se assemelharem as aplicações *desktop*, elas fazem com que o seu uso seja facilitado e oferecem uma interface mais rica em termos de recursos e funcionalidades para o usuário (LOOSLEY, 2006).

Nas RIAs uma parte relevante da lógica de negócio é realizada no cliente, tomando vantagem das potencialidades do Ajax, por exemplo. Duhl (2003) ressalta que as RIAs foram propostas com o objetivo de resolver problemas encontrados nas aplicações *web*. Elas provêm interface sofisticada para representar processos e dados complexos, ao mesmo tempo que minimizam a transferência de dados entre cliente e servidor (FUKUDA, YAMAMOTO, 2008).

As abordagens típicas para RIAs possuem quatro características principais (STEARN, 2007):

a) Ambiente de execução – relacionado ao software que o desenvolvedor escolhe para executar a aplicação e prover suas capacidades básicas.

b) Interface gráfica com o usuário – a linguagem selecionada para desenvolver a interface;

c) Lógica de negócio – a linguagem utilizada para manipular as interações do usuário com os dados e os controles da interface, além de implementar as regras de negócio;

d) Serviços de *backend* – os vínculos com conteúdos externos à aplicação. Esses vínculos podem ser com servidores que provem acesso local ou remoto de armazenamento, bancos de dados e meio de consumir e agregar recursos baseados em *web*, como os *webservices*.

Uma tecnologia RIA representa um ambiente de desenvolvimento no qual a aplicação é programada para o lado servidor e a mesma aplicação pode ser executada no lado cliente sem uma programação adicional para o cliente (PAVLIĆ; PAVLIĆ; JOVANOVIĆ, 2012).

As aplicações caracterizadas como RIAs são uma nova geração de aplicações Internet que combinam comportamento e características das aplicações *web* e *desktop* (COLOMBO-MENDOZA; ALOR-HERNÁNDEZ; RODRÍGUEZ-GONZÁLEZ, 2012). O desenvolvimento de RIAs é guiado por um conjunto de novas linguagens de *frameworks* como, por exemplo, Flex, Silverlight e Google Web Toolkit (BUSCH; KOCH, 2009). A demanda das RIAs está crescendo em decorrência do desenvolvimento de aplicações em nuvem. Computação em nuvem é um estilo de computação no qual recursos dinamicamente escaláveis e implantados são fornecidos como serviço através da rede (COLOMBO-MENDOZA; ALOR-HERNÁNDEZ; RODRÍGUEZ-GONZÁLEZ, 2012).

Na computação em nuvem os usuários não precisam ter conhecimento especializado ou controle da infraestrutura em rede que suporta os serviços fornecidos (ORACLE..., 2011). Muitos dos serviços disponibilizados chamados serviços em nuvem são acessados por meio de *Application Programing Interfaces* (API).

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados para a realização deste trabalho. Os materiais estão relacionados às tecnologias e às ferramentas utilizadas e o método apresenta a sequência das principais atividades realizadas.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias que foram utilizadas para modelar e implementar o sistema. A implementação tem o objetivo de apresentar o uso das tecnologias.

Ferramenta / Tecnologia	**Versão**	**Referência**	**Finalidade**
Astah-Professional	6.8.0 (model version: 37)	http://astah.net/editions/community	Documentação da modelagem baseada na UML.
Linguagem Java	JDK 1.7	http://www.oracle.com	Linguagem de programação.
Eclipse Kepler	4.3.2	http://www.eclipse.org	Ambiente de desenvolvimento.
Postgres	9.1.11	http://www.postgresql.org	Banco de dados.
PGAdmin	1.16.1	http://www.postgresql.org/download/linux/ubuntu	Administrador do banco de dados.
Apache Tomcat	7.0	http://www.apache.org	*Container web* para a aplicação.
PrimeFaces	4	http://primefaces.org	Biblioteca de componentes para *web*.
JSF	2.0	https://javaserverfaces-spec-public.java.net	*Framework* para *web.*
Hibernate	3	http://www.hibernate.org	*Framework* de mapeamento objeto-relacional e a persistência dos dados.

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

A seguir estão descritas as etapas definidas para o desenvolvimento do aplicativo e as principais atividades de cada uma dessas etapas.

a) Levantamento de requisitos

O levantamento dos requisitos foi realizado como estágio supervisionado pelo autor deste trabalho de conclusão de curso. Esses requisitos foram complementados porque funcionalidades haviam sido levantadas de forma incompleta.

b) Análise e projeto do sistema

Uma primeira versão da modelagem também foi realizada como trabalho de estágio. Neste trabalho de conclusão de curso a modelagem foi ajustada visando atender os requisitos complementados e outras funcionalidades não adequadamente analisadas e modeladas na fase anterior.

c) Implementação

A implementação foi realizada utilizando a ferramenta Eclipse Kepler. Como trabalho de estágio foram implementadas as funcionalidades básicas de cadastro (inclusão, exclusão, consulta e alteração) visando exemplificar o uso das tecnologias e o aprendizado das mesmas. Em termos de interface, o objetivo no estágio foi testar a melhor forma de compor os formulários e de disponibilizar as informações na tela para os usuários finais. Nesta etapa de trabalho de conclusão de curso as demais funcionalidades foram implementadas.

d) Testes

Os testes foram realizados pelo programador, visando identificar erros de código.

4 RESULTADO

Este capítulo apresenta o resultado deste trabalho que é a modelagem de um sistema para controle de lanchonetes. No capítulo também constam códigos que visam exemplificar como a implementação de operações básicas de cadastro, realizadas com o objetivo de estudo das tecnologias. A implementação será efetivada como trabalho de conclusão de curso.

4.1 ESCOPO DO SISTEMA

O sistema modelado como resultado deste trabalho automatizará os processos de controle da área de produção e de pessoal de lanchonetes. Como exemplos desses processos estão e realização de pedidos, a preparação dos itens do cardápio, a entrega dos pedidos, o controle do fluxo financeiro e de caixa e o gerenciamento dos funcionários. A solução proposta considera o contexto apresentado a seguir.

Um pedido pode ser feito por mais de um garçom. Contudo, cada garçom gera uma comanda no registro que posteriormente gerará uma entrega para o cliente. Os pedidos são separados por cliente e mesa, quando realizado no estabelecimento. Logo, um pedido só é encerrado quando os clientes saem da mesa. Após a emissão da comanda pelo garçom, ou confirmação do pedido quando para entrega, o mesmo chega à cozinha, quando envolve preparo, e são lidos pelos cozinheiros.

Os cozinheiros efetuam o preparo dos itens dos pedidos conforme a ordem que os mesmos chegam à cozinha. Após o término dos mesmos, são encaminhados para os garçons ou para a entrega. Nesse momento é atualizada a situação do pedido no sistema para identificar que o mesmo está sendo entregue. Quando o pedido é feito para entrega, o cliente pode controlar por meio da página da lanchonete todo o *status* do pedido. Após efetuada a entrega o pedido é atualizado o estado do pedido indicando que o mesmo foi entregue.

O funcionário do caixa da lanchonete solicita ao cliente qual a mesa e verifica os valores do pedido, informa os itens ao cliente para confirmação e solicita a forma de pagamento que pode ser dinheiro, cartão ou cheque. Após o cliente realizar o pagamento, o funcionário finaliza o pedido no sistema. Nesse momento, o pedido passa a ser contabilizado no fluxo de caixa e no controle financeiro da lanchonete.

O gerente da lanchonete tem a função de controlar o funcionamento do estabelecimento e o trabalho dos funcionários, desde os registros das atividades dos mesmos até o controle de ponto de cada funcionário.

4.2 MODELAGEM DO SISTEMA

O Quadro 2 apresenta a listagem dos requisitos funcionais identificados para o sistema. No Quadro 2 'RF' significa requisito funcional.

Identificação	Nome	Descrição
RF01	Cadastrar comandas	Um pedido pode ser composto por várias comandas. Uma comanda é composta por itens do cardápio ou por ingredientes adicionais que compõe os itens do cardápio.
RF02	Cadastrar pedidos	Um pedido pode ser composto por comandas ou por itens de cardápio.
RF03	Cadastrar pessoas	Um cadastro de pessoa pode ser dividido em cliente, funcionário ou fornecedor.
RF04	Cadastrar ingredientes	Ingredientes que podem compor itens de cardápio ou itens do pedido.
RF05	Cadastrar cardápios	Os cardápios são compostos por ingredientes que compõe o produto final.
RF06	Cadastrar tipos de movimentos	Os tipos de movimentos definem os documentos do negócio e das operações internas realizadas.
RF07	Cadastrar itens de financeiro	Os itens de financeiro geram movimentação de caixa.
RF08	Cadastrar itens de caixa	Os itens de caixa da empresa.
RF09	Cadastrar itens de estoque	Os itens do estoque são compostos por ingredientes, quantidade, validade, etc.
RF10	Cadastrar tabela de preços	As tabelas de preços são utilizadas para definir os valores dos itens de cardápio.
RF11	Cadastrar usuários	Usuários que utilizam as funcionalidades do sistema.
RF12	Cadastrar mesas	Mesas que serão utilizadas para controle dos pedidos no sistema.
RF13	Cadastrar Requisições	Requisições de produtos para serem compradas
RF14	Analisar Requisições	Efetuar a analise das requisições cadastradas a fim de verificar se todos os itens informados devem mesmo ser solicitados
RF15	Enviar Cotações	Enviar as cotações, para os fornecedores, geradas no momento da análise de requisição
RF16	Cotar	Efetuar a cotação dos itens a fim de definir qual fornecedor foi vencedor no processo de cotação
RF17	Autorizar Compra	Efetuar a autorização da compra pelos valores cotados

		pelos fornecedores.
RF18	Cadastrar Pedido de Compra	Pedidos de compra que serão gerados manualmente sem passar pelo processo de compras

Quadro 2 – Requisitos funcionais

A listagem do Quadro 3 apresenta os requisitos não-funcionais (RNF) identificados para o sistema como um todo, também denominados de requisitos suplementares.

Identificação	Nome	Descrição
RNF01	Acesso ao sistema	O acesso ao sistema será realizado por meio de *login* e senha.

Quadro 3 – Requisitos não funcionais

O diagrama de casos de uso apresentado na Figura 1 contém as funcionalidades essenciais do sistema realizadas pelos seus atores que são: cliente, garçom, cozinheiro, atendente de caixa e gerente.

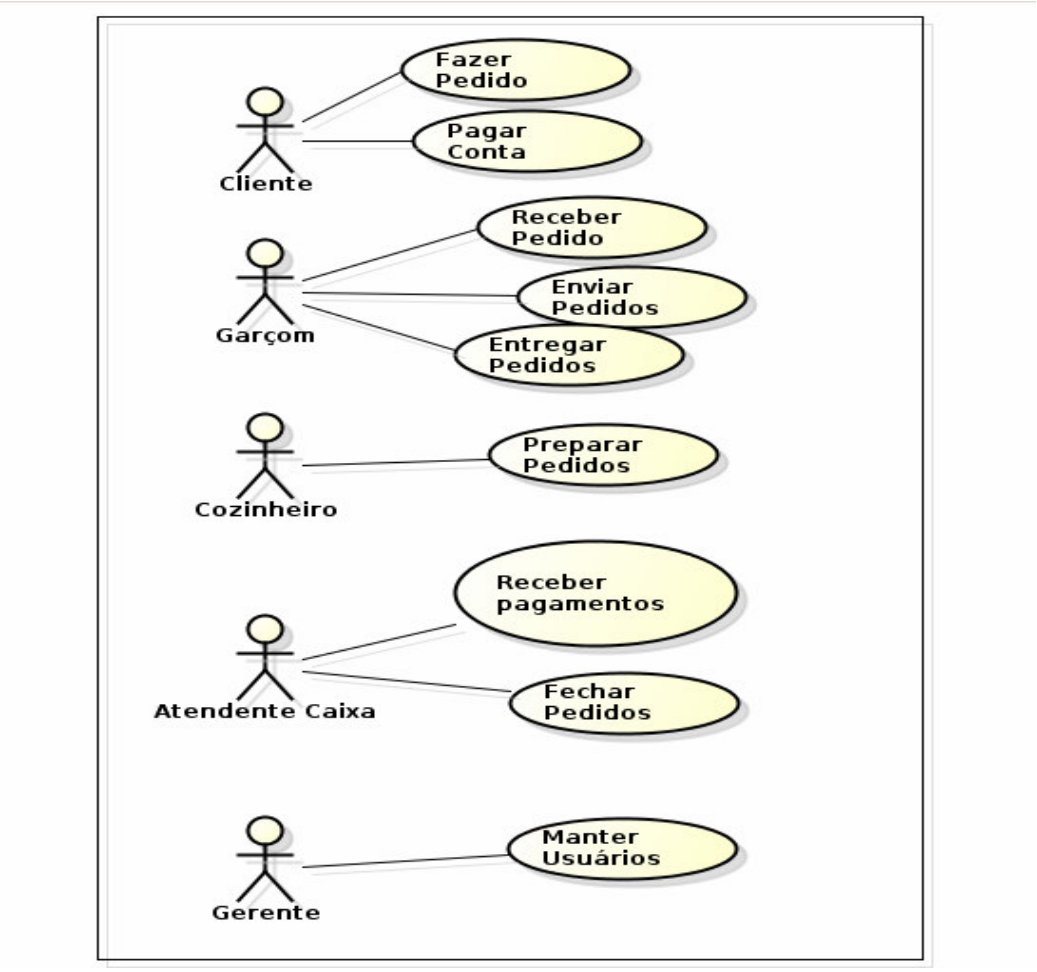


Figura 1 – Diagrama de casos de uso

O gerente é o responsável pelos cadastros dos usuários no sistema. O atendente do caixa é o responsável por efetuar o fechamento dos pedidos e por receber os pagamentos. O cozinheiro prepara os pedidos. O garçom é quem recebe os pedidos, envia para a cozinha e faz a entrega dos mesmos. O cliente faz o pedido e paga a conta.

A seguir os casos de uso apresentados na Figura 1 são descritos. O caso de uso fazer pedido é apresentado no Quadro 4.

Caso de uso:
Fazer pedido
Descrição:
O cliente faz um pedido na lanchonete. O pedido será encaminhado pelo garçom para a cozinha.
Evento Iniciador:
Não há
Atores:
Cliente
Pré-condição:
Não há.
Sequência de Eventos:
1. Cliente faz o pedido dos produtos que deseja.
2. Garçom registra o pedido.
3. Cliente é informado que o pedido será encaminhado para a cozinha e entregue ao mesmo assim que pronto.
Pós-Condição:
Pedido do cliente registrado para ser atendido pela cozinha.

Nome do fluxo alternativo (extensão)	**Descrição**
Linha 1: Não há ingredientes ou produtos solicitados pelo cliente em estoque.	1. O cliente é informado que o produto ou ingredientes para confecção do pedido não estão disponíveis. 2. Garçom solicita se o cliente quer fazer outro pedido, oferecendo alternativas.

Quadro 4 – Caso de uso fazer pedido

No Quadro 5 está a descrição do caso de uso pagar conta.

Caso de uso:
Pagar conta
Descrição:
O cliente paga a conta após ter os pedidos atendidos e antes de deixar o estabelecimento.
Evento Iniciador:
Garçom apresenta a conta para o cliente.
Atores:
Cliente
Pré-condição:
Conta ser apresentada para o cliente.
Sequência de Eventos:
1. Garçom apresenta a conta para o cliente.

2. Cliente confere a conta e faz o pagamento.
3. Garçom recebe o valor para o pagamento e efetua o registro do pagamento no sistema.
4. Cliente recebe o troco, se houver.
5. Garçom fecha o pedido do cliente.

Pós-Condição:
Pedido do cliente fechado em decorrência de pagamento.

Quadro 5 – Caso de uso pagar conta

O caso de uso receber pedido é apresentado no Quadro 6.

Caso de uso:
Receber pedido

Descrição:
O Garçom recebe o pedido do cliente

Evento Iniciador:
Pedido realizado pelo cliente.

Atores:
Garçom

Pré-condição:
Não há.

Sequência de Eventos:
1. Cliente faz o pedido.
2. Garçom registra o pedido no sistema.
3. Pedido é encaminhado à cozinha.

Pós-Condição:
Pedido encaminhado para a cozinha e colocado em fila para preparo.

Nome do fluxo alternativo (extensão)	**Descrição**
Linha 3: Pedido não precisa de preparo.	3.1 Garçom providencia o pedido 3.2 Pedido é entregue para o cliente

Quadro 6 – Caso de uso receber pedido

O caso de uso enviar pedido é apresentado no Quadro 7.

Caso de uso:
Enviar pedido

Descrição:
Pedido é enviado para preparo.

Evento Iniciador:
Pedido foi efetuado pelo cliente e há ingredientes para preparo.

Atores:
Garçom

Pré-condição:
Pedido colocado na fila para preparo.

Sequência de Eventos:
1. Pedido é enviado para a cozinha para preparo.
2. Pedido é colocado na fila para preparo.

Pós-Condição:
Pedido sendo preparado.

Quadro 7 – Caso de uso enviar pedido

No Quadro 8 é apresentado o caso de uso entregar pedido.

Caso de uso:
Entregar pedido
Descrição:
O pedido é entregue para o cliente.
Evento Iniciador:
Pedido foi preparado.
Atores:
Garçom
Pré-condição:
O preparo do pedido foi concluído.
Sequência de Eventos:
1. Garçom recebe informe que o pedido já está preparado.
2. Garçom apanha o pedido e entrega para o cliente.
3. Cliente recebe o pedido.
Pós-Condição:
Pedido entregue para o cliente.

Nome do fluxo alternativo (extensão)	**Descrição**
Linha 1: Pedido não precisa de preparo.	1.1 Garçom providencia os itens a serem entregues ao cliente 1.2 Garçom entrega os itens para o cliente.

Quadro 8 – Caso de uso entregar pedido

O caso de uso preparar pedido é apresentado no Quadro 9.

Caso de uso:
Preparar pedido
Descrição:
O pedido é encaminhado para preparo na cozinha.
Evento Iniciador:
Pedido registrado para ser preparado.
Atores:
Cozinheiro
Pré-condição:
Pedido registrado no sistema.
Sequência de Eventos:
1. Cozinha recebe pedido para ser preparado.
2. Pedido é colocado na fila para preparo.
3. Pedido é preparado.
4. É registrado no sistema que pedido está pronto (garçom informado).
Pós-Condição:
Pedido pronto para ser entregue para o cliente.

Quadro 9 – Caso de uso preparar pedido

A descrição do caso de uso receber pagamento é apresentada no Quadro 10.

Caso de uso:
Receber pagamento
Descrição:
O cliente paga a conta.
Evento Iniciador:

Cliente foi atendido e solicita fechamento da conta para pagamento.

Atores:

Atendente de caixa

Pré-condição:

Pedido do cliente atendido.

Sequência de Eventos:

1. Cliente é informado do valor a ser pago.
2. Cliente efetua o pagamento.
3. Cliente recebe o troco, se houver.
4. Conta do cliente é finalizada no sistema.

Pós-Condição:

Conta do cliente é encerrada.

Quadro 10 – Caso de uso receber pagamento

O caso de uso fechar pedido é apresentado no Quadro 11.

Caso de uso:

Fechar pedido

Descrição:

O cliente quer fechar o pedido para efetuar o pagamento.

Evento Iniciador:

Cliente solicita fechamento do pedido.

Atores:

Atendente de caixa

Pré-condição:

Cliente ter conta (pedido) aberta e atendida.

Sequência de Eventos:

1. Cliente solicita fechamento do pedido.
2. Atendente informa o valor para o cliente.

Pós-Condição:

Cliente informado do valor a ser pago em decorrência do fechamento do pedido

Quadro 11 – Caso de uso fechar pedido

O caso de uso manter usuários é apresentado no Quadro 6.

Caso de uso:

Manter usuários

Descrição:

Incluir, excluir, alterar e consultar usuários do sistema.

Evento Iniciador:

Necessidade de incluir, excluir e alterar dados e consultar dados de usuários do sistema.

Atores:

Gerente

Pré-condição:

Usuário para incluir ou excluir, dados de usuários para alterar, consulta de usuários para realizar.

Sequência de Eventos:

1. Sistema apresenta as operações possíveis de serem realizadas pelo usuário gerente.
2. Gerente escolhe a operação que pretende realizar.
3. Sistema solicita os dados necessários para realizar a respectiva operação.
4. Gerente informa os dados solicitados.

5. Sistema realiza a operação solicita e informa que a mesma foi realizada.
Pós-Condição:
Operação com cadastro de usuário realizada.

Quadro 12 – Caso de uso manter usuário

Na Figura 2 está o diagrama de classes de análise do sistema.

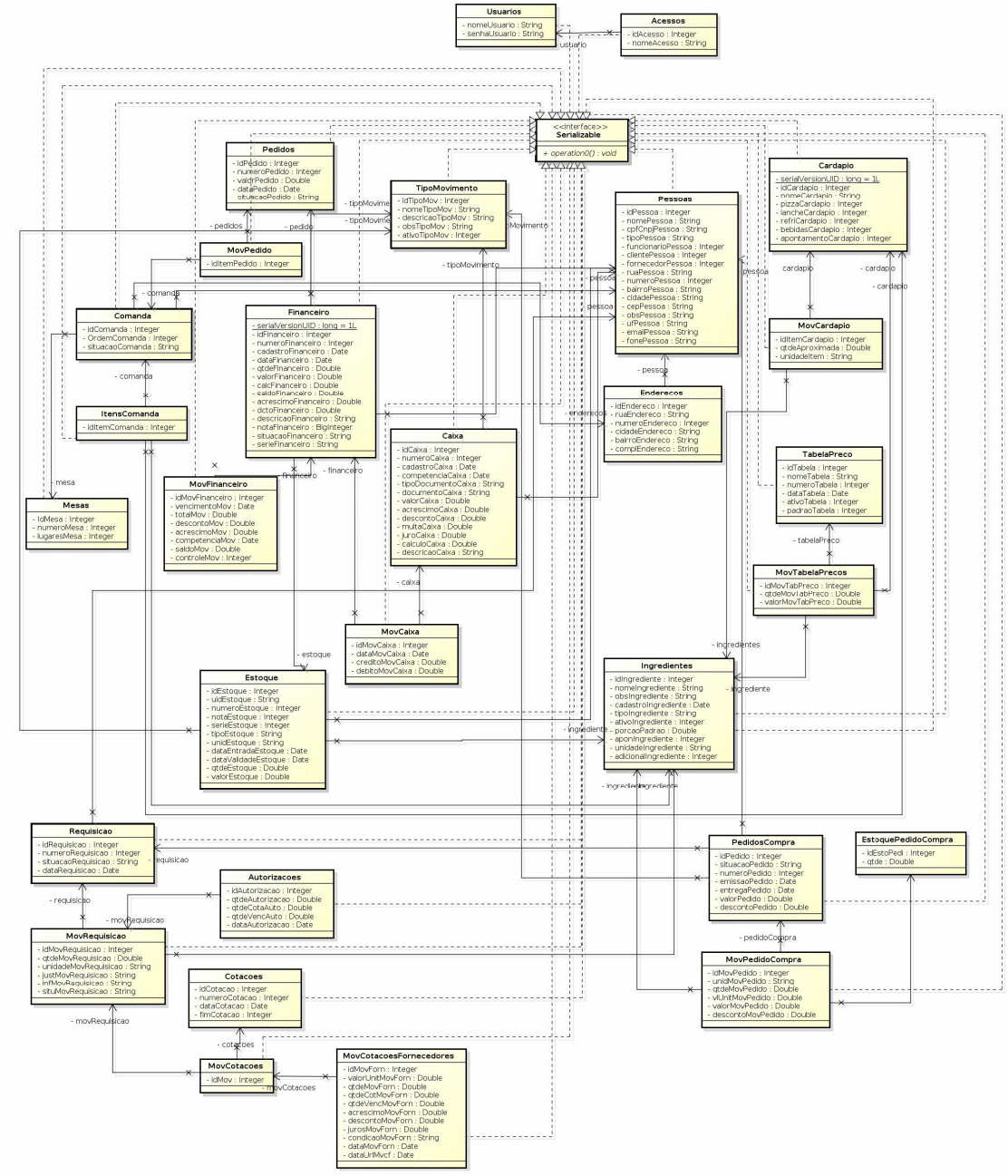


Figura 2 - Diagrama de classes de análise do sistema

As classes apresentadas no diagrama da Figura 2 estão documentadas a seguir. O

Quadro 13 apresenta a classe Aluno. O atributo “id” das classes constantes na Figura 2 é necessário para o tratamento pelo Hibernate. Os métodos de inclusão, alteração, consulta e exclusão estão todos na classe *Data Access Object* (DAO). Foi criada uma classe genérica que recebe outra classe por parâmetro e realiza as operações na mesma. Assim, não é necessário criar um DAO para cada classe, pelo menos para as operações básicas (incluir, alterar, excluir e listar todos os registros).

Identificação:	Usuario
Descrição:	Funcionários que podem utilizar o sistema
Requisitos:	RF11
Atributos:	idUsuario (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeUsuario (string): nome do usuário. pessoa (Pessoa): vinculo do usuário com a classe Pessoas. senhaUsuario (string): senha de acesso ao sistema do usuário
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); String validaUsuario(T t);

Quadro 13 – Descrição da classe Usuario

No Quadro 14 é apresentada a classe Acesso.

Identificação:	Acesso
Descrição:	Permissões dadas aos usuários do sistema.
Requisitos:	RNF01
Atributos:	idAcesso (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeAcesso (string): nome do acesso que o usuário terá. usuario(Usuario): vinculo do acesso com o usuário que terá o acesso.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 14 – Descrição da classe Acesso

No Quadro 15 é apresentada a classe Pessoas.

Identificação:	Pessoas
Descrição:	Pessoas que serão utilizadas nos cadastros do sistema.
Requisitos:	RF03
Atributos:	idPessoa (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomePessoa (String): nome da pessoa no sistema. cpfCnpjPessoa (String): CPF ou CNPJ da pessoa de acordo com o tipo da pessoa. tipoPessoa (String): tipo do cadastro da pessoa se será física ou jurídica.

	funcionarioPessoa (Integer): campo para marcar a pessoa como funcionário. clientePessoa (Integer): campo para marcar a pessoa como cliente. fornecedorPessoa (Integer): campo para marcar a pessoa como fornecedor. ruaPessoa (String): nome da rua da pessoa. numeroPessoa (Integer): número da casa da pessoa. bairroPessoa (String): nome do bairro. cidadePessoa (String): nome da cidade. cepPessoa (String): CEP da casa. obsPessoa (String): observação para o cadastro. ufPessoa (String): UF do cadastro da pessoa.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getPessoas(String t); List<T> getPessoasManui(List<T> t);

Quadro 15 – Descrição da classe Pessoas

No Quadro 16 é apresentada a classe Ingredientes.

Identificação:	Ingredientes
Descrição:	Ingredientes que serão usados nos cadastros de cardápios e adicionais do sistema.
Requisitos:	RF04
Atributos:	idIngredientes (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeIngrediente (String): nome do Ingrediente. obsIngrediente (String): observação para o cadastro. cadastroIngrediente (Date): data de cadastro do ingrediente. tipoIngrediente (String): tipo do ingrediente. ativoIngrediente (Integer): marca se está ativo. porcaoPadrao (Double): valor que será utilizado por padrão nos cadastros que o ingrediente for utilizado. aponIngrediente (Integer): código único informado no cadastro que poderá ser utilizado como busca para o ingrediente nas telas onde for referenciado. unidadeIngrediente (String): unidade de medida. adicionalIngrediente (Integer):campo para definir se pode ser um adicional em um pedido.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getIngredientes(Integer t); List<T> getIngredientesManu(List<T> t);

Quadro 16 – Descrição da classe Ingredientes

No Quadro 17 é apresentada a classe Caixa.

Identificação:	Caixa
Descrição:	Registros de movimentação de caixa da empresa.
Requisitos:	RF08

Atributos:	idCaixa (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. numeroCaixa (Integer): número do registro do caixa. tipoMovimento (TipoMovimento): vínculo com a classe TipoMovimento que define o tipo que caixa irá ter. pessoa (Pessoas): vínculo com a classe Pessoas, será mais utilizados para as saídas, para definir qual a pessoa do cadastro do caixa. cadastroCaixa (Date): data de cadastro do caixa. competenciaCaixa (Date): data que o caixa será validado. tipoDocumentoCaixa (String): tipo do documento de entrada no caixa, se dinheiro, cheque ou cartão. documentoCaixa (String):campo para se informar o documento, como um número de cheque. valorCaixa (Double): valor do registro do caixa. acrescimoCaixa (Double): acréscimo do registro. descontoCaixa (Double): desconto do registro. multaCaixa (Double): multa por pagamento em atraso. juroCaixa (Double): juros aplicado por pagamento atrasado. calculoCaixa (Double): campo que irá guardar o saldo do caixa (valor+acréscimo-desconto+multa+juros). descricaoCaixa (String):descrição do registro.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getCaixaManu(List<T> t);

Quadro 17 – Descrição da classe Caixa

No Quadro 18 é apresentada a classe MovCaixa.

Identificação:	MovCaixa
Descrição:	Itens das movimentações do caixa
Requisitos:	RF08
Atributos:	idMovCaixa (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. caixa (Caixa): Vinculo do movimento com a classe Caixa. financeiro (Financeiro): vinculo do movimento com a classe Financeiro. dataMovCaixa (Date): data da movimentação do item. creditoMovCaixa (Double): valor do credito do caixa. debitoMovCaixa (Double):.valor do débito do caixa.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getMovimentosCaixa(T t);

Quadro 18 – Descrição da classe MovCaixa

No Quadro 19 é apresentada a classe Cardapio.

Identificação:	Cardapio
Descrição:	Cadastro dos cardápios do sistema.
Requisitos:	RF05
Atributos:	idCardapio (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeCardapio (String): Nome do cardápio. pizzaCardapio (Integer): campo para definir se o mesmo é do tipo pizza. lancheCardapio (Integer):campo para definir se o mesmo é do tipo lanche. refriCardapio (Integer):campo para definir se o mesmo é do tipo refrigerante. bebidasCardapio (Integer): campo para definir se o mesmo é do tipo bebida. apontamentoCardapio (Integer): código único informado no cadastro que poderá ser utilizado como busca para o cardápio nas telas onde for referenciado.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getCardapio(Integer t);

Quadro 19 – Descrição da classe Cardapio

No Quadro 20 é apresentada a classe MovCardapio.

Identificação:	MovCardapio
Descrição:	Itens das movimentações dos cardápios.
Requisitos:	RF05
Atributos:	idItemCardapio (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. cardapio (Cardapio): Vinculo do item com a classe cardápio. ingredientes (Ingredientes): vínculo do item com a classe Ingredientes. qtdeAproximada (Double): quantidade aproximada que ingrediente terá no item. unidadeItem (String): unidade de medida.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getItensCardapio(T t); List<T> getCardapioManu(List<T> t);

Quadro 20 – Descrição da classe MovCardapio

No Quadro 21 é apresentada a classe Comanda.

Identificação:	Comanda
Descrição:	Registros das comandas retiradas nas mesas junto aos clientes, as comandas formam um pedido.
Requisitos:	RF01
Atributos:	idComanda (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. mesa (Mesas): mesa que a comanda está sendo cadastrada. OrdemComanda (Integer): número sequencial que define a ordem que os itens

	são preparados. situacaoComanda (String): campo para definir a situação atual da comanda. IdPessoa(Integer): campo para definir um código de pessoa quando comanda ‘e cadastrada para entrega. IdEndereco(Integer): Campo para definir um código de endereço vinculado com a pessoa que esta sendo incluído a comanda
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getComandasMesa(T t); List<T> getCardapioManu(List<T> t);

Quadro 21 – Descrição da classe Comanda

No Quadro 22 é apresentada a classe ItensComanda.

Identificação:	ItensComanda
Descrição:	Registros dos produtos contidos nas comandas.
Requisitos:	RF01
Atributos:	idItemComanda (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. comanda (Comanda): vínculo com a classe Comanda. cardapio (Cardapio): vínculo com a classe Cardapio. ingrediente (Ingredientes): vínculo com a classe Ingrediente.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getItensComandas(T t);

Quadro 22 – Descrição da classe ItensComanda

No Quadro 23 é apresentada a classe Financeiro.

Identificação:	Financeiro
Descrição:	Registros dos cadastros de financeiro do sistema.
Requisitos:	RF07
Atributos:	idItemComanda (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. pessoa (Pessoas): vínculo com a classe Pessoa, indica a pessoa a qual o financeiro estará vinculado. pedido (Pedidos): vínculo com a classe Pedido. estoque (Estoque): vínculo com a classe Estoque. numeroFinanceiro (Integer): número sequencial para identificação do registro cadastroFinanceiro (Date): data de cadastro. dataFinanceiro (Date): data de competência do registro. qtdeFinanceiro (Double): somatório das quantidades do movimento. valorFinanceiro (Double): somatório das valores do movimento. calcFinanceiro (Double): cálculo dos campos somatórios do registro. saldoFinanceiro (Double): valor ainda restante, não baixados do sistema. acrescimoFinanceiro (Double): somatório dos acréscimos do movimento.

	dctoFinanceiro (Double): somatório dos descontos do movimento. descricaoFinanceiro (String): campo texto para uma descrição do registro. notaFinanceiro (BigInteger): número da nota fiscal quando o financeiro é de entrada. situacaoFinanceiro (String): situação do registro. serieFinanceiro (String): série da nota. tipoMovimento (TipoMovimento): vínculo do registro com a classe TipoMovimento.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); Integer getNumero(); List<T> getFinanceiroManu(T t);

Quadro 23 – Descrição da classe Financeiro

No Quadro 24 é apresentada a classe MovFinanceiro.

Identificação:	MovFinanceiro
Descrição:	Registros das parcelas do financeiro.
Requisitos:	RF07
Atributos:	idMovFinanceiro (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. financeiro (Financeiro): vínculo com a classe financeiro. vencimentoMov (Date): data do vencimento da parcela. totalMov (Double): valor total da parcela. descontoMov (Double): valor do desconto. acrescimoMov (Double): valor do acréscimo. competenciaMov (Date): data de competência da parcela. saldoMov (Double): saldo da parcela. controleMov (Integer): campo de controle.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getMovFinanceiros(T t);

Quadro 24 – Descrição da classe MovFinanceiro

No Quadro 25 é apresentada a classe Estoque.

Identificação:	Estoque
Descrição:	Registros das itens do estoque.
Requisitos:	RF09
Atributos:	idEstoque (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. numeroEstoque (Integer): número sequencial para identificação do registro. notaEstoque (Integer): número da nota quando registro de entrada. serieEstoque (String): série da nota. pessoa (Pessoas): vínculo com a classe pessoa. ingrediente (Ingredientes): vínculo com a classe Ingrediente.

	tipoMovimento (TipoMovimento): vínculo com a classe TipoMovimento. dataEntradaEstoque (Date): data de entrada do estoque. dataValidadeEstoque (Date): data de validade do item. qtdeEstoque (Double): quantidade do registro. valorEstoque (Double): valor do registro.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getEstoqueManu(List<T> t);

Quadro 25 – Descrição da classe Estoque

No Quadro 26 é apresentada a classe Mesas.

Identificação:	Mesas
Descrição:	Registros das mesas cadastradas no sistema.
Requisitos:	RF12
Atributos:	IdMesa (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. numeroMesa (Integer): número da mesa. lugaresMesa (Integer): quantidade de lugares na mesa.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getMesasManu(List<T> t);

Quadro 26 – Descrição da classe Mesas

No Quadro 27 é apresentada a classe Tabela de Preços.

Identificação:	TabelaPreco
Descrição:	Registros das tabelas de preços cadastradas no sistema.
Requisitos:	RF10
Atributos:	idTabela (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeTabela (String): nome do registro da tabela de preços. numeroTabela (Integer): número sequencial para identificação. dataTabela (Date): data de cadastro do registro. ativoTabela (Integer): marca o registro como ativo. padraoTabela (Integer): define o registro que será utilizado por padrão.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); Integer getNumero(); List<T> getTabelaPrecosManu(List<T> t);

Quadro 27 – Descrição da classe Tabela de Precos

No Quadro 28 é apresentada a classe Itens da Tabela de Preços.

Identificação:	MovTabelaPrecos
Descrição:	Registros itens que compõe as tabelas de preços cadastradas no sistema.
Requisitos:	RF10
Atributos:	idMovTabPreco(número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. tabelaPreco (TabelaPreco): vínculo com a classe Tabela de Precos ingrediente (Ingredientes): vínculo com a classe Ingredientes. cardapio (Cardapio): vínculo com a classe Cardápio qtdeMovTabPreco (Double): quantidade do registro. valorMovTabPreco (Double): valor do registro.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getMovimentosTabelaPreco(T t);

Quadro 28 – Descrição da classe Itens Tabela de Precos

No Quadro 29 é apresentada a classe Tipo de Movimento.

Identificação:	TipoMovimento
Descrição:	Registros itens que compõe as tabelas de preços cadastradas no sistema.
Requisitos:	RF06
Atributos:	idTipoMov(número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. nomeTipoMov (String): nome do registro. descricaoTipoMov (String): descrição do campo, utilizado para validar saldos e valores em outros processos. obsTipoMov (String): observação do registro. ativoTipoMov (Integer): marca se está ativo.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getTipoMovimentoDesc(T t); List<T> getTipoMovimentoManu(List<T> t)

Quadro 29 – Descrição da classe Tipo de Movimento

No Quadro 30 é apresentada a classe Pedidos.

Identificação:	Pedidos
Descrição:	Registros dos pedidos do sistema, um pedido tem vários itens que podem ser compostos por uma ou mais comandas.
Requisitos:	RF02
Atributos:	idPedido(número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. numeroPedido (Integer): número sequencial do registro. valorPedido (Double): valor total do registro. dataPedido (Date): data do registro. situacaoPedido (String): situação atual do registro.

Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); List<T> getPedidosManu(List<T> t)

Quadro 30 – Descrição da classe Pedidos

No Quadro 31 é apresentada a classe Itens dos Pedidos.

Identificação:	MovPedido
Descrição:	Registros dos Itens que compõe um pedido.
Requisitos:	RF02
Atributos:	idItemPedido(número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. pedidos (Pedidos): vínculo com a classe Pedido. comanda (Comanda): vínculo com a classe Comanda.
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 31 – Descrição da classe Itens do Pedido

No Quadro 32 é apresentada a classe de Requisiçao

Identificação:	Requisicao
Descrição:	Registros das requisições para um pedido de compra.
Requisitos:	RF13,RF14
Atributos:	idRequisicao (número): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate. pessoa (Pessoas) : Vinculo com a classe pessoa numeroRequisicao (número) : número seqüencial para controle situacaoRequisicao (String) : situação da requisição conforme etapa dataRequisicao (Date) : data da inclusão da requisição
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 32 – Descrição da classe Requisição

No Quadro 33 é apresentada a classe de Itens da Requisiçao

Identificação:	MovRequisição
Descrição:	Registros dos movimentos de requisições para um pedido de compra.
Requisitos:	RF13,RF14
Atributos:	idMovRequisicao (Integer) : mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate requisicao (Requisicao) :cinculo com a classe requisicao ingrediente (Ingredientes) : vinculo com a classe ingredientes qtdeMovRequisicao (Double) : quantidade de ingredientes no movimento da

	requisição. unidadeMovRequisicao(String) : unidade de medida usada no movimento justMovRequisicao (String) : justificativa para a solicitação da compra infMovRequisicao (String): informações adicionais para o item da requisição situMovRequisicao (String) : Situação do movimento da requisição
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 33 – Descrição da classe Itens da Requisiçào

No Quadro 34 é apresentada a classe de Cotações

Identificação:	Cotacao
Descrição:	Registro das cotações geradas na análise
Requisitos:	RF14,RF15,RF16
Atributos:	idCotacao (Integer): : mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate requisicao (Requisicao) : vinculo com a classe requisicao numeroCotacao(Integer): número da cotação dataCotacao (Date): data do cadastro da cotação fimCotacao (Integer): campo para determinar que a cotação está finalizada requisicao (Requisicao): vinculo com a classe requisição
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); Void<T > cotar();

Quadro 34 – Descrição da classe Cotação

No Quadro 35 é apresentada a classe de Itens Cotações

Identificação:	MovCotacao
Descrição:	Itens das cotações vinculados aos produtos
Requisitos:	RF14, RF15, RF15
Atributos:	idMov(Integer): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate cotacoes(Cotacoes): vinculo com a classe de cotações movRequisicao(MovRequisicao) : vinculo com a classe de itens de requisição
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 35 – Descrição da classe Itens de Cotações

No Quadro 36 é apresentada a classe de Itens das Cotações por Fornecedor.

Identificação:	MovCotacoesFornecedores
Descrição:	Itens das cotações vinculados aos produtos para cada fornecedor
Requisitos:	RF14, RF15, RF15
Atributos:	idMovForn(Integer): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate movCotacoes(MovCotacoes): vinculo com a classe movCotacoes pessoas(Pessoas): vinculo com a classe pessoas ingredientes(Ingredientes): vinculo com a classe ingredientes valorUnitMovForn(Double): valor unitário do item cotado qtdeMovForn(Double): quantidade solicitada na requisicao qtdeCotMovForn(Double): quantidade cotada pelo fornecedor qtdeVencMovForn(Double): quantidade vencida da cotação pelo fornecedor acrescimoMovForn(Double): valor do acrescimo do item descontoMovForn(Double): valor do desconto do item jurosMovForn(Double): valor dos juros do item condicaoMovForn(String): condição de pagamento do item cotado dataMovForn(Date): data da cotação dataUrlMvcf(Date):data para vencimento da url enviada ao fornecedor
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 36 – Descrição da classe Itens das Cotações por Fornecedor

No Quadro 37 é apresentada a classe de Autorizações.

Identificação:	Autorizacao
Descrição:	Autorização das Requisições de compra já cotadas.
Requisitos:	RF17
Atributos:	idAutorizacao (Integer) : mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate movRequisicao(MovRequisicao): vinculo com a classe movRequisicao pessoas(Pessoas): vinculo com a classe pessoas qtdeAutorizacao(Double): quantidade da autorização de compras qtdeCotaAuto(Double): quantidade cotada para a autorização qtdeVencAuto(Double): quantidade vencida para a cotação dataAutorizacao(Date): data da autorização movCotacoesFornecedores(MovCotacoesFornecedores): vinculo com a classe movCotacoesFornecedores
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 37 – Descrição da classe Autorização

No Quadro 38 é apresentada a classe de Pedidos de compra.

Identificação:	PedidoCompras
Descrição:	Registros de pedidos de compra
Requisitos:	RF17, RF18
Atributos:	idPedido(Integer): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate pessoa(Pessoas): vinculo com a classe pessoa situacaoPedido(String): situação do pedido de compra numeroPedido(Integer): numero do pedido de compra para controle tipoMovimento(TipoMovimento): vinculo com a classe tipo de movimento emissaoPedido(Date): data de emissão do pedido entregaPedido(Date): data de entrega do pedido valorPedido(Double): valor total do pedido descontoPedido(Double): desconto do pedido requisicao(Requisicao): vinculo com a classe requisição
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos(); Void<T> gerarPedidos();

Quadro 38 – Descrição da classe Pedidos de Compra

No Quadro 39 é apresentada a classe de Itens de pedido de compra

Identificação:	MovPedidosCompras
Descrição:	Registros dos itens inclusos em um pedido de compra
Requisitos:	RF17, RF18
Atributos:	idMovPedido(Integer): mapear a classe com a anotação "@Entity" ("@Id") para id da classe para o Hibernate pedidoCompra(PedidosCompra): vinculo com a classe pedido de compra ingrediente(Ingredientes): vinculo com a classe ingrediente unidMovPedido(String): unidade de medida do movimento do pedido qtdeMovPedido(Double): quantidade do item no movimento do pedido vlUnitMovPedido(Double): valor unitário do movimento do pedido valorMovPedido(Double): valor total do movimento do pedido descontoMovPedido(Double): desconto do movimento do pedido estoquePedidoCompra(EstoquePedidoCompra): vinculo com a classe estoque pedido compra
Métodos:	String inclui(T t); String atualiza(T t); String exclui(T t); List<T> getTodos();

Quadro 34 – Descrição da classe Itens da Requisiçào

A Figura 3 apresenta o digrama de entidades e relacionamentos do banco de dados.

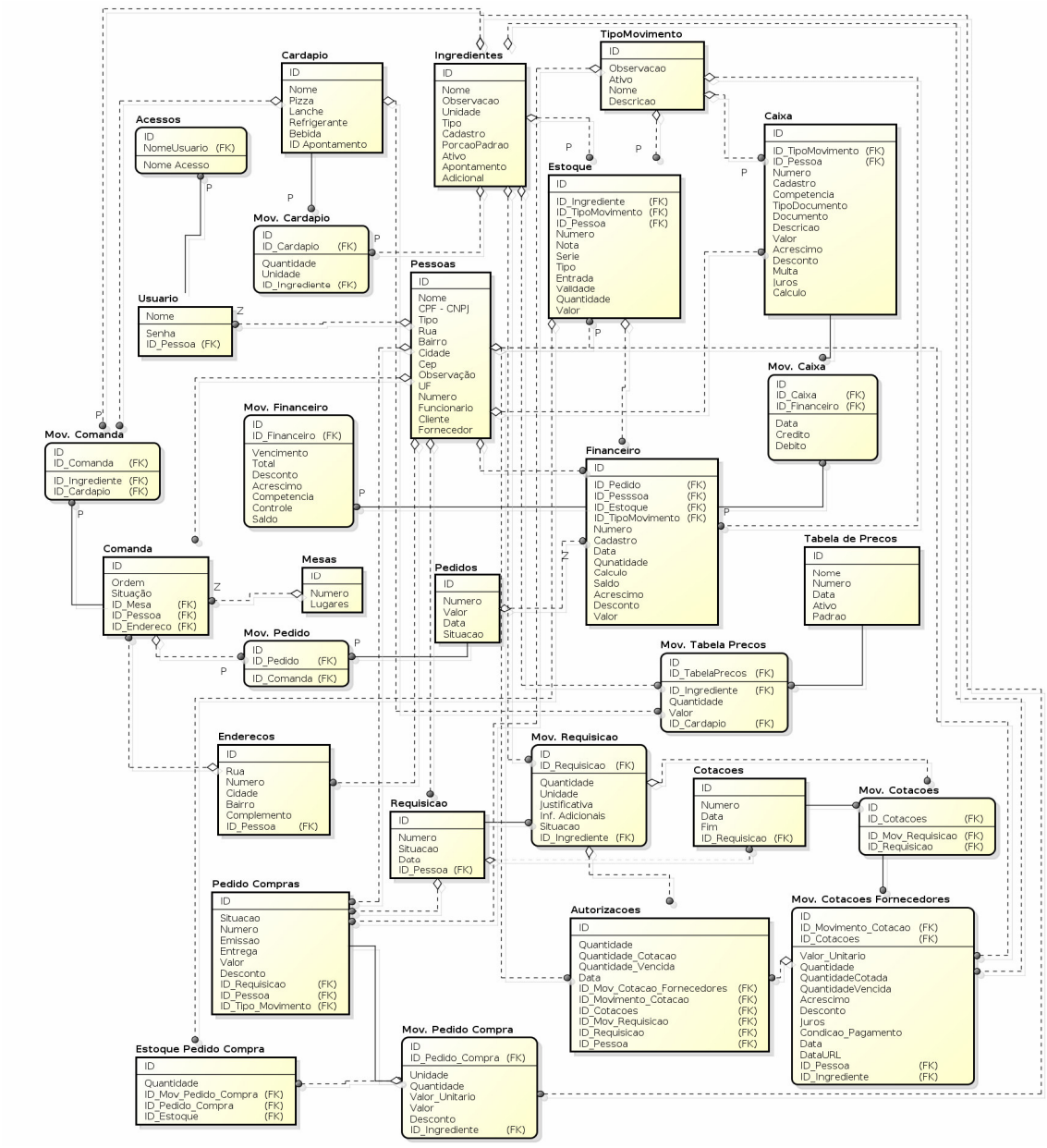


Figura 3 - Diagrama de entidades e relacionamento do banco de dados

No Quadro 31 estão os campos da tabela Usuário.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID_Pessoa	Numérico	Não	Sim	Não	
Nome	Texto	Não	Não	Não	
Senha	Texto	Não	Não	Não	

Quadro 32 – Campos da tabela Usuario

No Quadro 33 estão os campos da tabela Acessos.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
NomeUsuario	Texto	Não	Não	Sim	
NomeAcesso	Numérico	Não	Não	Não	

Quadro 33 – Campos da tabela Usuario

No Quadro 34 estão os campos da tabela Cardapio.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Apontamento	Numérico	Não	Não	Não	Campo com valor único na tabela, utilizado para facilitar em pesquisas. É definido pelos usuários.
Nome	Texto	Não	Não	Não	
Pizza	Numérico	Não	Não	Não	
Lanche	Numérico	Não	Não	Não	
Refrigerante	Numérico	Não	Não	Não	
Bebida	Numérico	Não	Não	Não	

Quadro 34 – Campos da tabela Cadapio

No Quadro 35 estão os campos da tabela Mov.Cardapio.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
IDCardapio	Numérico	Não	Não	Sim	Vem da tabela Cardapio
IDIngrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes
Quantidade	Numérico	Não	Não	Não	
Unidade	Numérico	Não	Não	Não	

Quadro 35 – Campos da tabela Mov. Cardapio

No Quadro 36 estão os campos da tabela Ingredientes.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Nome	Texto	Não	Não	Não	

Observação	Texto	Não	Não	Não	
Unidade	Numérico	Não	Não	Não	
Tipo	Numérico	Não	Não	Não	
Cadastro	Numérico	Não	Não	Não	
PorcaoPadrao	Numérico	Não	Não	Não	
Ativo	Numérico	Não	Não	Não	
Apontamento	Numérico	Não	Não	Não	Campo com valor único na tabela, utilizado para facilitar em pesquisas. É definido pelos usuários.
Adicional	Numérico	Não	Não	Não	

Quadro 36 – Campos da tabela Ingredientes

No Quadro 37 estão os campos da tipoMovimento.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
Observação	Texto	Não	Não	Não	
Ativo	Numérico	Não	Não	Não	
Nome	Texto	Não	Não	Não	
Descrição	Texto	Não	Não	Não	

Quadro 37 – Campos da tabela tipoMovimento

No Quadro 38 estão os campos da caixa.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Tipo_Movimento	Numérico	Não	Não	Sim	Vem da tabela TipoMovimento
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoas
Numero	Numérico	Não	Não	Não	
Cadastro	Numérico	Não	Não	Não	
Competencia	Numérico	Não	Não	Não	
TipoDocumetno	Numérico	Não	Não	Não	
Documento	Texto	Não	Não	Não	
Descricao	Texto	Não	Não	Não	
Valor	Numérico	Não	Não	Não	
Acrescimo	Numérico	Não	Não	Não	
Desconto	Numérico	Não	Não	Não	

Multa	Numérico	Não	Não	Não	
Juros	Numérico	Não	Não	Não	
Calculo	Numérico	Não	Não	Não	

Quadro 38 – Campos da tabela Caixa

No Quadro 39 estão os campos da tabela Estoque.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Ingrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes
ID_TipoMovimento	Numérico	Não	Não	Sim	Vem da tabela TipoMovimento
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoa
Numero	Numérico	Não	Não	Não	
Nota	Numérico	Não	Não	Não	
Serie	Texto	Não	Não	Não	
Tipo	Numérico	Não	Não	Não	
Entrada	Numérico	Não	Não	Não	
Validade	Data	Não	Não	Não	
Quantidade	Numérico	Não	Não	Não	
Valor	Numérico	Não	Não	Não	

Quadro 39 – Campos da tabela Estoque

No Quadro 40 estão os campos da tabela Pessoas.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
Nome	Texto	Não	Não	Não	
CPF_CNPJ	Texto	Não	Não	Não	
Tipo	Numérico	Não	Não	Não	
Rua	Texto	Não	Não	Não	
Bairro	Texto	Não	Não	Não	
Cidade	Texto	Não	Não	Não	
Cep	Texto	Não	Não	Não	
Observacao	Texto	Não	Não	Não	
UF	Texto	Não	Não	Não	
Numero	Numérico	Não	Não	Não	

Funcionario	Numérico	Não	Não	Não	
Cliente	Numérico	Não	Não	Não	
Fornecedor	Numérico	Não	Não	Não	

Quadro 40 – Campos da tabela Pessoas

No Quadro 41 estão os campos da tabela Mov.Caixa.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Caixa	Numérico	Não	Não	Sim	Vem da tabela Caixa
ID_Financeiro	Numérico	Não	Não	Sim	Vem da tabela Financeiro
Data	Data	Não	Não	Não	
Debito	Numérico	Não	Não	Não	
Credito	Numérico	Não	Não	Não	

Quadro 41 – Campos da tabela Mov.Caixa

No Quadro 42 estão os campos da tabela Financeiro.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Pedido	Numérico	Não	Não	Sim	Vem da tabela Pedido
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoa
ID_Estoque	Numérico	Não	Não	Sim	Vem da tabela Estoque
ID_TipoMovimento	Numérico	Não	Não	Sim	Vem da tabela TipoMovimento
Numero	Numérico	Não	Não	Não	
Cadastro	Texto	Não	Não	Não	
Data	Data	Não	Não	Não	
Quantidade	Numérico	Não	Não	Não	
Calculo	Numérico	Não	Não	Não	
Saldo	Numérico	Não	Não	Não	
Acrescimo	Numérico	Não	Não	Não	
Desconto	Numérico	Não	Não	Não	
Valor	Numérico	Não	Não	Não	

Quadro 42 – Campos da tabela Financeiro

No Quadro 43 estão os campos da tabela Mov. Financeiro.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Financeiro	Numérico	Não	Não	Sim	Vem da tabela Financeiro
Vencimento	Data	Não	Não	Não	
Total	Numérico	Não	Não	Não	
Desconto	Numérico	Não	Não	Não	
Acrescimo	Numérico	Não	Não	Não	
Competencia	Numérico	Não	Não	Não	
Controle	Numérico	Não	Não	Não	
Saldo	Numérico	Não	Não	Não	

Quadro 43 – Campos da tabela Mov. Financeiro

No Quadro 44 estão os campos da tabela Mov. Comanda.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Comanda	Numérico	Não	Não	Sim	Vem da tabela Comanda
ID_Ingrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes
ID_Cardapio	Numérico	Não	Não	Sim	Vem da tabela Cardapio

Quadro 44 – Campos da tabela Mov. Comanda

No Quadro 45 estão os campos da tabela Comanda.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Mesa	Numérico	Não	Não	Sim	Vem da tabela Mesas
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoas
ID_Endereco	Numérico	Não	Não	Sim	Vem da tabela Enderecos
Ordem	Numérico	Não	Não	Não	
Situação	Numérico	Não	Não	Não	

Quadro 45 – Campos da tabela Comanda

No Quadro 46 estão os campos da tabela Mesas.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Numero	Numérico	Não	Não	Não	
Lugares	Numérico	Não	Não	Não	

Quadro 46 – Campos da tabela Mesas

No Quadro 46 estão os campos da tabela Pedidos.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Numero	Numérico	Não	Não	Não	
Valor	Numérico	Não	Não	Não	
Data	Data	Não	Não	Não	
Situacao	Numérico	Não	Não	Não	

Quadro 46 – Campos da tabela Pedidos

No Quadro 47 estão os campos da tabela Mov. Pedido.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Pedido	Numérico	Não	Sim	Sim	Vem da tabela Pedidos
ID_Comanda	Numérico	Não	Não	Sim	Vem da tabela Comanda

Quadro 47 – Campos da tabela Mov. Pedido

No Quadro 48 estão os campos da tabela Precos.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Nome	Texto	Não	Não	Não	
Numero	Numérico	Não	Não	Não	
Data	Data	Não	Não	Não	
Ativo	Numérico	Não	Não	Não	
Padrao	Numérico	Não	Não	Não	

Quadro 48 – Campos da tabela Precos

No Quadro 49 estão os campos da tabela Mov. Tabela Precos.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Tabela_Precos	Numérico		Sim	Sim	Vem da tabela Tabela de Precos
ID_Ingrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes
ID_Cardapio	Numérico	Não	Não	Sim	Vem da tabela Cardapio
Valor	Numérico	Não	Não	Não	
Quantidade	Numérico	Não	Não	Não	

Quadro 49 – Campos da tabela Mov. Tabela Precos

No quadro 50 estão os campos da tabela Requisição.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Numero	Numérico	Não	Não	Não	
Situacao	Texto	Sim	Não	Não	
Data	Data	Sim	Não	Não	
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela de Pessoas

Quadro 50 – Campos da tabela Requisicao

No quadro 51 estão os campos da tabela Mov. Requisição.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Requisicao	Numérico	Não	Não	Sim	Vem da tabela Requisicao
Quantidade	Numérico	Não	Não	Não	
Justificativa	Texto	Sim	Não	Não	
Inf. Adicionais	Texto	Sim	Não	Não	
Situacao	Texto	Não	Não	Não	
ID_Ingrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes

Quadro 51 – Campos da tabela Mov. Requisicao

No quadro 52 estão os campos da tabela Cotações.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Numero	Numérico	Não	Não	Não	

Data	Data	Sim	Não	Não	
Fim	Numérico	Sim	Não	Não	
ID_Requisicao	Numérico	Não	Não	Sim	Vem da tabela Requisicao

Quadro 52 – Campos da tabela Cotacoes

No quadro 53 estão os campos da tabela Mov. Cotações.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Cotacoes	Numérico	Não	Não	Sim	Vem da tabela Cotacoes
ID_MovRequisicoes	Numérico	Não	Não	Sim	Vem da tabela Mov. Requsicoes
ID_Requisicao	Numérico	Não	Não	Sim	Vem da tabela Requisicoes

Quadro 53 – Campos da tabela Mov. Cotacoes

No quadro 54 estão os campos da tabela Mov. Cotacoes Fornecedores.

Campo	**Tipo**	**Nulo**	**Chave primária**	**Chave estrangeira**	**Observações**
ID	Numérico	Não	Sim	Não	
ID_Mov_Cotacoes	Numérico	Não	Não	Sim	Vem da tabela Mov. Cotacoes
ID_Cotacoes	Numérico	Não	Não	Sim	Vem da tabela Cotacoes
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoas
ID_Ingredientes	Numérico	Não	Não	Sim	Vem da tabela Ingredientes
ValorUnitario	Numérico	Não	Não	Não	
Quantidade	Numérico	Não	Não	Não	
QuantidadeCotada	Numérico	Não	Não	Não	
QuantidadeVencida	Numérico	Não	Não	Não	
Acrescimo	Numérico	Sim	Não	Não	
Desconto	Numérico	Sim	Não	Não	
Juros	Numérico	Sim	Não	Não	
CondicaoPagamento	Texto	Não	Não	Não	
Data	Data	Não	Não	Não	
DataURL	Data	Sim	Não	Não	

Quadro 54 – Campos da tabela Mov. Cotacoes Fornecedores

No quadro 55 estão os campos da tabela Autorizacoes

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Quantidade	Numérico	Não	Não	Não	
QuantidaeCotada	Numérico	Sim	Não	Não	
QuantidadeVencida	Numérico	Sim	Não	Não	
Data	Data	Não	Não	Não	
ID_MovCotacaoFornecedores	Numérico	Não	Não	Sim	Vem da tabela Mov. CotacaoFornecedores
ID_MovCotacao	Numérico	Não	Não	Sim	Vem da tabela Mov.Cotacao
ID_Cotacao	Numérico	Não	Não	Sim	Vem da tabela Cotacoes
ID_MovRequisicao	Numérico	Não	Não	Sim	Vem da tabela Mov. Requisicao
ID_Requisicao	Numérico	Não	Não	Sim	Vem da tabela Requisicao
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela Pessoas

Quadro 55 – Campos da tabela Autorizacoes

No quadro 56 estão os campos da tabela Pedido Compra

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Situacao	Texto	Não	Não	Não	
Numero	Numérico	Não	Não	Não	
Entrega	Data	Não	Não	Não	
Valor	Numérico	Não	Não	Não	
Desconto	Numérico	Sim	Não	Não	
ID_Requisicao	Numérico	Não	Não	Sim	Vem da tabela Requisição
ID_Pessoa	Numérico	Não	Não	Sim	Vem da tabela de Pessoas
ID_TipoMovimento	Numérico	Não	Não	Sim	Vem da tabela TipoMovimento

Quadro 56 – Campos da tabela Pedido Compra

No quadro 57 estão os campos da tabela Mov. Pedido Compra.

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
ID_Pedido_Compra	Numérico	Não	Não	Sim	Vem da tabela Pedido Compra
Unidade	Texto	Não	Não	Não	
Quantidade	Numérico	Não	Não	Não	
ValorUnitario	Numérico	Não	Não	Não	
Valor	Numérico	Não	Não	Não	
Desconto	Numérico	Sim	Não	Não	
ID_Ingrediente	Numérico	Não	Não	Sim	Vem da tabela Ingredientes

Quadro 57 – Campos da tabela Mov. Pedido Compra

No quadro 58 estão os campos da tabela Mov. Estoque por Pedido Compra

Campo	Tipo	Nulo	Chave primária	Chave estrangeira	Observações
ID	Numérico	Não	Sim	Não	
Quantidade	Numérico	Não	Não	Não	
ID_MovPedidoCompra	Numérico	Não	Não	Sim	Vem da tabela Mov. Pedido Compra
ID_PedidoCompra	Numérico	Não	Não	Sim	Vem da tabela Pedido Compra
ID_Estoque	Numérico	Não	Não	Sim	Vem da tabela Estque

Quadro 58 – Campos da tabela Mov. Estoque Pedido Compra

4.3 APRESENTAÇÃO DO SISTEMA

A tela inicial do sistema é a tela de *login* apresentada na Figura 4, por meio desta realizado o acesso ao sistema.

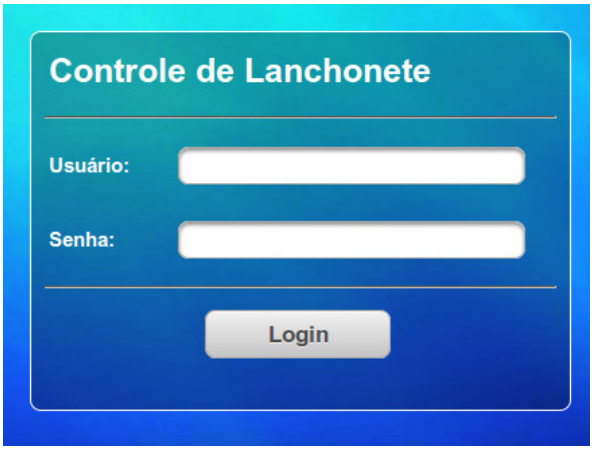


Figura 4 - Login do sistema

Ao efetuar *login* é feito uma validação de usuário e senha, caso as informações estejam incorretas o usuário é informado por meio de mensagem na tela que as informações estão incorretas, como mostra a Figura 5.

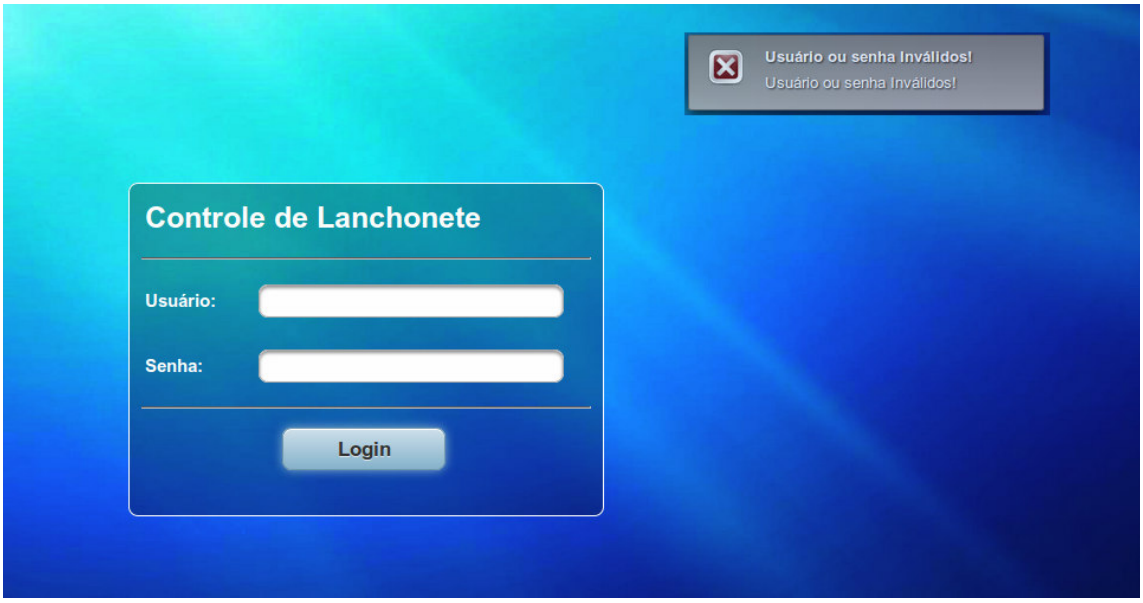


Figura 5 - Login inválido

Ao efetuar *login* com usuário e senha corretos, o usuário é direcionado para a página principal (*home)* (Figura 6).

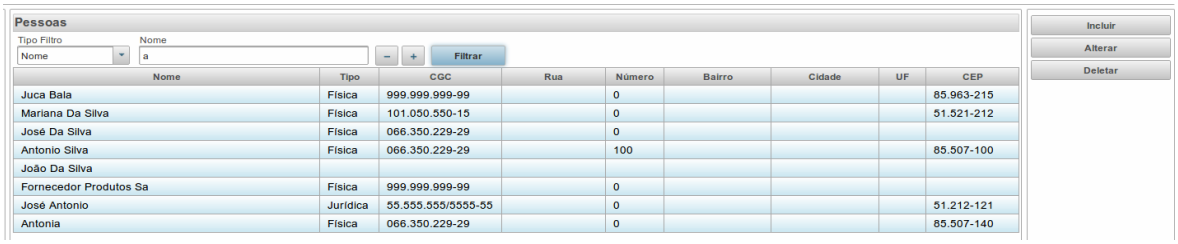
Figura 6 - Tela principal do sistema

O leiaute do sistema é composto por cinco setores: o setor superior que contém a logo da empresa; o setor lateral esquerdo que contém o menu de navegação; o setor central com o conteúdo da página que está sendo navegada; o setor lateral direito que apresenta os controles dos formulários; e o setor inferior que contém o rodapé com algumas informações. A Figura 7 apresenta os cinco setores nos quais a página está organizada.

Figura 7 - Setores da página

A Figura 8 mostra o setor central com o conteúdo da página de manutenção de pessoas

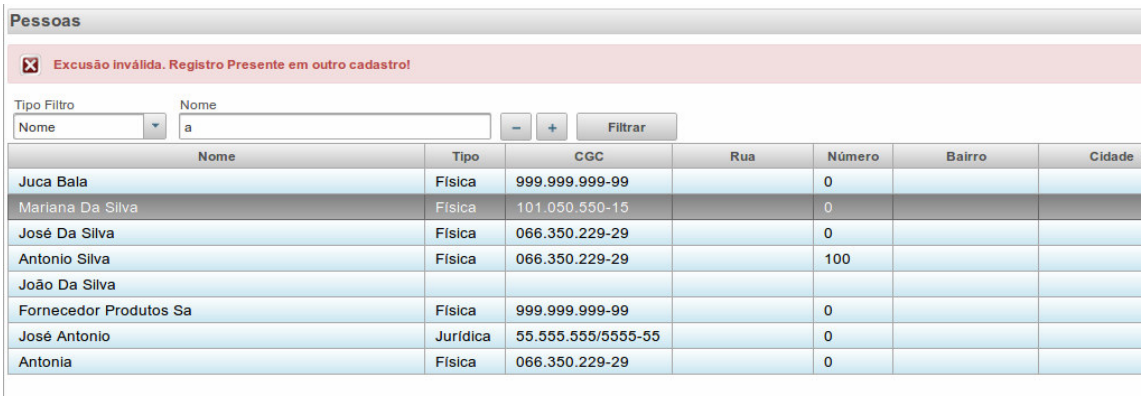
juntamente com o setor lateral direito com os controles da página. No setor central podem ser filtrados os itens já cadastrados, podendo adicionar mais filtros à seleção clicando no botão mais (+) ou remover filtros clicando no botão menos (-). No setor lateral direito podem ser realizadas as operações básicas de cadastro como incluir, alterar e excluir.



Nome	Tipo	CGC	Rua	Número	Bairro	Cidade	UF	CEP
Juca Bala	Física	999.999.999-99		0				85.963-215
Mariana Da Silva	Física	101.050.550-15		0				51.521-212
José Da Silva	Física	066.350.229-29		0				
Antonio Silva	Física	066.350.229-29		100				85.507-100
João Da Silva								
Fornecedor Produtos Sa	Física	999.999.999-99		0				
José Antonio	Jurídica	55.555.555/5555-55		0				51.212-121
Antonia	Física	066.350.229-29		0				85.507-140

Figura 8 - Manutenção de pessoas

A Figura 9 mostra uma restrição ao tentar excluir um registro que já foi utilizado em algum outro cadastro do sistema.



Nome	Tipo	CGC	Rua	Número	Bairro	Cidade
Juca Bala	Física	999.999.999-99		0		
Mariana Da Silva	Física	101.050.550-15		0		
José Da Silva	Física	066.350.229-29		0		
Antonio Silva	Física	066.350.229-29		100		
João Da Silva						
Fornecedor Produtos Sa	Física	999.999.999-99		0		
José Antonio	Jurídica	55.555.555/5555-55		0		
Antonia	Física	066.350.229-29		0		

Figura 9 - Restrição de exclusão do registro

Na Figura 10 está a tela de inclusão de pessoas com a marcação de um asterisco ao lado dos campos obrigatórios.

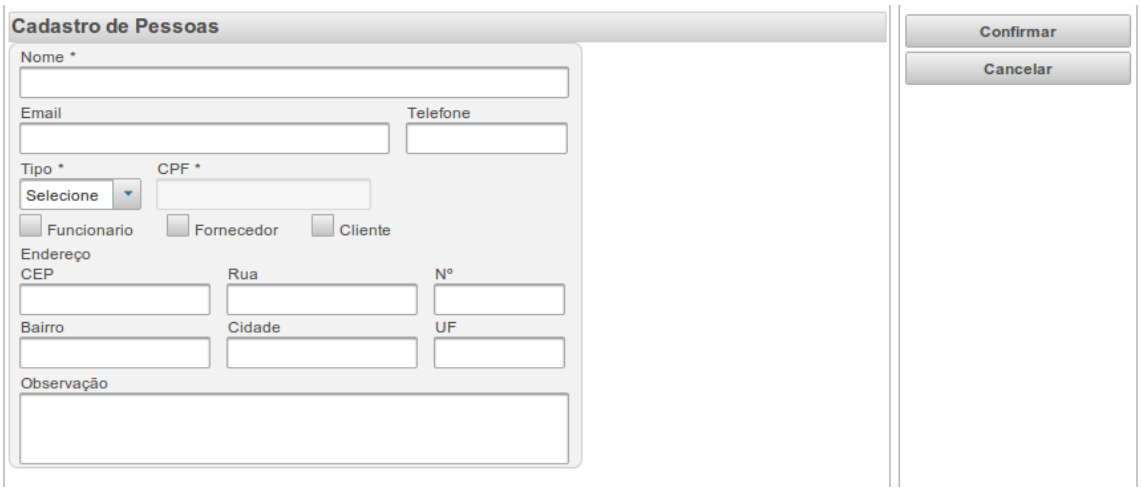


Figura 10 - Inclusão de pessoas

A Figura 11 mostra uma validação para os campos obrigatórios que não foram informados na tela, destacando os mesmos com uma borda vermelha.

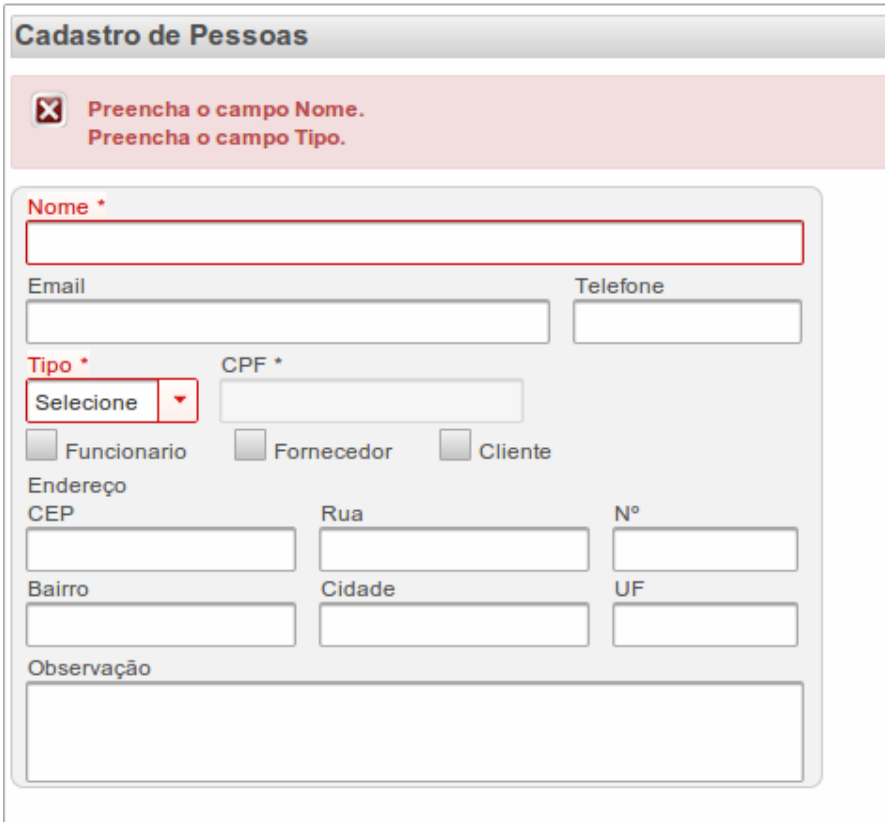


Figura 11 - Campos obrigatórios cadastro pessoas

A Figura 12 mostra uma validação em relação ao CPF ou CNPJ informado para o

cadastro da pessoa, caso o mesmo seja inválido apresenta a mensagem na tela.

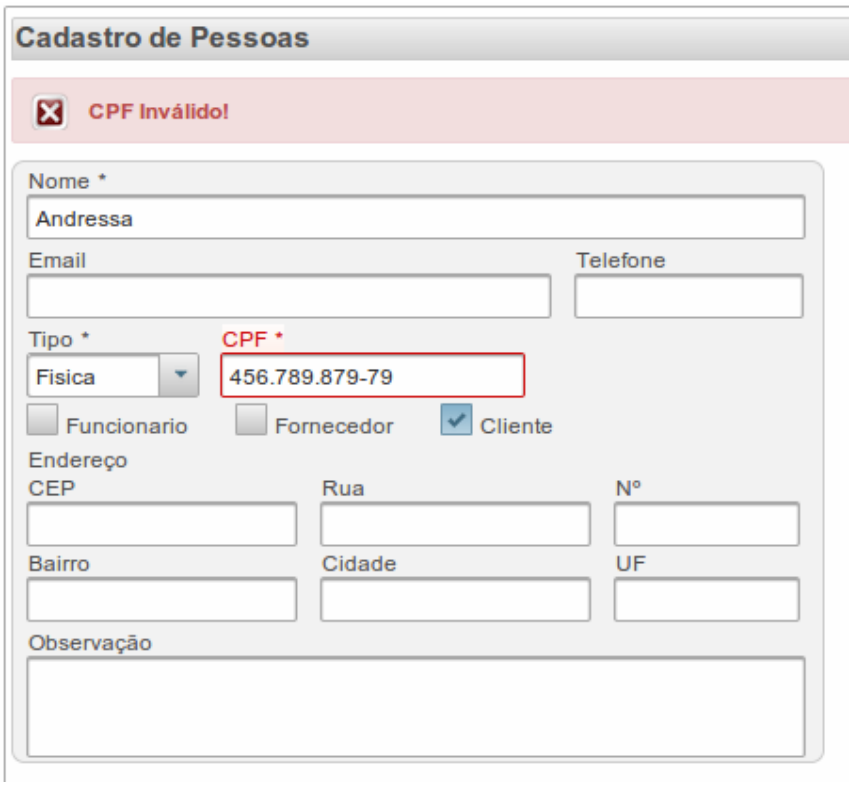


Figura 12 - Validação CPF-CNPJ cadastro pessoas

Ao confirmar a inserção do registro de um novo cadastro de pessoas, a tela mostra uma mensagem informando que o registro foi incluído com sucesso, como mostra a Figura 13.

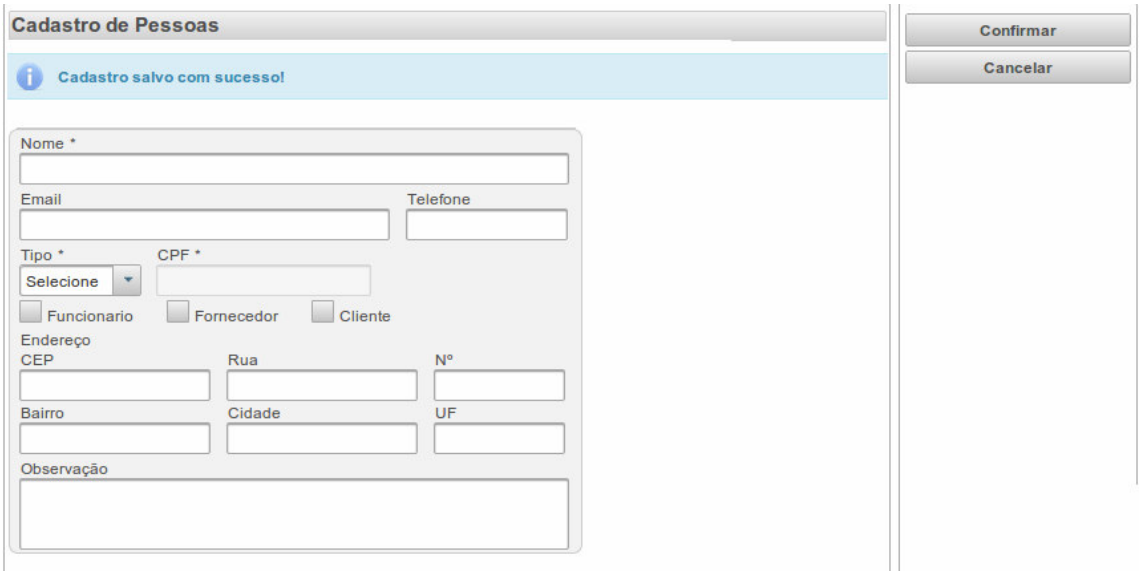


Figura 13 - Registro de pessoa incluído com sucesso

A Figura 14 mostra o conteúdo da página de manutenção de ingredientes, apresentado o *grid* com a listagem dos produtos já cadastrados.

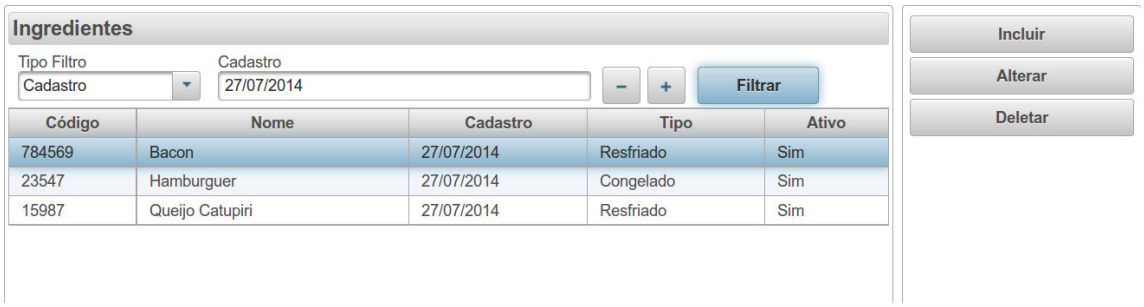


Figura 14 – Lista de ingredientes

Ao clicar no botão incluir, o sistema direcionará para a página de cadastro apresentando os campos necessários para a inclusão de um ingrediente, juntamente com os botões confirmar e cancelar e o menu lateral no setor lateral esquerdo. A navegação entre os menus está disponível em todos as operações realizadas. A Figura 15 apresenta a tela de cadastro de ingredientes.

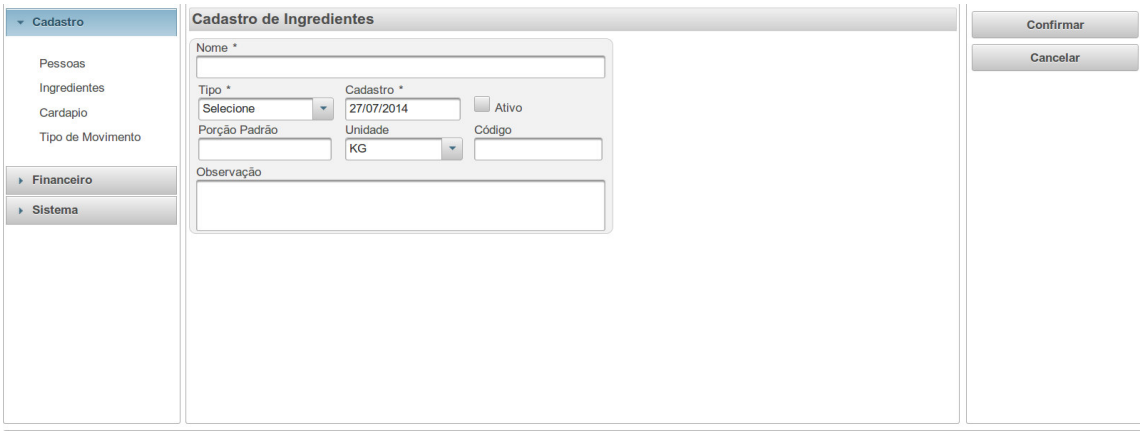


Figura 15 – Cadastro de ingredientes

Se o usuário clicar no botão “Confirmar” sem preencher os campos obrigatórios, o sistema mostra na tela uma mensagem informando os campos obrigatórios não preenchidos. Na Figura 16 é apresentada a mensagem localizada na parte superior do formulário. E os campos que possuem preenchimento obrigatório também são destacados em cor vermelha. Por padrão, todos os campos obrigatórios dos formulários são marcados com um asterisco.

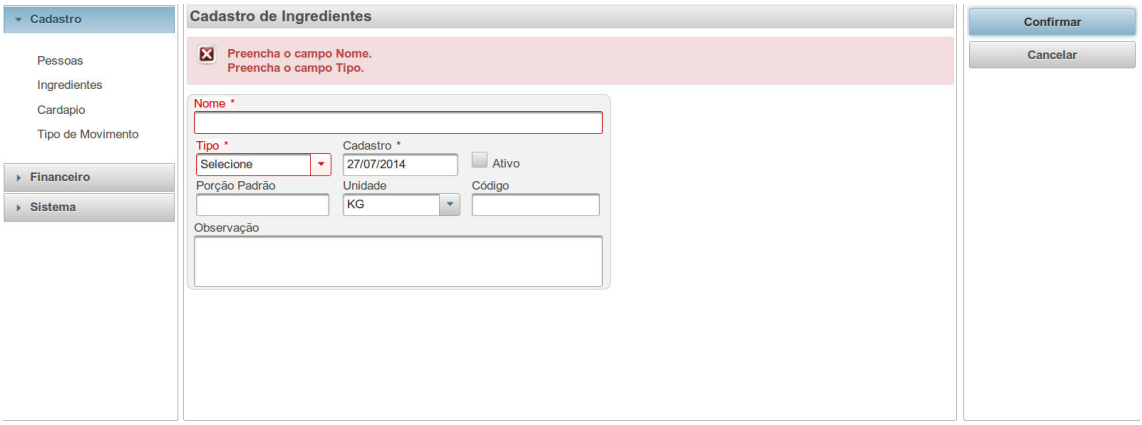


Figura 16 – Exemplo de validação de campos de preenchimento obrigatórios

Se todos os campos forem validados, o sistema gravará o novo ingrediente e mostrará uma mensagem de sucesso, como mostra a Figura 17. A mensagem é apresentada na parte superior do formulário, mesma localização da mensagem de campos obrigatórios e outras. No caso de operação realizada com sucesso o ícone indicador da mensagem e a cor da mesma são diferentes.

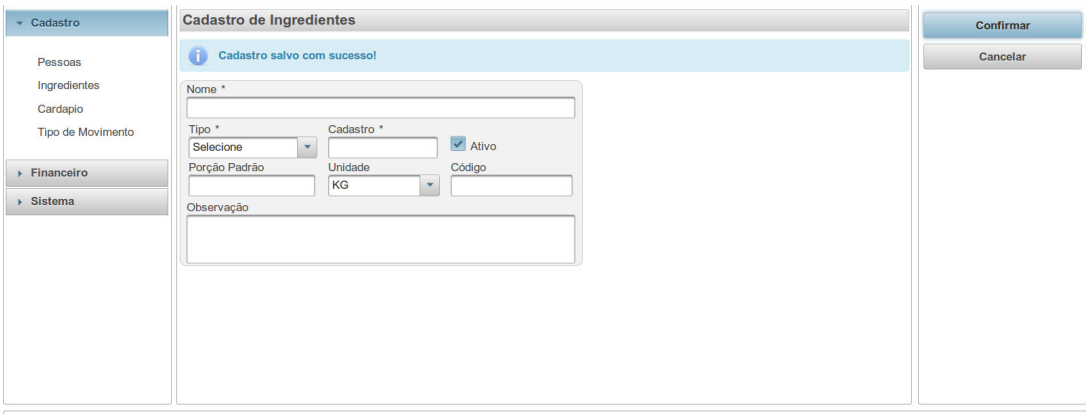


Figura 17 – Cadastro de Ingrediente realizado

A operação de alteração é semelhante à operação de inclusão, sendo que para alterar um registro o usuário deve selecionar o item no *grid* e clicar no botão Alterar. A Figura 18 apresenta um item selecionado no *grid*. Um item selecionado pode ser excluído ou alterado, como indicam os respectivos botões na lateral direita do formulário.

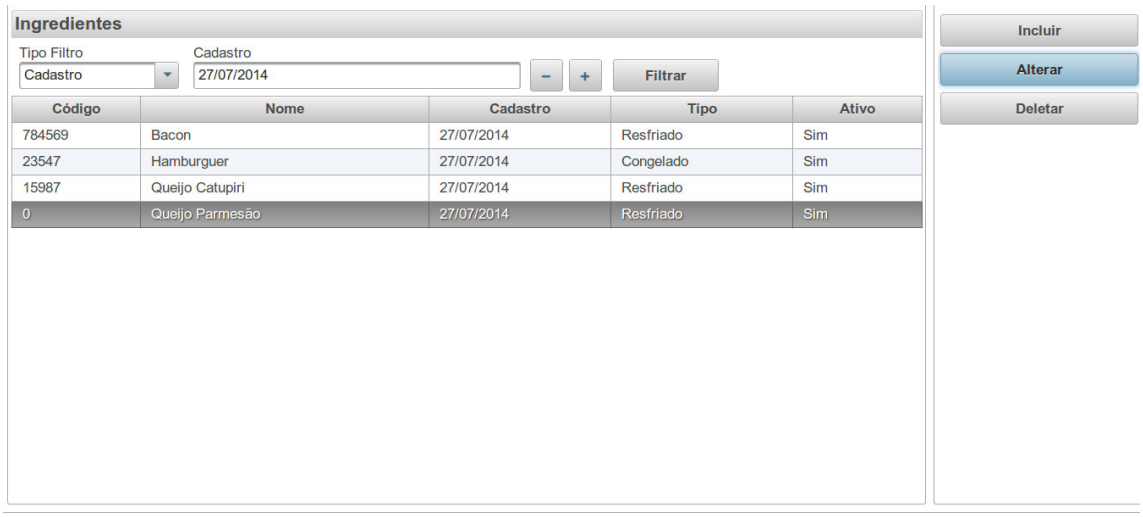


Figura 18 – Procedimento para atualização

Se clicado no botão “Alterar” os dados do registro selecionado no *grid* são carregados e o formulário é aberto para edição. O usuário pode realizar alterações nos dados, confirmando a alteração caso queira que a atualização seja realizada. O botão “Cancelar” faz o retorno para a página anterior e as alterações que possam ter sido realizadas são desconsideradas. A Figura 19 apresenta a edição de um cadastro de ingredientes.

Figura 19 – Tela de alteração de ingrediente

Os tratamentos feitos pelo sistema na alteração são os mesmos da inclusão e é realizado o mesmo processo. A única diferença é que ao confirmar a alteração, o sistema redireciona para a página da manutenção de ingredientes com o valor que foi atualizado correto, como mostra a Figura 20.

Figura 20 – Manutenção de ingredientes já com o valor atualizado

Para realizar a operação de exclusão, o usuário deve selecionar um registro no *grid* e clicar no botão “Excluir”. O sistema apresentará uma mensagem de confirmação avisando o usuário que a operação não poderá ser desfeita, como mostra a Figura 21. A mensagem é apresentada em uma janela sobreposta: o usuário deve aceitar ou cancelar a exclusão.

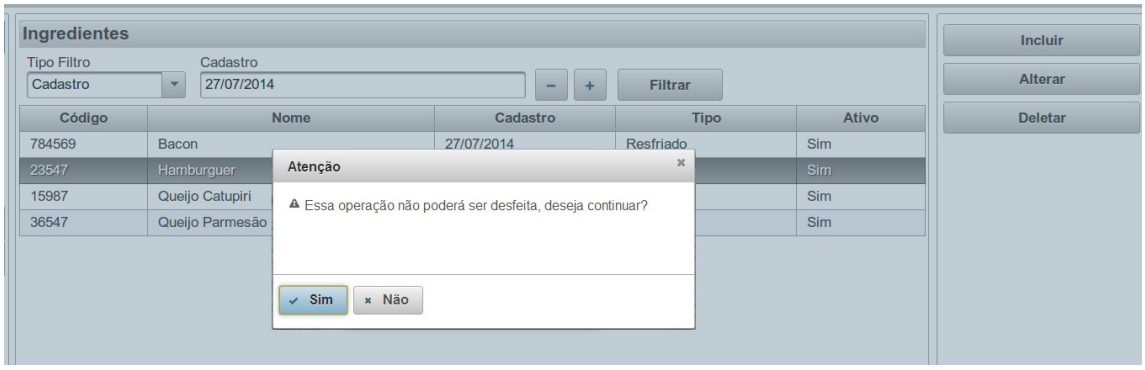


Figura 21 – Operação de exclusão

Se o usuário confirmar a exclusão e o registro não estiver sendo usado em nenhum outro registro, o sistema removerá o registro selecionado e atualizará o *grid* com os itens como mostra a Figura 22.

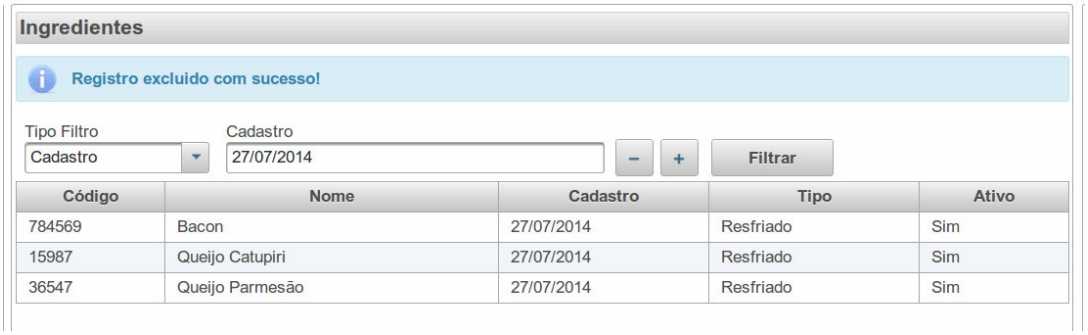


Figura 22 – Operação de exclusão

Caso o registro esteja sendo utilizado por outra tabela no banco de dados e o vinculo do mesmo para exclusão não seja em cascata, o sistema emitirá um alerta informando que o mesmo não pode ser excluído, como mostra a Figura 23.

Figura 23 – Operação de exclusão

As validações e mensagens apresentadas nos processos de alterar e excluir registros são padrão e mostradas em todas as telas de manutenção do sistema.

A Figura 24 mostra a tela de manutenção de cardápio. Essa manutenção é diferente das demais, apresenta os registros dos movimentos agrupados pelos cabeçalhos.

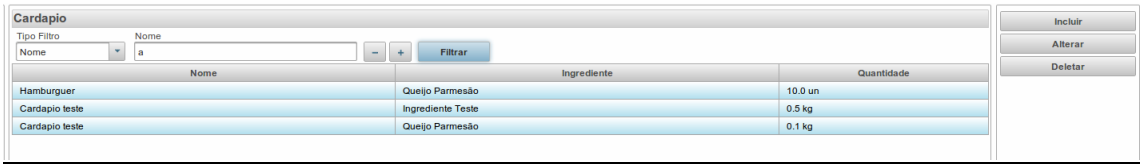


Figura 24 - Manutenção cardápio

Após clicar no botão incluir ou alterar da manutenção, o usuário é direcionado para a tela de cadastro de cardápio. Os campos obrigatórios vem marcados com um asterisco por padrão. Essa tela possui um novo formato, pois grava itens em um cabeçalho e para esse cabeçalho, podem ser informados ‘n’ movimentos, informando os valores na parte inferior da tela e clicando em enviar, como apresentado na Figura 25.

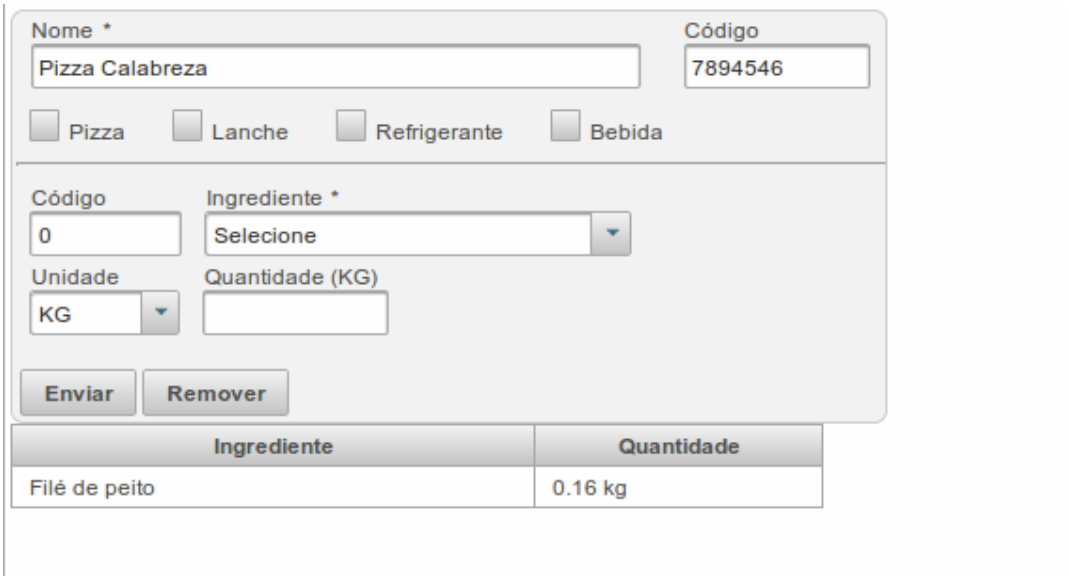


Figura 25 - Cadastro de cardápio

Caso seja efetuado clique no botão Confirmar sem informar os dados o obrigatórios do cabeçalho é apresentada uma mensagem como mostra a Figura 26.

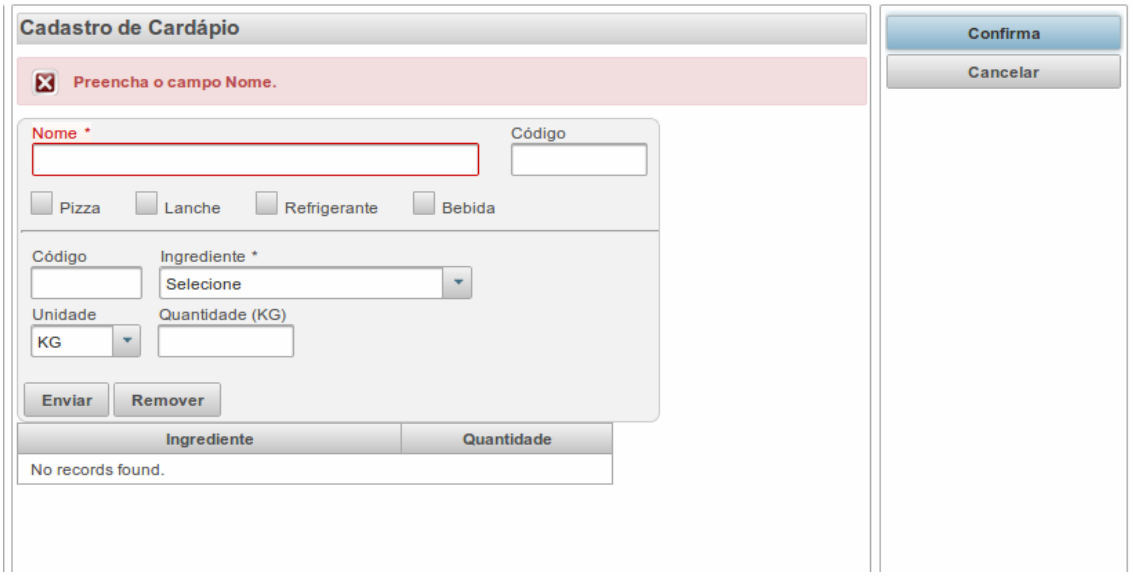


Figura 26 - Cadastro de cardápio campos obrigatórios

A Figura 27 mostra a validação dos campos obrigatórios do movimento da tela. Essa divisão de obrigatoriedade só é possível com o atributo *process* do JSF. Como a tela é dividida em duas partes, o botão confirma processa a parte do cabeçalho e o botão enviar processa a parte do movimento.

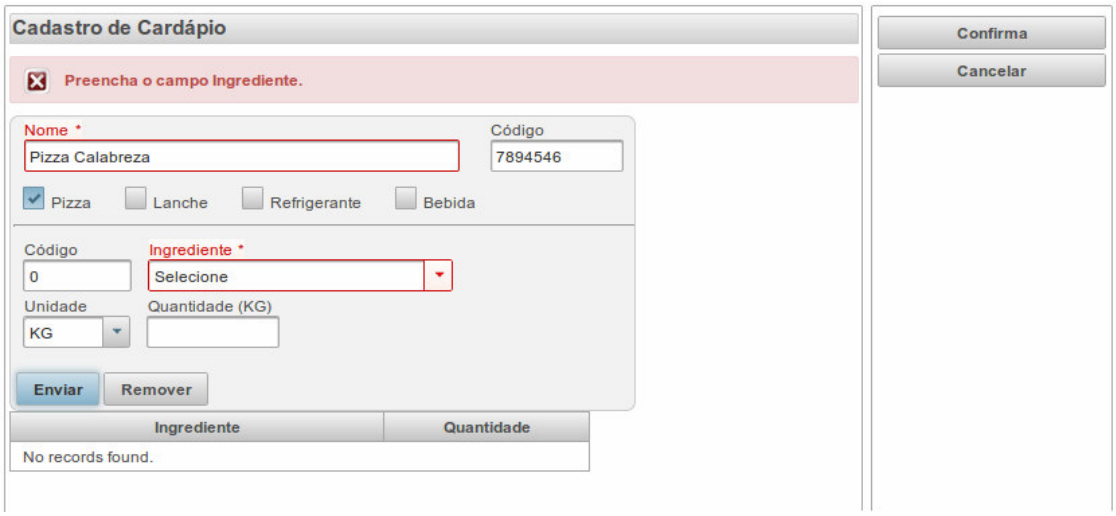


Figura 27 - Validação dos campos obrigatórios no movimento do cardápio

A Figura 28 mostra a manutenção da tela de tipos de movimentos, os quais são utilizados para definir os tipos das operações internas no sistema.

Figura 28 - Manutenção tipo de movimento

A tela de cadastro de tipo de movimento ao confirmar sem informar os campos obrigatórios é mostrada na Figura 29.

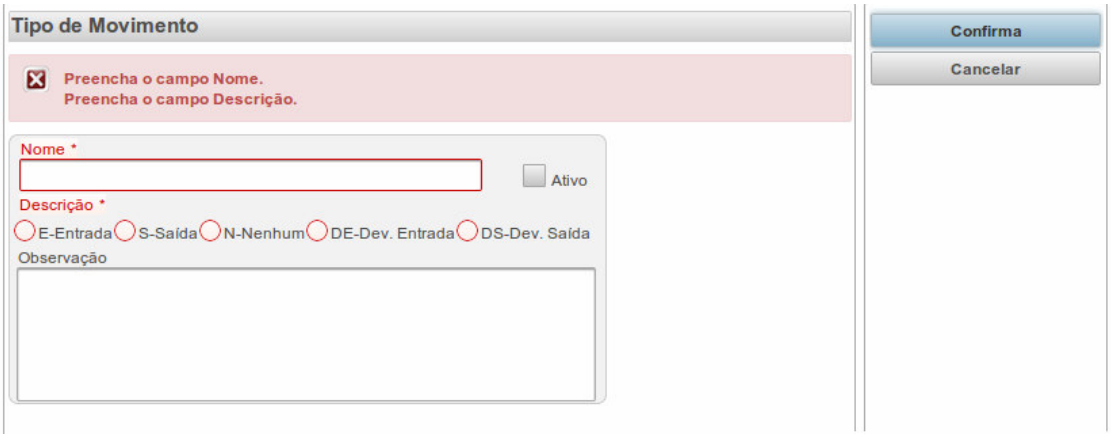


Figura 29 - Cadastro tipo de movimento campos obrigatórios

Ao confirmar estando as informações corretamente preenchidas, é informado ao usuário que o registro foi incluído com sucesso, como mostra a Figura 30.

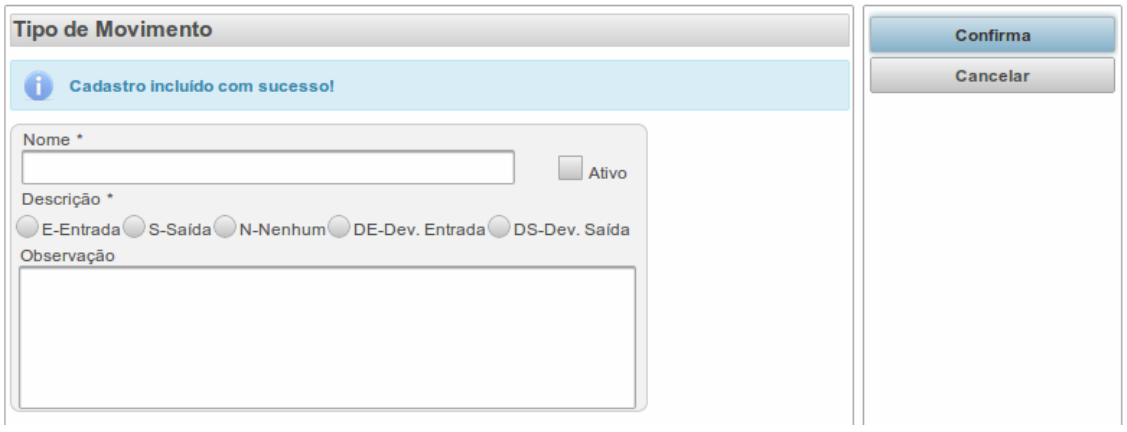
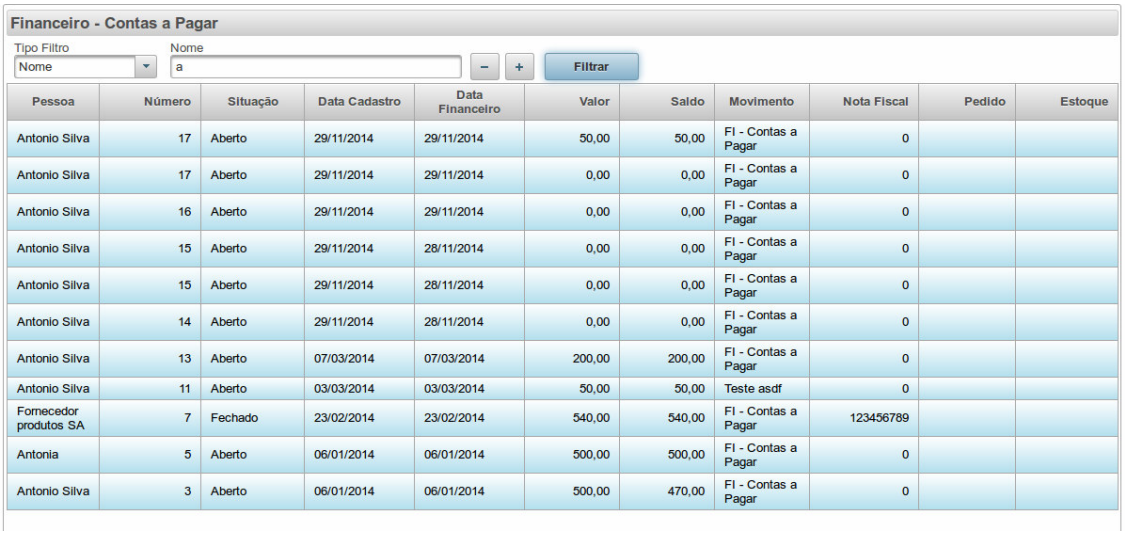


Figura 30 - Cadastro tipo de movimento registro incluído

A Figura 31 exemplifica a tela de manutenção de contas a pagar, mas é utilizada também como manutenção da tela de contas a receber, o que diferencia é um parâmetro definido no clique no menu lateral.



Pessoa	Número	Situação	Data Cadastro	Data Financeiro	Valor	Saldo	Movimento	Nota Fiscal	Pedido	Estoque
Antonio Silva	17	Aberto	29/11/2014	29/11/2014	50,00	50,00	FI - Contas a Pagar	0		
Antonio Silva	17	Aberto	29/11/2014	29/11/2014	0,00	0,00	FI - Contas a Pagar	0		
Antonio Silva	16	Aberto	29/11/2014	29/11/2014	0,00	0,00	FI - Contas a Pagar	0		
Antonio Silva	15	Aberto	29/11/2014	28/11/2014	0,00	0,00	FI - Contas a Pagar	0		
Antonio Silva	15	Aberto	29/11/2014	28/11/2014	0,00	0,00	FI - Contas a Pagar	0		
Antonio Silva	14	Aberto	29/11/2014	28/11/2014	0,00	0,00	FI - Contas a Pagar	0		
Antonio Silva	13	Aberto	07/03/2014	07/03/2014	200,00	200,00	FI - Contas a Pagar	0		
Antonio Silva	11	Aberto	03/03/2014	03/03/2014	50,00	50,00	Teste asdf	0		
Fornecedor produtos SA	7	Fechado	23/02/2014	23/02/2014	540,00	540,00	FI - Contas a Pagar	123456789		
Antonia	5	Aberto	06/01/2014	06/01/2014	500,00	500,00	FI - Contas a Pagar	0		
Antonio Silva	3	Aberto	06/01/2014	06/01/2014	500,00	470,00	FI - Contas a Pagar	0		

Figura 31 - Manutenção contas a pagar

A Figura 32 mostra a tela de cadastro de contas a pagar que utiliza o mesmo parâmetro definido no clique do menu lateral, para definir qual tipo de tratamento a tela terá. Caso seja a pagar traz os tipos de movimento com descrição saída e as pessoas listadas são as marcadas como fornecedor. Caso seja a receber, traz os tipos de movimentos de entrada e as pessoas marcadas como cliente.

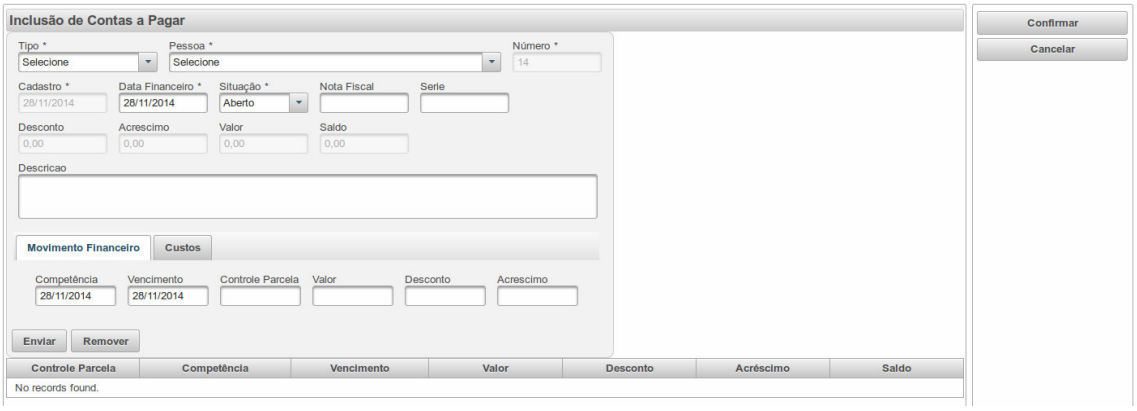


Figura 32 - Cadastro de contas a pagar

A Figura 33 mostra uma validação dos campos obrigatórios no bloco do cabeçalho da tela, por *default* os campos obrigatórios vem marcados com o símbolo asterisco.

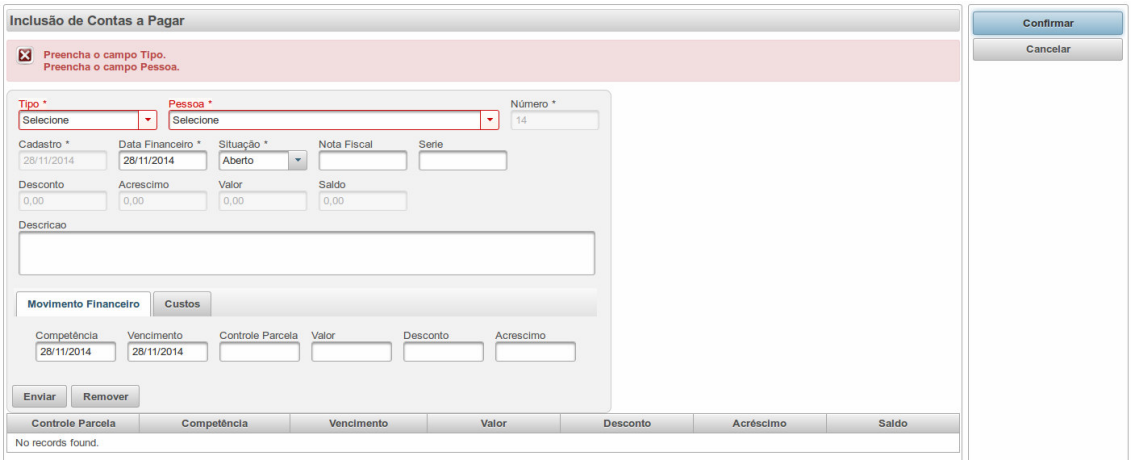


Figura 33 - Validação cabeçalho contas a pagar

A tela de cadastros para contas a pagar e a receber traz um novo formato de formulário, pois devem ser informados dois tipos de movimentos distintos na inclusão do registro. Esses movimentos são apresentados nas abas Movimento Financeiro e Custos. A aba de Movimento Financeiro é usada para informar as parcelas do financeiro com seus respectivos valores e datas. Já a guia de custos é utilizada para informar os valores de movimentação de caixa, como nos mostra a Figura 34.

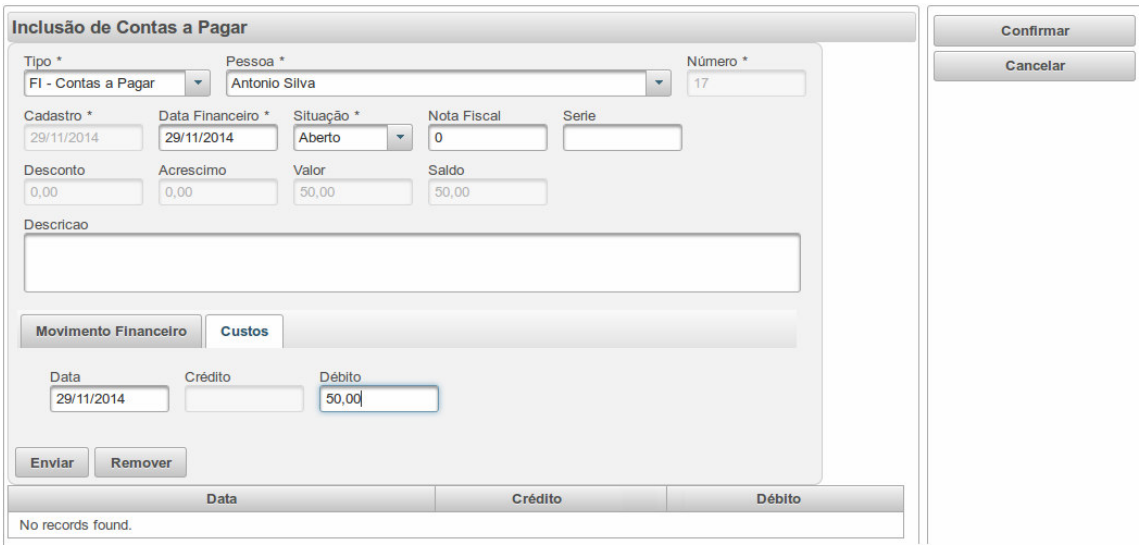


Figura 34 - Custos Contas a pagar

Caso o usuário confirme sem informar os Movimentos do Financeiro e os Custos o sistema apresenta uma mensagem informando que não foi incluído um registro no movimento, como mostra a Figura 35.

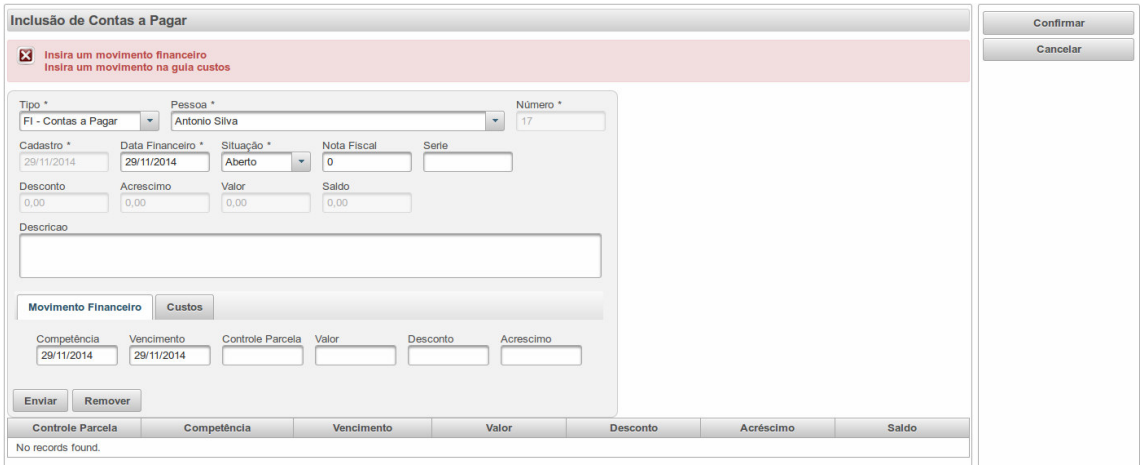


Figura 35 - Validação movimentos vazios

A Figura 36 nos mostra a tela utilizada para o cadastro de caixa manual, sem depender de nenhum registro do financeiro.

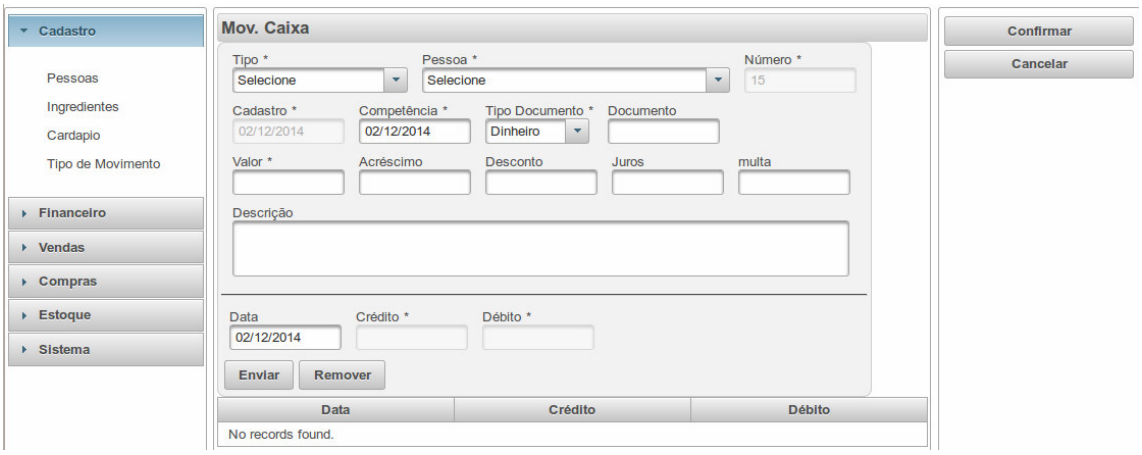


Figura 36 - Cadastro de caixa

Os campos obrigatórios no cadastro de caixa, marcados pelo asterisco, ao confirmar a inclusão valida caso estejam vazios e mostra uma mensagem na tela, como na Figura 37.

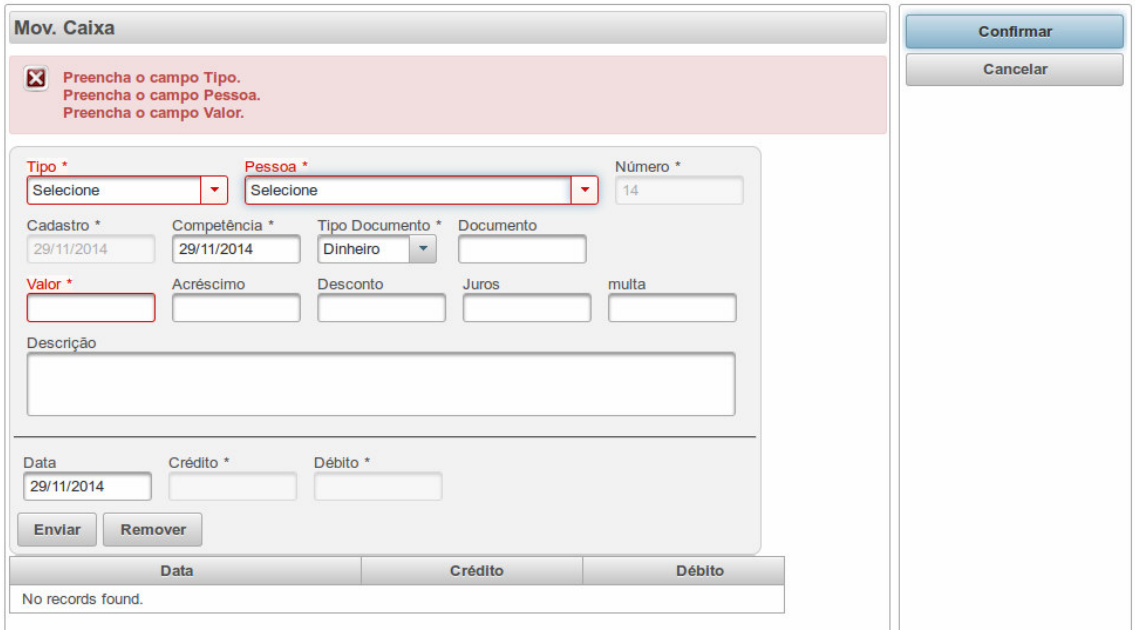


Figura 37 - Cadastro caixa validação itens obrigatórios

Conforme o tipo de movimento selecionado na tela de Cadastro de Caixa, os campos obrigatórios do movimento são modificados. Quando selecionado um tipo de movimento com descrição saída, habilita o campo debito e o marca como obrigatório. Caso o tipo de movimento seja de entrada, habilita o campo credito o tornando obrigatório, como mostram as Figuras 38 e 39, respectivamente.

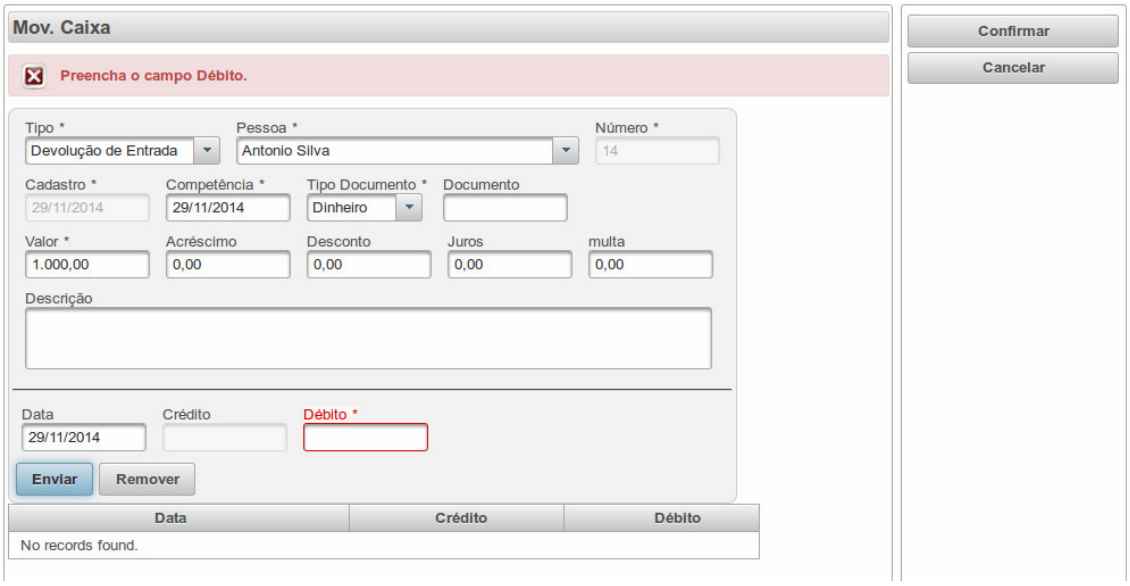


Figura 38 - Validação movimento de caixa

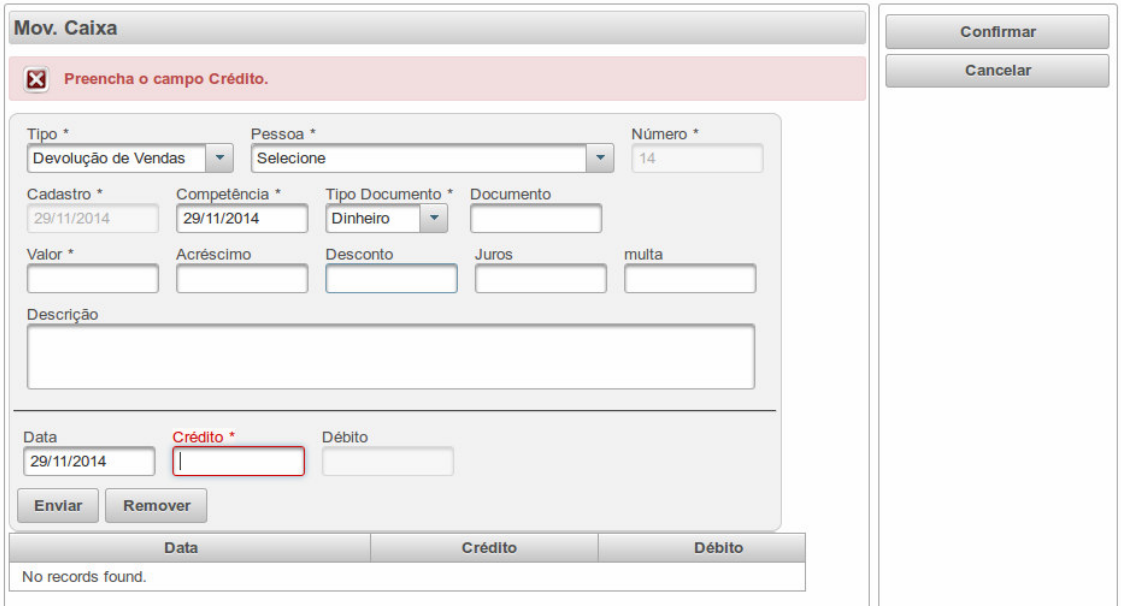


Figura 39 - Validação movimento de caixa

A Figura 40 mostra a tela de Manutenção de Requisições. Essa tela mostra os registros separando as requisições por pedido gerado, que podem ser gerados vários para uma mesma requisição, validados conforme as cotações feitas pelos fornecedores.

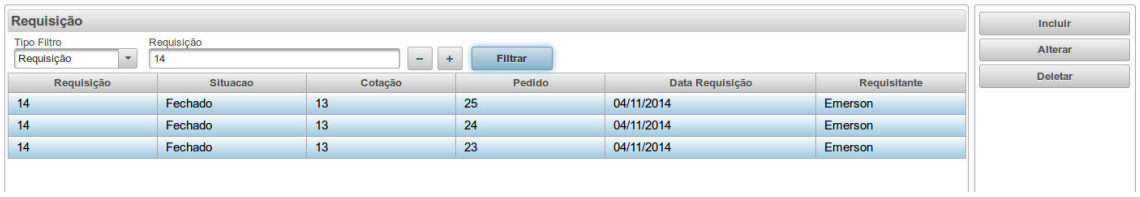


Figura 40 - Manutenção de requisições

A Figura 41 mostra a tela de cadastro de requisições.

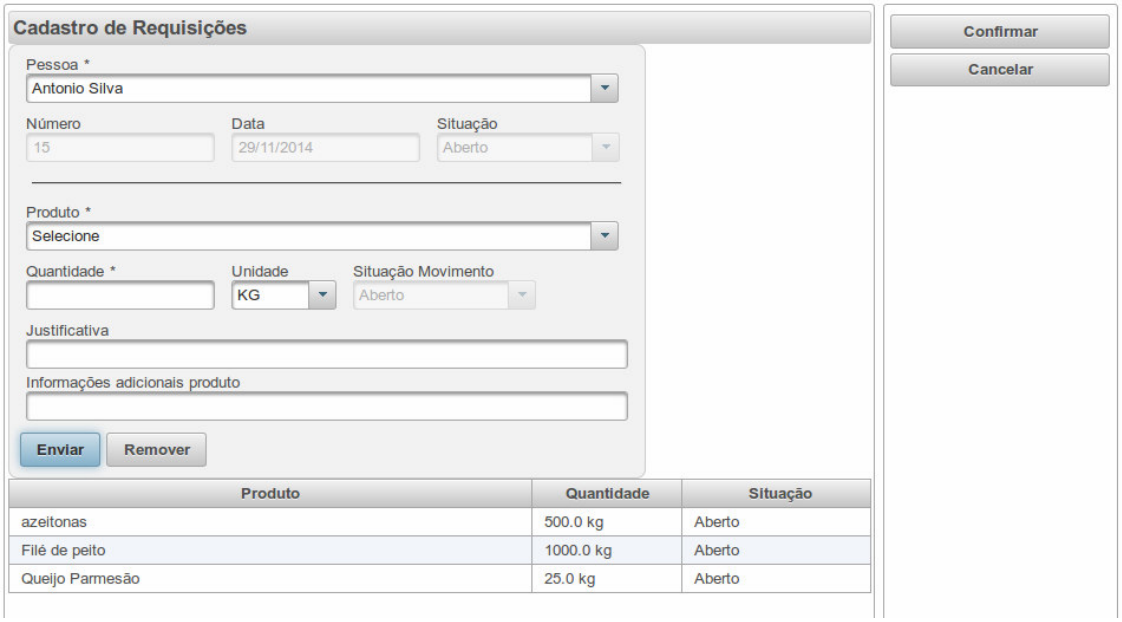


Figura 41 - Cadastro de requisições

A Figura 42 apresenta validações de inclusão no cabeçalho da requisição para os campos obrigatórios. Após clicar no botão Confirmar, o sistema mostra mensagens informando quais campos devem ser preenchidos.

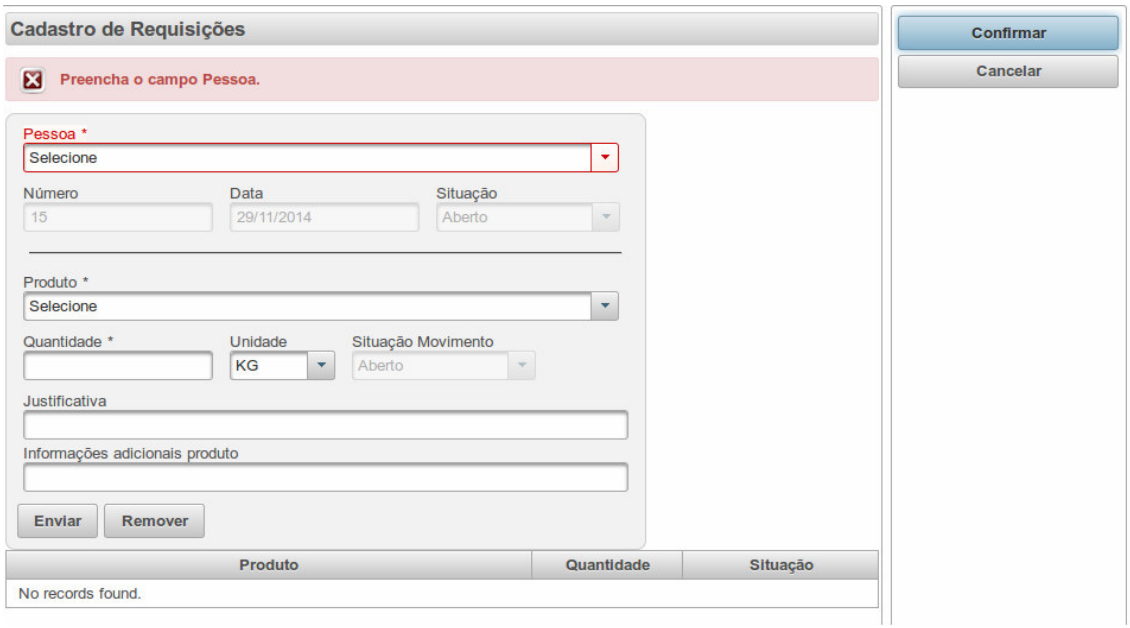


Figura 42 - Cadastro requisição validação cabeçalho

Caso sejam informados os dados do cabeçalho corretamente e confirmado sem

informar nenhum registro no movimento o sistema solicita a inclusão de um registro conforme Figura 43. Caso usuário esteja tentando incluir algum movimento sem preencher todos os campos obrigatórios para o movimento, o sistema solicita o preenchimento dos mesmos, conforme Figura 44.

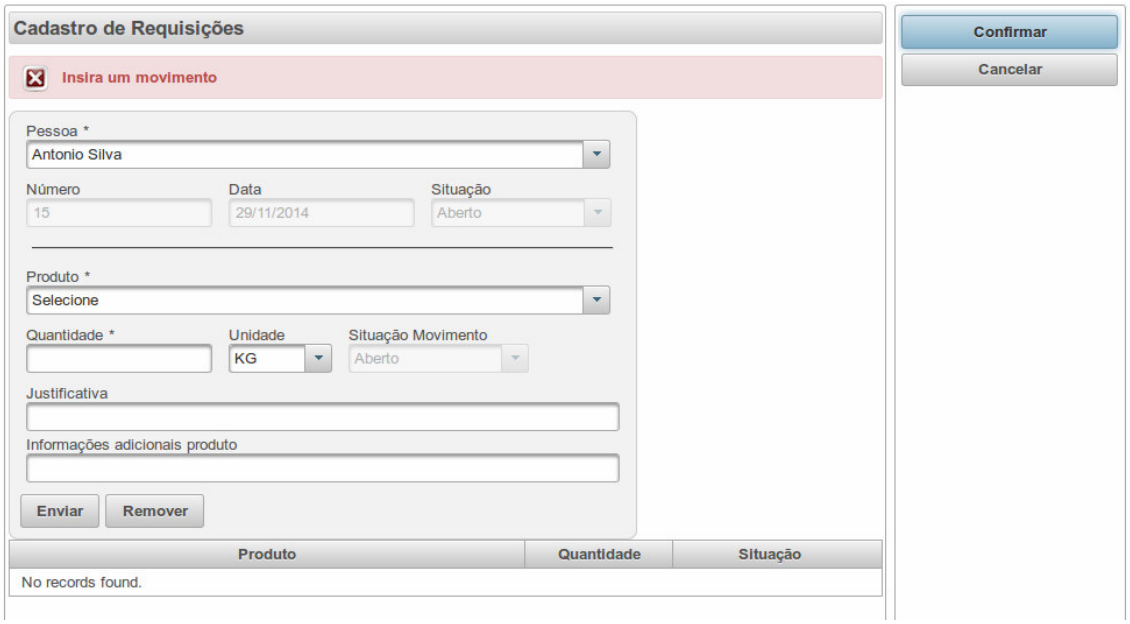


Figura 43 - Cadastro requisição validação sem movimentos

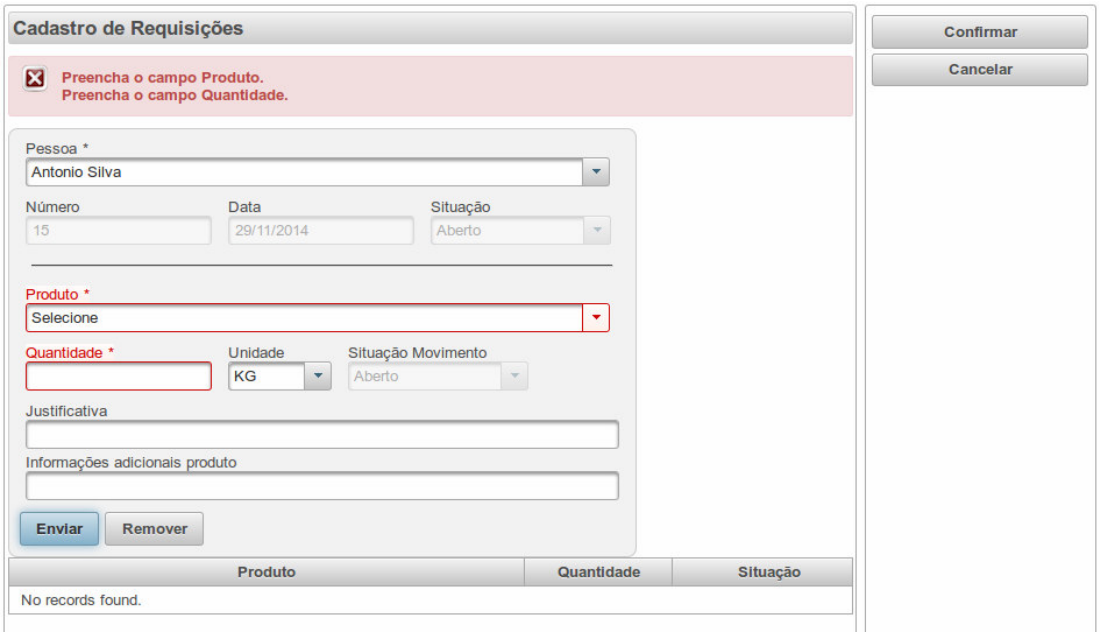


Figura 44 - Cadastro requisição validação movimento

Ao confirmar a requisição a mesma modifica a situação estando disponível para o próximo passo do processo que é a análise da requisição. Na tela de Analise de Requisição o usuário filtra a requisição mostrando no *grid* abaixo todos os itens incluídos na mesma, com suas quantidades e justificativas. A tela possui um botão confirmar todos, que modifica a situação de todos os itens para analisados na sessão. Para confirmar a operação alterando o valor no banco de dados deve se clicar no botão confirma, conforma mostra a Figura 45.

Figura 45 - Análise de requisição

Caso a pessoa responsável pela análise das requisições não queira analisar todos de uma única vez, pode clicar na linha do item no *grid* que irá abrir um formulário com as informações do item e fazer a analise item a item, podendo alterar a quantidade e até mesmo recusar a requisição do item, como mostra a Figura 46.

Figura 46 - Análise requisição item

Ao analisar um item e confirmar o mesmo fica com situação em análise, que ao confirmar a análise irá mudar a situação da requisição e dos movimentos deixando-os disponíveis ou não para os próximos processos. Quando confirmado o item analisado é gerada uma cotação para que a mesma possa ser enviada aos fornecedores. Conforme mostra a Figura 47.

Figura 47 - Analise requisições item analisado

Após confirmar a análise, a requisição fica disponível para ser enviada para os fornecedores efetuarem as cotações, mas para efetuar esse envio devem ser configurados alguns parâmetros anteriormente, como mostra a Figura 48.

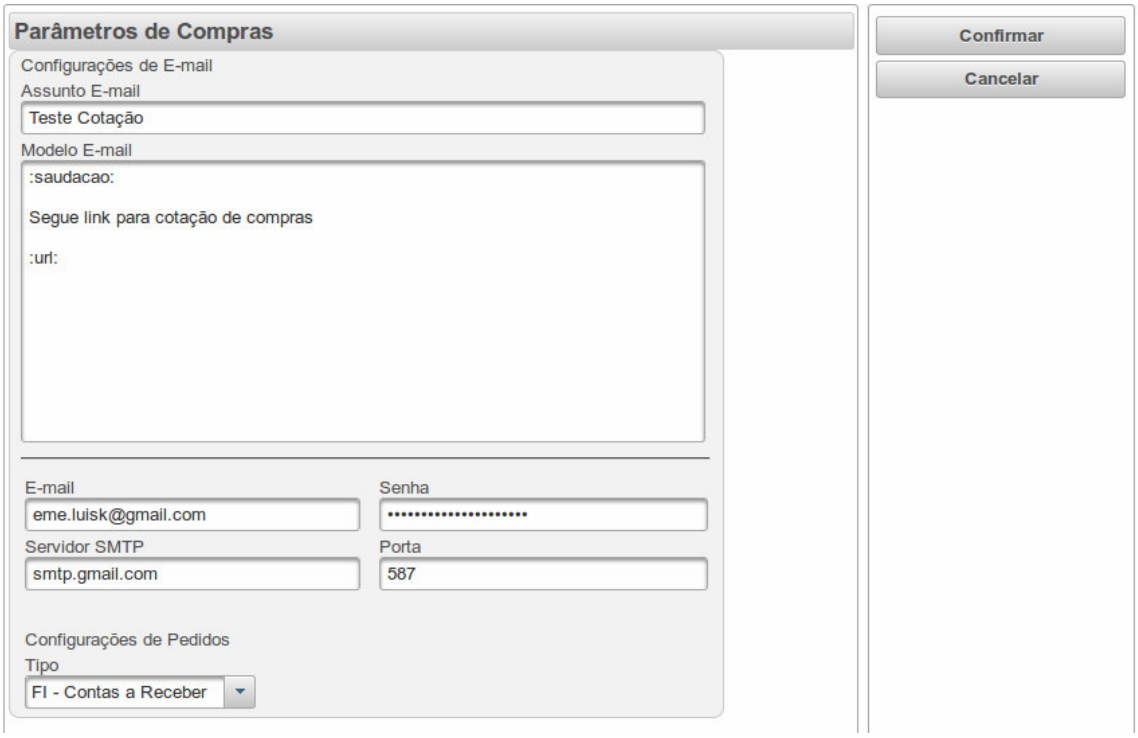


Figura 48 - Parâmetros de compras

Após todas as configurações estarem corretas, podem ser enviadas as cotações para os fornecedores informarem seus preços e condições. A tela de Envio de Cotações lista todos os fornecedores do sistema e as cotações geradas no sistema que ainda não foram finalizadas, conforme Figura 49. E ao selecionar a requisição/cotação gerada devem se marcados os fornecedores para os quais a cotação será enviada. Esse processo monta um *link* que direciona o fornecedor para uma página do sistema na qual ele pode informar suas ofertas. As

informações do link são criptografadas. Caso o usuário clique em enviar sem selecionar a requisição/cotação, o sistema solicita ao mesmo que informe os campos obrigatórios, como mostra Figura 50. Caso todos os campos estejam preenchidos, mas no momento do envio exista algum problema de Internet, o sistema retornará uma mensagem informando sobre o erro na conexão, conforme Figura 51.

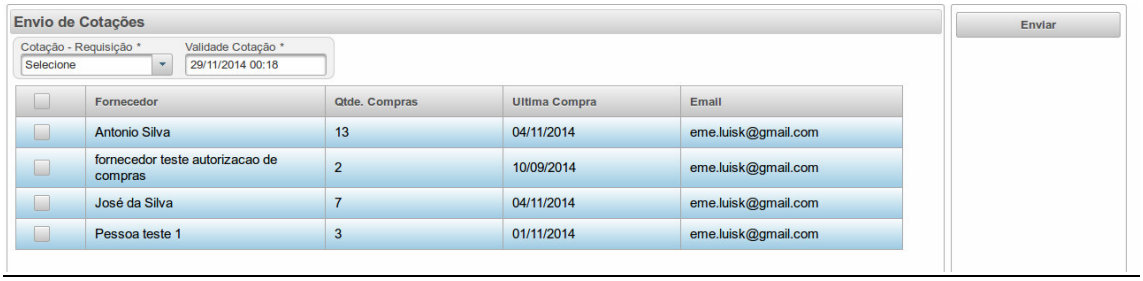


Figura 49 - Envio de cotações

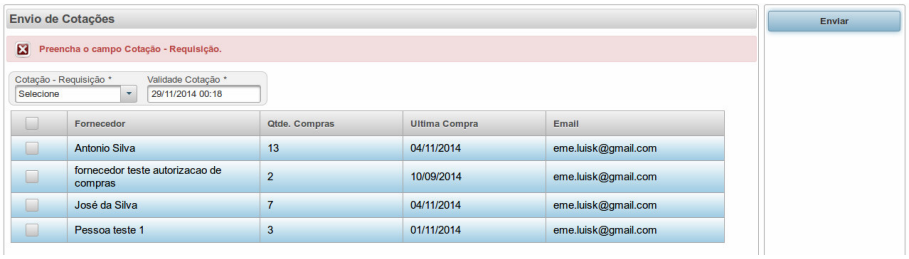


Figura 50 - Envio de cotações validação requisição

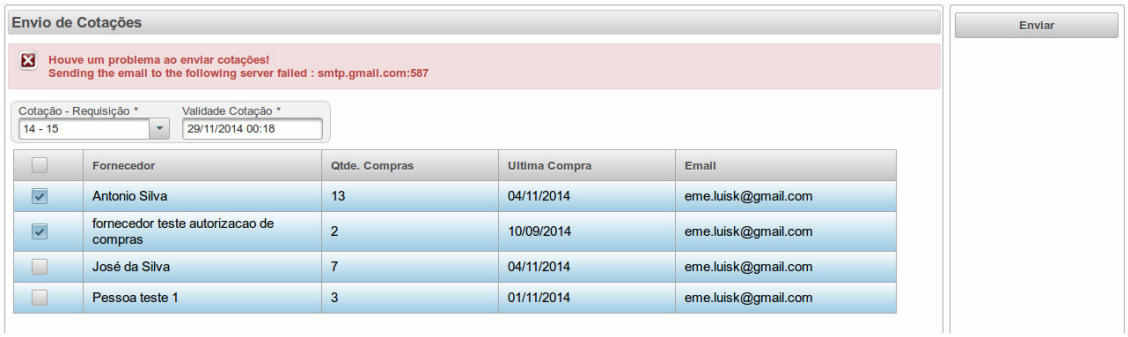


Figura 51 - Envio cotação erro conexão

Após efetuar o envio das cotações para o fornecedor, o sistema modifica a situação da requisição e da cotação para que possam ser utilizadas nos processos posteriores. A cotação chega ao fornecedor por meio de um *link* com algumas informações criptografadas.

Ao acessar o *link* o fornecedor será direcionado a uma página onde pode informar suas

ofertas para os produtos solicitados como mostra a Figura 52.

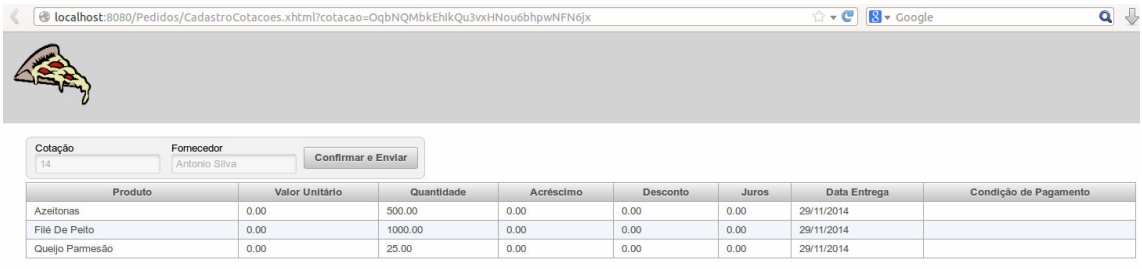


Figura 52 - Cotação fornecedores

Caso a cotação informada nos parâmetros da *Uniform Resource Locator* (URL) já tenha sido cotada, o sistema mostrará uma mensagem informando o usuário, como mostrado na Figura 53. Caso a data de validade da URL informada na tela de envio de cotação já tenha vencido, o sistema informará o usuário que a cotação já está vencida como mostra Figura 54. Para informar as condições para os itens da cotação, o fornecedor deve clicar na linha do item e será aberto um formulário para informação dos valores como mostra Figura 55. Ao confirmar a cotação o sistema informa ao usuário que a cotação foi enviada com sucesso conforme Figura 56.

Figura 53 - Cotação fornecedores validação cotação finalizada

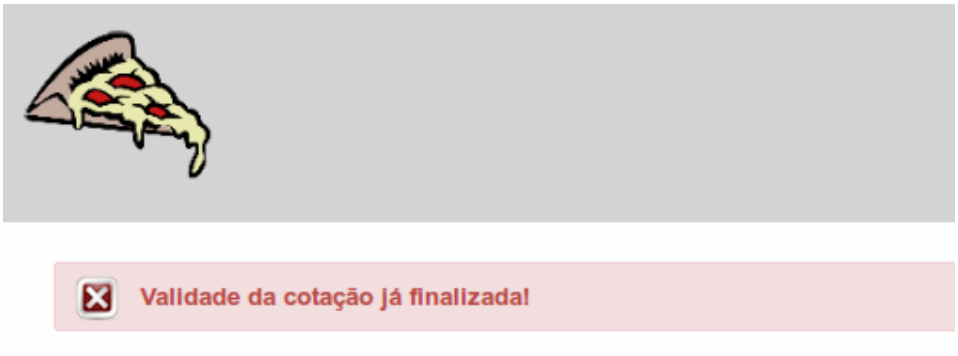


Figura 54 - Cotação fornecedores validade cotação

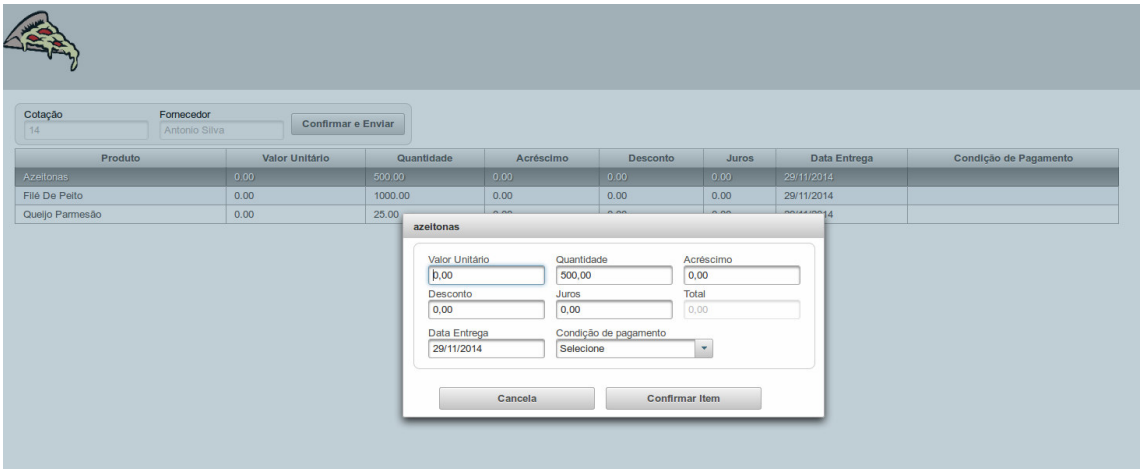


Figura 55 - Cotação fornecedores item

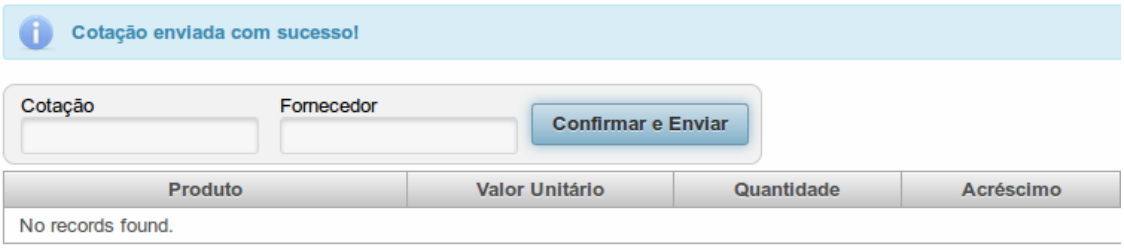


Figura 56 - Cotação fornecedores finalizada

Para verificar quais fornecedores fizeram as melhores propostas é necessário acessar a tela de Receber Cotações. Nessa tela, após filtrar uma requisição, são mostrados todos os itens cotados pelos fornecedores. E ao clicar em cotar, serão feitos cálculos para validar quais fornecedores venceram as cotações, como mostra a Figura 57.

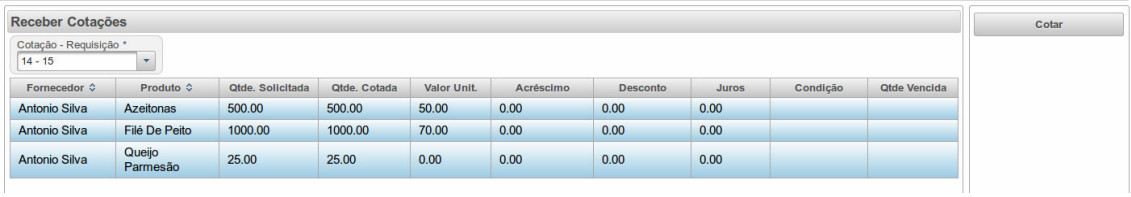


Figura 57 - Receber cotações

Após realizar as cotações são gerados registros de autorização de compras, os quais podem ser visualizados na tela de Autorizações, os itens são agrupados por fornecedor, como mostra Figura 58. A quantidade autorizada pode ser alterada, clicando na lupa na linha. Após isso é apresentado um formulário no qual o usuário pode alterar a quantidade autorizada, como mostra a Figura 59.

Figura 58 - Autorizações de compras

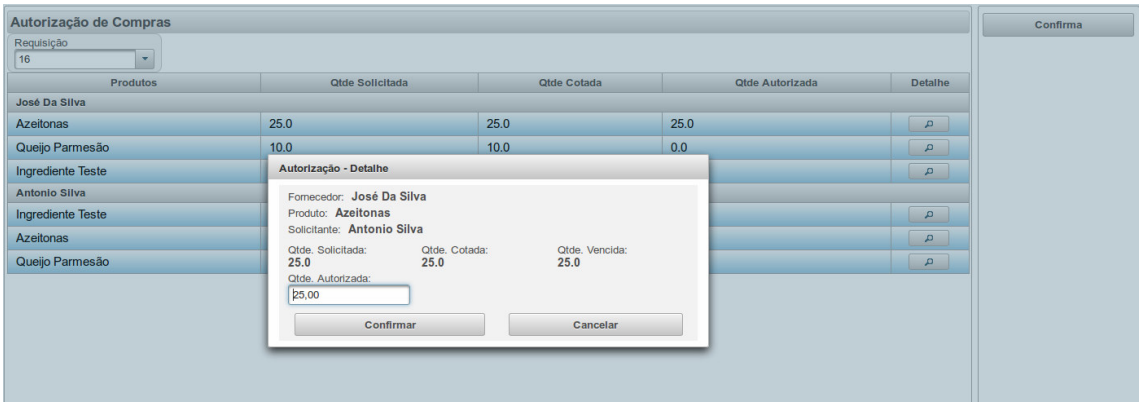


Figura 59 - Autorização de compras alteração item

Ao finalizar a autorização e clicar no confirma o sistema faz algumas verificações, uma delas é em relação a quantidade autorizada em relação a quantidade cotada. Caso a quantidade autorizada seja maior que a quantidade cotada pelo fornecedor, o sistema informa ao usuário sobre as quantidades, como mostra a Figura 60.

Figura 60 - Autorização de compras validação quantidade

Outra validação é em relação a quantidade requerida. Caso a quantidade autorizada seja maior que a quantidade requerida o sistema insere uma mensagem de confirmação para o usuário definir se deseja prosseguir ou não, como mostra a Figura 61. Após todas as validações o sistema confirma a autorização mostrando uma mensagem de sucesso, como mostra a Figura 62. Após todo esse processo são gerados pedidos de compra conforme as opções definidas na cotação e na autorização.

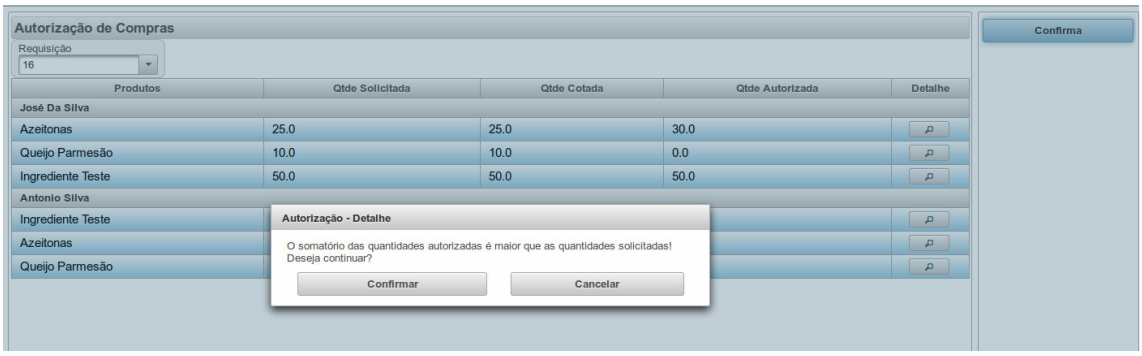


Figura 61 - Autorização de compras validação quantidade maior que a solicitada

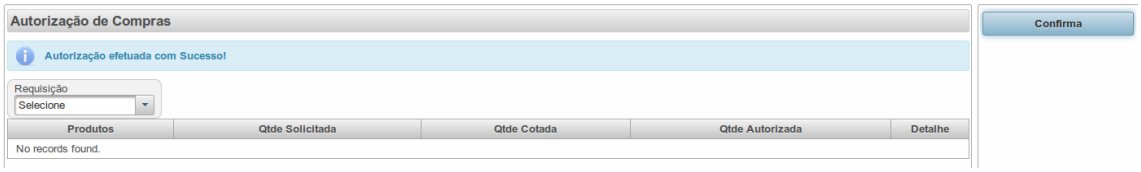


Figura 62 - Autorização de compras finalizada

Após serem gerados os pedidos de compra, quando os mesmos chegam para a entrega na empresa, os registros dos produtos devem ser lançados no sistema. Para isso existe a tela de Entrada Pedido de Compras que lista os fornecedores. E ao selecionar um fornecedor,

todos os pedidos de compras com quantidades a serem entregues são listados, como mostra Figura 63. Quando os pedidos são selecionados, no movimento são listados apenas os produtos relacionados nesses pedidos para facilitar a entrada dos mesmo no estoque, como mostrado na Figura 64.

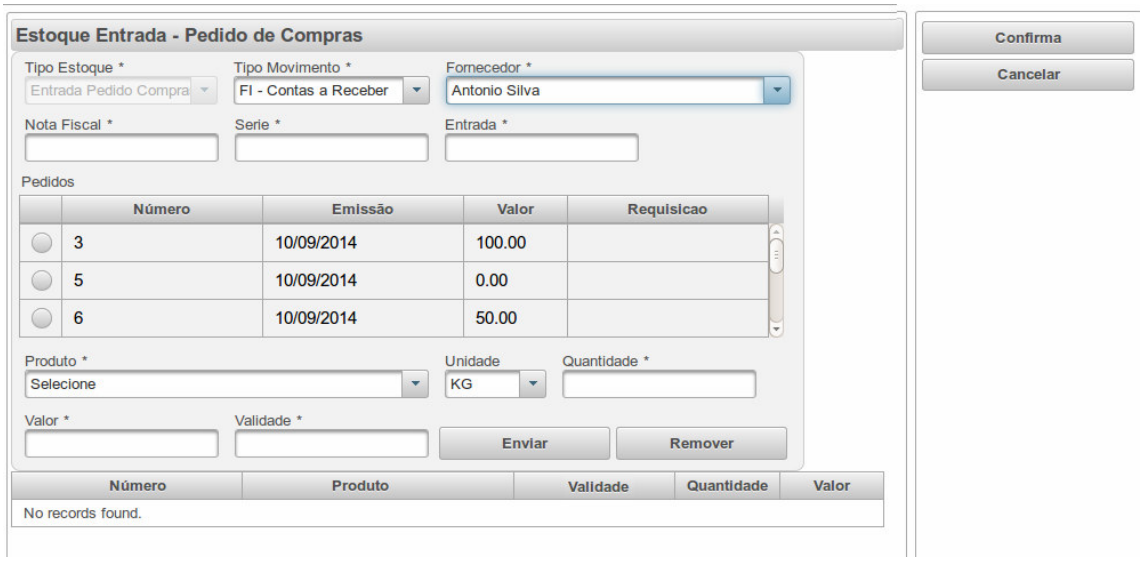


Figura 63 - Estoque entradas - pedido de compra

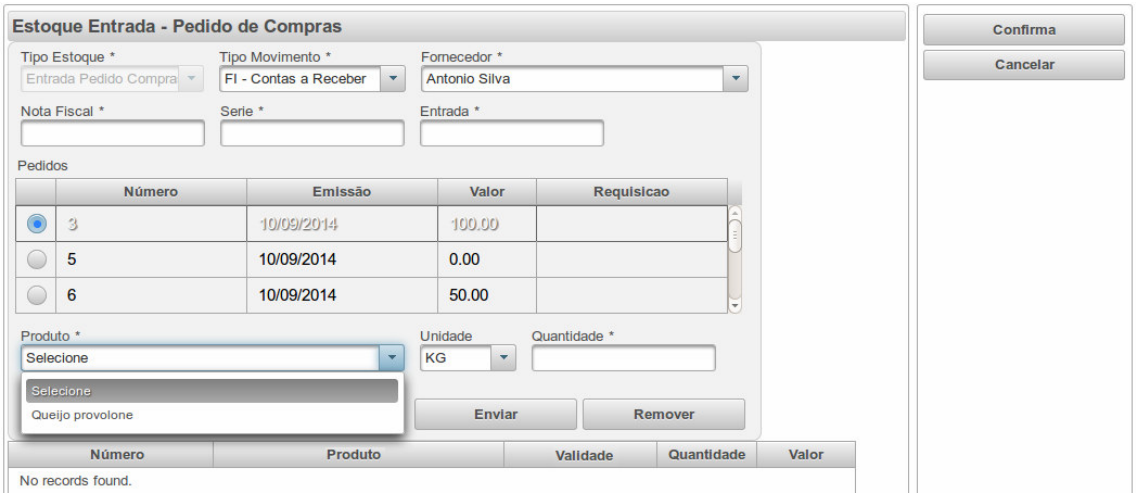


Figura 64 - Estoque entrada - pedido de compra itens pedidos

Os menus relacionados a estoque direcionam para a mesma tela de manutenção de estoque, o que diferencia é um parâmetro setado no clique do menu, como no financeiro. As operações de inclusão e alteração realizadas em seguida se baseiam nesse parâmetro para definir a obrigatoriedade dos campos. Por exemplo, na tela de Estoque - Entradas é

obrigatório informar nota fiscal, série e data da entrada, como mostra a Figura 65. Já na tela de Estoque - Saídas, são obrigatórios somente os campos tipo de movimento e cliente, como mostra a Figura 66.

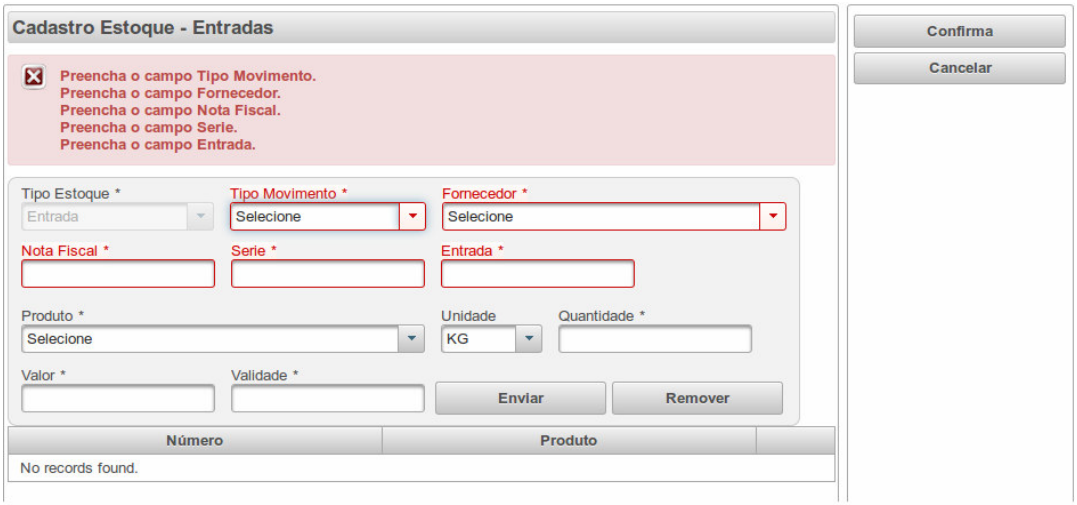


Figura 65 - Estoque saídas

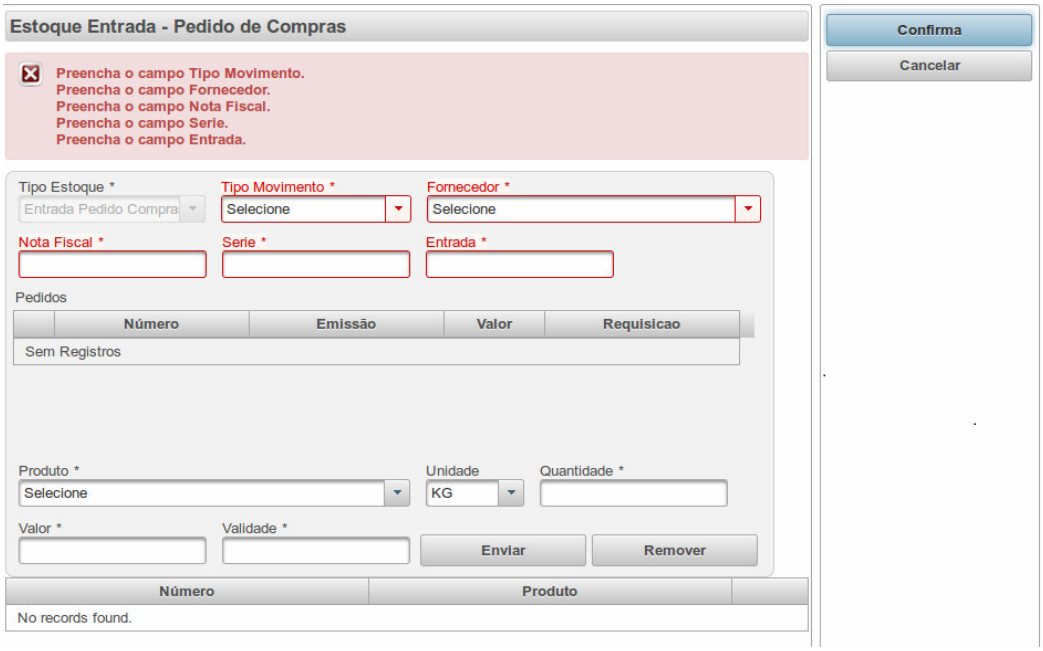


Figura 66 - Estoque entradas

4.4 IMPLEMENTAÇÃO DO SISTEMA

Como apresentado na Seção 4.3, o leiaute do sistema é composto por cinco setores, isso é feito através do uso do componente "*layout*" do PrimeFaces que permite divide a tela em regiões sem a *tag* "*div*" do HTML. Esses setores são chamados de "layoutUnits" e podem ter as seguintes posições: "*north*", "*south*", "*east*", "*west*" e "*center*".

Para facilitar o desenvolvimento, foi utilizado um *template* padrão chamado "LayoutPadrao.xhtml" (Listagem 1). Esse *template* possui o leiaute básico do sistema, que será utilizado em todas as páginas.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui"
      xmlns:pe="http://primefaces.org/ui/extensions">
<h:head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
      <title><ui:insert name="titulo">Controle de Lanchonetes</ui:insert></title>
      <h:outputStylesheet library="css" name="sistema.css" />
</h:head>
<h:body>

      <p:layout fullPage="true">

            <p:layoutUnit id="unitNorte" position="north" size="100"
                  resizable="false" closable="false" collapsible="true"
                  styleClass="cabecalho" style="background-color: #D3D3D3;">

                  <h:graphicImage value="/imagens/logo.png" />
            </p:layoutUnit>

            <p:layoutUnit position="west" resizable="true" size="200" minSize="40"
                  maxSize="200">
                  <ui:include src="/template/MenuLateral.xhtml" />
            </p:layoutUnit>
            <h:form id="formulario">
                  <p:layoutUnit id="centroUnit" position="center">
                        <ui:insert name="nomeTela" />
                        <ui:insert name="conteudo" />
                  </p:layoutUnit>

                  <p:layoutUnit id="eastUnit" position="east" size="350"
                        resizable="false">
                        <ui:insert name="controle" />
                  </p:layoutUnit>
            </h:form>
            <p:layoutUnit position="south" size="50" resizable="false"
```

```
                    collapsible="true">
                    <ui:include src="/template/Rodape.xhtml" />
                </p:layoutUnit>

        </p:layout>
</h:body>
</html>
```

Listagem 1 – LayoutPadrao.xhtml

Como visto na listagem 1, foi utilizado somente o componente "layout" do PrimeFaces, com o modo "*fullpage*" ativado. Dentro do modo "*fullpage*" foram incluídos cinco "layouUnits" posicionadas em "*north*", "*west*", "*center*", "*east*" e "*south*". O "layouteUnit" posicionado em "*north*" possui o componente "graphicImage" do *Java Server Faces* (JSF), que é responsável por renderizar imagens. É esse componente que apresenta a logo da empresa. A *unit* "*west*" faz a inclusão da página "MenuLateral.xhtml" através da *tag include* do Facelet. A *unit* "*center*" possui a *tag* "*insert*" do Facelet, que define um espaço que será utilizado por outras páginas para definir o conteúdo da *unit*. A *unit* "*east*" possui a *tag* "*insert*" para serem definidos os controles das páginas. As *units* "*center*" e "*east*" definem os limites do formulário. Todos os componentes dessas *units* são processados ao efetuar o *submit* do formulário. Por fim está a *unit* "*south*" que faz a inclusão da página "Rodape.xhtml".

O menu principal foi desenvolvido em uma página chamada "MenuLateral.xhtml", cujo código é apresentado na Listagem 2. Essa página utiliza a *tag* "*composition*" do Facelet. Essa página é uma composição que pode ser incluída em outras páginas. Também foi utilizado o componente "accordionPanel". Esse componente pode se comportar como um menu que fecha os outros menus ao ser selecionado, também foi incluído o componente "menu" que pode conter "menuItem" que são as opções do menu.

Cada "menuItem" está invocando o método "onClickMenu" do "managedBean" "controleMenuBean" que espera um parâmetro do tipo inteiro. Esse parâmetro é quem define para qual tela será direcionado o sistema.

```
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:p="http://primefaces.org/ui">

        <div class="menuBar">
                <h:form>
                        <p:accordionPanel style="width:100%;">
                                <p:tab title="Cadastro">
                                        <p:menu
                                                style="width:100%; margin-
```

```
left:0px;border:0px;background-color:#FFFFFF;">
                                        <p:menuitem value="Pessoas"
                                        action="#{controleMenuBean.onClickMenu(1)}" />
                                        <p:menuitem value="Ingredientes"
                                        action="#{controleMenuBean.onClickMenu(2)}" />
                                        <p:menuitem value="Cardapio"
                                        action="#{controleMenuBean.onClickMenu(3)}" />
                                        <p:menuitem value="Tipo de Movimento"
                                        action="#{controleMenuBean.onClickMenu(4)}" />
                                    </p:menu>
                                </p:tab>
                                <p:tab title="Financeiro">
                                    <p:menu
                                    style="width:100%; margin-left:0px;border:0px;background-
color:#FFFFFF;">
                                        <p:menuitem value="Contas a Pagar"
                                        action="#{controleMenuBean.onClickMenu(5)}" />
                                        <p:menuitem value="Contas a Receber"
                                        action="#{controleMenuBean.onClickMenu(6)}" />
                                        <p:menuitem value="Mov. Caixa"
                                        action="#{controleMenuBean.onClickMenu(7)}" />
                                    </p:menu>
                                </p:tab>
                                <p:tab title="Sistema">
                                    <p:menu
                                    style="width:100%; margin-left:0px;border:0px;background-
color:#FFFFFF;">
                                        <p:menuitem value="Usuários" outcome="Usuarios" />
                                        </p:menu>
                                </p:tab>
                            </p:accordionPanel>
                        </h:form>
                </div>
</ui:composition>
```

Listagem 2 – MenuLateral.xhtml

A página que lista os Ingredientes é chamada de “ManutencaoIngredientes.xhtml” e utiliza as *tags* “composition” para dizer que é uma composição de outra página, no caso o “LayoutPadrao.xhtml”. Quando é utilizada a *tag* “define” é necessário usar o atributo “*name*”, com o mesmo valor do atributo “*name*” da *tag* “*insert*” que se pretende usar. A listagem 3 mostra a página de manutenção de Ingredientes com o atributo “*name*” da *tag* “define” que possui o mesmo valor que o atributo “*name*” da *tag* “*insert*” utilizado no *template* desenvolvido. Dessa forma, todo o conteúdo da página de Manutenção de Ingredientes será incluído no espaço utilizado pela *tag* “*insert*” do *template*.

```
<ui:composition template="/template/LayoutPadrao.xhtml"
        xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:p="http://primefaces.org/ui"
        xmlns:pe="http://primefaces.org/ui/extensions">


        <ui:define name="nomeTela">
                <h:panelGrid id="gridNome" styleClass="grid-nome">
                        <h:outputLabel id="nomeTelas" styleClass="nomeTela">
                                <b>Ingredientes</b>
                        </h:outputLabel>
                </h:panelGrid>
        </ui:define>

        <ui:define name="conteudo">
                <p:messages id="msgs" showDetail="false" showSummary="true"
                        styleClass="form-messages" errorClass="error" infoClass="info" />

                <h:panelGrid columns="5">
                        <h:panelGroup>
                                <p:outputLabel value="Tipo Filtro"
                                        style="margin-left: 5px; margin-right: 5px; width: 50px;" />
                                <br />
                                <p:selectOneMenu id="tipoFiltro"
                                        value="#{manuIngredientesBean.tipo}"
                                        style="margin-left: 5px; margin-right: 5px; width: 150px;">
                                        <f:selectItem itemLabel="Nome"
itemValue="Nome:nomeIngrediente:s" />
                                        <f:selectItem itemLabel="Tipo"
itemValue="Tipo:tipoIngrediente:s" />
                                        <f:selectItem itemLabel="Cadastro"
                                                itemValue="Cadastro:cadastroIngrediente:d" />
                                        <f:selectItem itemLabel="Ativo"
                                                itemValue="Ativo:ativoIngrediente:i" />
                                        <f:selectItem itemLabel="Codigo"
                                                itemValue="Codigo:aponIngrediente:i" />
                                        <p:ajax listener="#{manuIngredientesBean.onSelectFiltro}"
                                                update="filtro" process="@this" />

                                </p:selectOneMenu>
                        </h:panelGroup>
                        <h:panelGroup id="filtro">
                                <p:outputLabel value="#{manuIngredientesBean.nomeFiltro}"
                                        style="margin-left: 5px; margin-right: 5px; width: 50px;" />
                                <br />
                                <p:inputMask mask="#{manuIngredientesBean.mascara}"
                                        value="#{manuIngredientesBean.filtro}"
                                        style="margin-left: 5px; margin-right: 5px; width: 300px;">
                                        <p:ajax event="keyup" process="@this" />
                                </p:inputMask>
                        </h:panelGroup>
                        <h:panelGroup>
                                <br />
                                <p:commandButton icon="ui-icon-minus"
                                        action="#{manuIngredientesBean.onRemoveFiltro}" />
```

```
                        </h:panelGroup>
                        <h:panelGroup>
                                <br />
                                <p:commandButton icon="ui-icon-plus"
                                        action="#{manuIngredientesBean.onAddFiltro}" />
                        </h:panelGroup>
                        <h:panelGroup>
                                <br />
                                <p:commandButton id="btnFiltro" value="Filtrar"
                                        style="margin-left: 5px; margin-right: 5px; width: 100px;"
                                        update="grid" action="#{manuIngredientesBean.onFiltrar}" />
                        </h:panelGroup>
                </h:panelGrid>
                <p:outputPanel id="grid">
                        <p:dataTable id="dataTableItens" var="itemAdicionado"
                                emptyMessage="Sem Registros"
                                value="#{manuIngredientesBean.ingredienteDtm}"
                                selection="#{manuIngredientesBean.ingredienteSelecionado}"
                                selectionMode="single">
                                <p:ajax event="rowSelect"
                                        listener="#{manuIngredientesBean.onRowSelect}" />
                                <p:column headerText="Código"
width="100">#{itemAdicionado.aponIngrediente}</p:column>
                                <p:column headerText="Nome"
width="250">#{itemAdicionado.nomeIngrediente}</p:column>
                                <p:column width="150">
                                        <f:facet name="header">Cadastro</f:facet>
                                        <h:outputText value="#{itemAdicionado.cadastroIngrediente}">
                                                <f:convertDateTime pattern="dd/MM/yyyy"
dateStyle="medium"
                                                        locale="pt_BR" timeZone="GMT+3" />
                                        </h:outputText>
                                </p:column>
                                <p:column headerText="Tipo"
width="150">#{manuIngredientesBean.tipoIngrediente(itemAdicionado.tipoIngrediente)}</p:column>
                                <p:column headerText="Ativo"
width="100">#{manuIngredientesBean.ativoProduto(itemAdicionado.ativoIngrediente)}</p:column>
                        </p:dataTable>
                </p:outputPanel>
        </ui:define>
        <ui:define name="controle">
                <h:panelGrid style="width: 100%;">
                        <p:commandButton id="btnInclui" value="Incluir" style="width: 100%;"
                                action="#{manuIngredientesBean.onIncluir}" />
                        <p:commandButton id="btnAlterar" value="Alterar" process="@this"
                                style="width: 100%;" action="#{manuIngredientesBean.onAlterar}">
                                <f:param name="tipoFinanceiro" value="P" />
                        </p:commandButton>
                        <p:commandButton id="btnDeletar" value="Deletar" process="@this"
                                update="msgs grid" style="width: 100%;"
                                action="#{manuIngredientesBean.onDelete}" />
                </h:panelGrid>

        </ui:define>
</ui:composition>
```

Listagem 3 – ManutencaoIngredientes.xhtml

Como apresentado na Listagem 3, a página Manutenção de Ingredientes possui três *tags* "*define*": a primeira é para atribuir o nome da tela, pois o mesmo deve ser informado em cada janela que é inserida; a segunda é para o conteúdo da página; e a terceira é para os controles, para os botões confirmar e cancelar, pois os mesmo devem invocar os métodos do "ManagedBean" que está sendo utilizado.

A definição do conteúdo possui um componente "panelGrid" nativo do JSF para organizar os campos tipo filtro, filtro e os botões (-), (+) e filtrar. Logo abaixo está um componente "outputPanel" que possui um "dataTable", ambos do PrimeFaces. Os botões acionam métodos descritos no ManagedBean que controla as operações da tela. A Listagem 4 mostra o ManagedBean que controla a página manutenção de ingredientes.

```
package br.com.emerson.pedidosWeb.view;

import java.text.Normalizer;
import java.util.ArrayList;
import java.util.List;

import javax.annotation.PostConstruct;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.context.FacesContext;

import org.primefaces.event.SelectEvent;

import br.com.emerson.pedidosWeb.DataModels.ManuIngredientesDTM;
import br.com.emerson.pedidosWeb.model.Filtro;
import br.com.emerson.pedidosWeb.model.Ingredientes;
import br.com.emerson.pedidosWeb.repository.IIngredientes;
import br.com.emerson.pedidosWeb.util.FacesUtil;
import br.com.emerson.pedidosWeb.util.Repositorios;

@ManagedBean
@SessionScoped
public class ManuIngredientesBean {
	private String tipo;
	private String nomeFiltro;
	private String mascara;
	private String filtro;
	private List<Filtro> filtros;
	private Repositorios repositorio;
	private ManuIngredientesDTM ingredienteDtm;
	private Ingredientes ingredienteSelecionado;

	@PostConstruct
	public void init() {
		this.nomeFiltro = "Nome";
		this.mascara = "";
		this.filtros = new ArrayList<Filtro>();
		this.repositorio = new Repositorios();
		this.ingredienteDtm = new ManuIngredientesDTM();
```

```
	}

	public void onSelectFiltro() {
		this.nomeFiltro = this.tipo.split(":")[0];
		this.filtro = "";
		if (this.tipo.split(":")[2].equals("d")) {
			this.mascara = "99/99/9999";
		} else {
			this.mascara = "";
		}
	}

	public void onAddFiltro() {
		Filtro f = new Filtro();
		this.onRemoveFiltro();
		f.setChave(this.tipo.split(":")[1]);
		f.setTipo(this.tipo.split(":")[2]);
		if (this.tipo.split(":")[2].equals("s"))
			f.setFiltro(this.tipoIngredienteC(this.filtro));
		else if (this.tipo.split(":")[2].equals("i"))
			f.setFiltro(this.ativoProduto(this.filtro));
		else
			f.setFiltro(this.filtro);
		this.filtros.add(f);
	}

	public void onFiltrar() {
		this.onAddFiltro();
		IIngredientes iIngredientes = repositorio.getIngredientes();
		List<Ingredientes> lst = iIngredientes.filtrar(Ingredientes.class,
				filtros);
		this.ingredienteDtm = new ManuIngredientesDTM(lst);
		// this.filtros.clear();
	}

	public void onRemoveFiltro() {
		if (this.filtros.size() > 0) {
			List<Filtro> lstAux = new ArrayList<Filtro>();
			for (Filtro f : this.filtros) {
				if (!f.getChave().equals(this.tipo.split(":")[1])) {
					lstAux.add(f);
				}
			}
			this.filtros.clear();
			if (lstAux.size() > 0) {
				this.filtros = lstAux;
			}
		}
	}

	public void onDelete() {
		String msg;
		if (this.ingredienteSelecionado == null) {
			msg = "Selecione um registro!";
			FacesContext.getCurrentInstance().addMessage(null,
				new FacesMessage(FacesMessage.SEVERITY_WARN, msg, msg));
		} else {
			IIngredientes iIngrediente = repositorio.getIngredientes();
```

```
                        String confirmacao;
                        confirmacao =
iIngrediente.excluir(Ingredientes.class,this.ingredienteSelecionado.getIdIngrediente());
                        if (confirmacao.equals("")) {
                                msg = "Registro excluido com sucesso!";
                                FacesContext.getCurrentInstance().addMessage(null,
                                        new FacesMessage(FacesMessage.SEVERITY_INFO, msg, msg));
                                onFiltrar();
                        } else {
                                msg = confirmacao;
                                FacesContext.getCurrentInstance()
                                        .addMessage(null, new
FacesMessage(FacesMessage.SEVERITY_ERROR,  msg, msg));
                        }
                }
        }

        public void onIncluir() {
                FacesUtil.facesRedirect("CadastroIngredientes.xhtml");
                FacesUtil.setRequestAttributeSession("ingrediente", null, null);
                FacesUtil.setRequestAttributeSession("alterarRegistro", null, false);
        }

        public void onAlterar() {
                FacesUtil.facesRedirect("CadastroIngredientes.xhtml");
                FacesUtil.setRequestAttributeSession("ingrediente",
                        this.ingredienteSelecionado, null);
                FacesUtil.setRequestAttributeSession("alterarRegistro", null, true);
        }

        public String tipoIngrediente(String tipo) {
                if (tipo.equals("C")) {
                        return "Congelado";
                } else if (tipo.equals("R")) {
                        return "Resfriado";
                } else {
                        return "Seco";
                }
        }

        private String tipoIngredienteC(String tipo) {
                if ("congelado".indexOf(tipo.toLowerCase()) > 0) {
                        return "C";
                }else if ("resfriado".indexOf(tipo.toLowerCase())>0){
                        return "R";
                }else if ("seco".indexOf(tipo.toLowerCase())>0){
                        return "S";
                }else {
                        return tipo;
                }
        }

        public String ativoProduto(Integer ativo) {
                if (ativo == 0)
                        return "Não";
                else
                        return "Sim";
        }
```

```
private String ativoProduto(String ativo) {
        if (Normalizer.normalize(ativo, Normalizer.Form.NFD)
                        .replaceAll("[^\\p{ASCII}]", "").toLowerCase().equals("sim")) {
                return "1";
        } else if (Normalizer.normalize(ativo, Normalizer.Form.NFD)
                        .replaceAll("[^\\p{ASCII}]", "").toLowerCase().equals("nao")) {
                return "0";
        } else {
                return ativo;
        }
}

public void onRowSelect(SelectEvent event) {
        this.ingredienteSelecionado = (Ingredientes) event.getObject();
}

public String getTipo() {
        return tipo;
}

public void setTipo(String tipo) {
        this.tipo = tipo;
}

public String getNomeFiltro() {
        return nomeFiltro;
}

public void setNomeFiltro(String nomeFiltro) {
        this.nomeFiltro = nomeFiltro;
}

public String getMascara() {
        return mascara;
}

public void setMascara(String mascara) {
        this.mascara = mascara;
}

public String getFiltro() {
        return filtro;
}

public void setFiltro(String filtro) {
        this.filtro = filtro;
}

public List<Filtro> getFiltros() {
        return filtros;
}

public void setFiltros(List<Filtro> filtros) {
        this.filtros = filtros;
}

public ManuIngredientesDTM getIngredienteDtm() {
```

```
		return ingredienteDtm;
	}

	public void setIngredienteDtm(ManuIngredientesDTM ingredienteDtm) {
		this.ingredienteDtm = ingredienteDtm;
	}

	public Ingredientes getIngredienteSelecionado() {
		return ingredienteSelecionado;
	}

	public void setIngredienteSelecionado(Ingredientes ingredienteSelecionado) {
		this.ingredienteSelecionado = ingredienteSelecionado;
	}
}
```

Listagem 4 – ManuIngredientesBean.java

O ManagedBean “ManuIngredientesBean” possui a anotação “SessionScoped”, dessa forma essa classe ficará criada enquanto não for finalizada a sessão no navegador. A classe foi implementada com escopo de sessão, pois a tela de inclusão de ingredientes necessita de informações da mesma, as quais não estariam mais disponíveis em um escopo de visão, por exemplo. Esse “ManagedBean” possui algumas variáveis declaradas, são elas “tipo”, “nomeFiltro”, “mascara”, “filtro” e um *list* de filtros “filtros” que são utilizados para aplicar os filtros no “dataTable” da tela, também possui uma variável “repositorio” que faz referência a uma classe que instância os DAOs que é a camada de comunicação com o banco de dados, possui ainda as variáveis “ingredienteDtm” e ingredienteSelecionado. A variável “ingredienteDtm” faz referência uma classe “ManuIngredientesDTM”, tal classe estende a classe “ListDataModel” do PrimeFaces e implementa a classe “SelectableDataModel”, como mostra a Listagem 5. Essa classe é necessária para habilitar a opção “selection” no “dataTable” da página “ManutencaoIngredientes.xhtml”. Já a variável “IngredienteSelecionado” recebe o valor do objeto selecionado no “dataTable”, essa variável é necessária para as operações de alteração e exclusão.

```
package br.com.emerson.pedidosWeb.DataModels;

import java.io.Serializable;
import java.util.List;

import javax.faces.model.ListDataModel;

import org.primefaces.model.SelectableDataModel;

import br.com.emerson.pedidosWeb.model.Ingredientes;

public class ManuIngredientesDTM extends ListDataModel<Ingredientes> implements
SelectableDataModel<Ingredientes>,Serializable{
```

```
	public ManuIngredientesDTM(){}

	public ManuIngredientesDTM(List<Ingredientes> lstIngredientes){
		super(lstIngredientes);
	}

	@Override
	public Ingredientes getRowData(String RowKey) {

		@SuppressWarnings("unchecked")
		List<Ingredientes> ingrediente = (List<Ingredientes>) getWrappedData();
		for(Ingredientes i: ingrediente){
			if(i.getIdIngrediente().toString().equals(RowKey)){
				return i;
			}
		}
		return null;
	}

	@Override
	public Object getRowKey(Ingredientes ingrediente) {
		return ingrediente.getIdIngrediente();
	}
}
```

Listagem 5 – ManuIngredientesDtm.java

5 CONCLUSÃO

O objetivo deste trabalho foi implementar um sistema para controle de uma lanchonete, incluindo o controle operacional aos controles financeiros e gerenciais. O sistema foi desenvolvido conforme o planejamento composto por diversas funcionalidades que são úteis ao negócio de bares, restaurantes e similares. Além das funcionalidades comuns de estoque, compras e vendas, a possibilidade de fornecedores realizarem cotações também foi implementada. O sistema também provê o gerenciamento dos pedidos realizados e enviados para a cozinha.

A aplicação foi desenvolvida para a plataforma *web*, para facilidade de acesso e manutenção, utilizando recursos que caracterizam como aplicação *RIA*. As tecnologias utilizadas tornam a aplicação não dependente de uma arquitetura de sistema especifica. foram utilizadas diversas tecnologias e ferramentas para facilitar na implementação.

Uma dessas ferramentas utilizadas foi o Hibernate, um *framework Object Relational Mapping* (ORM), que tem como principal finalidade mapear as classes Java "orientadas a objeto" para o "mundo relacional" automaticamente de forma simples para o desenvolvedor. Assim, não é necessário a utilização de linguagem SQL dentro do código.

Uma outra ferramenta é o PrimeFaces, uma biblioteca que implementa as especificações do JSF. O JSF busca unir as principais vantagens entre as aplicações *desktop* e *web,* utilizando componentes ricos, sem a necessidade de utilização de HTML e JavaScript. O PrimeFaces possui uma aceitação muito boa pela comunidade e uma vasta documentação (que pode ser encontrada, inclusive, na página oficial http://primefaces.org/), facilitando o desenvolvimento e a solução de possíveis problemas. .

Como trabalhos futuros para melhorar as funcionalidades desenvolvidas, cita-se a possibilidade de o cliente poder fazer pedidos pela *web* ou por telefone. Assim, um módulo para o cliente consultar cardápios pela web, dever ser implementado. Para o atendimento por telefone, o funcionário atendente registra o pedido no sistema, que é fechado quando o entregador presta contas do pedido. Os entregadores também poderiam contar com um módulo, implementado para dispositivos móveis, para o gerenciamento das entregas e dos pagamentos efetuados.

REFERÊNCIAS

BUSCH, Marianne; KOCH, Nora. **Rich Internet Applications, State of the Art**. Technical Report 0902, 2009, p. 1-18.

COLOMBO-MENDOZA, Luis Omar; ALOR-HERNÁNDEZ, Giner; RODRÍGUEZ-GONZÁLEZ, Alejandro. **A Novel Approach for Generating Multi-device Rich Internet Applications. IEEE, 2012, p. 361-367.**

DUHL J., **Rich internet applications**, IDC white papers. Disponível em <http://www.idc.com>, 2003.

FUKUDA, H., YAMAMOTO, Y. **A system for supporting development of large scaled rich internet applications**. In: 23rd IEEE/ACM International Conference on Automated Software *Engine*ering (ASE 2008), 2008, p. 459-462.

LOOSLEY, C. **Rich internet applications: design, measurement and management chalenges** 2006. Disponível em: <http://www.keynote.com/docs/whitepapers/> Acesso em: 5 de abril de 2014.

ORACLE CLOUD RESOURCE MODEL API. 2011. Disponível em: <http://www.oracle.com/technetwork/topics/cloud/oraclecloud-resource-model-api-154279.pdf>. Acesso em: 24 jul. 2014.

PAVLIĆ, Daniel; PAVLIĆ, Mile; JOVANOVIĆ, Vladan. **Future of Internet technologies. International Convention** on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2012, p. 1366-1371.

STEARN, Brent. **XULRunner: A New Approach for Developing Rich Internet Applications.** (Eds.) SINGH, Munindar P.; RAMAMRITHAM, Krithi. Spotlight. may/jun 2007, p. 67-73.

TRAMONTANA, Porfirio; AMALFITANO, Domenico; FASOLINO, Anna Rita. **Reverse Engineering Techniques: from Web Applications to Rich Internet Applications**. IEEE, 2013, p. 83-86.