INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - CAMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

ARIEL WILKOMM

DESENVOLVIMENTO DE UM SISTEMA DE PEDIDOS PARA A EMPRESA VALDEMAR DETERGENTES

Jorge Luís Boeira Bavaresco

PASSO FUNDO
2017

ARIEL WILKOMM

DESENVOLVIMENTO DE UM SISTEMA DE PEDIDOS PARA A EMPRESA VALDEMAR DETERGENTES

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Campus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador (a): Jorge L. Boeira Bavaresco

PASSO FUNDO

2017

ARIEL WILKOMM

DESENVOLVIMENTO DE UM SISTEMA DE PEDIDOS PARA A EMPRESA VALDEMAR DETERGENTES

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Nome do Professor(a) Orientador(a)

_____

Nome do Professor(a) Convidado(a)

_____

Nome do Professor(a) Convidado(a)

_____

Coordenação do Curso

PASSO FUNDO

2017

DEDICATÓRIA

A minha mãe,
pela compreensão e pelo estímulo
em todos os momentos.

AGRADECIMENTOS

Primeiramente agradeço a minha mãe, Nilza Luiz dos Santos, pelo apoio, carinho e educação, que permitiram que eu chegasse até esse momento.

Agradeço ao professor Jorge Luís Boeira Bavaresco que com seu conhecimento e experiência, contribuiu imensamente na execução deste projeto.

Agradeço aos meus colegas pela amizade, pelas trocas de experiências e conhecimentos que contribuiriam com nossas trajetórias durante o curso.

Por fim, agradeço aos demais professores e funcionários do IFSUL por proporcionarem as condições necessárias para o que ensino e o aprendizado sejam de qualidade nesta instituição.

EPÍGRAFE

"Qualquer coisa que possa ocorrer mal,
ocorrerá mal,
no pior momento possível."
Edward A. Murphy.

RESUMO

No presente projeto de conclusão buscou-se desenvolver uma aplicação web para controle de pedidos da empresa Valdemar Detergentes, com o objetivo de melhorar a gestão do negócio. Para redução de erros manuais, buscou-se a partir do presente estudo, soluções para otimizar o processo. O sistema web foi desenvolvido utilizando a plataforma Java EE, disponibilizando recursos por meio de contêineres EJB e interface com design responsivo. Utilizou-se também a API JPA para a persistência dos dados, o framework de componentes Model-View-Control JavaServer Faces e o banco de dados relacional PostgreSQL. Após a implementação da aplicação obteve-se um resultado positivo, pois com a aplicação a empresa poderá ter controle mais efetivo de suas compras e vendas, controle dos clientes e de suas compras a pagar e a receber.

Palavras-chave: ERP. Java. Responsivo.

ABSTRACT

In the present project of conclusion, we tried to develop a web application for order control of the company Valdemar Detergentes, aiming to improve the management of the business. For the reduction of manual errors, we sought from the present study, solutions to optimize the process. The web system was developed using the Java EE platform, providing resources through EJB containers and interface with responsive design. We also used the JPA API for data persistence, the JavaServer Faces Model-View-Control component framework, and the PostgreSQL relational database. After the implementation of the application a positive result was obtained, because with the application the company can have more effective control of its purchases and sales, control of customers and their purchases payable and receivable.

Keywords: ERP. Java. Responsive.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

APIs – Application Programming Interfaces
DAO – Data Access Object
EJB – Enterprise Java Beans
ERP – Enterprise Resource Planning
Java EE – Java Enterprise Edition
Java SE – Java Standard Edition
JPA – Java Persistence API
JSF – Java Server Faces
JSP – JavaServer Pages
JSR – Requisição de Especificação Java
MRP – Material Requirement Planning
MRP II – Material Requirement Planning
OMT – Object Modeling Technique
UI – interface de usuário
VD – Valdemar Detergentes

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho abordou o desenvolvimento de um sistema de pedidos com o conceito de planejamento de recurso corporativo ou *Enterprise Resource Planning* (ERP). Proporcionando com isso que a empresa tivesse um crescimento tecnológico e um melhor controle dos pedidos realizados.

A empresa Valdemar Detergentes (VD) estava passando por problemas na gestão do negócio, já que todos os cálculos de venda e compra de produtos eram feitos manualmente. Por esse motivo a gestão era dificultada e propensa a erros. Então com isso pensou-se: como aperfeiçoar a gestão da empresa VD, agilizando os processos existentes e reduzindo erros da gestão manual?

Com o desenvolvimento do sistema para a gestão, esperou-se minimizar os problemas que a gestão manual apresentava, automatizando o processo de fazer pedidos e transformando os cálculos que até então eram manuais para que a aplicação faça automaticamente.

A empresa VD não possuía qualquer tipo de sistema computacional para gestão de seus negócios. Com a falta de um sistema web para controlar os pedidos e a empresa, os clientes ligavam encomendando para a empresa quando necessitavam de algum produto, que gerava uma conta de telefone com um valor elevado para os clientes e para a empresa. Também convém citar o fato de não possuir sistema acarretava em um grande número de documentos escritos ou impressos para o controle da empresa. Segundo Geremias e Papior (2006), “...o uso de um sistema ERP para microempresas é recomendado, pois apoia a gestão de todas as atividades proporcionando uma visão mais detalhada do negócio. A necessidade de implantação desse tipo de sistema foi identificada quando os antigos procedimentos já não estavam mais sendo tão eficientes.”. Com essa afirmação, de Geremias e de Papior, não apenas a empresa VD tinha a necessidade de possuir um sistema ERP, mas como é recomendado para todas as microempresas possuir um sistema como esse, para controlar os seus negócios. Com o uso do sistema ERP que foi desenvolvido, a empresa pode ter seus processos melhorados e isso pode ajudar no seu crescimento.

1.1 OJETIVOS

A seguir serão apresentados os objetivos gerais e específicos do trabalho proposto.

1.1.1 Objetivo geral

Desenvolver uma aplicação web para pedidos, usando o conceito de ERP para gestão de negócio da empresa VD, para um melhor controle dos pedidos feitos na empresa.

1.1.2 Objetivos específicos

- Estudar o funcionamento da empresa;
- Modelar o software;
- Desenvolver uma aplicação web para realizar a gestão dos processos;

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos e tecnologias que foram utilizadas no desenvolvimento do projeto, baseando-se em pesquisas bibliográficas.

2.1 ERP

O ERP surgiu da evolução de outras duas siglas que também representavam o programa de produção: *Material Requirement Planning* (MRP) e *Material Requirement Planning* (MRP II). Em 1990 o MRP II incorporou aspectos de recursos humanos e financeiro. Com a grande mudança começou a ser chamado de ERP (TOSTES, 2009).

Segundo Colangelo (apud. Campos, 2006), MRP surgiu com o objetivo de dar apoio às funções de planejamento de produção e compras. Seus procedimentos eram baseados em listas de matérias e um plano de produção. Após algum tempo o MRP englobou aspectos financeiros, orçamentos e custeio de produção, mas ainda não englobava todos os aspectos do sistema da empresa e o suporte completo e integrado da empresa era difícil. Em 1990 sofreu mais modificações e passou a ser chamado de ERP. O ERP trabalha em um ambiente cliente-servidor, com base de dados única e integrado, que possibilita a empresa ter uma visão mais ampla da utilização de recursos em diversas áreas.

Ainda segundo Colangelo (apud. Campos, 2006), à medida que o ERP ampliava suas áreas de atuação dentro das organizações acabou por determinar a classificação para uma determinada categoria de software, que é possível ver “ERP para comércio” ou “ERP para serviços”.

2.2 Java

O Java é uma linguagem de programação usada em diversos projetos, sendo web ou *desktop*, foi lançado em 1995 pela empresa Sun Microsystems. Essa nova linguagem cresceu rapidamente entre os desenvolvedores, pois poderia ser usada em diversos sistemas operacionais. O Java é encontrado atualmente em inúmeros

servidores web, bancos de dados relacionais, computadores de grande porte, telefones móveis, sistemas de cartão de crédito entre outros (GONÇALVES, 2007).

Java é uma tecnologia usada para desenvolver aplicações que tornam a Web um lugar mais divertido e útil. Java não é igual a JavaScript, que só executado em seu browser. Java permite executar jogos, fazer uploads de fotos e fazer bate papos on-line entre outras funções. Se você não possuir o Java em seu computador muitas aplicações web simplesmente não iriam funcionar (JAVA, 2017).

Programas Java consistem em partes conhecidos como classes. As classes possuem métodos que realizam tarefas e retornam informações quando são concluídas. A linguagem Java já possui várias bibliotecas para usar em seus programas, que também são conhecidas como *Java Application Programming Interfaces* (APIs). Em geral programas Java passam por 5 fases: edição, compilação, carregamento, verificação e execução (DEITEL, DEITEL, 2010).

Segundo Deitel e Deitel (2010, p.11), “Java é uma poderosa linguagem de programação”. Por esse motivo que foi utilizado o Java na aplicação e por ela sempre ter crescido junto com os desenvolvedores existe um grande número de desenvolvedores Java e um grande número de documentação para o desenvolvimento.

2.2.1 Java EE

Java Enterprise Edition (EE) é uma plataforma padrão para desenvolver aplicações Java de grande porte para a internet, que inclui bibliotecas e funcionalidade para implementar software Java distribuído. Possui uma série de especificações com objetivos distintos (FARIA, 2013). As especificações mais conhecidas são:

- Servlet;
- JavaServer Faces (JSF);
- JavaServer Pages (JSP);
- Java Persistence API (JPA);
- Enterprise Java Beans (EJB);

A primeira versão do Java EE conhecida originalmente como J2EE, focava nas questões que as empresas enfrentavam em 1999 que era componentes distribuídos.

Com o tempo ela evoluiu se tronou mais rica em conteúdo, com maior desempenho, mais leve, e mais portátil (GONCALVES, 2011).

O J2EE foi visto como um sistema pesado: com dificuldades de codificar, de distribuir e de se executar. Então surgiram Struts, Springs, ou Hibernate e mostraram uma nova maneira de desenvolver aplicações. Após fim do primeiro trimestre de 2006 o Java EE foi liberado e se mostrou uma melhoria notável. Para os desenvolvedores o EJB3 e a nova JPA foram mais como um salto enorme para a plataforma e com isso foi introduzido o JSF (GONCALVES, 2011).

O Java EE é um subconjunto do Java Standard Edition (SE) então todas as funções Java que havia no Java SE também estão disponíveis no Java EE. O Java SE 6 foi lançado em 2006. Ele foi desenvolvido sob Requisição de Especificação Java 270 (JSR) e trouxe muitas novas funcionalidades. O Java SE 6 oferece novas ferramentas para diagnósticos, gerenciamento e monitoramento de aplicações (GONCALVES, 2011).

O sistema foi desenvolvido para ser uma aplicação web e como o Java é a linguagem de programação escolhida para a criação dos códigos de programação, o Java EE é a plataforma padrão para o desenvolvimento em aplicações web. Então por esses motivos o Java EE foi a melhor escolha para o desenvolvimento da aplicação.

2.3 Java Persistence API

O JPA é um framework para persistência em Java, que oferece uma API de mapeamento objeto-relacional e soluções para integrar persistência com sistemas corporativos escaláveis. A JPA é uma especificação, não um produto, para trabalharmos com a JPA precisaremos de uma implementação (FARIA, 2013).

A persistência dos dados de um programa é um conceito fundamental no desenvolvimento de um sistema. Se um sistema não preservasse os dados após o encerramento do programa, ele não seria prático e usual (FARIA, 2013).

Em 1998 o EJB 1.0 foi lançado, mas era um componente pesado, distribuído e usado para lógica funcional transacional. A persistência foi introduzida e aprimorada até a EJB 2.1. Depois de muitas reclamações dos componentes e em reconhecimento ao sucesso e simplicidade de estrutura de código aberto, o modelo de persistência do *Enterprise Edition* foi rearquitetado no Java EE 5. A JPA nasceu

com uma abordagem muito mais leve, que adotou muitos dos princípios de desempenho do Hibernate (GONCALVES, 2011).

A JPA 1.0 foi criada com o Java EE 5 para resolver problemas de persistência de dados. Ela possui um modelo de orientação por objetos relacionais. A JPA 2.0 segue o mesmo caminho de simplicidade e adiciona novas funcionalidades. O desenvolvedor usa a API para acessar e manipular dados relacionais a partir do EJB, componentes web e aplicações Java SE (GONCALVES, 2011).

Para uso funcional da aplicação que foi desenvolvida é necessário ter persistência dos dados que serão gravados no banco de dados. Para isso a JPA facilitará nesse sentido e por isso foi escolhida nesse sistema.

2.4 JavaServer Faces

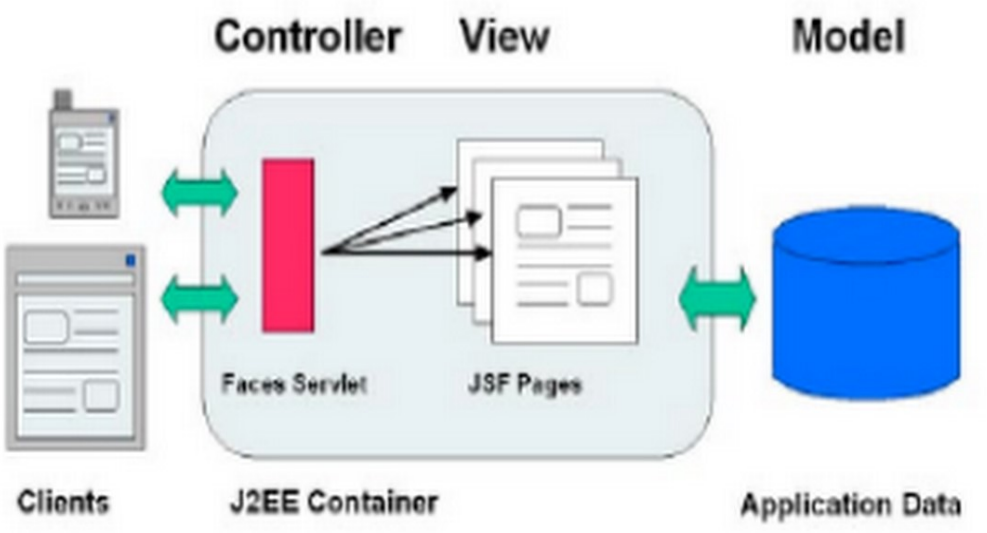
As aplicações web usando Java vem sendo desenvolvido desde que foi implementado em 1997 a especificações de Servlet. Nessa época o HTML era escrito junto ao código fonte Java, o que era incoerente com o modelo MVC (Model-View-Control). Então 1998 foi introduzido o JSP. Mesmo com ajuda do Struct o desenvolvimento de aplicações web era precário, por isso a Sun decidiu criar uma solução MVC padrão que resultou no JSF que tem sido formalmente disponível a partir de 2004 (DENONCURT, 2008).

O JSF é um framework de apresentação orientado a eventos. Por exemplo se o usuário clicar em um botão da aplicação, um *ActionEventJSF* é disparado e passados para receptores registrados. Esses receptores são os componentes de aplicação gerenciados pelo JSF. Também podemos dizer que o JSF simplifica a construção de interfaces web com o usuário em Java (BAUER, 2007).

Segundo GONÇALVES (2007, p.511) “JavaServer Faces com certeza é uma das melhores tecnologias visuais para construção de páginas web através de componentes.”. Por esse motivo o uso do JSF na aplicação foi de grande ajuda, pois facilitou a criação das páginas web já que o framework é um dos melhores.

O modelo MVC do JSF é representado pela Figura 1:

Figura 1: Modelo Model-View-Control no JavaSever Faces



Fonte: FARIA, 2013.

O verdadeiro poder do JSF está em seu modelo de componentes de interface do usuário, que gera grande produtividade aos desenvolvedores, permitindo a contrução de interfaces web com um conjunto de componentes pré-construido (FARIA, 2013).

2.5 PrimeFaces

O PrimeFaces é uma biblioteca de componentes do JSF. Inclui diversos componentes como: campos de entrada, botões, tabelas de dados, árvores, gráficos, diálogos entre outros (FARIA, 2013).

O PrimeFaces é desenvolvido e mantido pela Prime Technology, que é uma empresa de desenvolvimento de software, juntamente com atividades de consultoria e de treinamento. O PrimeFaces é uma biblioteca leve e todas as decisões tomadas pela empresa são baseadas em manter ela tão leve quanto possível. Caso adicionar soluções de terceiros ao PrimeFaces não vai haver sobrecarga do sistema (PRIMEFACES, 2017).

O PrimeFaces é um componente de interface de usuário (UI) para aplicações web usando o JSF. Possui um conjunto de mais de 100 componentes UI, que não requer configuração inicial e nem uma dependência. Porém possui com conjunto de

pacotes de extensão chamado de PrimeFaces Extensions. Essa extensão visa ser leve e rápida para o JSF 2 (JONNA, 2014).

Para agilizar o processo de criação de páginas web é recomendável usar um framework e o PrimeFaces sendo uma biblioteca leve e com inúmeros componentes pode ajudar no desenvolvimento do projeto. Caso o desenvolvedor for adicionar soluções de terceiros, o PrimeFaces não irá fazer a aplicação se tornar lenta.

2.6 PostgreSQL

O PostgreSQL começou a sua construção em meados de 1986 na Universidade Norte-Americana da Califórnia, a Universidade de Berkeley. Em 1995 obteve um interpretador SQL que recebeu o nome de Postgres95, onde teve um aumento de 25% em seu código fonte. Então em 1996 teve uma abertura completa de seu código fonte. Nessa época já era conhecido como PostgreSQL (Pereira Neto, 2007). Com a abertura do sistema, vários usuários começaram a colaborar com a melhoria do software.

O PostgreSQL tem suporte para Windows, Linux e Solaris e MAC. Possui a maioria dos tipos de dados SQL como: INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL e TIMESTAMP. Também suporta armazenamento de imagens, sons ou vídeo. Possui interfaces de programação nativas para C / C ++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre outros (POSTGRESQL, 2017).

Toda a aplicação é preciso ter um banco de dados para a persistência dos dados. O PostgreSQL é suportado em diversos sistemas operacionais e como possui licença aberta foi a escolha cabível no desenvolvimento do projeto, para guardar os dados que serão gerados pela aplicação.

2.7 UML

O Unified Modeling Language (UML) é uma linguagem visual para modelagem de softwares baseados em orientação a objetos. É uma linguagem de propósito geral e pode ser usado em todos os estágios do desenvolvimento da aplicação. Atualmente é utilizada em todo o mundo pelas empresas de desenvolvimento de softwares. A UML não é uma linguagem de programação e sim uma linguagem de

modelagem, cujo propósito é auxiliar os desenvolvedores na construção da aplicação (GUEDES, 2011).

A UML surgiu da união de 3 métodos: o método de Booch, o método Object Modeling Technique (OMT) e o método Object-Oriented Sofware Engineering (OOSE). A união desses três métodos começou em 1990 e só foi concluída em 1996. A versão 2.0 da UML foi lançada em 2005 e atualmente encontra-se na 2.3 Beta (GUEDES, 2011).

Segundo Guedes (2011), a UML possui diversos diagramas para uma visão ampla de todo o sistema. Cada diagrama demonstra uma parte do sistema, como se o sistema fosse modelado por camadas. Dentre eles temos:

- O diagrama de classes;
- O diagrama de caso de uso;
- O diagrama de objeto;
- O diagrama de pacote;
- O diagrama de sequência;
- O diagrama de comunicação;
- O diagrama de máquina de estado;
- O diagrama de atividade;
- O diagrama de visão geral de interação;
- O diagrama de componentes;
- O diagrama de implantação;
- O diagrama de estrutura composta;
- O diagrama de tempo.

A UML é importante para este projeto de conclusão, pois é necessário em qualquer projeto pelo mundo, arquitetar o que será desenvolvido e para isso será usado a UML na criação dos diagramas de classes, digramas de casos de uso e diagramas de sequência. Também podemos citar que será feito a descrição dos casos de uso.

3 MODELAGEM E ESTUDO DE CASO

Para elaboração deste trabalho de conclusão analisou-se a história, funcionamento da empresa VD e requisitos do sistema pedido. A partir dos dados levantados foi modelado o sistema para seu desenvolvimento, fazendo uso da linguagem de modelagem UML.

Foram escolhidas diversas tecnologias baseadas nas pesquisas bibliográficas como: Banco de dados PostgreSQL para gerenciamento dos dados, Java Persistence API para persistência dos dados e a parte web foi escolhido o Framework JSF com a biblioteca BootsFaces, estes últimos que auxiliam na implementação de um design responsivo na aplicação.

3.1 Estudo de Caso

Neste capítulo será apresentado o estudo de caso efetuado, e será descrito o método utilizado atualmente pela empresa para gestão do negócio e as falhas encontradas para execução da gestão.

3.1.1 História da Empresa

A empresa VD surgiu no ano de 2010 tendo como membro fundador Valdemar Soares. A empresa atua na venda de detergentes em regiões de Santa Catarina e em algumas cidades do Rio Grande do Sul.

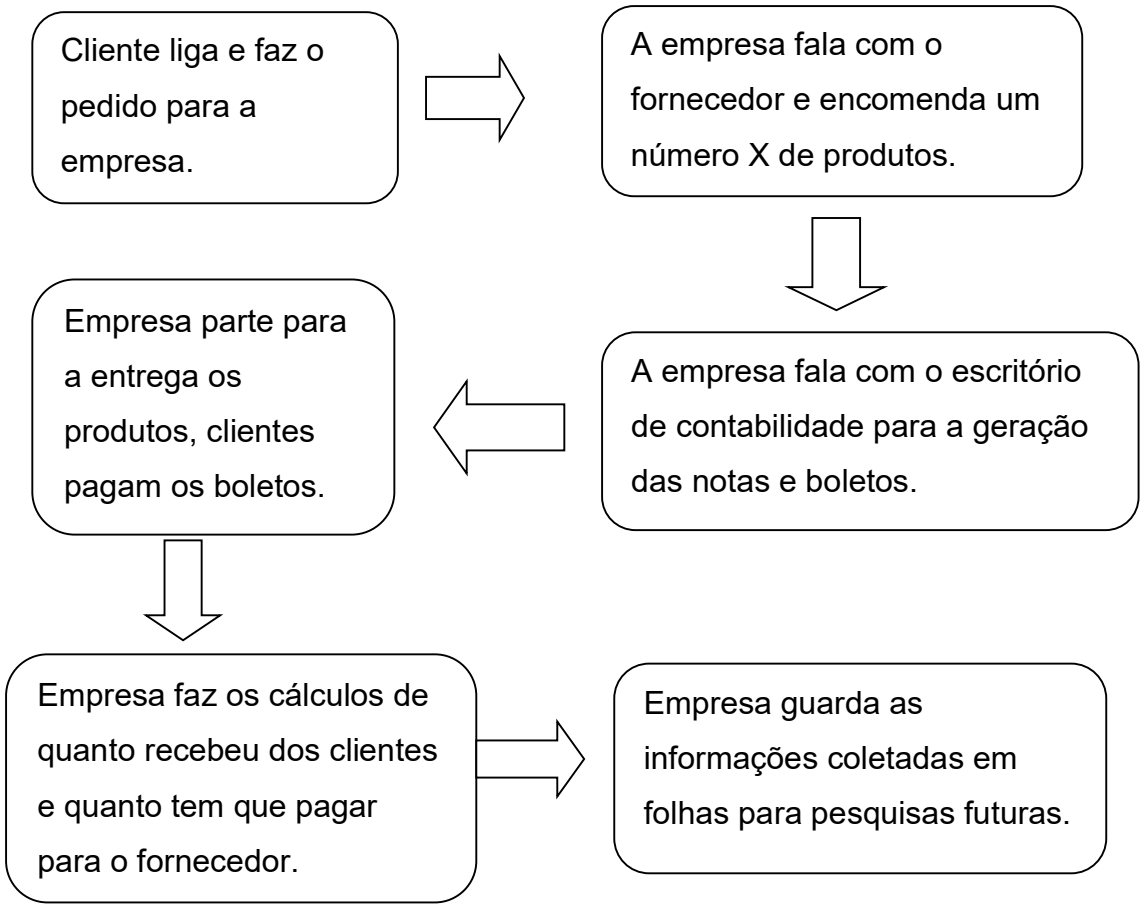
A empresa é de pequeno porte com apenas um funcionário, este sendo o Valdemar. A empresa compra produtos de um fornecedor da cidade de Tuparendi e revende nas áreas de atuação citadas anteriormente. As entregas são feitas por um caminhão Truck 1111.

3.1.2 Gestão do negócio.

A empresa até o momento em que foi escrito esta monografia não possuía um sistema computacional, e toda a gestão do negócio se dava por processo manual e a contabilidade é realizada por um escritório do ramo. Com isso gerava inúmeros documentos impressos e escritos à mão. As contas a receber eram geradas pelo empregado conforme ele escrevia ou memorizava seu o valor, informando a cada cliente. Já as contas a pagar eram geradas feita verificando com o fornecedor o total a pagar.

Todos os cálculos feitos a punho são sujeitos a falhas, sendo muito comum o fornecedor dos produtos dizer o total de um valor e a empresa VD ter como o valor a pagar sendo um valor divergente dependendo dos erros de cálculos relatados, demonstrado na figura 2.

Figura 2: Fluxograma para fazer pedido



Fonte: do Autor

3.1.3 Funcionamento da empresa

Até então a empresa funcionava da seguinte forma: os clientes ligavam para um número particular do funcionário Valdemar e faziam o pedido de quais produtos desejam, informando o nome e quantidade em litros de cada produto, onde as bombonas[1] podem variar de 20 a 1000 litros.

[1] Recipiente plástico resistente, revestido pelas laterais com ferro, usado para armazenamento de produtos químicos.

Com os pedidos de inúmeros clientes de uma determinada região a empresa fazia o pedido para os fornecedores fabricarem os produtos e carregarem o caminhão. Enquanto este processo é feito a empresa solicita a emissão dos boletos para o escritório de contabilidade.

Com o caminhão carregado com todos os produtos pedidos e alguns a mais caso conquiste novos clientes ou algum cliente deseje algo a mais do que foi pedido, também com os boletos em mão o funcionário parte para a entrega do produto.

Após entregar os produtos o funcionário volta para a sua cidade e faz a conta de quanto deve para o fornecedor, quanto deu de lucro e quais foram os gastos extras. Com isso em mãos ele paga o fornecedor, descarrega as bombonas vazias e deposita o lucro na conta da empresa.

3.1.4 Problemas existentes na empresa

A empresa apresenta como principal problema a falta de gerenciamento. O proprietário da empresa não possui funcionários. Ele é responsável por tudo: vender, faturar, entregar, controlar estoques, entrada e saída de caixa. Devido à falta de formação do proprietário, seus métodos de controle são antigos, não fazendo uso de novos métodos e das tecnologias que facilitam a vida dos empresários.

Outro problema enfrentado é a má gestão da empresa em controlar suas contas a receber causando a inadimplência dos clientes. Onde muitas vezes empresário recebe cheques pré-datados e boletos com vencimentos em 30 e 60 dias. No dia do pagamento não existindo um controle para o recebimento destes chegues ou boletos os clientes, acabam muitas vezes não cumprindo com suas obrigações.

3.2 Modelagem do Sistema

Para a modelagem do sistema foi marcado uma reunião com o representante da empresa e foram vistos pontos relevantes para colocar na aplicação. Nesta seção apresenta-se os requisitos funcionais e não-funcionais e os diagramas que ilustram os funcionamento e estrutura do software.

3.2.1 Requisitos Funcionais

RF1. O sistema deve manter um cadastro de cliente.

RF2. O sistema deve manter um cadastro dos produtos vendidos pela empresa.

RF3. O sistema deve permitir visualizar as contas a receber das vendas realizadas para os clientes.

RF4. O sistema deve possuir uma página para fazer pedidos.

RF5. O sistema deverá dispor de mecanismos de buscas para os produtos.

RF6. O sistema deve manter um cadastro dos produtos comprados pela empresa.

3.2.2 Requisitos Não Funcionais

RNF1. O sistema deverá utilizar design responsivo, para o maior número de dispositivos compatíveis.

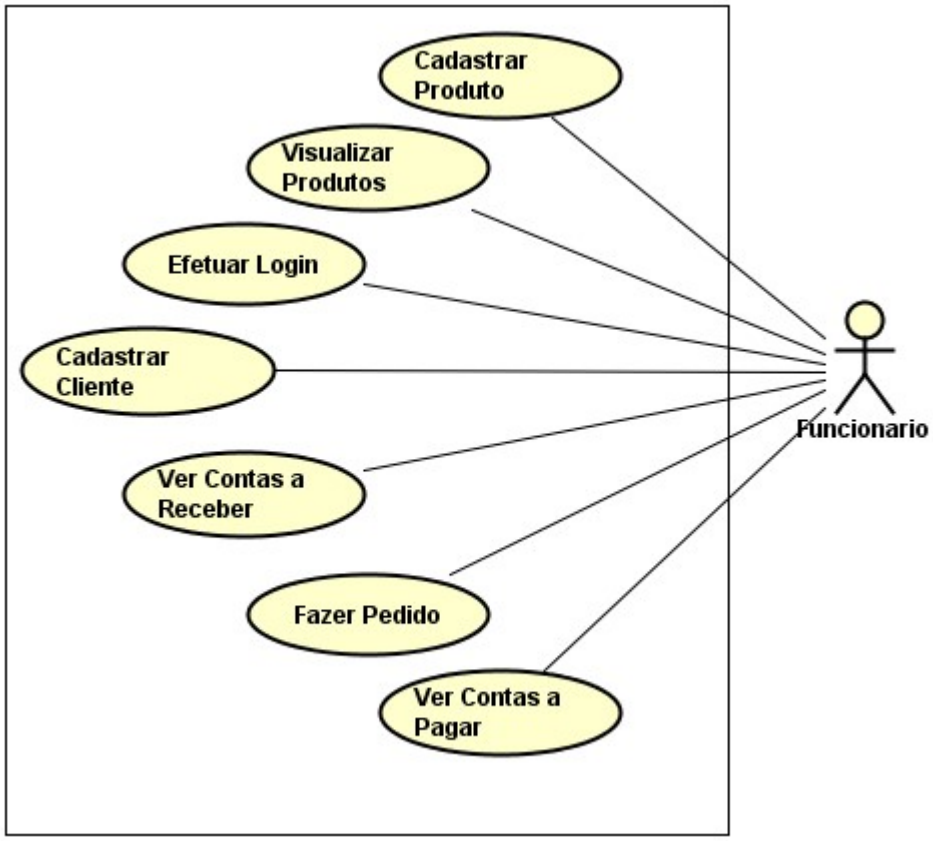
RNF2. O sistema deverá ser compatível com o maior número de plataformas de sistemas operacional.

RNF3. O sistema deverá utilizar de tecnologias compatíveis aos padrões atuais da computação.

3.3 Diagrama de Casos de Uso

O diagrama de casos de uso tem como objetivo especificar como o sistema funcionará do ponto de vista do usuário e é fundamental para a visualização simplificada das funções dos atores no programa. O diagrama demonstrado a seguir na Figura 3.

Figura 3: Casos de Uso



Fonte: Do Autor

3.4 Descrição do Caso de Uso

A descrição dos casos de uso tem como objetivo detalhar as funções do autor com os casos de uso.

3.4.1 Caso de Uso Fazer Pedidos

Descrição do Caso de Uso	**Fazer Pedidos**
Ator Principal	Funcionário.
Resumo	Esse caso de uso descreve como é feito o pedido de novos produtos pelos clientes.
Pré-condições	Fazer *Login*
Pós-condições	Pedido cadastrado para enviar ao cliente.
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Cria uma nova venda	
	2. Adicionar o produto a venda.
3. Visualizar a venda e finalizar pedido.	
	4. Salvar pedido no sistema.
Fluxo Secundário	
Ações do Ator	**Ações do Sistema**
1. Selecionar a venda para altera-la.	
	2. Editar o que for necessário.
3. Visualizar alterações e salvar.	
	4. Salvar alteração no sistema.
Fluxo Secundário	
Ações do Ator	**Ações do Sistema**
1. Procurar a vendo e clicar no botão de excluir.	
	2. Salvar alteração no sistema.

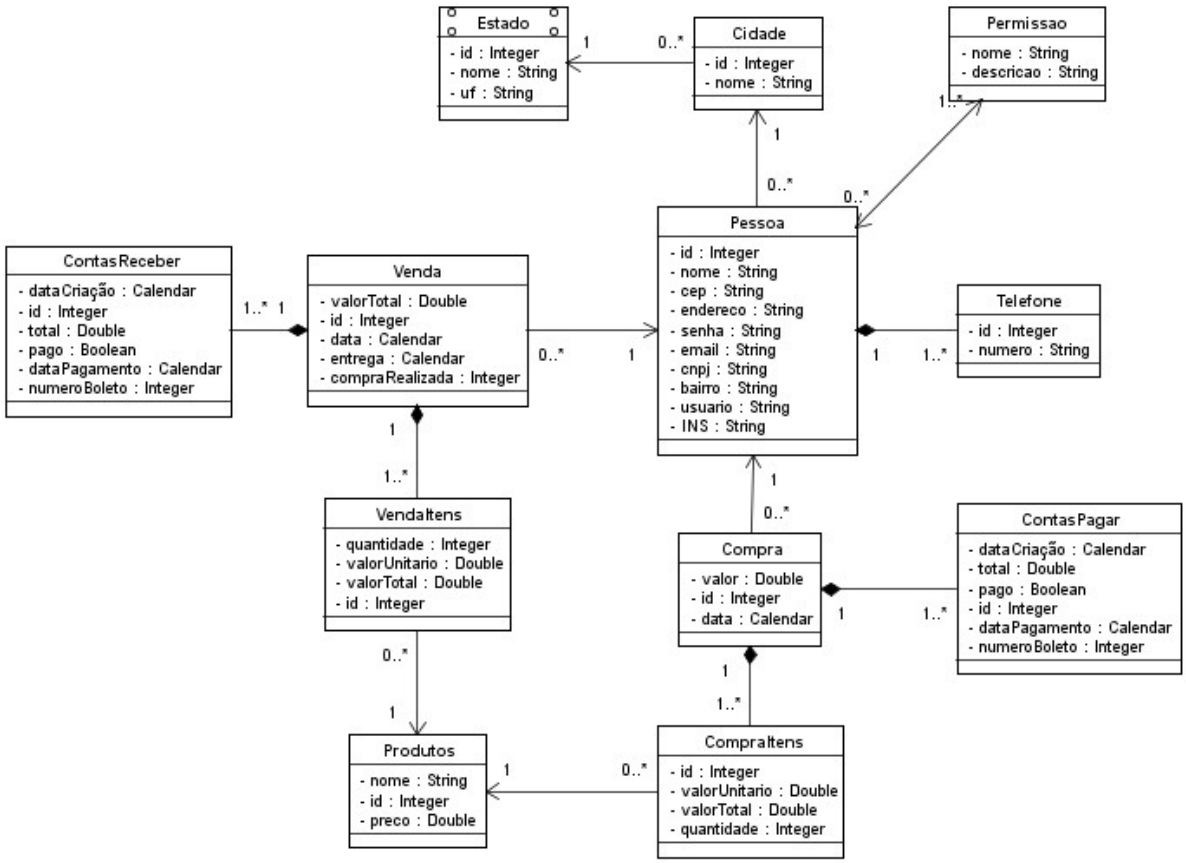
3.4.2 Caso de Uso Manter Contas a Receber

Descrição do Caso de Uso	**Manter Contas a Receber**
Ator Principal	Funcionário.
Resumo	Esse caso de uso descreve como são controladas as contas que a empresa tem para receber de seus clientes.
Pré-condições	Fazer *Login*
Pós-condições	
Fluxo Principal	
Ações do Ator	**Ações do Sistema**
1. Selecionar a conta de um determinado cliente.	
	2. Mostrar conta detalhada.
3. Informar se está totalmente pago.	
	4. Atualizar as contas do cliente.
Fluxo Secundário	
Ações do Ator	**Ações do Sistema**

3.5 Diagrama de Classes

A seguir será apresentado o diagrama de classes, que será usado para o desenvolvimento do sistema, demonstrado na Figura 4.

Figura 4: Diagrama de Classes



Fonte: Do Autor

A classe “Pessoa” contém os dados dos clientes, que usarão o sistema, como “email” e “senha” para fazer login no sistema. Também possui os dados pessoais e de localização da empresa, para poder levar os produtos até o local informado. A classe “Pessoa” possui uma relação muitos para muitos com a tabela de “Permissao” para que seja feito o *login* na aplicação.

As classes “Venda” e “Compra” são para informar o que a empresa VD vendeu para os clientes ou o que ele comprou do fornecedor. Possui o atributo “data” para informar o dia que foi efetuado a compra ou a venda. As classes “VendaItens” e “CompraItens” são para informar respectivamente quais produtos foram vendidos pela empresa ou comprados do fornecedor, possui os atributos de “quantidade”, “valotTotal”, e “valorUnitario” para ter o valor de cada produto e quantos produtos foram vendidos ou comprados.

Temos também as classes “ContasReceber” e “ContaPagar” que são referentes as contas que os clientes devem pagar para a empresa e as contas que a

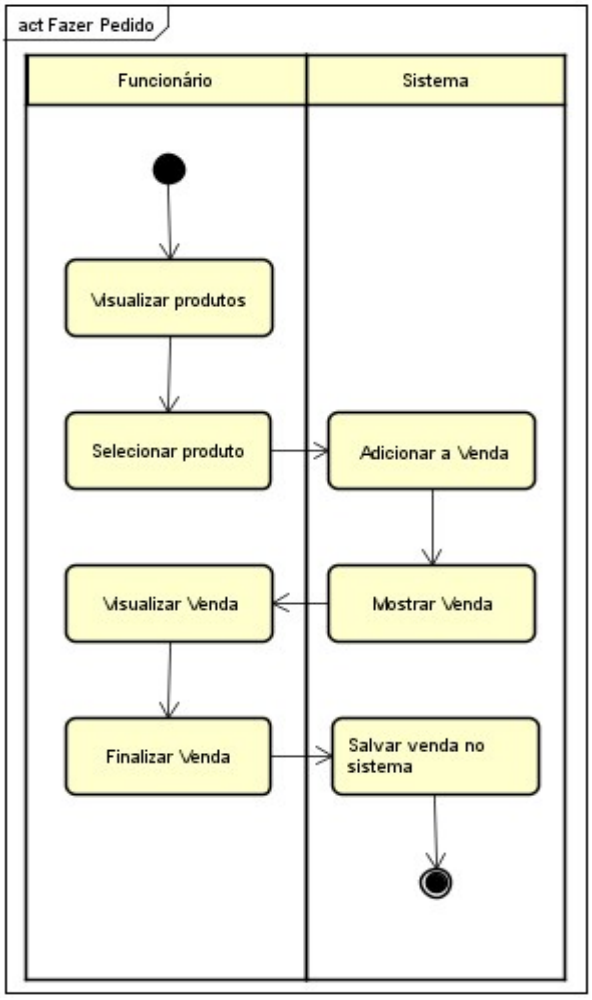
empresa deve pagar para o fornecedor. Com essas classes podemos ter um controle financeiro, e saber qual dia foi gerado cada conta, tirando assim erros da gestão manual que poderiam vir a ocorrer.

Por fim a classe “Produto” que tem os dados de cada produto que a empresa pode vender. Tem os atributos “preco” e “nome” para informar o preço do produto e nome do mesmo, para ser disponibilizado para venda na página inicial do site.

3.6 Diagrama de Atividades

Os diagramas de atividades são essenciais para demostrar as etapas envolvidas em um determinado processo. O diagrama a seguir ilustrado na Figura 5 demostra como será o processo de fazer um pedido.

Figura 5:Fazer Pedido

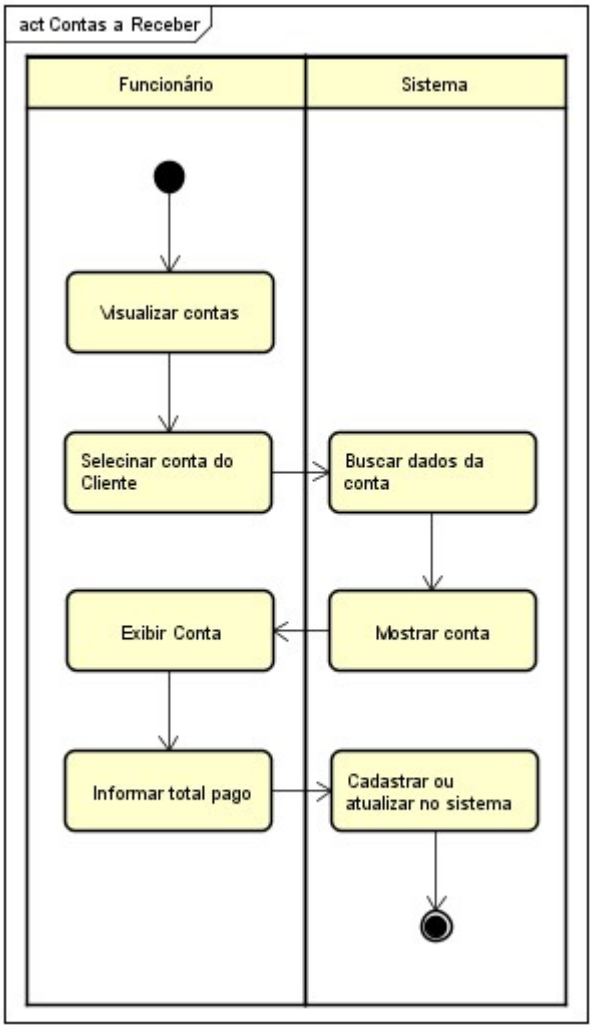


Fonte: Do Autor

O diagrama apresentado demonstra o funcionário visualizando os produtos, então seleciona o produto para adiciona-lo a venda, que por sua vez o sistema retorna mostrando o produto ou os produtos que o funcionário selecionou, por fim o cliente finaliza o pedido após conferir os dados do pedido.

O próximo diagrama demostra as contas a receber que são geradas pelo sistema após um cliente fazer um pedido. Demonstrado na Figura 6.

Figura 6: Contas a Receber



Fonte: Do Autor

O diagrama de contas a receber demostra o funcionario visualizando todas as contas que estão faltando pagamentos. Então ele seleciona a conta de um determinado cliente e informa o total que aquele cliente pagou, para enfim atualizar os dados do cliente.

4 DESENVOLVIMENTO

Neste capítulo serão descritos os procedimentos utilizados, a estrutura da camada da aplicação e os elementos ocupados para o desenvolvimento da aplicação web.

4.1 Ambiente de Desenvolvimento

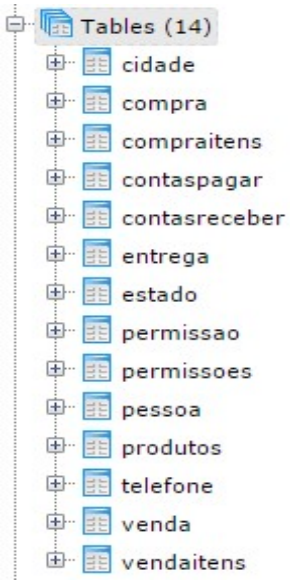
Nesta seção serão descritos os a tecnologias usadas para o desenvolvimento do trabalho.

4.1.1 IDE e Servidor

Para o desenvolvimento do sistema de pedidos, utilizou-se uma IDE e um servidor. O servidor escolhido para a aplicação foi o GlassFish server versão 4.0. A IDE escolhida foi o NetBeans IDE versão 8.2 que tem a possibilidade de interação com o servidor GlassFish. Ambos são necessários para a construção do código e administração do sistema.

Para gerenciar e administrar o banco de dados da aplicação utilizou-se os Sistema gerenciador de banco PostgreSQl. Por meio do servidor GlassFish, deste sistema ERP, pode-se guardar os dados que seriam gerados na aplicação. Como Ilustrado na Figura 7.

Figura 7: Tabelas do banco de dados



Fonte: do Autor

4.1.2 Bibliotecas e Frameworks

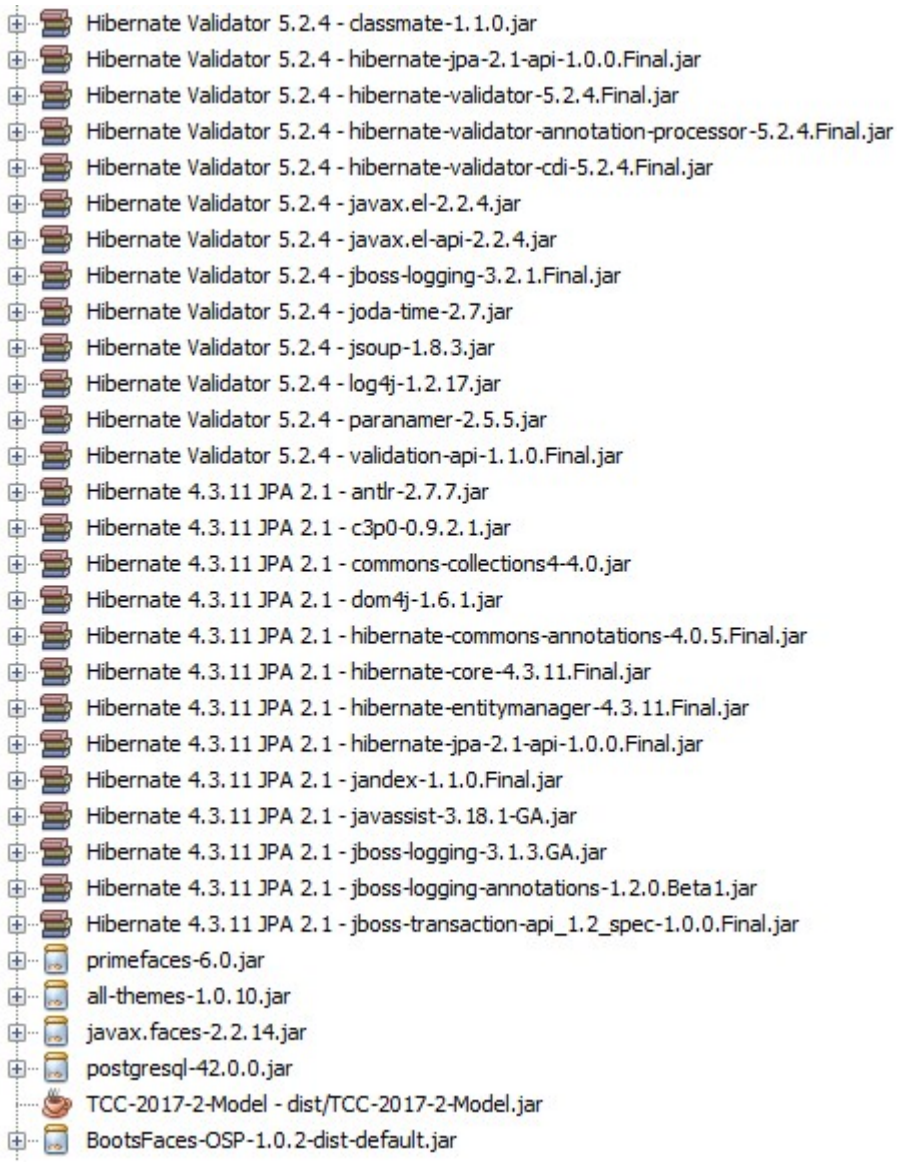
Para que fosse possível implementar o sistema foi necessário utilizar-se de algumas bibliotecas e frameworks. Para maior agilidade no desenvolvimento da aplicação e para implementação de um design responsivo.

- **Biblioteca Hibernate Validator 5.2.4:** A biblioteca Hibernate Validator 5.2.4, foi utilizada para validar os dados persistidos, impedindo dados incoerentes com as anotações feitas na camada de modelo.
- **Biblioteca Hibernate JPA 4.3.11 JPA 2.1:** A biblioteca Hibernate JPA é uma biblioteca de persistência de dados e foi aplicada para mapeamento de objeto-relacional.
- **Biblioteca BootsFaces 1.0.2:** A biblioteca BootsFaces é uma biblioteca de recursos gráficos para o JSF fundamentada no estilo visual da biblioteca Bootstrap. A finalidade de empregar essa biblioteca ao sistema consiste em que seus recursos são elaborados para a utilização em layout responsivo, sendo este um dos objetivos da aplicação web.
- **Biblioteca PrimeFaces 6.0:** A biblioteca PrimeFaces possui como sua principal funcionalidade, auxiliar na implementação de recursos gráficos.

- **Driver JDBC PostgreSQL:** O driver JDBC do PostgreSQL dispõe de elementos para a conexão do sistema web com o banco de dados, possibilitando gerenciar e manipular os dados que compõe as tabelas da aplicação.

Na Figura 8 podemos ver as Bibliotecas que integrantes do sistema.

Figura 8: Bibliotecas do projeto



Fonte: do Autor

4.2 Padrões do Sistema

Assim que o sistema é acessado pode-se ver na parte superior da tela um menu para navegação onde aparece um *link* para acessar o início da página. Um *drop menu* com os cadastros de usuário e controles de vendas e compra, um *drop menu* com as contas a receber e contas a pagar e então um *drop menu* com o usuário conectado, como visto na Figura 9.

Figura 9: Navegação

Fonte: do Autor

Nas páginas de manutenção do sistema temos como padrão os seguintes campos (Figura 10):

1. Botão inserir: para inserir um novo registro;
2. Botão editar: para editar um registro;
3. Botão excluir: para excluir o registro selecionado;
4. Campo de texto: para pesquisas na página em que o usuário está usando no momento.

Figura 10: Novo, Alterar, Excluir, Pesquisar

Fonte: do Autor

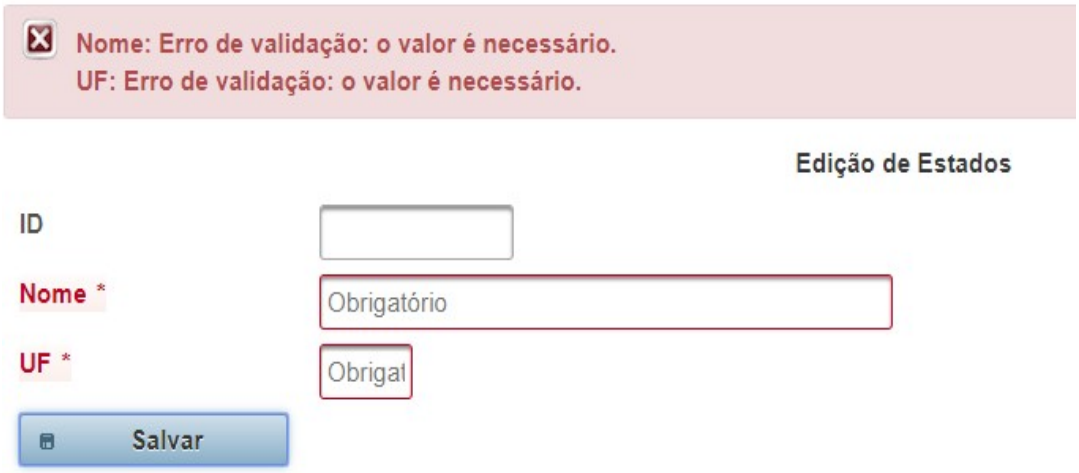
Quando alguma ação do sistema é executada como por exemplo: alterar um registro uma mensagem é disparada para informar ao usuário que executou a ação saiba o que está acontecendo. Também são disparadas mensagem por validação de

campos como por exemplo um campo não poder estar em branco, demonstrado na Figura 11 e Figura 12.

Figura 11: Objeto persistido

Fonte: do Autor

Figura 12: Validação dos campos



Fonte: do Autor

4.3 Estrutura e Layout da Aplicação

Nessa seção será apresentado o layout da aplicação e sua arquitetura.

4.3.1 Camada de Modelo

A camada de modelo da aplicação é encarregada de manter as classes que se tornarão objetos instanciados pela aplicação e serão persistidos. Por meio da JPA criam-se relações entre os objetos e a tabela do banco de dados, fazendo uso de anotações feitas nas classes da aplicação. Dessa forma, determina-se propriedades, tipos, nomes e restrições de dados que serão arquivadas nas tabelas do banco de dados.

No caso das classes, devemos seguir o padrão JavaBeans, com esse padrão define-se as características que as classes devem possuir, sendo persistente ou não. O padrão JavaBeans consiste em implementar a interface *Serializable* e conter atributos encapsulados, para fazer uso por meio de métodos *getter* e *setter* como podemos ver na Figura 13.

Figura 13: Classe Java Pessoa

```
@Entity
@Table(name = "pessoa")
public class Pessoa implements Serializable {

    @Id
    @SequenceGenerator(name = "seq_pessoa", sequenceName = "seq_pessoa_id", allocationSize = 1)
    @GeneratedValue(generator = "seq_pessoa", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @NotNull(message = "O nome não pode ser nulo")
    @NotBlank(message = "O nome não pode ser em branco")
    @Length(max = 100, message = "O nome não pode ter mais que {max} caracteres")
    @Column(name = "nome", length = 100, nullable = false)
    private String nome;
    @NotNull(message = "A cidade deve ser informada")
    @ManyToOne
    @JoinColumn(name = "cidade", referencedColumnName = "id", nullable = false)
    private Cidade cidade;
    @NotNull(message = "O cep não pode ser nulo")
    @NotBlank(message = "O cep não pode ser em branco")
    @Length(max = 8, message = "O cep não pode ter mais que {max} caracteres")
    @Column(name = "cep", length = 8, nullable = false)
    private String cep;
    @NotNull(message = "O endereço não pode ser nulo")
    @NotBlank(message = "O endereço não pode ser em branco")
    @Length(max = 100, message = "O endereço não pode ter mais que {max} caracteres")
    @Column(name = "endereco", length = 100, nullable = false)
    private String endreco;
```

Fonte: do Autor

No trecho de código mostrado na figura 8 podemos ver as anotações @*Entity* e @*Table*(*name*=”pessoa”), onde são definidos respectivamente que este objeto é uma entidade do banco. As anotações @ID informam que a variável id declarada nesta classe é o identificador da classe e será criada no banco como chave primaria; @*SequenceGenerator* e @*GeneratedValue* servem para criar uma *sequence* no banco que irá gerar o id automaticamente ao inserir um novo registro, somando mais um a cada chamada da *sequence* e atribui este valor à variável id. Também temos a anotação @*column* que serve para representar que este campo é uma coluna na tabela de pessoas no banco de dados.

Para validação dos dados temos as anotações @*NotNull* para que os valores inseridos não sejam nulos; @*NotBlank* para os valores são serem em branco e

@Length para os valores ter um tamanho máximo quando forem inseridos. Se os valores inseridos não estiverem de acordo retorna uma mensagem ao usuário que é definida em *message* de forma personalizada. Na Figura 14 podemos ver a estrutura da camada de modelo.

Figura 14: Classes da camada de modelo

Fonte: do Autor

Um arquivo essencial para implementação do projeto é o arquivo “persistence.xml”. Este arquivo possui informações relevantes a unidade de persistência e as configurações com a finalidade de efetivar a comunicação com o banco de dados. Neste arquivo possuímos o elemento <*persistence-unit*> que define qual a nossa unidade de persistência e o elemento <*provider*> definiu o Hibernate como provedor de persistência. Também possui os elementos <*property*> onde define-se que o banco de dados fará a atualização automática das tabelas e informa que o dialeto padrão usado para consultas SQL é o do PostgreSQL. Como podemos ver na figura 15.

Figura 15: Trecho de cógigo do arquivo persistence.xml

```
<persistence-unit name="TCC-2017-2-WebPU" transaction-type="JTA">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <jta-data-source>jdbc/tcc-2017</jta-data-source>
  <class>br.edu.ifsul.modelo.Cidade</class>
  <class>br.edu.ifsul.modelo.Compra</class>
  <class>br.edu.ifsul.modelo.CompraItens</class>
  <class>br.edu.ifsul.modelo.ContasPagar</class>
  <class>br.edu.ifsul.modelo.ContasReceber</class>
  <class>br.edu.ifsul.modelo.Entrega</class>
  <class>br.edu.ifsul.modelo.Estado</class>
  <class>br.edu.ifsul.modelo.Pessoa</class>
  <class>br.edu.ifsul.modelo.Produtos</class>
  <class>br.edu.ifsul.modelo.Telefone</class>
  <class>br.edu.ifsul.modelo.Venda</class>
  <class>br.edu.ifsul.modelo.VendaItens</class>
  <class>br.edu.ifsul.modelo.Permissao</class>
  <properties>
    <property name="hibernate.hbm2ddl.auto" value="update"/>
    <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
    <property name="hibernate.transaction.jta.platform"
              value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
    <property name="hibernate.classloading.use_current_tccl_as_parent" value="false"/>
  </properties>
</persistence-unit>
```

Fonte: do Autor

O arquivo “glassfish_resource.xml” é responsável por ter definições referentes a conexão com o banco, auxilia na criação de pool de conexões e recursos JDBC no servidor. Neste arquivo tem propriedades referentes a conexão com o banco de dados, são elas: *portNumber* (informa a porta usada para conexão com banco); *serverName* (nome do servidor utilizado); user (usuário do banco); *dataBaseName* (nome do banco de dados); *password* (senha do banco); URL (caminho para acesso ao banco); *driveClass* (classe responsável pela conexão JDBC). Como visto na figura 16.

Figura 16: trecho de código do arquivo glassfish_resource.xml

```
<property name="serverName" value="localhost"/>
<property name="portNumber" value="5432"/>
<property name="databaseName" value="TCC-2017-2"/>
<property name="User" value="postgres"/>
<property name="Password" value="ariel"/>
<property name="URL" value="jdbc:postgresql://localhost:5432/TCC-2017-2"/>
<property name="driverClass" value="org.postgresql.Driver"/>
```

Fonte: do Autor

4.3.2 Classes DAO

As classes Data Access Object (DAO) da aplicação web desenvolvida, são responsáveis por gerenciar a comunicação entre o banco de dados, a unidade de persistência e os demais elementos do sistema que utilizam a persistência. As classes DAO possuem um gerenciador de entidades para acessar as classes da camada de modelo chamado de *EntityManager*. Com ele pode-se manusear os dados por intermédio da unidade de persistência determinada pela anotação *@PersistenceContext*. Visualizado na figura 17.

Figura 17: Trecho de código do arquivo DAOGenerico

```
public class DAOGenerico<T> implements Serializable {

    // lista paginada
    private List<T> listaObjetos;
    // lista com todos os objetos
    private List<T> listaTodos;
    @PersistenceContext(unitName = "TCC-2017-2-WebPU")
    protected EntityManager em;
    protected Class classePersistente;
    protected String ordem = "id";
    protected String filtro = "";
    protected Integer maximoObjetos = 20;
    protected Integer posicaoAtual = 0;
    protected Integer totalObjetos = 0;
```

Fonte: do Autor

Na aplicação web, pode-se reaproveitar o código da classe DAO genérica, que contém métodos de listar, deletar, atualizar e criar classes da camada de modelo no banco de dados. A classe DAO genérica dispõe de alguns recursos, tais como filtro e ordenação dos resultados da pesquisa, quantidade de registros por página a ser visualizada e paginação. Todas as classes da camada DAO do projeto implementam o padrão JavaBeans, por isso implementam a interface *Serializable*, as classes distintas receberam *@Stateful*, para poder guardar o estado da sessão, tornando-se assim EJBs para o uso na camada de controle. Como pode ser visto na Figura 18:

Figura 18: Trecho de código estendendo do ContasReceberDAO

```
@Stateful
public class ContasReceberDAO<T> extends DAOGenerico<ContasReceber> implements Serializable{
```

Fonte: do Autor

Para que fosse possível visualizar apenas as contas que ainda estavam por receber, o método “getListaObjetos” foi reescrito na classe contas a receber e adicionado a cláusula de “where pago = a” onde a é uma variável booleana que pode ser *true* ou *false*, assim ele pode listar as contas pagas e as contas que ainda estão para receber. O trecho de código descrito pode ser visto na Figura 19:

Figura 19: Trecho de código getListaObjetos

```
public List<ContasReceber> getListaObjetos() {
    String jpql = "from " + classePersistente.getSimpleName();
    String where = " where pago = " + a;
    // limpando o filtro contra injeção de SQL
    filtro = filtro.replaceAll("[';-]","");
    if (filtro.length() > 0){
        if (ordem.equals("id")){
            try {
                Integer.parseInt(filtro);
                where += " and " + ordem + " = '" + filtro + "' ";
            }catch (Exception e){}
        } else {
            where += " and upper(" + ordem + ") like '" + filtro.toUpperCase() + "%' ";
        }
    }
    jpql += where;
    jpql += " order by "+ordem;
    totalObjetos = em.createQuery(jpql).getResultList().size();
    return em.createQuery(jpql).setFirstResult(posicaoAtual).
            setMaxResults(maximoObjetos).getResultList();
}
```

Fonte: do Autor

4.3.3 Conversores

Como as páginas web entendem somente texto, então para que compreendam os arquivos Java quando listado na aplicação web, torna-se necessária a conversão dos dados da aplicação para o tipo *String*, para que seja possível exibí-los na interface web. Citando um dos casos, tendo-se dados do tipo *Calendar*, que é utilizado para representar dados referentes a um determinado tempo, somente pode-

se visualizar na tela com a conversão para o tipo *String*, pois os elementos da interface não manipulam dados do tipo temporal. Como pode ser analisado no trecho de código mostrado na Figura 20:

Figura 20: Trecho de código ConverterCalendar

```
@FacesConverter(value = "converterCalendar")
public class ConverterCalendar implements Serializable, Converter{

    private SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");

    @Override
    public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
        try {
            Calendar retorno = Calendar.getInstance();
            retorno.setTime(sdf.parse(string));
            return retorno;
        } catch (Exception e) {
            return null;
        }
    }

    @Override
    public String getAsString(FacesContext fc, UIComponent uic, Object o) {
        if(o == null){
            return null;
        }
        Calendar obj = (Calendar) o;
        return sdf.format(obj.getTime());
    }
```

Fonte: do Autor

No ERP foram feitos conversores para fazer a conversão de String para objetos Java e de objetos Java para *String*. Por meio dos métodos abstratos implementados pela interface Converter, pode-se receber uma *String* por parâmetro e retornar um objeto da classe que está sendo convertida, por intermédio do método “getAsObject()”. Para fazer a opção inversa tem-se o método “getAsString()”, responsável por receber um objeto e retornar uma *String*. Para utilizar os conversores, emprega-se uma instância da classe *EntityManager*, com a finalidade de acessar os dados da camada de modelo e converte-los em *String* como visto na Figura 21:

Figura 21: Trecho de código ConverterCidade

```
@FacesConverter(value = "converterCidade")
public class ConverterCidade implements Serializable, Converter {

    @PersistenceContext(unitName = "TCC-2017-2-WebPU")
    private EntityManager em;

    @Override
    public Object getAsObject(FacesContext fc, UIComponent uic, String string) {
        if(string == null || string.equals("Selecione um registro")){
            return null;
        }
        return em.find(Cidade.class, Integer.parseInt(string));
    }

    @Override
    public String getAsString(FacesContext fc, UIComponent uic, Object o) {
        if(o == null){
            return null;
        }
        Cidade obj = (Cidade) o;
        return obj.getId().toString();
    }
```

Fonte: do Autor

4.3.4 Camada de Controle

A camada de controle é responsável pela comunicação entre os elementos da camada de persistência com a interface gráfica. Por meio dos métodos contidos dentro dos controladores da aplicação a camada de visão é capaz de solicitar dados à camada de controle, que por sua vez solicita a camada DAO, os dados contidos na camada de modelo, transmitindo por intermédio da interface, com o propósito de manipular e persistir esses dados.

No começo da classe de controle, determina-se o nome pelo qual a classe será chamada, onde usou-se o atributo *value* da anotação *@Named* para definir o nome. Para definir o ciclo de vida do *ManagedBean* dos *controler* da aplicação, utilizou-se a anotação *@SessionScoped*, com isso o *bean* é mantido até a sessão do usuário encerrar ou expirar. Os atributos que integram essas classes podem ser EJBs da camada DAO, dessa forma pode-se trabalhar com os dados do banco de dados. Entre as atividades que as classes de controle possuem, apresentam-se as funções

de listar, inserir, editar e excluir dados, tal como a aplicação de paginação em listas como demonstrado na Figura 22:

Figura 22: Trecho de código ControleCidade

```
@Named(value = "controleCidade")
@SessionScoped
public class ControleCidade implements Serializable{
    @EJB
    private CidadeDAO<Cidade> dao;
    private Cidade objeto;
    private Boolean editando;
    @EJB
    private EstadoDAO<Estado> estadoDAO;
```

Fonte: do Autor

4.3.5 Camada de Visão

A camada de visão do ERP, possui a finalidade de adaptar-se ao tamanho da tela de qualquer dispositivo, com a utilização do design responsivo na configuração do layout. A interface gráfica da aplicação foi implementada com a utilização do framework JSF juntamente com as bibliotecas PrimeFaces e BootsFaces. Desta forma de implementação a construção do layout das páginas fica simplificado, pois abstrai o desenvolvimento por meio das linguagens HTML e CSS, permitindo a utilização de componentes prontos disponibilizados pelas bibliotecas.

O arquivo utilizado como base para a implementação do layout é o “template.xhtml”, este arquivo contém locais editáveis que foram usados pelos arquivos que fazem parte do layout do sistema. O menu do sistema foi construído por intermédio do elemento *<b:navbarLinks>*, no caso do local para onde foi destinado o conteúdo das páginas, foi implementado por meio do elemento *<b:dropMenu>* e suas tag filhas *<b:navCommandLink>* onde tem como ação redirecionar o usuário para as tabelas que tem seu nome descrito no *value* desse elemento e buscar seus respectivos dados. Trecho do código pode ser visualizado na Figura 23:

Figura 23: Trecho de código template.xhtml

```
<h:head>
    <title><ui:insert name="titulo">Título da Página</ui:insert></title>
</h:head>
<h:body>
    <b:container>
        <h:form id="formMenu">
            <b:navBar brand="VD">
                <b:navbarLinks>
                    <b:navLink icon="home" value="Início" outcome="inicio"/>
                    <b:dropMenu icon="cog" value="Cadastros">
                        <b:navCommandLink icon="globe" value="Estado"
                        <b:navCommandLink icon="globe" value="Cidade"
                        <b:navCommandLink icon="book" value="Produtos"
                        <b:navCommandLink icon="book" value="Contas a pagar"
                        <b:navCommandLink icon="book" value="Contas a receber"
                        <b:navCommandLink icon="user" value="Pessoas"
                        <b:navCommandLink icon="credit-card" value="Vendas"
                        <b:navCommandLink icon="credit-card" value="Compra"
                    </b:dropMenu>
                </b:navbarLinks>
            </b:navBar>
        </h:form>
        <h:panelGroup id="conteudo">
            <ui:insert name="conteudo">
                Aqui vai o conteudo das paginas
            </ui:insert>
        </h:panelGroup>
    </b:container>
</h:body>
```

Fonte: do Autor

Para apresentar a tela inicial do sistema, utilizou-se o arquivo “index.xhtml”, para isso empregaram-se elementos do JSF por meio de facelets. Com o elemento *<ui:composition>* viabilizou-se a inserção no arquivo template.xhtml, por meio do elemento *<ui:define>* foram definidos o título e o conteúdo. A Figura 24 apresenta trecho do código do arquivo “index.xhtml”:

Figura 24: Trecho de código index.xhtml

```
<ui:composition template="/template/template.xhtml">
    <ui:define name="titulo">TCC</ui:define>
    <ui:define name="conteudo">

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

Fonte: do Autor

As páginas de listagens de registros da aplicação web, possuem botões que proporcionam ações de criação, edição e exclusão de registros. Pode-se selecionar a ordem de visualização dos registros por via de caixas de seleção, e existem campos que possibilitam a aplicação de filtros relacionados aos registros e a quantidade de registros exibidos por página. Por fim, as páginas de listagens também incluem botões de paginação, esses botões possibilitam a navegação entre as páginas como visto na figura 25:

Figura 25: Trecho de código listagem de cidade

```
<p:messages/>
<p:commandButton value="Novo" icon="ui-icon-plus"
                 actionListener="#{controleCidade.novo()}"
                 update="formEdicao formListagem"/>
<p:dataTable value="#{controleCidade.dao.listaObjetos}" var="obj"
             reflow="true"
             rows="20" id="listagem">
    <f:facet ...23 linhas />
    <f:facet ...11 linhas />
    <p:column headerText="ID">
        <p:outputLabel value="#{obj.id}"/>
    </p:column>
    <p:column headerText="Nome">
        <p:outputLabel value="#{obj.nome}"/>
    </p:column>
    <p:column headerText="Estado">
        <p:outputLabel value="#{obj.estado.nome}"/>
    </p:column>
    <p:column headerText="Ações">
        <div align="center">
            <p:commandButton icon="ui-icon-pencil"
                             actionListener="#{controleCidade.alterar(obj.id)}"
                             process="@form"
                             update=":formEdicao formListagem"/>
            <p:commandButton icon="ui-icon-trash"
                             actionListener="#{controleCidade.excluir(obj.id)}"
                             process="@form" update=":formListagem"/>
        </div>
    </p:column>
</p:dataTable>
```

Fonte: do Autor

No caso das inclusões e edições de registros na lista, utilizou-se o arquivo “formulario.xhtml” para realizar essas ações. Por meio dos formulários elaborados nesse arquivo pode-se incluir novos registros ou editar os dados de algum registro que já consta no sistema. Para que se tornasse possível integrar os formulários de edição aos formulários do arquivo “listar.xhtml”, utilizou-se o elemento *<ui:include>* dentro do código de “listar.xhtml”. A Figura 26 apresenta um trecho de código do arquivo “formulário.xhtml”:

Figura 26: Trecho de código formulário de edição de cidade

```
<h:form id="formEdicao">
    <h:panelGroup rendered="#{controleCidade.editando}">
        <div class="ui-fluid">
            <p:messages/>
            <p:panelGrid columns="2" columnClasses="ui-grid-col-2, ui-grid-col-20"
                         layout="grid" styleClass="ui-panelgrid-blank">
                <f:facet name="header">
                    <p:outputLabel value="Edição de Cidades"/>
                </f:facet>
                <p:outputLabel for="txtID" value="ID"/>
                <p:inputText id="txtID" value="#{controleCidade.objeto.id}" readonly="true"
                             size="10"/>
                <p:outputLabel for="txtNome" value="Nome"/>
                <p:inputText id="txtNome" value="#{controleCidade.objeto.nome}"
                             size="40" maxlength="40"
                             required="true"
                             placeholder="Obrigatório"/>
                <p:outputLabel for="txtEmail" value="Estado"/>
                <p:selectOneMenu id="txtEmail" value="#{controleCidade.objeto.estado}">
                    <f:converter converterId="converterEstado"/>
                    <f:selectItem itemLabel="Selecione um registro" noSelectionOption="true"/>
                    <f:selectItems value="#{controleCidade.estadoDAO.listaTodos}"
                                   var="s" itemLabel="#{s.nome}"/>
                </p:selectOneMenu>
                <p:commandButton value="Salvar" icon="ui-icon-disk"
                                 actionListener="#{controleCidade.salvar()}"
                                 update="formEdicao :formListagem"/>
            </p:panelGrid>
        </div>
    </h:panelGroup>
</h:form>
```

Fonte: do Autor

5 RESULTADOS

Com o objetivo de otimizar o processo da gestão manual da empresa VD, criou-se um sistema web para auxiliar nessa tarefa. Inicialmente apresenta-se a tela de navegação da aplicação, ela é composta por links que possuem a função de voltar a tela inicial, acessar os cadastros. O sistema desenvolvido possui um nível de acesso, o usuário administrador. O usuário administrador pode executar qualquer ação na aplicação. O usuário do sistema é um usuário administrador, então por sua vez pode alterar, criar e deletar dados como visto na Figura 27:

Figura 27: Listagem da tabela cidade com os botões de alterar excluir e criar

Fonte: do Autor

O cadastro de vendas refere-se as vendas feitas pela empresa VD para seus clientes, onde tem a disponibilidade de editar, alterar ou excluir uma venda. Nesses cadastros de vendas também podem ser adicionados itens a venda e no momento de salvar cria-se uma conta a receber. Já no caso das compras, são as compras feitas pela empresa com seu fornecedor e gera automaticamente uma conta a pagar como visto na Figura 28:

Figura 28: Listagem da tabela de vendas

Fonte: do Autor

O sistema também possui duas tabelas para a visualização das contas, uma para as contas a receber e a outra para as contas a pagar. Nessas tabelas o usuário pode visualizar as contas que ainda não foram pagas, possuindo um botão onde o usuário pode alterar entre as contas pagas e as não pagas, fazendo assim uma forma mais fácil de visualizar suas contas. Também possui um botão para informar se ouve o pagamento de uma determinada conta, como visto na Figura 29 e na Figura 30.

Figura 29: Formulário da tabela de vendas

Fonte: do Autor

Figura 30: Formulário da tabela de vendas com o formulário de itens da venda

Fonte: do Autor

Ao clicar no botão novo ou editar de alguma tabela, será aberto o formulário para preenchimento. No caso do botão novo, o formulário apresenta-se em branco, já no caso do botão de edição o formulário será apresentado com os dados carregados do banco de dados, conforme pode-se visualizar na Figura 31.

Figura 31: Formulário da tabela de vendas com o formulário de itens da venda com dados

Fonte: do Autor

Se o usuário optar pela exclusão do registro deve clicar no botão excluir para executar esta ação, a ação será executada e posteriormente a tela será atualizada com os registros que permanecem cadastrados. Na tela de formulário, pode-se preencher os dados a determinada tabela. Mostrado na figura 32:

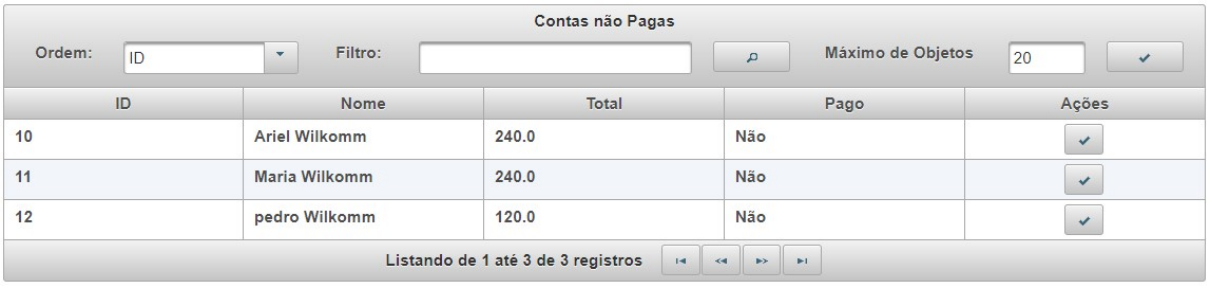
Figura 32: Tabela lista de itens da venda

Fonte: do Autor

O formulário de vendas e de compras possui algumas peculiaridades similares e que merecem ser descritas. Como por exemplo quando o usuário do sistema clica em nova venda em alterar uma venda o formulário aparece e logo abaixo dele

possui duas tabelas uma para os itens da venda e uma para as contas geradas por aquela venda. O usuário ao clicar em novo item desaparece a tabela e aparece mais uma parte do formulário onde ele pode selecionar o produto e a quantidade, informando um valor quando estiver comprando o item do fornecedor e recebendo um valor por parâmetro quando estiver vendendo o produto ao cliente. Ao clicar no botão salvar item o formulário do item some e aparece a tabela novamente com o item desejado. Então com isso pode-se salvar a venda ou a compra e será gerado uma conta com o dia, valor, e que a conta não foi paga, para respectiva ação. Como visto na figura 33, onde está sendo alterado o dado para mostrar a conta a receber e os item daquela venda.

Figura 33: Contas a receber não pagas



Fonte: do Autor

Durante o processo de desenvolvimento do ERP, procurou-se a resolução de todas as questões relacionadas ao estudo de caso, resolvendo erros da gestão manual e passando a possuir um sistema para melhor controle dos pedidos.

6 CONSIDERAÇÕES FINAIS

O presente estudo teve como objetivo o desenvolvimento de um ERP para pedidos da empresa Valdemar detergentes. A aplicação apresentada buscou reduzir a processo manual e otimizar o controle de pedidos. Após a implementação da aplicação obteve-se um resultado positivo, pois com a aplicação a empresa terá controle mais efetivo de suas compras e venda, controle dos clientes e de suas compras a pagar ou a receber.

Destaca-se que as atividades de levantamento de requisitos e coleta de dados, foram realizadas juntamente com o dono da empresa para entender melhor o funcionamento e métodos utilizados na empresa. Seguidamente realizou-se a modelagem de acordo com o estudo de caso realizado anteriormente. Por fim, realizou-se o desenvolvimento da solução web, contendo métodos específicos para realizar as operações necessárias ao objetivo de facilitar os processos da geração de pedidos da empresa.

Deste modo, o desenvolvimento da aplicação deu-se pela integração de variadas tecnologias, tais como: a plataforma Java EE, que disponibiliza recursos através de containers EJB e possibilita a criação de interfaces com layout responsivo, o layout do sistema foi elaborado com o auxílio das bibliotecas PrimeFaces e BootsFaces. Também se utilizou a API JPA com o servidor de persistência Hibernate, o JavaServer Faces e o banco de dados objeto relacional PostgreSQL.

Com o desenvolvimento desta monografia foi possível aplicar os conhecimentos da plataforma Java EE e suas tecnologias como o JSF. Outro ponto relevante foram as práticas de atividades de levantamento de requisitos e analise de estudos de caso, possibilitando a aquisição de conhecimento mais amplo sobre esses temas.

Após a conclusão do projeto não foi possível implementar algumas funcionalidades. Entre elas pode-se listar geração de boletos e geração de relatórios. Tal fato se deve a fala de tempo hábil para a conclusão destas funcionalidades até o momento da apresentação do trabalho.

Por fim, as funcionalidades propostas no presente estudo, foram implementadas com sucesso, mas seria plausível implementar uma interface para o

cliente da empresa utilizar e fazer testes da aplicação com a empresa, para que seja aprovado. Seria interessante desenvolver um sistema de geração de relatórios para que a empresa possa imprimir as contas a receber, as vendas a realizadas e as vendas realizadas em um determinado período. Como desenvolvimentos futuros fazer a parte de pedidos para a geração de boletos, por parte do escritório de contabilidade. Também poderia ser desenvolvido em um próximo estudo o desenvolvimento de um E-commerce baseado na modelagem mostrada no presente trabalho.

REFERÊNCIAS

BAUER, Christian; KING, Gavin. **Java persistence com hibernate.** Rio de Janeiro, RJ: Ciência Moderna, 2007. 844 p.

CAMPOS, Ronaldo Ribeiro de. **Características se sistema integrado de gestão empresarial desenvolvido sobre modelo de software livre:** informação para suporte à fase de seleção e viabilidade de instalação em pequenas empresas. São Paulo, São Carlos, 2006.

DEITEL, Paul; DEITEL, Harvey. **Java:** como programar. 8. ed. São Paulo, SP: Pearson, 2010. 1144 p.

DENONCOURT, Don. **Introduction to Java Server Faces: create a simple JSF application with WDSc 7.0**. Disponivel em <http://go-galegroup.ez131.periodicos.capes.gov.br/ps/i.do?&id=GALE|A176902406&v=2.1&u=capes&it=r&p=AONE&sw=w>. Acesso em 24 de maio de 2017.

FARIA, Thiago. **Java EE 7:** com JSF, PrimeFaces, CDI. 2013. 199 p.

GEREMIAS, Felipe; PAPIOR, Luiz Henrique. **Resumo de TCC** “Desenvolvimento de um sistema ERP com foco nas tecnologias de software livre / código aberto”. UNISUL. 2006.

GONCALVES, Antônio. **Introdução à plataforma Java (TM) EE 6 com o glassFish (TM) 3.** 2. ed. Rio de Janeiro: Ciência Moderna, 2011. 563 p.

GONÇALVES, Edson. **Desenvolvendo aplicações web com JSP, Servlets, Java Server Faces, Hibernate, EJB3 Persistence e Ajax.** Rio de Janeiro, RJ: Ciência Moderna, 2007. 735 p.

GUEDES, Gilleanes T. A. **UML 2:** uma abordagem prática. 2. ed. São Paulo, SP: Novatec, 2011. 484 p.

JAVA. **O que é java?**. Disponível em <https://www.java.com/pt_BR/about/whatis_java.jsp>. Acesso em 19 de maio de 2017.

JONNA, Sudheer. **Learning PrimeFaces Extensions Development. Packt Publishing**, 2014.

PEREIRA NETO, Álvaro. **PostgreSQL técnicas avançadas:** versões Open Source 7.x e 8.x: soluções para desenvolvedores e administradores de bancos de dados. 4. ed. São Paulo, SP: Érica, 2007. 284 p.

POSTGRESQL. **About**. Disponível em < https://www.postgresql.org/about/>. Acesso em 17 de maio de 2017.

PRIMEFACES. **Why PrimeFaces**. Disponível em <https://www.primefaces.org/whyprimefaces/>. Acesso em 22 de maio de 2017.

TOSTES, Luís Eduardo Fernandes Rogério. **Análise da implementação de sistema ERP em pequenas empresas auxiliada por software livre.** São Paulo, São Carlos, 2009.